Introduction

The problem we are trying to solve is when to buy, sell, or hold a single stock. This is an issue that most people will face in their life whether it be in a 401k retirement account, individual investing account, or certified financial planners. There are an extensive amount of theories and strategies to try and time the market correct to maximize profits by buying low and selling high. Also, there are passive strategies that employ buying and holding over extended periods. We are trying to leverage machine learning techniques to find the ideal time to buy, sell, or hold. We were not worried about asset selection or asset allocation, which are separate problems. We prioritized risk mitigation – meaning we didn't care a whole lot about returns so as not to lose capital. Today, artificial intelligence and machine learning are at the forefront of finance. Currently, there are not many full autonomous stock trading systems currently in the market. Although, artificial intelligence is used to complement the analysis of financial data. Most intelligent portfolios like Wealthfront use algorithms in regards to MPT (modern portfolio theory) and use passive strategies to diversify clients' portfolios. One example exchange-traded fund (ETF) named BUZZ, uses AI/ML to gauge social sentiment on publicly traded stocks from social media like Twitter and Reddit.

We choose to use reinforcement learning methods to do our analysis because of the inherent nature of an agent traversing an environment with actions trying to maximize its reward. It parallels in the real world to an investor making trading actions in the stock market to try and maximize profit. We also selected to use a broad range of assets to test on because of their difference in implied volatility.

