STOCK MARKET ANALYSIS USING PYTHON

A Project Report of Capstone Project - 2

Submitted by

SIDDHANT SAURABH

1613112047/16SCSE112051

in partial fulfillment for the award of the degree

of

Bachelor of Technology

IN

Computer Science and Engineering With Specialization in Data Analytics

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

Under the Supervision of

Mr. M. Vivek Anand, M.E.

Assistant Professor

APRIL/MAY-2020

GALGOTIAS UNIVERSITY

SCHOOL OF COMPUTING AND SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project report "STOCK MARKET ANALYSIS USING PYTHON" is the bonafide work of "SIDDHANT SAURABH" who carried out the project work under my supervision.

SIGNATURE OF HEAD OF DEPARTMENT

Dr. MUNISH SHABARWAL

PhD (Management), PhD (CS)

Professor & Dean

School of Computer Science and Engineering

SIGNATURE OF THE SUPERVISOR

Mr. M. Vivek Anand, M.E.

Professor

School of Computer Science and Engineering

TABLE OF CONTENTS

ABSTRACT

Stock analysis is the technique utilized by a merchant or financial specialist to look at and assess the securities exchange. It is then used to settle on educated choices about purchasing and selling shares. Stock analysis can likewise be alluded to as market investigation, or value analysis. Stock analysis can be utilized to increase knowledge of the economy overall, the securities exchange, a particular division or an individual stock.

Stock analysis depends on the possibility that by contemplating market information from over a wide span of time, brokers can make a technique for picking which stocks to concentrate on, just as an approach to distinguish sections and leave focuses for their exchanges.

Importance of Stock Analysis

It is extremely important to carry out a comprehensive research work before making an investment. It is only after in-depth research work, you can evaluate or predict the future performance of a share, specific sector or the stock market. Even if you are going through the stock market tips, then also it is imperative to perform a thorough research just to have a great peace of mind that the investment you are planning to undertake will yield profitable returns or not.

When you are buying shares, then you are purchasing some portions of the business with an expectation to make profits if there is an increase in the business value. Before buying a cloth or phone, you may be carrying out research to analyze their quality and performance. Similarly, when you are taking a stock market investment decision, then you must ensure that your hard-earned money is invested in the right place and does not go wasted.

Types of stock analysis

Although stock analysis can take different forms, there are two main types that traders tend to favour. These are:

Technical analysis, which takes a gander at the recorded value outlines of an advantage, and studies past market designs so as to anticipate future developments. Merchants will utilize key devices, similar to help and obstruction lines, to find out market patterns

Fundamental analysis, which takes a gander at information from the organization and from its macroeconomic condition to evaluate potential benefits from exchanges. It centers around information sources that are accessible to people in general, for example, an organization's monetary record and income streams

The two assortments of stock analysis have the equivalent planned result: to settle on the right purchasing and selling choices and pick the ideal occasions to put exchanges.

A few dealers will commit the entirety of their opportunity to specialized examination, while different merchants may adhere to a simply crucial investigation of business sectors. Nonetheless, it is entirely expected to utilize a blend of the two.

INTRODUCTION

What is the Stock Market?

The stock market refers to the collection of markets and exchanges where regular activities of buying, selling, and issuance of shares of publicly-held companies take place. Such financial activities are conducted through institutionalized formal exchanges or over-the-counter (OTC) marketplaces which operate under a defined set of regulations. There can be multiple stock trading venues in a country or a region which allow transactions in stocks and other forms of securities.

While both terms - stock market and stock exchange - are used interchangeably, the latter term is generally a subset of the former. If one says that she trades in the stock market, it means that she buys and sells shares/equities on one (or more) of the stock exchange(s) that are part of the overall stock market. The leading stock exchanges in the U.S. include the New York Stock Exchange (NYSE), Nasdaq, and the Chicago Board Options Exchange (CBOE). These leading national exchanges, along with several other exchanges operating in the country, form the stock market of the U.S.

Though it is called a stock market or equity market and is primarily known for trading stocks/equities, other financial securities - like exchange traded funds (ETF), corporate bonds and derivatives based on stocks, commodities, currencies, and bonds - are also traded in the stock markets.

Understanding the Stock Market

While today it is possible to purchase almost everything online, there is usually a designated market for every commodity. For instance, people drive to city outskirts and farmlands to purchase Christmas trees, visit the local timber market to buy wood and other necessary material for home furniture and renovations, and go to stores like Walmart for their regular grocery supplies.

Such dedicated markets serve as a platform where numerous buyers and sellers meet, interact and transact. Since the number of market participants is huge, one is assured of a fair price. For example, if there is only one seller of Christmas trees in the entire city, he will have the liberty to charge any price he pleases as the buyers won't have anywhere else to go. If the number of tree sellers is large in a common marketplace, they will have to compete against each other to attract buyers. The buyers will be spoiled for choice with low- or optimum-pricing making it a fair market with price transparency. Even while shopping online, buyers compare prices offered by different sellers on the same shopping portal or across different portals to get the best deals, forcing the various online sellers to offer the best price.

A stock market is a similar designated market for trading various kinds of securities in a controlled, secure and managed environment. Since the stock market brings together hundreds of thousands of market participants who wish to buy and sell shares, it ensures fair pricing practices and transparency in transactions. While earlier stock markets used to issue and deal in paper-based physical share certificates, the modern day computer-aided stock markets operate electronically.

How the Stock Market Works?

In a nutshell, stock markets provide a secure and regulated environment where market participants can transact in shares and other eligible financial instruments with confidence with zero- to low-operational

risk. Operating under the defined rules as stated by the regulator, the stock markets act as primary markets and as secondary markets.

As a primary market, the stock market allows companies to issue and sell their shares to the common public for the first time through the process of initial public offerings (IPO). This activity helps companies raise necessary capital from investors. It essentially means that a company divides itself into a number of shares (say, 20 million shares) and sells a part of those shares (say, 5 million shares) to the common public at a price (say, $10 per share).

To facilitate this process, a company needs a marketplace where these shares can be sold. This marketplace is provided by the stock market. If everything goes as per the plans, the company will successfully sell the 5 million shares at a price of $10 per share and collect $50 million worth of funds. Investors will get the company shares which they can expect to hold for their preferred duration, in anticipation of rising share price and any potential income in the form of dividend payments. The stock exchange acts as a facilitator for this capital raising process and receives a fee for its services from the company and its financial partners.

Following the first-time share issuance IPO exercise called the listing process, the stock exchange also serves as the trading platform that facilitates regular buying and selling of the listed shares. This constitutes the secondary market. The stock exchange earns a fee for every trade that occurs on its platform during the secondary market activity.

The stock exchange shoulders the responsibility of ensuring price transparency, liquidity, price discovery and fair dealings in such trading activities. As almost all major stock markets across the globe now operate electronically, the exchange maintains trading systems that efficiently manage the buy and sell orders from various market participants. They perform the price matching function to facilitate trade execution at a price fair to both buyers and sellers.

A listed company may also offer new, additional shares through other offerings at a later stage, like through rights issue or through follow-on offers. They may even buy back or delist their shares. The stock exchange facilitates such transactions.

The stock exchange often creates and maintains various market-level and sector-specific indicators, like the S&P 500 index or Nasdaq 100 index, which provide a measure to track the movement of the overall market. Other methods include the Stochastic Oscillator and Stochastic Momentum Index.

The stock exchanges also maintain all company news, announcements, and financial reporting, which can be usually accessed on their official websites. A stock exchange also supports various other corporate-level, transaction-related activities. For instance, profitable companies may reward investors by paying dividends which usually comes from a part of the company’s earnings. The exchange maintains all such information and may support its processing to a certain extent.

Functions of a Stock Market

A stock market primarily serves the following functions:

Fair Dealing in Securities Transactions: Depending on the standard rules of demand and supply, the stock exchange needs to ensure that all interested market participants have instant access to data for all buy and sell orders thereby helping in the fair and transparent pricing of securities. Additionally, it should also perform efficient matching of appropriate buy and sell orders.

For example, there may be three buyers who have placed orders for buying Microsoft shares at $100, $105 and $110, and there may be four sellers who are willing to sell Microsoft shares at $110, $112, $115 and $120. The exchange (through their computer operated automated trading systems) needs to ensure that the best buy and best sell are matched, which in this case is at $110 for the given quantity of trade.

Efficient Price Discovery: Stock markets need to support an efficient mechanism for price discovery, which refers to the act of deciding the proper price of a security and is usually performed by assessing market supply and demand and other factors associated with the transactions.

Say, a U.S.-based software company is trading at a price of $100 and has a market capitalization of $5 billion. A news item comes in that the EU regulator has imposed a fine of $2 billion on the company which essentially means that 40 percent of the company's value may be wiped out. While the stock market may have imposed a trading price range of $90 and $110 on the company's share price, it should efficiently change the permissible trading price limit to accommodate for the possible changes in the share price, else shareholders may struggle to trade at a fair price.

Liquidity Maintenance: While getting the number of buyers and sellers for a particular financial security are out of control for the stock market, it needs to ensure that whosoever is qualified and willing to trade gets instant access to place orders which should get executed at the fair price.

Security and Validity of Transactions: While more participants are important for efficient working of a market, the same market needs to ensure that all participants are verified and remain compliant with the necessary rules and regulations, leaving no room for default by any of the parties. Additionally, it should ensure that all associated entities operating in the market must also adhere to the rules, and work within the legal framework given by the regulator.