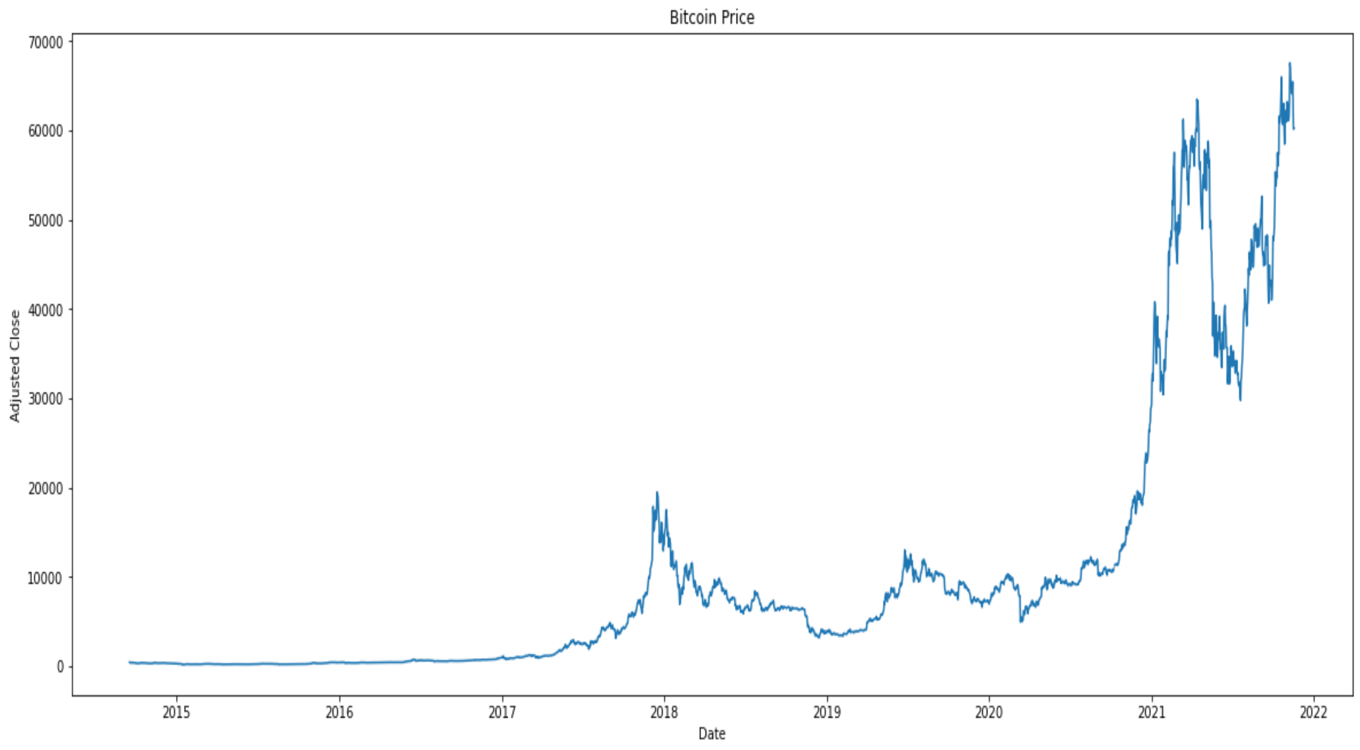
Data Collection

For our project, we obtained the equity data from the Yahoo Finance API. There are two ways to obtain this data, the y*finance* python library or manually through the Yahoo Finance website. From either source, the same historical data can be obtained from a selected date range and selected frequency, which can be daily, weekly, monthly, or yearly.

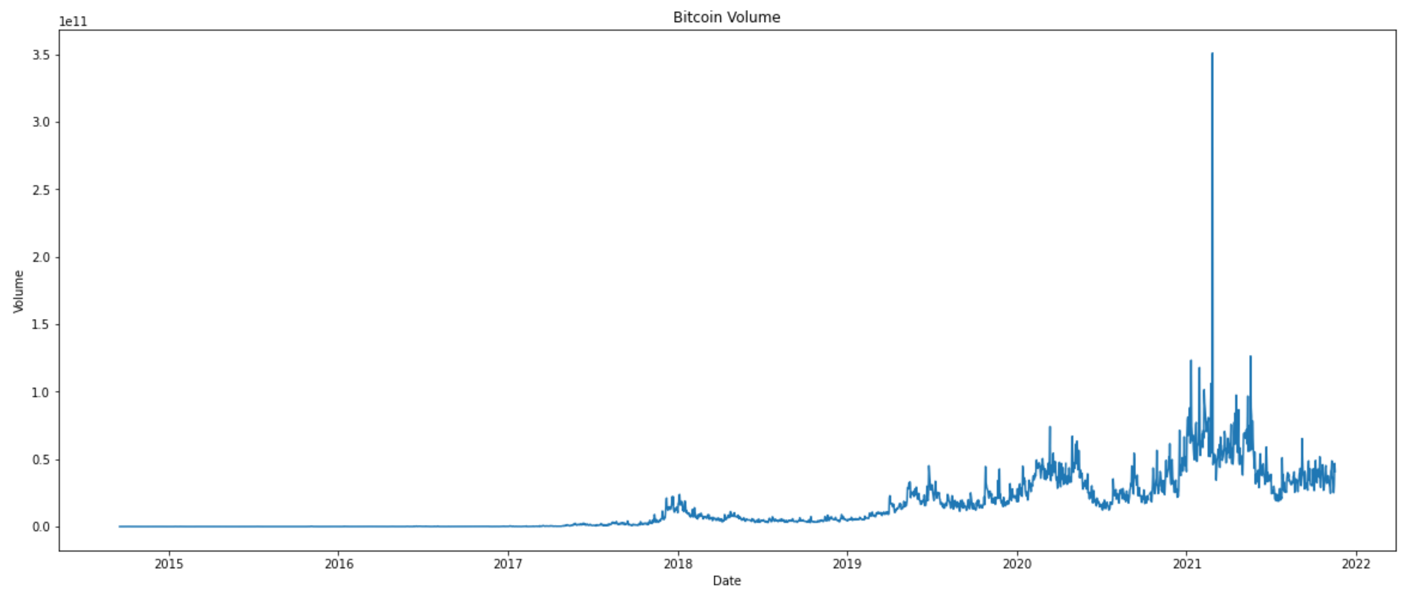
The data downloaded from Yahoo Finance contains the following columns;
Date: A DateTime object corresponding to the stock data, in the format of year-month-day. This column is incredibly useful when plotting time series data.
Open: The price point where the equity started trading for the chosen frequency.
High: The highest price point where the equity traded for the chosen frequency.
Low: The lowest price point where the equity traded for the chosen frequency.
Close: Price point the equity closed for the chosen frequency.
Adj Close: The adjusted close is much like the close column but it also factors in any adjustments that may have been made by executives during the trading period. These actions include stock splits, dividends, and the issuance of new stock.
Volume: The trading volume where the equity traded for the chosen frequency.

Exploratory Data Analysis

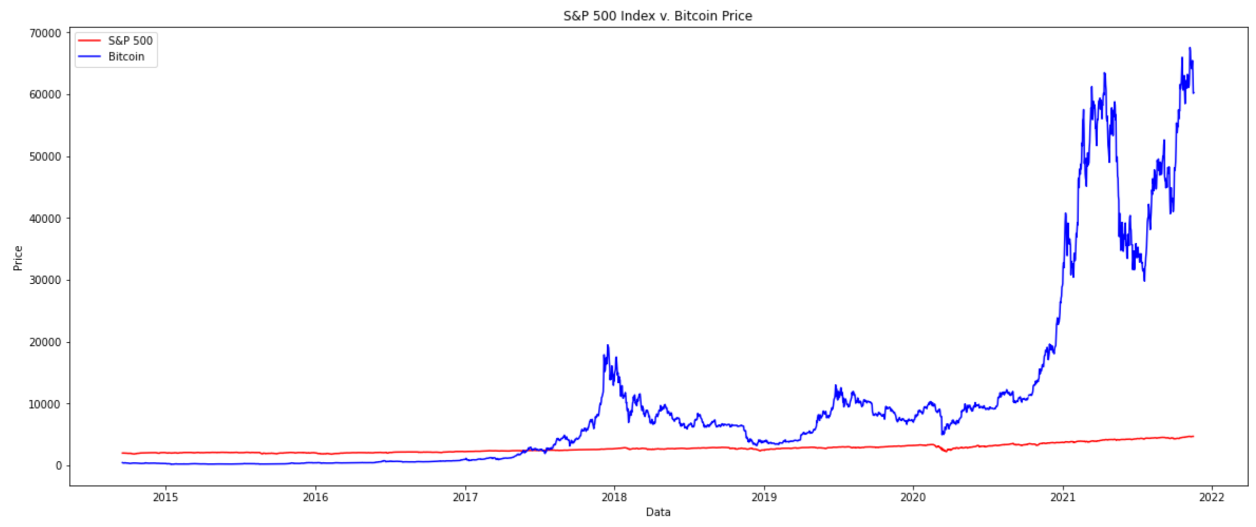
After obtaining our dataset, the obvious next step was to perform exploratory data analysis on it. When it comes to equity data, some of the measures to consider are the following;
Price: This may be the most important measure when it comes to equity data. The price represents what the valuation of the equity is at the given date. The following graph shows the daily stock price for Bitcoin from 2014 to 2021;



Volume: The volume represents the daily traded amounts of an equity. From an equities' volume, we can conclude things like the public interest and developments and buzz that are surrounding the security. The following is the volume of bitcoin from 2014 to 2021;