Support All Eligible Types of Participants: A marketplace is made by a variety of participants, which include market makers, investors, traders, speculators, and hedgers. All these participants operate in the stock market with different roles and functions. For instance, an investor may buy stocks and hold them for long term spanning many years, while a trader may enter and exit a position within seconds. A market maker provides necessary liquidity in the market, while a hedger may like to trade in derivatives for mitigating the risk involved in investments. The stock market should ensure that all such participants are able to operate seamlessly fulfilling their desired roles to ensure the market continues to operate efficiently.

Investor Protection: Along with wealthy and institutional investors, a very large number of small investors are also served by the stock market for their small amount of investments. These investors may have limited financial knowledge, and may not be fully aware of the pitfalls of investing in stocks and other listed instruments. The stock exchange must implement necessary measures to offer the necessary protection to such investors to shield them from financial loss and ensure customer trust.

For instance, a stock exchange may categorize stocks in various segments depending on their risk profiles and allow limited or no trading by common investors in high-risk stocks. Exchanges often impose restrictions to prevent individuals with limited income and knowledge from getting into risky bets of derivatives.

Balanced Regulation: Listed companies are largely regulated and their dealings are monitored by market regulators, like the Securities and Exchange Commission (SEC) of the U.S. Additionally, exchanges also mandate certain requirements – like, timely filing of quarterly financial reports and instant reporting of any relevant developments - to ensure all market participants become aware of

corporate happenings. Failure to adhere to the regulations can lead to suspension of trading by the exchanges and other disciplinary measures.

Regulating the Stock Market

A local financial regulator or competent monetary authority or institute is assigned the task of regulating the stock market of a country. The Securities and Exchange Commission (SEC) is the regulatory body charged with overseeing the U.S. stock markets. The SEC is a federal agency that works independently of the government and political pressure. The mission of the SEC is stated as: "to protect investors, maintain fair, orderly, and efficient markets, and facilitate capital formation."

Stock Market Participants

Along with long-term investors and short term traders, there are many different types of players associated with the stock market. Each has a unique role, but many of the roles are intertwined and depend on each other to make the market run effectively.

- Stockbrokers, also known as registered representatives in the U.S., are the licensed professionals who buy and sell securities on behalf of investors. The brokers act as intermediaries between the stock exchanges and the investors by buying and selling stocks on the investors' behalf. An account with a retail broker is needed to gain access to the markets.
- Portfolio managers are professionals who invest portfolios, or collections of securities, for clients. These managers get recommendations from analysts and make the buy or sell decisions for the portfolio. Mutual fund companies, hedge funds, and pension plans use portfolio managers to make decisions and set the investment strategies for the money they hold.
- Investment bankers represent companies in various capacities, such as private companies that want to go public via an IPO or companies that are involved in pending mergers and acquisitions. They take care of the listing process in compliance with the regulatory requirements of the stock market.
- Custodian and depot service providers, which are institutions holding customers' securities for safekeeping so as to minimize the risk of their theft or loss, also operate in sync with the exchange to transfer shares to/from the respective accounts of transacting parties based on trading on the stock market.
- Market maker: A market maker is a broker-dealer who facilitates the trading of shares by posting bid and ask prices along with maintaining an inventory of shares. He ensures sufficient liquidity in the market for a particular (set of) share(s), and profits from the difference between the bid and the ask price he quotes.

How Stock Exchanges Make Money?

Stock exchanges operate as for-profit institutes and charge a fee for their services. The primary source of income for these stock exchanges are the revenues from the transaction fees that are charged for each trade carried out on its platform. Additionally, exchanges earn revenue from the listing fee charged to companies during the IPO process and other follow-on offerings.

The exchange also earns from selling market data generated on its platform - like real-time data, historical data, summary data, and reference data – which is vital for equity research and other uses. Many exchanges will also sell technology products, like a trading terminal and dedicated network connection to the exchange, to the interested parties for a suitable fee.

The exchange may offer privileged services like high-frequency trading to larger clients like mutual funds and asset management companies (AMC), and earn money accordingly. There are provisions for

regulatory fee and registration fee for different profiles of market participants, like the market maker and broker, which form other sources of income for the stock exchanges.

The exchange also makes profits by licensing their indexes (and their methodology) which are commonly used as a benchmark for launching various products like mutual funds and ETFs by AMCs.

Many exchanges also provide courses and certification on various financial topics to industry participants and earn revenues from such subscriptions.

Competition for Stock Markets

While individual stock exchanges compete against each other to get maximum transaction volume, they are facing threat on two fronts.

Dark Pools: Dark pools, which are private exchanges or forums for securities trading and operate within private groups, are posing a challenge to public stock markets. Though their legal validity is subject to local regulations, they are gaining popularity as participants save big on transaction fees.

Blockchain Ventures: Amid rising popularity of blockchains, many crypto exchanges have emerged. Such exchanges are venues for trading cryptocurrencies and derivatives associated with that asset class. Though their popularity remains limited, they pose a threat to the traditional stock market model by automating a bulk of the work done by various stock market participants and by offering zero- to low-cost services.

Significance of the Stock Market

The stock market is one of the most vital components of a free-market economy.

It allows companies to raise money by offering stock shares and corporate bonds. It lets common investors participate in the financial achievements of the companies, make profits through capital gains, and earn money through dividends, although losses are also possible. While institutional investors and professional money managers do enjoy some privileges owing to their deep pockets, better knowledge and higher risk taking abilities, the stock market attempts to offer a level playing field to common individuals.

The stock market works as a platform through which savings and investments of individuals are channelized into the productive investment proposals. In the long term, it helps in capital formation & economic growth for the country.

Examples of Stock Markets

The first stock market in the world was the London stock exchange. It was started in a coffeehouse, where traders used to meet to exchange shares, in 1773. The first stock exchange in the United States of America was started in Philadelphia in 1790. The Buttonwood agreement, so named because it was signed under a buttonwood tree, marked the beginnings of New York's Wall Street in 1792. The agreement was signed by 24 traders and was the first American organization of its kind to trade in securities. The traders renamed their venture as New York Stock and Exchange Board in 1817.

What is Stock Market Analysis?

Stock market analysis enables investors to identify the intrinsic worth of a security even before investing in it. All stock market tips are formulated after thorough research by experts. Stock analysts try to find out activity of an instrument/sector/market in future.

By using stock analysis, investors and traders arrive at equity buying and selling decisions. Studying and evaluating past and current data helps investors and traders to gain an edge in the markets to make informed decisions. Fundamental Research and Technical Research are two types of research used to first analyse and then value a security.

Why is Stock Market Analysis important?

Performing a research before making an investment is a must. It is only after a thorough research that you can make some assumptions into the value and future performance of an investment. Even if you are following stock trading tips, it ideal to do some research, just to ensure that you are making an investment that's expected to get you maximum returns.

When you invest in equity, you purchase some portions of a business expecting to make money upon increase in the value of the business. Before buying anything, be it a car or phone, you do some degree of research about its performance and quality. An investment is no different. It is your hard earned money that you are about to invest, so you must have a fair knowledge of what you are investing in.

What is Fundamental Research?

In fundamental research, you try to find out value of an equity share using the information provided in the financial statements of the company. The investor tries to analyse various aspects of the business like competitive advantage, financial soundness, quality of management and competition. The main aim is to ascertain the relative attractiveness of the underlying business.

Here, it is assumed that the market price doesn't reflect the true value of the company due to some uncontrollable external factors like investor sentiments. As the market will attain equilibrium, the real value will be equal to its market price in the long run. It believes that paying a higher price for a stock will affect return on investment adversely. Thus, by means of financial ratios, investors try to arrive at the true value at which a stock should ideally trade in the market.

Which key indicators are used in Fundamental Research?

Financial ratios form the pillars of fundamental research. Some of them are as follows:

Return On Equity (ROE)

Return On Equity tells you about how much does a company earns on shareholders' equity. It gives you information apart from a simple profit figure. It shows whether the operation of the company are efficient or not.

*Return On Equity = [(Income – Preference Dividend)/ (Average Shareholders' Equity)]*100*

While looking for this metric, an ideal ROE is one which is consistent, high and increasing. ROE of one company can be compared with its own past performance and with performance of other companies within the same industry. You may use it irrespective of the type of industry.

Debt-Equity Ratio (DER)

Debt-Equity Ratio shows the proportion of assets which is being used to finance the assets of the company. It indicates how much funds have been provided by the borrowers and owners of the company. This ratio can be expressed in numbers and in percentage.

Debt-Equity (D/E) Ratio = Total Debt/Total Equity

While looking for a debt-equity ratio, go for the ones which are lower than others and are decreasing in a consistent manner. You can compare D/E of one company with its own past performance and with performance of other companies within the same industry. You may use it to analyse performance of capital intensive industries like capital goods, metals, oil and gas.

Earning Per Share (EPS)

Earning Per Share is one such useful measure which the investors look for all the time. It shows the amount of money which the company is earning on every share. EPS of a company needs to increase in a consistent manner to show superior management performance.

Earning Per Share = (Net Income – Preference Dividend)/Weighted Average Number of Shares Outstanding

EPS of one company can be compared with its past performance and with that of other companies in the same industry. It can be used to ascertain what portion of profit is the company allocating to each outstanding share. Investors usually go for companies which have steadily increasing earnings per share. It can be easily used to compare performance across industries.