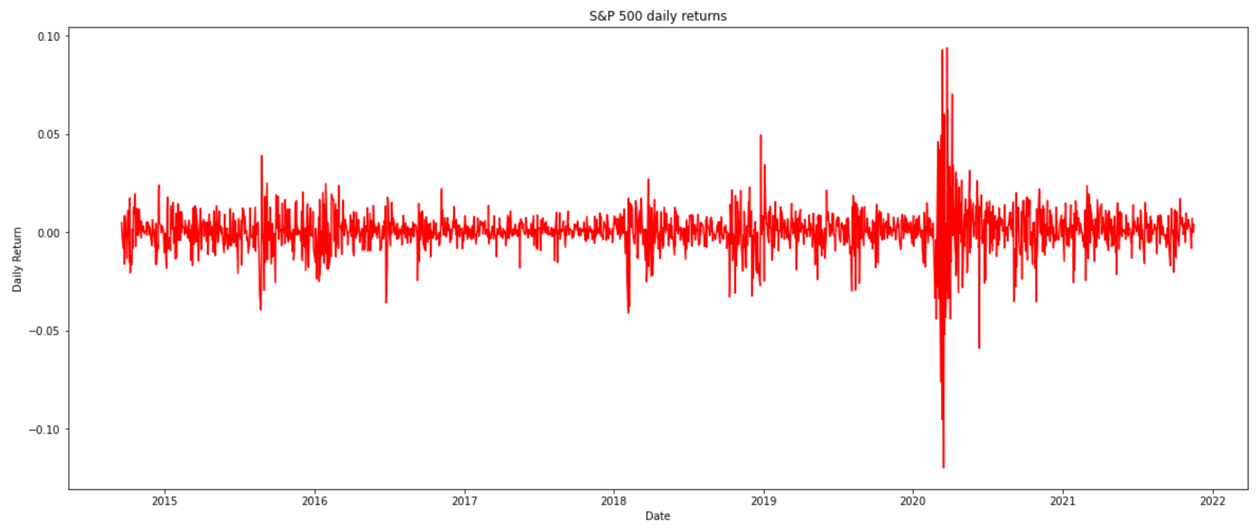


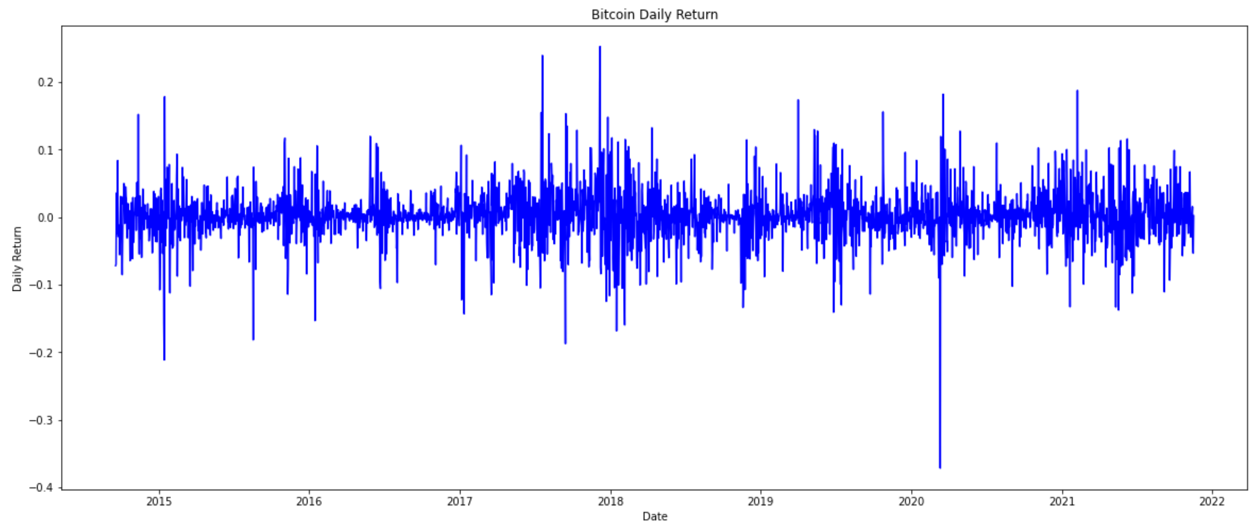
As the biggest stock index in the world, S&P 500 is a common benchmark to compare other stock data to, here are our findings when we did performance analysis against it;



Daily Returns:

When looking at stock data, to measure performance, the first thing that comes to mind is the daily returns. Daily returns are the percent change of each trading for the stock, the following are the daily returns for the S&P 500 index and Bitcoin:





As we can see from the plot, S&P 500 had the more stable returns while constantly putting out decent ones. The story is different with Bitcoin, it has a very diverse number of returns, meaning it is high-risk equity. Passive traders usually get scared of these types of returns, while active traders chase the dips to buy and the spikes to sell their equities.

Model

The model used is Deep Q-Learning. Deep Q-Learning is a subset of Deep Reinforcement Learning Models. The procedure of the Deep Q-Learning algorithm is fairly simple. First, the weights are initialized, then an action is chosen based on the state, then the weights of the neural network are updated using the Bellman Equation. The model was constructed using the Keras library. The model's task is to trade on a given security and learn to produce good profits by exploring and exploiting the market data.

Parameters

There are six main parameters that must be inputted for a Deep Q-Learning model constructed through the Keras Library. The six parameters are Gamma, Epsilon, Action Size, Neural Network Layer Count, Optimizer (although optional, it is highly recommended to use), and the Learning Rate.

The Gamma was set to 0.95 because a Gamma closer to 1.00 causes the agent to consider future rewards over recent rewards. Consequently, this gives the agent more incentive to wait or hold onto the security and sell it at a future date rather than selling it at the immediate price increase. This will ensure that the model will learn to anticipate huge potential growths in the market.

The Epsilon was set to 1.00 because this is the standard number used in Reinforcement Learning Algorithms. The Epsilon is mainly used for determining whether the agent should make a random action or an exploitive action using the neural network.

The Action Size was set to three because when trading there are only three possible actions: buy, sell, and hold.

The Neural Network Layer Count was set to four since this is usually the optimal number of layers needed for most deep learning problems. For the first layer, the input dimension was set to the state size, and the activation function used was ReLu due to the model requiring this layer to provide a yes or no type of output. Moreover, the next two layers in the neural network used ReLu as well. The last layer or output layer used the linear activation function because the linear activation function gives real-valued expected results which the network uses to determine whether it would be best to buy, sell, or hold the security.

The optimizer used was Adam. Adam was used because it has been demonstrated to have the best performance among optimizers.

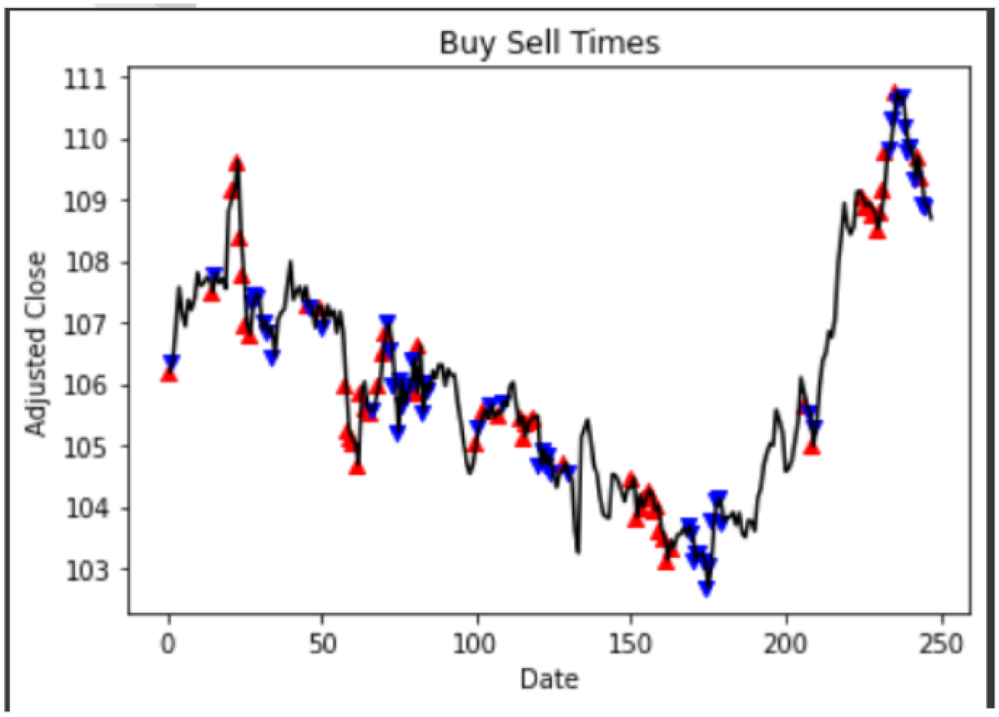
Lastly, the learning rate was set to 0.001, because this is the proven optimal standard value for the learning rate.

Overall, we did not play around with the hyperparameters, because the changes would not make much difference on the performance. Moreover, research has shown that the hyperparameters we have set are optimal. Therefore, the main testing of this model was training it on various episode counts and state sizes.

Results

The starting capital for all the tests was one million USD and the risk was 20%
All of these results are run on a daily data frame of a year, from December 9th, 2020 to December 9th, 2021.