Price to Earning Ratio (PER)

Price to Earning Ratio compares the current market price of the share with the earnings per share. It tells you the price which the investors are willing to pay for the share depending on the current earnings.

Price to Earning Ratio = Current Share Price/Earning Per Share

This ratio also indicates the number of years that will be required to get back the initial invested capital by way of returns. You need to look for stocks which have a low price to earnings ratio. You can easily compare the P/E ratio of a company with its past performance and also with other companies operational in the same industry. Ideally, this ratio is suitable to analyse performance of companies present in the FMCG, pharmaceutical and technology sector.

What is Technical Research?

Technical research relates to the study of past stock prices to predict the trend of prices in future. It shows you the direction of movement of the share prices. With the help of technical research, you can identify whether there will be sharp rise or fall in the price of share. It is not dependent on recent news or events which have already been incorporated in the price of the share.

As the stock prices are dependent on investor psychology which keeps changing according to news and events, technical research emphasises the use of Stop-losses. It will save investors from suffering a big loss in future. Technical research gives meaningful results only for stocks which are high in demand and traded in huge volumes.

Technical research uses different types of charts like bar charts, candlestick charts; to understand the pattern of stock prices. Daily charts are used by short term traders to examine the immediate movement in the stock prices. Weekly / monthly charts are used by medium/long term traders to ascertain the probability to earn higher more in the long run.

EXISTING SYSTEM

Machine learning

With the advent of the digital computer, stock market prediction has since moved into the technological realm. The most prominent technique involves the use of artificial neural networks (ANNs) and Genetic Algorithms(GA). Scholars found that the bacterial chemotaxis optimization method may perform better than GA. ANNs can be thought of as mathematical function approximators. The most common form of ANN in use for stock market prediction is the feed forward network utilizing the backward propagation of errors algorithm to update the network weights. These networks are commonly referred to as Backpropagation networks. Another form of ANN that is more appropriate for stock prediction is the time recurrent neural network (RNN) or time delay neural network (TDNN). Examples of RNN and TDNN are the Elman, Jordan, and Elman-Jordan networks. (See the Elman And Jordan Networks)..

For stock prediction with ANNs, there are usually two approaches taken for forecasting different time horizons: independent and joint. The independent approach employs a single ANN for each time horizon, for example, 1-day, 2-day, or 5-day. The advantage of this approach is that network forecasting error for one horizon won't impact the error for another horizon—since each time horizon is typically a unique problem. The joint approach, however, incorporates multiple time horizons together so that they are determined simultaneously. In this approach, forecasting error for one time horizon may share its error with that of another horizon, which can decrease performance. There are also more parameters required for a joint model, which increases the risk of overfitting.

Of late, the majority of academic research groups studying ANNs for stock forecasting seem to be using an ensemble of independent ANNs methods more frequently, with greater success. An ensemble of ANNs would use low price and time lags to predict future lows, while another network would use lagged highs to predict future highs. The predicted low and high predictions are then used to form stop prices for buying or selling. Outputs from the individual "low" and "high" networks can also be input into a final network that would also incorporate volume, intermarket data or statistical summaries of prices, leading to a final ensemble output that would trigger buying, selling, or market directional change. A major finding with ANNs and stock prediction is that a classification approach (vs. function approximation) using outputs in the form of buy(y=+1) and sell(y=-1) results in better predictive reliability than a quantitative output such as low or high price.

Since NNs require training and can have a large parameter space; it is useful to optimize the network for optimal predictive ability.

Data sources for market prediction

Tobias Preis et al. introduced a method to identify online precursors for stock market moves, using trading strategies based on search volume data provided by Google Trends. Their analysis of Google search volume for 98 terms of varying financial relevance, published in *Scientific Reports*, suggests that increases in search volume for financially relevant search terms tend to precede large losses in financial markets. Out of these terms, three were significant at the 5% level ($|z| > 1.96$). The best term in the negative direction was "debt", followed by "color".

In a study published in *Scientific Reports* in 2013, Helen Susannah Moat, Tobias Preis and colleagues demonstrated a link between changes in the number of views of English Wikipedia articles relating to financial topics and subsequent large stock market moves.

The use of Text Mining together with Machine Learning algorithms received more attention in the last years, with the use of textual content from the Internet as input to predict price changes in Stocks, Stocks and other financial markets.

The collective mood of Twitter messages has been linked to stock market performance. The study, however, has been criticized for its methodology.

The activity in stock message boards has been mined in order to predict asset returns. The enterprise headlines from Yahoo! Finance and Google Finance were used as news feeding in a Text mining process, to forecast the Stocks price movements from the Dow Jones Industrial Average.

Market mimicry

Using new statistical analysis tools of complexity theory, researchers at the New England Complex Systems Institute (NECSI) performed research on predicting stock market crashes. It has long been thought that market crashes are triggered by panics that may or may not be justified by external news. This research indicates that it is the internal structure of the market, not external crises, which is primarily responsible for crashes. The number of different stocks that move up or down together were shown to be an indicator of the mimicry within the market, how much investors look to one another for cues. When the mimicry is high, many stocks follow each other's movements - a prime reason for panic to take hold. It was shown that a dramatic increase in market mimicry occurred during the entire year before each market crash of the past 25 years, including the financial crisis of 2007–08.

Time series aspect structuring

Aspect structuring, also referred to as Jacaruso Aspect Structuring (JAS) is a trend forecasting method which has been shown to be valid for anticipating trend changes on various stock market and geopolitical time series datasets. The method addresses the challenge that arises with high dimensional data in which exogenous variables are too numerous or immeasurable to be accounted for and used to make a forecast. The method identifies the single variable of primary influence on the time series, or "primary factor", and observes trend changes that occur during times of decreased significance in the said primary variable. Presumably, trend changes in these instances are instead due to so-called "background factors". Although this method cannot elucidate the multivariate nature of background factors, it can gauge the effects they have on the time-series at a given point in time even without measuring them. This observation can be used to make a forecast.

PROPOSED SYSTEM

Downloading the Data

We will be using data from the following sources:
1. Alpha Vantage. Before you start, however, you will first need an API key, which we can obtain for free here. After that, we can assign that key to the api_key variable. 2. Use the data from this page. we will need to copy the *Stocks* folder in the zip file to your project home folder.

Stock prices come in several different flavours. They are,

• Open: Opening stock price of the day

• Close: Closing stock price of the day

• High: Highest stock price of the data

• Low: Lowest stock price of the day

Getting Data from Alphavantage

We will first load in the data from Alpha Vantage. Since you're going to make use of the American Airlines Stock market prices to make your predictions, you set the ticker to "AAL". Additionally, you also define a url_string, which will return a JSON file with all the stock market data for American Airlines within the last 20 years, and a file_to_save, which will be the file to which you save the data. You'll use the ticker variable that you defined beforehand to help name this file.

Next, you're going to specify a condition: if you haven't already saved data, you will go ahead and grab the data from the URL that you set in url_string; You'll store the date, low, high, volume, close, open values to a pandas DataFrame df and you'll save it to file_to_save. However, if the data is already there, you'll just load it from the CSV.

Getting Data from Kaggle Data found on Kaggle is a collection of csv files and you don't have to do any preprocessing, so you can directly load the data into a Pandas DataFrame.

Data Exploration Here you will print the data you collected in to the DataFrame. You should also make sure that the data is sorted by date, because the order of the data is crucial in time series modelling.

Splitting Data into a Training set and a Test set

You will use the mid price calculated by taking the average of the highest and lowest recorded prices on a day. Now you can split the training data and test data. The training data will be the first 11,000 data points of the time series and rest will be test data.
Normalizing the Data Now you need to define a scaler to normalize the data. MinMaxScalar scales all the data to be in the region of 0 and 1. You can also reshape the training and test data to be in the shape [data_size, num_features].

Due to the observation you made earlier, that is, different time periods of data have different value ranges, you normalize the data by splitting the full series into windows. If you don't do this, the earlier data will be close to 0 and will not add much value to the learning process. Here you choose a window size of 2500.

Tip: when choosing the window size make sure it's not too small, because when you perform windowed-normalization, it can introduce a break at the very end of each window, as each window is normalized independently.

In this example, 4 data points will be affected by this. But given you have 11,000 data points, 4 points will not cause any issue Reshape the data back to the shape of [data_size]

You can now smooth the data using the exponential moving average. This helps you to get rid of the inherent raggedness of the data in stock prices and produce a smoother curve.

Note that you should only smooth training data.

One-Step Ahead Prediction via Averaging

Averaging mechanisms allow you to predict (often one time step ahead) by representing the future stock price as an average of the previously observed stock prices. Doing this for more than one time step can produce quite bad results. You will look at two averaging techniques below; standard averaging and exponential moving average. You will evaluate both qualitatively (visual inspection) and quantitatively (Mean Squared Error) the results produced by the two algorithms.

The Mean Squared Error (MSE) can be calculated by taking the Squared Error between the true value at one step ahead and the predicted value and averaging it over all the predictions.

Standard Average

You can understand the difficulty of this problem by first trying to model this as an average calculation problem. First you will try to predict the future stock market prices (for example, xt+1) as an average of the previously observed stock market prices within a fixed size window (for example, xt-N, ..., xt) (say previous 100 days). Thereafter you will try a bit more fancier "exponential moving average" method and see how well that does. Then you will move on to the "holy-grail" of time-series prediction; Long Short-Term Memory models.