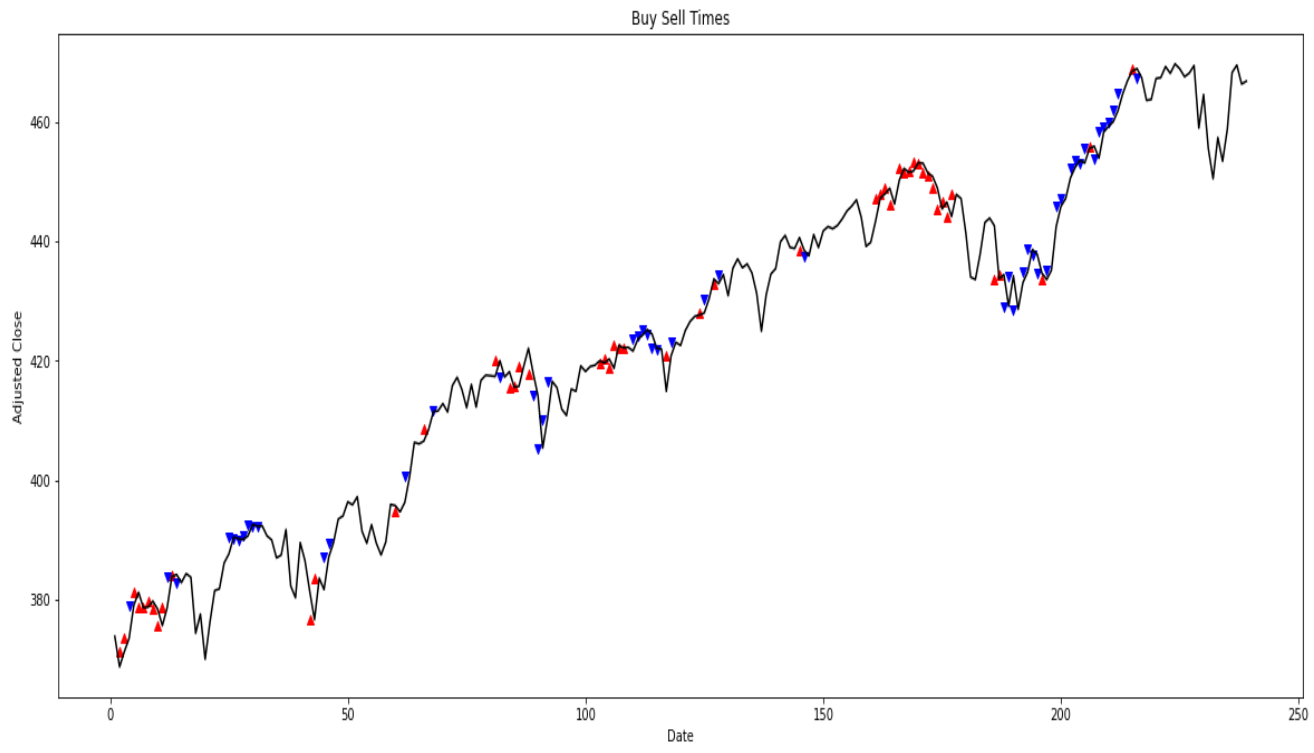
JPY-USD (Japanese Yen)



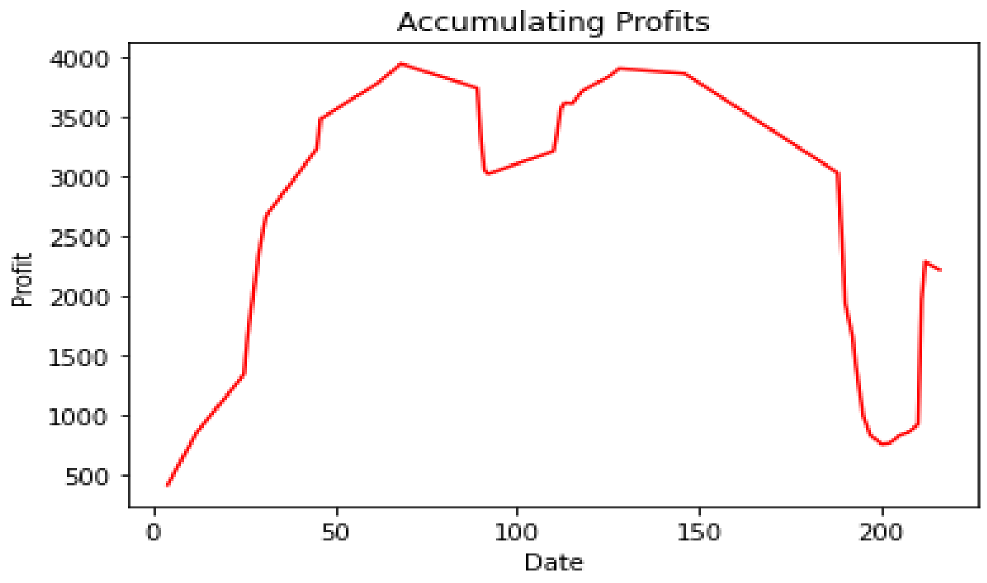
This chart displays the buy and sell times for JPY-USD. The x-coordinate is the number of days of data the model was run on, and the y-coordinate is the adjusted close price. Normally in trading, swing traders (traders who trade once a week or once in two weeks) look at the adjusted close price on the daily timeframe to make their decisions to buy or sell a security; hence, we adhered to their principles for this problem.

The red triangles represent when the model bought shares of JPY-USD, and the blue triangles represent when the model sold shares of JPY-USD. Based on all the buy and sell actions made by the model, it did not perform well. Many times, the model made sell-actions when the price was decreasing and made buy-actions when the price was increasing or at the top. Consequently, this led to a lot of loss in profit. Looking at the days 150 to 200, the model made buy-actions when the price was going down, but made sell-actions when the price went further down causing the model to gain negative profits during this timeframe. Looking at day 25, the model made a buy-action at the peak of the bull market (bull market is when the price is increasing) and made sell-actions after the price fell on days 30 to 40. This is another area where the model failed to produce positive profits.

SPY ETF

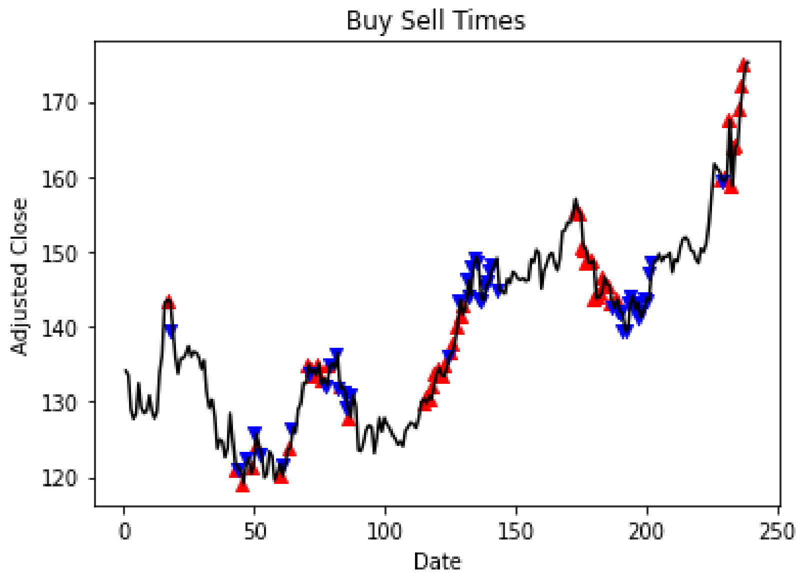


Following the information given in the section above, we plotted the decisions our model made for the SPY ETF. As mentioned in the EDA, this index follows the most popular stocks in the world and is often a benchmark when running any machine learning model on stock dataThe model performed quite better on SPY than the Japanese Yen, making better decisions overall. As we can see from days 0 to 50, it made perfect decisions by buying when the price of the stock was low and selling when it was high, profiting greatly. When we look at another interval of days from 150 to 200, the model made some terrible choices by buying at the absolute top and also selling at the lowest low for the interval.