In other words, you say the prediction at t+1t+1 is the average value of all the stock prices you observed within a window of tt to t−Nt−N.
t seems that it is not too bad of a model for very short predictions (one day ahead). Given that stock prices don't change from 0 to 100 overnight, this behavior is sensible. Next, you will look at a fancier averaging technique known as exponential moving average.

Exponential Moving Average

You might have seen some articles on the internet using very complex models and predicting almost the exact behavior of the stock market. But **beware!** These are just optical illusions and not due to learning something useful. You will see below how you can replicate that behavior with a simple averaging method.

In the exponential moving average method, you calculate xt+1xt+1 as,

• $x_{t+1} = EMA_t = \gamma \times EMA_{t-1} + (1-\gamma)\, x_t$ where $EMA_0 = 0$ and EMA is the exponential moving average value you maintain over time.

The above equation basically calculates the exponential moving average from t+1t+1 time step and uses that as the one step ahead prediction. γγ decides what the contribution of the most recent prediction is to the EMA. For example, a γ=0.1γ=0.1 gets only 10% of the current value into the EMA. Because you take only a very small fraction of the most recent, it allows to preserve much older values you saw very early in the average.

If Exponential Moving Average is this Good, Why do You Need Better Models? You see that it fits a perfect line that follows the True distribution (and justified by the very low MSE). Practically speaking, you can't do much with just the stock market value of the next day. Personally what I'd like is not the exact stock market price for the next day, but *would the stock market prices go up or down in the next 30 days*. Try to do this, and you will expose the incapability of the EMA method.

You will now try to make predictions in windows (say you predict the next 2 days window, instead of just the next day). Then you will realize how wrong EMA can go. Here is an example:

Predict More Than One Step into the Future

To make things concrete, let's assume values, say xt=0.4xt=0.4, EMA=0.5EMA=0.5 and γ=0.5γ=0.5

• Say you get the output with the following equation o $X_{t+1} = EMA_t = \gamma \times EMA_{t-1} + (1 - \gamma)X_t$ o So you have $x_{t+1}=0.5\times0.5+(1-0.5)\times0.4=0.45$ o So $x_{t+1}=EMA_t=0.45$

• So the next prediction x_{t+2} becomes, o $X_{t+2} = \gamma \times EMA_t + (1-\gamma)X_{t+1}$ o Which is $x_{t+2}=\gamma\times EMA_t+(1-\gamma)EMA_t=EMA_t$

o Or in this example, $X_{t+2} = X_{t+1} = 0.45$

So no matter how many steps you predict in to the future, you'll keep getting the same answer for all the future prediction steps.

One solution you have that will output useful information is to look at **momentum-based algorithms**. They make predictions based on whether the past recent values were going up or going down (not the exact values). For example, they will say the next day price is likely to be lower, if the prices have been dropping for the past days, which sounds reasonable. However, you will use a more complex model: an LSTM model.

These models have taken the realm of time series prediction by storm, because they are so good at modelling time series data. You will see if there actually are patterns hidden in the data that you can exploit.

Introduction to LSTMs: Making Stock Movement Predictions Far into the Future
Long Short-Term Memory models are extremely powerful time-series models. They can predict an arbitrary number of steps into the future. An LSTM module (or cell) has 5 essential components which allows it to model both long-term and short-term data.

• Cell state (ct) - This represents the internal memory of the cell which stores both short term memory and long-term memories

• Hidden state (ht) - This is output state information calculated w.r.t. current input, previous hidden state and current cell input which you eventually use to predict the future stock market prices. Additionally, the hidden state can decide to only retrive the short or long-term or both types of memory stored in the cell state to make the next prediction.
• Input gate (it) - Decides how much information from current input flows to the cell state
• Forget gate (ft) - Decides how much information from the current input and the previous cell state flows into the current cell state
• Output gate (ot) - Decides how much information from the current cell state flows into the hidden state, so that if needed LSTM can only pick the long-term memories or short-term memories and long-term memories

And the equations for calculating each of these entities are as follows.

• $it = \sigma(W{ix}xt + W{ih}h_{t-1}+b_i)$

• $\tilde{c}t = \sigma(W{cx}xt + W{ch}h_{t-1} + b_c)$

• $ft = \sigma(W{fx}xt + W{fh}h_{t-1}+b_f)$

• $c_t = ft c{t-1} + i_t \tilde{c}_t$

• $ot = \sigma(W{ox}xt + W{oh}h_{t-1}+b_o)$
• ht=ottanh(ct)ht=ottanh(ct)

For a better (more technical) understanding about LSTMs you can refer to this article.

TensorFlow provides a nice sub API (called RNN API) for implementing time series models. You will be using that for your implementations.

Data Generator

You are first going to implement a data generator to train your model. This data generator will have a method called .unroll_batches(...) which will output a set of *num_unrollings* batches of input data obtained sequentially, where a batch of data is of size [batch_size, 1]. Then each batch of input data will have a corresponding output batch of data.

For example if num_unrollings=3 and batch_size=4 a set of unrolled batches it might look like,
• input data: [x0,x10,x20,x30],[x1,x11,x21,x31],[x2,x12,x22,x32][x0,x10,x20,x30],[x1,x11,x21,x31],[x2,x12,x22,x32]
• output data: [x1,x11,x21,x31],[x2,x12,x22,x32],[x3,x13,x23,x33][x1,x11,x21,x31],[x2,x12,x22,x32],[x3,x13,x23,x33]

Data Augmentation

Also to make your model robust you will not make the output for $xtalwaysalways{t+1}.Rather you will randomly sample an output from the set.

x{t+1},x{t+2},\ldots,x_{t+N} where N is a small window size.

Here you are making the following assumption:

• $x{*t+1*},*x*{t+2},\ldots,x_{t+N}will not be very far from each other

CODE:-

```
# Make sure that you have all these libaries available to run the code successfully

from pandas_datareader import data
import matplotlib.pyplot as plt
import as pd
import datetime as dt
import urllib.request
import os
import numpy as np
import tensorflow as tf # This code has been tested with TensorFlow 1.6

from sklearn.preprocessing import MinMaxscaler

data_source='kaggle'# alphavantage or kaggle


if data_source=='alphavantage'

# ====================== Loading Data from Alpha Vantage ==================================

url_string=
"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=%s&outputsize=full&apikey=%
s" %tkinter, api_key
# Save data to this
file_to_save= 'stock_market_data-%s.csv %ticker

# If you haven't already saved data,
# Go ahead and grab the data from the url
# And store date, low, high, volume, close, open values to a Pandas DataFrame
if not os.path.exists(file_to_save):
 with urllib.request.urlopen(url_string) as url:
 data = json.loads(url.read().decode())
 # extract stock market data
 data = data['Time Series (Daily)']
 df = pd.DataFrame(columns=['Date','Low','High','Close','Open'])
 for k,v in data.items():
 date = dt.datetime.strptime(k, '%Y-%m-%d')
 data_row = [date.date(),float(v['3. low']),float(v['2. high']),
 float(v['4. close']),float(v['1. open'])]
 df.loc[-1,:] = data_row
 df.index = df.index + 1
 print('Data saved to : %s'%file_to_save)
 df.to_csv(file_to_save)

 # If the data is already there, just load it from the CSV
 else:
 print('File already exists. Loading data from CSV')
 df = pd.read_csv(file_to_save)
```

```
else:

 # ====================== Loading Data from Kaggle ==================================

 # You will be using HP's data. Feel free to experiment with other data.
 # But while doing so, be careful to have a large enough dataset and also pay attention to the data normalization
 df = pd.read_csv(os.path.join('Stocks','hpq.us.txt'),delimiter=',',usecols=['Date','Open','High','Low','Close'])

 print('Loaded data from the Kaggle repository')


# Sort DataFrame by date
df = df.sort_values('Date')

# Double check the result

df.head()

plt.figure(figsize = (18,9))
plt.plot(range(df.shape[0]),(df['Low']+df['High'])/2.0)
plt.xticks(range(0,df.shape[0],500),df['Date'].loc[::500],rotation=45)
plt.xlabel('Date',fontsize=18)
plt.ylabel('Mid Price',fontsize=18)

plt.show()

# First calculate the mid prices from the highest and lowest
high_prices = df.loc[:,'High'].as_matrix()
low_prices = df.loc[:,'Low'].as_matrix()

mid_prices = (high_prices+low_prices)/2.0

train_data = mid_prices[:11000]

test_data = mid_prices[11000:]


# Scale the data to be between 0 and 1
# When scaling remember! You normalize both test and train data with respect to training data
# Because you are not supposed to have access to test data
scaler = MinMaxScaler()
train_data = train_data.reshape(-1,1)

test_data = test_data.reshape(-1,1)


# Train the Scaler with training data and smooth data
smoothing_window_size = 2500
for di in range(0,10000,smoothing_window_size):
 scaler.fit(train_data[di:di+smoothing_window_size,:])
 train_data[di:di+smoothing_window_size,:] = scaler.transform(train_data[di:di+smoothing_window_size,:])

# You normalize the last bit of remaining data
scaler.fit(train_data[di+smoothing_window_size:,:])

train_data[di+smoothing_window_size:,:] = scaler.transform(train_data[di+smoothing_window_size:,:])
```

```
# Reshape both train and test data
train_data = train_data.reshape(-1)

# Normalize test data

test_data = scaler.transform(test_data).reshape(-1)


# Now perform exponential moving average smoothing
# So the data will have a smoother curve than the original ragged data
EMA = 0.0
gamma = 0.1
for ti in range(11000):
 EMA = gamma*train_data[ti] + (1-gamma)*EMA
 train_data[ti] = EMA

# Used for visualization and test purposes

all_mid_data = np.concatenate([train_data,test_data],axis=0)


window_size = 100
N = train_data.size
std_avg_predictions = []
std_avg_x = []
mse_errors = []

for pred_idx in range(window_size,N):

 if pred_idx >= N:
 date = dt.datetime.strptime(k, '%Y-%m-%d').date() + dt.timedelta(days=1)
 else:
 date = df.loc[pred_idx,'Date']

 std_avg_predictions.append(np.mean(train_data[pred_idx-window_size:pred_idx]))
 mse_errors.append((std_avg_predictions[-1]-train_data[pred_idx])**2)
 std_avg_x.append(date)


print('MSE error for standard averaging: %.5f'%(0.5*np.mean(mse_errors)))


plt.figure(figsize = (18,9))
plt.plot(range(df.shape[0]),all_mid_data,color='b',label='True')
plt.plot(range(window_size,N),std_avg_predictions,color='orange',label='Prediction')
#plt.xticks(range(0,df.shape[0],50),df['Date'].loc[::50],rotation=45)
plt.xlabel('Date')
plt.ylabel('Mid Price')
plt.legend(fontsize=18)

plt.show()


window_size = 100
N = train_data.size
```

```
run_avg_predictions = []
run_avg_x = []

mse_errors = []

running_mean = 0.0
run_avg_predictions.append(running_mean)

decay = 0.5

for pred_idx in range(1,N):

 running_mean = running_mean*decay + (1.0-decay)*train_data[pred_idx-1]
 run_avg_predictions.append(running_mean)
 mse_errors.append((run_avg_predictions[-1]-train_data[pred_idx])**2)
 run_avg_x.append(date)


print('MSE error for EMA averaging: %.5f'%(0.5*np.mean(mse_errors)))


plt.figure(figsize = (18,9))
plt.plot(range(df.shape[0]),all_mid_data,color='b',label='True')
plt.plot(range(0,N),run_avg_predictions,color='orange', label='Prediction')
#plt.xticks(range(0,df.shape[0],50),df['Date'].loc[::50],rotation=45)
plt.xlabel('Date')
plt.ylabel('Mid Price')
plt.legend(fontsize=18)
plt.show()
class DataGeneratorSeq(object):

 def __init__(self,prices,batch_size,num_unroll):
 self._prices = prices
 self._prices_length = len(self._prices) - num_unroll
 self._batch_size = batch_size
 self._num_unroll = num_unroll
 self._segments = self._prices_length //self._batch_size
 self._cursor = [offset * self._segments for offset in range(self._batch_size)]

 def next_batch(self):

 batch_data = np.zeros((self._batch_size),dtype=np.float32)
 batch_labels = np.zeros((self._batch_size),dtype=np.float32)

 for b in range(self._batch_size):
 if self._cursor[b]+1>=self._prices_length:
 #self._cursor[b] = b * self._segments
 self._cursor[b] = np.random.randint(0,(b+1)*self._segments)

 batch_data[b] = self._prices[self._cursor[b]]
 batch_labels[b]= self._prices[self._cursor[b]+np.random.randint(0,5)]

 self._cursor[b] = (self._cursor[b]+1)%self._prices_length
```

```
 return batch_data,batch_labels

 def unroll_batches(self):

 unroll_data,unroll_labels = [],[]
 init_data, init_label = None,None
 for ui in range(self._num_unroll):

 data, labels = self.next_batch()

 unroll_data.append(data)
 unroll_labels.append(labels)

 return unroll_data, unroll_labels

 def reset_indices(self):
 for b in range(self._batch_size):
 self._cursor[b] = np.random.randint(0,min((b+1)*self._segments,self._prices_length-1))



dg = DataGeneratorSeq(train_data,5,5)
u_data, u_labels = dg.unroll_batches()

for ui,(dat,lbl) in enumerate(zip(u_data,u_labels)):
 print('\n\nUnrolled index %d'%ui)
 dat_ind = dat
 lbl_ind = lbl
 print('\tInputs: ',dat )
 print('\n\tOutput:',lbl)


D = 1 # Dimensionality of the data. Since your data is 1-D this would be 1
num_unrollings = 50 # Number of time steps you look into the future.
batch_size = 500 # Number of samples in a batch
num_nodes = [200,200,150] # Number of hidden nodes in each layer of the deep LSTM stack we're using
n_layers = len(num_nodes) # number of layers
dropout = 0.2 # dropout amount


tf.reset_default_graph() # This is important in case you run this multiple times


# Input data.
train_inputs, train_outputs = [],[]

# You unroll the input over time defining placeholders for each time step
for ui in range(num_unrollings):
 train_inputs.append(tf.placeholder(tf.float32, shape=[batch_size,D],name='train_inputs_%d'%ui))
 train_outputs.append(tf.placeholder(tf.float32, shape=[batch_size,1], name = 'train_outputs_%d'%ui))
```

```
lstm_cells = [
 tf.contrib.rnn.LSTMCell(num_units=num_nodes[li],
 state_is_tuple=True,
 initializer= tf.contrib.layers.xavier_initializer()
 )
 for li in range(n_layers)]

drop_lstm_cells = [tf.contrib.rnn.DropoutWrapper(
 lstm, input_keep_prob=1.0,output_keep_prob=1.0-dropout, state_keep_prob=1.0-dropout
) for lstm in lstm_cells]
drop_multi_cell = tf.contrib.rnn.MultiRNNCell(drop_lstm_cells)
multi_cell = tf.contrib.rnn.MultiRNNCell(lstm_cells)

w = tf.get_variable('w',shape=[num_nodes[-1], 1], initializer=tf.contrib.layers.xavier_initializer())

b = tf.get_variable('b',initializer=tf.random_uniform([1],-0.1,0.1))


# Create cell state and hidden state variables to maintain the state of the LSTM
c, h = [],[]
initial_state = []
for li in range(n_layers):
 c.append(tf.Variable(tf.zeros([batch_size, num_nodes[li]]), trainable=False))
 h.append(tf.Variable(tf.zeros([batch_size, num_nodes[li]]), trainable=False))
 initial_state.append(tf.contrib.rnn.LSTMStateTuple(c[li], h[li]))

# Do several tensor transofmations, because the function dynamic_rnn requires the output to be of
# a specific format. Read more at: https://www.tensorflow.org/api_docs/python/tf/nn/dynamic_rnn
all_inputs = tf.concat([tf.expand_dims(t,0) for t in train_inputs],axis=0)

# all_outputs is [seq_length, batch_size, num_nodes]
all_lstm_outputs, state = tf.nn.dynamic_rnn(
 drop_multi_cell, all_inputs, initial_state=tuple(initial_state),
 time_major = True, dtype=tf.float32)

all_lstm_outputs = tf.reshape(all_lstm_outputs, [batch_size*num_unrollings,num_nodes[-1]])

all_outputs = tf.nn.xw_plus_b(all_lstm_outputs,w,b)


split_outputs = tf.split(all_outputs,num_unrollings,axis=0)


# When calculating the loss you need to be careful about the exact form, because you calculate
# loss of all the unrolled steps at the same time
# Therefore, take the mean error or each batch and get the sum of that over all the unrolled steps

print('Defining training Loss')
loss = 0.0
with tf.control_dependencies([tf.assign(c[li], state[li][0]) for li in range(n_layers)]+
 [tf.assign(h[li], state[li][1]) for li in range(n_layers)]):
 for ui in range(num_unrollings):
 loss += tf.reduce_mean(0.5*(split_outputs[ui]-train_outputs[ui])**2)

print('Learning rate decay operations')
```

```
global_step = tf.Variable(0, trainable=False)
inc_gstep = tf.assign(global_step,global_step + 1)
tf_learning_rate = tf.placeholder(shape=None,dtype=tf.float32)
tf_min_learning_rate = tf.placeholder(shape=None,dtype=tf.float32)

learning_rate = tf.maximum(
 tf.train.exponential_decay(tf_learning_rate, global_step, decay_steps=1, decay_rate=0.5, staircase=True),
 tf_min_learning_rate)

# Optimizer.
print('TF Optimization operations')
optimizer = tf.train.AdamOptimizer(learning_rate)
gradients, v = zip(*optimizer.compute_gradients(loss))
gradients, _ = tf.clip_by_global_norm(gradients, 5.0)
optimizer = optimizer.apply_gradients(
 zip(gradients, v))


print('\tAll done')


print('Defining prediction related TF functions')

sample_inputs = tf.placeholder(tf.float32, shape=[1,D])

# Maintaining LSTM state for prediction stage
sample_c, sample_h, initial_sample_state = [],[],[]
for li in range(n_layers):
 sample_c.append(tf.Variable(tf.zeros([1, num_nodes[li]]), trainable=False))
 sample_h.append(tf.Variable(tf.zeros([1, num_nodes[li]]), trainable=False))
 initial_sample_state.append(tf.contrib.rnn.LSTMStateTuple(sample_c[li],sample_h[li]))

reset_sample_states = tf.group(*[tf.assign(sample_c[li],tf.zeros([1, num_nodes[li]])) for li in range(n_layers)],
 *[tf.assign(sample_h[li],tf.zeros([1, num_nodes[li]])) for li in range(n_layers)])

sample_outputs, sample_state = tf.nn.dynamic_rnn(multi_cell, tf.expand_dims(sample_inputs,0),
 initial_state=tuple(initial_sample_state),
 time_major = True,
 dtype=tf.float32)

with tf.control_dependencies([tf.assign(sample_c[li],sample_state[li][0]) for li in range(n_layers)]+
 [tf.assign(sample_h[li],sample_state[li][1]) for li in range(n_layers)]):
 sample_prediction = tf.nn.xw_plus_b(tf.reshape(sample_outputs,[1,-1]), w, b)


print('\tAll done')


epochs = 30
valid_summary = 1 # Interval you make test predictions

n_predict_once = 50 # Number of steps you continously predict for

train_seq_length = train_data.size # Full length of the training data
```

```
train_mse_ot = [] # Accumulate Train losses
test_mse_ot = [] # Accumulate Test loss
predictions_over_time = [] # Accumulate predictions

session = tf.InteractiveSession()

tf.global_variables_initializer().run()

# Used for decaying learning rate
loss_nondecrease_count = 0
loss_nondecrease_threshold = 2 # If the test error hasn't increased in this many steps, decrease learning rate

print('Initialized')
average_loss = 0

# Define data generator
data_gen = DataGeneratorSeq(train_data,batch_size,num_unrollings)

x_axis_seq = []

# Points you start your test predictions from
test_points_seq = np.arange(11000,12000,50).tolist()

for ep in range(epochs):

 # ======================== Training ====================================

 for step in range(train_seq_length//batch_size):

 u_data, u_labels = data_gen.unroll_batches()

 feed_dict = {}
 for ui,(dat,lbl) in enumerate(zip(u_data,u_labels)):
 feed_dict[train_inputs[ui]] = dat.reshape(-1,1)
 feed_dict[train_outputs[ui]] = lbl.reshape(-1,1)

 feed_dict.update({tf_learning_rate: 0.0001, tf_min_learning_rate:0.000001})

 _, l = session.run([optimizer, loss], feed_dict=feed_dict)

 average_loss += l

 # ============================ Validation ==============================

 if (ep+1) % valid_summary == 0:

 average_loss = average_loss/(valid_summary*(train_seq_length//batch_size))

 # The average loss
 if (ep+1)%valid_summary==0:
 print('Average loss at step %d: %f' % (ep+1, average_loss))

 train_mse_ot.append(average_loss)
```

```
average_loss = 0 # reset loss

predictions_seq = []

mse_test_loss_seq = []

# ==================== Updating State and Making Predicitons ========================

for w_i in test_points_seq:
mse_test_loss = 0.0
our_predictions = []

if (ep+1)-valid_summary==0:
# Only calculate x_axis values in the first validation epoch
x_axis=[]

# Feed in the recent past behavior of stock prices
# to make predictions from that point onwards
for tr_i in range(w_i-num_unrollings+1,w_i-1):
current_price = all_mid_data[tr_i]
feed_dict[sample_inputs] = np.array(current_price).reshape(1,1)
_ = session.run(sample_prediction,feed_dict=feed_dict)

feed_dict = {}

current_price = all_mid_data[w_i-1]

feed_dict[sample_inputs] = np.array(current_price).reshape(1,1)

# Make predictions for this many steps
# Each prediction uses previous prediciton as it's current input
for pred_i in range(n_predict_once):

pred = session.run(sample_prediction,feed_dict=feed_dict)

our_predictions.append(np.asscalar(pred))

feed_dict[sample_inputs] = np.asarray(pred).reshape(-1,1)

if (ep+1)-valid_summary==0:
# Only calculate x_axis values in the first validation epoch
x_axis.append(w_i+pred_i)

mse_test_loss += 0.5*(pred-all_mid_data[w_i+pred_i])**2

session.run(reset_sample_states)

predictions_seq.append(np.array(our_predictions))

mse_test_loss /= n_predict_once
mse_test_loss_seq.append(mse_test_loss)
```

```
if (ep+1)-valid_summary==0:
x_axis_seq.append(x_axis)
current_test_mse = np.mean(mse_test_loss_seq)

# Learning rate decay logic
if len(test_mse_ot)>0 and current_test_mse > min(test_mse_ot):
loss_nondecrease_count += 1
else:
loss_nondecrease_count = 0

if loss_nondecrease_count > loss_nondecrease_threshold :
session.run(inc_gstep)
loss_nondecrease_count = 0
print('\tDecreasing learning rate by 0.5')

test_mse_ot.append(current_test_mse)
print('\tTest MSE: %.5f'%np.mean(mse_test_loss_seq))
predictions_over_time.append(predictions_seq)

print('\tFinished Predictions')


best_prediction_epoch = 28 # replace this with the epoch that you got the best results when running the plotting code

plt.figure(figsize = (18,18))
plt.subplot(2,1,1)
plt.plot(range(df.shape[0]),all_mid_data,color='b')

# Plotting how the predictions change over time
# Plot older predictions with low alpha and newer predictions with high alpha
start_alpha = 0.25
alpha = np.arange(start_alpha,1.1,(1.0-start_alpha)/len(predictions_over_time[::3]))
for p_i,p in enumerate(predictions_over_time[::3]):
 for xval,yval in zip(x_axis_seq,p):
 plt.plot(xval,yval,color='r',alpha=alpha[p_i])

plt.title('Evolution of Test Predictions Over Time',fontsize=18)
plt.xlabel('Date',fontsize=18)
plt.ylabel('Mid Price',fontsize=18)
plt.xlim(11000,12500)

plt.subplot(2,1,2)

# Predicting the best test prediction you got
plt.plot(range(df.shape[0]),all_mid_data,color='b')
for xval,yval in zip(x_axis_seq,predictions_over_time[best_prediction_epoch]):
 plt.plot(xval,yval,color='r')

plt.title('Best Test Predictions Over Time',fontsize=18)
plt.xlabel('Date',fontsize=18)
plt.ylabel('Mid Price',fontsize=18)
plt.xlim(11000,12500)

plt.show()
```

OUTPUT/RESULT/SCREEN SHOT

Step 1: Choosing the data

One of the most important steps in machine learning and predictive modeling is gathering good data, performing the appropriate cleaning steps and realizing the limitations.

For this example I will be using stock price data from a single stock, Zimmer Biomet (ticker: ZBH). Simply go too finance.yahoo.com, search for the desired ticker. Once you are on the home page of the desired stock, simple navigate to the "Historical Data" tab, input the range of dates you would like to include, and select "Download Data." I chose 5 years, but you can choose as far back as you would like.

Now that we have out data, let's go ahead and see what we have. Simply open the file in Excel.

	1	2	3	4	5	6	7
1	Date	Open	High	Low	Close	Adj Close	Volume
2	11/11/13	88.519997	89.559998	88.410004	89.230003	85.394432	1151800
3	11/12/13	88.779999	89.309998	88.349998	89.25	85.413544	1102700
4	11/13/13	89.07	89.650002	88.519997	89.639999	85.786797	743000
5	11/14/13	89.470001	90.75	89.400002	90.300003	86.418427	1068800

Looks like we have some goodies here. You may notice that all of the fields are numerical values, except that pesky date value. We need to fix this. The values that we are going to pass into our model need to be in a format that can be most easily understood. So, we need to perform some "data preprocessing" steps. In our case we are going to insert a new column after 1, name it "Date Value," and copy all of the dates from column 1 into column 2. Then select all of the data and change the type from "Date" to "Text." The results should look like the following:

	1	2	3	4	5	6	7	8
1	Date	Date Value	Open	High	Low	Close	Adj Close	Volume
2	11/11/13	41589	88.519997	89.559998	88.410004	89.230003	85.394432	1151800
3	11/12/13	41590	88.779999	89.309998	88.349998	89.25	85.413544	1102700
4	11/13/13	41591	89.07	89.650002	88.519997	89.639999	85.786797	743000
5	11/14/13	41592	89.470001	90.75	89.400002	90.300003	86.418427	1068800

Ok, so now save the file as "choose_a_name.csv" (make sure it is a ".csv "and not one of the excel default formats).

Before we start, lets talk about limitations. You will notice that the only data we feed this model is date and price. There are many external factors that affect the price outside of the historical price. Highly robust models might utilize external data such as news, time of the year, social media sentiment, weather, price of competitors, market volatility, market indices, etc. This model is very basic, but in time you can learn the skills to build a model that is more "aware" of the overall marketplace. That being said, let's move on.

Step 2: Choosing the model

So now that we have data cleaned up, we need to choose a model. In this case we are going to use a neural network to perform a regression function. A regression will spit out a numerical value on a continuous scale, a apposed to a model that may be used for classification efforts, which would yield a categorical output. In this situation, we are trying to predict the price of a stock on any given day (and if you are trying to make money, a day that hasn't happened yet).

To build our model we are going to use TensorFlow… well, a simplified module called TFANN which stands for "TensorFlow Artificial Neural Network." In order to do this, we are going to use Google Colab. If you are not familiar with Colab, simply navigate to colab.research.google.com, it is a *free* virtual python notebook environment. (For those of you that will be following along and don't know what you are doing, just copy paste the code below into a "cell" and then hit run before creating a new one and copying more code).

Step 3: Building the Model

First we need to install TFANN. Open a new Colab notebook (python 3). Colab has numerous libraries which can be accessed without installation; however, TFANN is not one of them so we need to execute the following command:

```
pip install TFANN
```

Now let's import our dependencies:

```
import numpy as np
import matplotlib.pyplot as mpl
from sklearn.preprocessing import scale
from TFANN import ANNR
from google.colab import files
```

NumPy will be used for our matrix operations, Matplotlib for graphs, sykit-learn for data processing, TFANN for the ML goodness, and google.colab files will help us upload data from the local machine to the virtual environment.

Now we need to import the data that we have already processed. To do this we will execute the following command, which will provide us with a window to upload the .csv file.

```
files.upload()
```

Now we can finally get to the meat of this project. Execute the following commands:

```
#reads data from the file and ceates a matrix with only the dates and the prices
 stock_data = np.loadtxt('ZBH_5y.csv', delimiter=",", skiprows=1, usecols=(1, 4))
 #scales the data to smaller values
 stock_data=scale(stock_data)
 #gets the price and dates from the matrix
 prices = stock_data[:, 1].reshape(-1, 1)
 dates = stock_data[:, 0].reshape(-1, 1)
 #creates a plot of the data and then displays it
 mpl.plot(dates[:, 0], prices[:, 0])
```

You should get a nice graph that looks like this:

Note, that the scale is no longer in dollars on the y-axis and those arbitrary integer-date values on the x-axis. We have scaled the data down to make the learning process more effective. Try writing some code to return the scale of the y-axis back to dollars and the x-axis to years!

Now, we need to construct the model. In this case we will use *one* input and output neuron (input date, output price) and will have *three* hidden layers of 25 neurons each. Each layer will have an “tanh” activation function. If you do not understand these concepts, feel free to google it and come back, understanding the basics for neural network principals will be very helpful as you progress.

```
#Number of neurons in the input, output, and hidden layers
 input = 1
 output = 1
 hidden = 50
 #array of layers, 3 hidden and 1 output, along with the tanh activation function
 layers = [('F', hidden), ('AF', 'tanh'), ('F', hidden), ('AF', 'tanh'), ('F', hidden), ('AF', 'tanh'), ('F',
output)]
```

We have now initialized the model and are ready to train!

Step 4: Training the Model

```
number of days for the hold-out period used to access progress
 holdDays = 5
 totalDays = len(dates)
 #fit the model to the data "Learning"
 mlpr.fit(dates[0:(totalDays-holdDays)], prices[0:(totalDays-holdDays)])
```

Once the training is complete, we can execute the following commands to see how we did.

```
#Predict the stock price using the model
 price Predict = mlpr.predict(dates)
 #Display the predicted reuslts agains the actual data
 mpl.plot(dates, prices)
```

Let's think about some ways in which we can increase the fidelity of the model. We can think of about this as "what knobs can we turn" to tune our model. Well the first is to simply decrease the error tolerance.

The first trial, the error tolerance was set as .2; however, we can lower this to a smaller number, say .1, lets give that a try!
Simply make the following changes. Note that I am also updating the name of the variables so that the values we already created/observed do not change. Certainly not the most effective method here.

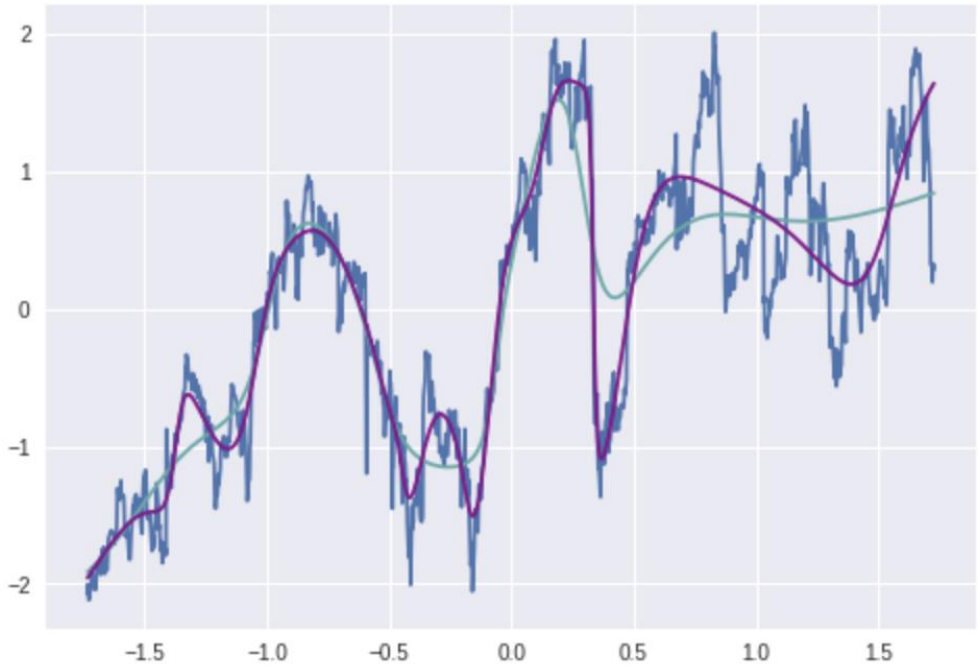
```
#Number of neurons in the input, output, and hidden layers
 input2 = 1
 output2 = 1
 hidden2 = 50
 #array of layers, 3 hidden and 1 output, along with the tanh activation function
 layers = [('F', hidden2), ('AF', 'tanh'), ('F', hidden2), ('AF', 'tanh'), ('F', hidden2), ('AF', 'tanh'), ('F',
output2)]
 #construct the model and dictate params
```

Run the model again with the following commands and we get new results:

```
holdDays = 5
totalDays = len(dates)
mlpr2.fit(dates[0:(totalDays-holdDays)], prices[0:(totalDays-holdDays)])
```

Once it has finished training:

```
pricePredict2 = mlpr2.predict(dates)
 mpl.plot(dates, prices)
 mpl.plot(dates, pricePredict, c='#5aa9ab')
 mpl.plot(dates, pricePredict2, c='#8B008B')
 mpl.show()
```



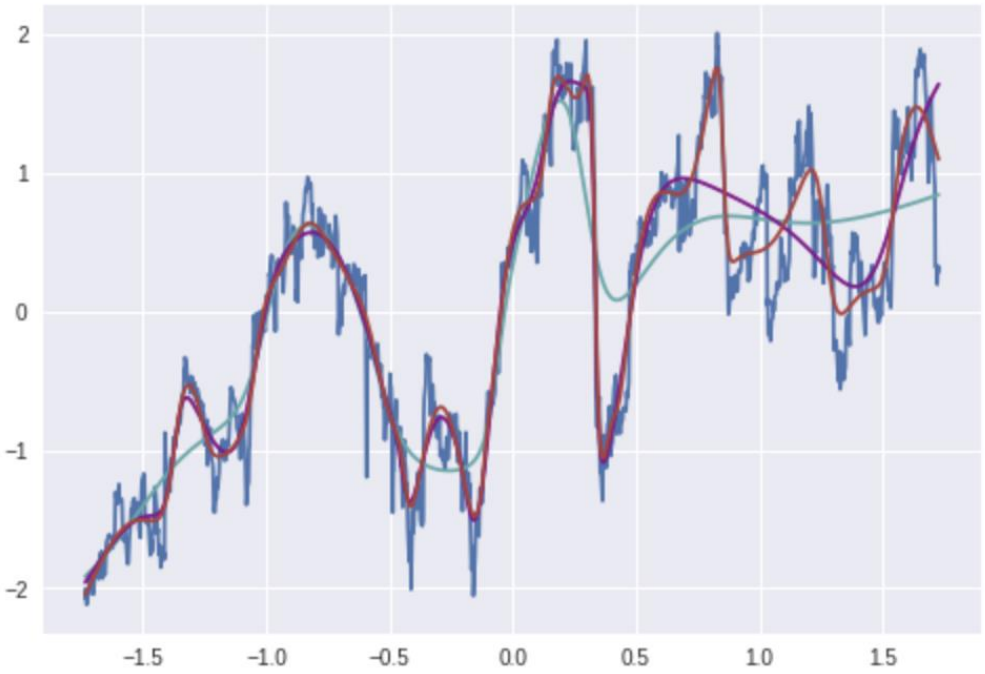
As you can see, lovering the error tolerance… well… lowered the error. So you might be wondering "why not just set the error to a really small number?" and that would be a great question. Go ahead and

try it for yourself, re-execute the code you just ran with the tolerance set at .05. What you will observe is that the maximum number of iterations you use will stop the execution before it reached the desired level of error. Ok then, why not just increase the maximum number of iterations? Well, the problem lies in the given model parameters. The model itself has limitations, the lowest achievable error for the model we constructed may only be .8 (I have not checked exactly for this for this model). In this situation, it does not matter how many more iterations you add, the structure of the model will not yield better results, no matter how many iterations are run. It is simply capped out.

The next logical question to ask here would be "how can we change the model to achieve greater error?" and that is what we are going to explore!

Models have what are known as "hyperparameters." These are the parameters that govern the model, they define how the model is created. Altering these can give us better (or perhaps worse) results. Examples include: number of neurons in each hidden layer, the number of hidden layers, the activation function, etc.

Our goal here is to "tune" these hyperparameters to achieve a lower error tolerance than was possible with our first model. The simplest way to do this, in my opinion, is do increase the number of neurons in the hidden layers. I am by no means a leading source of knowledge on this topic, but I will venture far enough to say that increasing the number of neurons and/or the number of hidden layers increases the level of abstraction with which the model can represent the given data. So lets give that a try!
Increasing the number of neurons in each hidden layer from 50 to 100 and setting the tolerance to .075:



Much much better! The orange line is out newest prediction. Notice how much better it tracks the more recent prices than the last model did.

Try changing the activation function to something besides "tanh", or perhaps adding an additional layer.

To add another layer, reference this line of code:

```
layers = [('F', hidden), ('AF', 'tanh'), ('F', hidden), ('AF', 'tanh'), ('F', hidden), ('AF', 'tanh'), ('F', output)]
```

Add one additional layer by adding another

(‘AF’, hidden), (‘AF’, ‘tanh’)

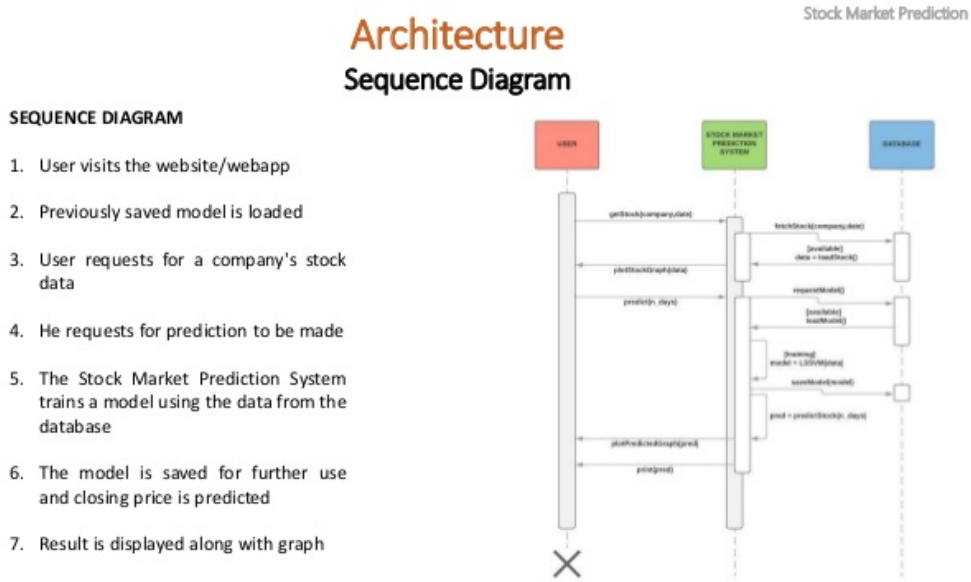
before the output node. This adds the layer and the activation function applied to it before it is fed into the next layer.

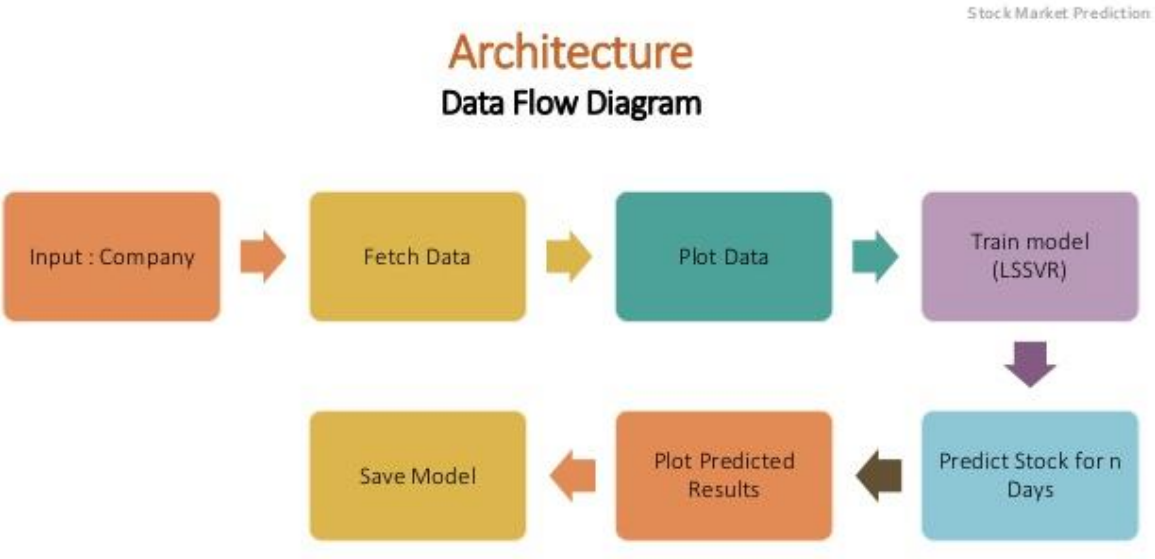
layers = [('F', hidden), ('AF', 'tanh'), ('F', hidden), ('AF', 'tanh'), ('F', hidden), ('AF', 'tanh'),('F', hidden), ('AF', 'tanh'), ('F', output)]

Or perhaps you want a different number of neurons at each hidden layer, tapering them down is a common method. The example below tapers from 100 down to 25 nodes before the output:

layers = [('F', 100), ('AF', 'tanh'), ('F', 50), ('AF', 'tanh'), ('F', 25), ('AF', 'tanh'), ('F', output)]

IMPLIMANTION AND ARCHITECTURE DIAGRAM



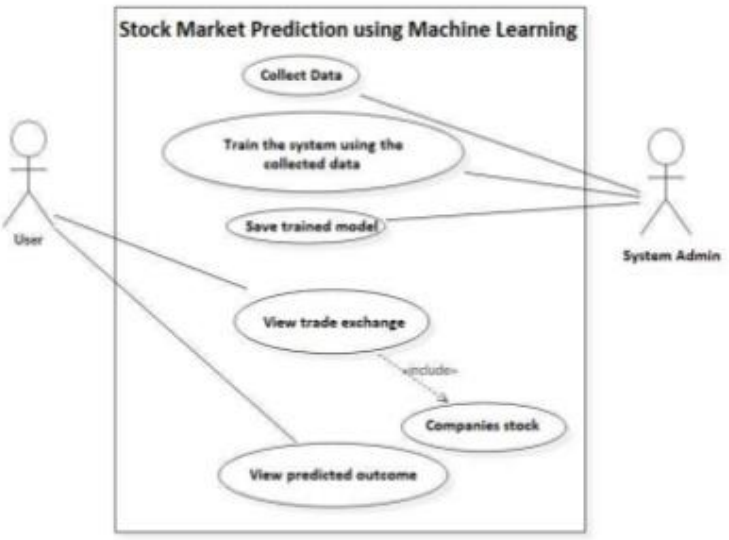


Architecture

System Design

USE CASE DIAGRAM

1. Data is initially collected from online sources or the stock exchange
2. The data is then used to train the system
3. Trained model is saved
4. User views the trade exchange and stock of a company
5. Using the model, closing prices are predicted



CONCLUSION

In this, we learnt how difficult it can be to device a model that is able to correctly predict stock price movements.We started with a motivation for why we need to model stock prices. This was followed by an explanation and code for downloading data. Then we looked at two averaging techniques that allow you to make predictions one step into the future. we next saw that these methods are futile when you need to predict more than one step into the future. Thereafter we discussed how you can use LSTMs to make predictions many steps into the future. Finally we visualized the results and saw that your model (though not perfect) is quite good at correctly predicting stock price movements.

Here, I'm stating several takeaways.

1. Stock price/movement prediction is an extremely difficult task. Personally I don't think any of the stock prediction models out there shouldn't be taken for granted and blindly rely on them. However models might be able to predict stock price movement correctly most of the time, but not always.

2. Do not be fooled by articles out there that shows predictions curves that perfectly overlaps the true stock prices. This can be replicated with a simple averaging technique and in practice it's useless. A more sensible thing to do is predicting the stock price movements.

3. The model's hyperparameters are extremely sensitive to the results you obtain. So a very good thing to do would be to run some hyperparameter optimization technique (for example, Grid search / Random search) on the hyperparameters. Below I listed some of the most critical hyperparameters

o The learning rate of the optimizer

o Number of layers and the number of hidden units in each layer

o The optimizer. I found Adam to perform the best

o Type of the model. You can try GRU/ Standard LSTM/ LSTM with Peepholes and evaluation performance difference

4. In this we did something faulty (due to the small size of data)! That is we used the test loss to decay the learning rate. This indirectly leaks information about test set into the training procedure. A better way of handling this is to have a separate validation set (apart from the test set) and decay learning rate with respect to performance of the validation set.

REFERENCES

- file:///C:/Users/Visitor/Desktop/Paper_109.pdf
- https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119197652.oth1
- http://thestockmarketwatch.com/learn/stocks-basics-conclusion/
- https://www.datacamp.com/community/tutorials/lstm-python-stock-market
- http://shodhganga.inflibnet.ac.in/bitstream/10603/25949/14/14_chapter%208.pdf
- https://link.springer.com/content/pdf/bbm%3A978-81-322-1590-5%2F1.pdf
- https://www.scribd.com/document/110661247/Conclusion
- https://www.pantechsolutions.net/stock-market-prediction-using-machine-learning
- https://www.ijeat.org/wp-content/uploads/papers/v8i4/D6321048419.pdf
- https://markdunne.github.io/public/mark-dunne-stock-market-prediction.pdf
- https://www.investopedia.com/terms/s/stock-analysis.asp