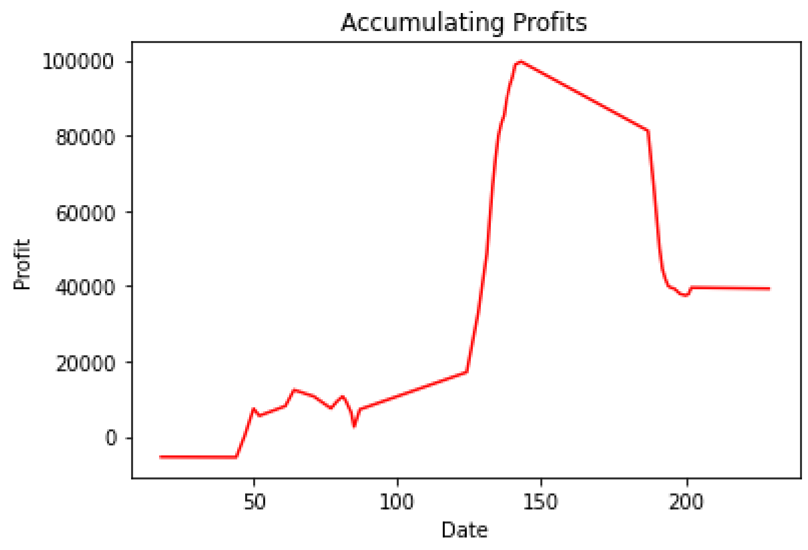


As we can see from the accumulating profits, the model actually profited on the daily data frame. While not returning the greatest results, it still managed to profit.

Apple Stock (AAPL)



Now we tried a single stock asset (AAPL) and it performed the best out of all three assets tested. It does a decent job at buying low, holding till the price increases, and finally sells. One area for more analysis would be around the 175 date mark where the model buys while the price is decreasing and sells at a lower price. This time frame is around September/October which could be of interest since that is when Apple releases new products and reports its quarterly earnings. We can see this decline in profit from the model in the accumulating profit graph below.



It would be interesting to see if the profit would stabilize or increase if other data and features were included. This is mentioned in one of our next steps.

Findings

After running our model on three kinds of equity data, we realized the following findings:

- The model performed better on the SPY index and Apple stock than it did on the Japanese Yen, we think that this is because the implied volatility of stocks like SPY and AAPL are lower than that of a currency.
- Our model consistently held its own when the equities crashed in price. It did not meet its objective and make a profit during these times but it is interesting to note that it barely lost any money on it. The model achieved this through its hold action, where it waited for a better price to sell the stock rather than panic selling.
- Using the reinforcement model in its current state in real-world trading would not be suitable because of the profitability of passive strategies. If you bought shares of SPY a year ago today, you would have a return of around 28% versus using our model would yield around 5% in the same time period.
- Our model does perform better than the popular AI social sentiment exchange-traded fund BUZZ by about a 3% yield.

Next Steps

When trading stocks, many other datasets, and features can be taken into consideration rather than just financial time series data. As described in the introduction, social sentiment on social media could reveal underlying price movements in the market. Certain political climates could lead to price gains or losses of publicly traded companies like natural resource companies if a Democratic president is elected.

Another next step could be comparing the use of different reinforcement learning methods with how much profit they yield. Along these lines, more systematic testing of hyperparameters would have to be put in place to guide consistent results throughout the models. This would likely increase running time for testing so code optimization will need to be done to help with this issue.

The last next step for this project would be to use a live trading API like Interactive Brokers or Alpaca, both of which come with Python packages, to vigorously backtest the strategy on a bigger data frame of equity data, giving us a better idea of its capabilities. Once the backtesting is done, it is a smart idea to forward test as well, using the paper trading functions of either of the mentioned APIs. This is incredibly helpful when tweaking the parameters and even changing up the model since it trades the security without any risk using fake money. Once the model returns profits that we are happy with, we can start trading with real money.

Conclusion

Artificial intelligence and machine learning have come a long way in the field of finance. Many cutting-edge machine learning models are paving the way for future innovation and it is truly an exciting time. From our results, we concluded that our current model is not feasible for real-world autonomous and intelligent trading because of more profitable strategies. This does not take away the fact that the model can be used in a strategic way to assist with financial analysis. As shown, the model is able to mitigate loss during large price drops. Using the described next steps, the model may be able to have an increased profitability to possibly beat a passive strategy.

Project Retrospect

There are two main takeaways from this project: don’t get caught up on hyperparameter testing and plan accordingly to collect results for further analysis. Hyperparameter testing can almost become obsessive due to trying to increase profitability little by little – a better approach would be a systematic one. Collecting results from the model needs to be accounted for in the project schedule due to the sheer time it takes to run these large models. If any tweaks are made to the model, then all the tests that were run prior need to be run again on the new model. The central theme here is to have a defined project schedule with all aspects of a machine learning project. These projects are much different than a normal software project due to the inherent nature of machine learning.

Resources

https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc