Portfolio Optimization

Isaac Lee, Walter Williams, Lara Zeng, Lucy Li

August 28, 2023

Abstract

The Markowitz mean-variance portfolio construction problem is a widely used method for optimizing portfolio allocation based on expected returns and risks. It involves selecting a portfolio that minimizes the variance of returns while achieving a desired level of expected return, resulting in the efficient frontier, the set of optimal portfolios for a given level of expected return. However, there are few occasions where the model is adopted to the crypto market.

In our research project, we seek to develop a complete workflow from alpha research to portfolio construction using crypto data. First, we examine the application of the classical Markowitz portfolio construction problem on crypto dataset, treating portfolio optimization as a single-period problem and experimenting with various configurations of the optimization problem. Then, we adopt LSTM to predict returns, which we train using daily market returns, r_t, computed from closing prices and volume, V_t. Finally, we test our model portfolio against the Monte-Carlo-simulated-benchmark portfolio. By analyzing the results of our simulations, we hope to gain a deeper understanding of the trade-offs involved in portfolio optimization and to identify strategies for improving portfolio performance.

1 Introduction

In the financial sector, an important question is how to construct and maintain an investment portfolio. A diversified collection of assets, or investment instruments that can be bought and sold, minimizes risk and balances constraints with the goal of creating profit for the investor. In 1952, H. Markowitz published his seminal work Portfolio Selection, which introduced the mathematical relation between the return and the risk of a security.

This led to the development of portfolio management strategies, including the mean-variance model, which aims to reduce risk by diversifying investments. Regardless, financial instruments remain vulnerable to market fluctuations. Managing portfolios is challenging due to the large number of securities available for investment.

Nowadays in the industry, quantitative asset managers and hedge funds seek to develop sources of "alpha," the excess return of their investment relative to an index. Indexes like the S&P and Dow Jones measure the price performance of a basket of securities using a standardized metric and methodology. These indexes are portfolios of extremely well-performing stocks that withstand the test of time. Achieving index-beating performance is extremely challenging. Hence, quant funds are faced with an especially difficult task. First, quants are faced with the challenge of using noisy financial data to forecast prices, both directly and indirectly. Second, they are confronted with the predicament of portfolio construction and optimization, in deciding how to combine the "signals" generated from modelling individual securities and the correlations between them to solve the mean variance optimization problem.

In this project, we seek to demonstrate a workflow for quantitative research in the crypto-market, from signal generation to portfolio construction. The discussion of the project is split into three sections: first, we introduce the sub-problems of classical portfolio optimization, followed by a discussion on how we can implement them in the lens of numerical linear algebra. Then, we proceed to a discussion of the application of LSTM and neural networks for forecasting individual assets. Finally, we combine the portfolio construction and signals to back-test the performance of our portfolio against the average performance of 25,000 randomly generated portfolios. We find that our optimization strategy and signal construction yields a 12.7 percent improvement in portfolio returns.

1.1 Cryptocurrency

Cryptocurrency is a relatively new asset class that has gained significant attention in recent years. The first cryptocurrency, Bitcoin, was introduced in 2008, and since then, the market has exploded, with over 10,000 cryptocurrencies in existence and a collective market cap of over $3 trillion. While many individuals have made fortunes investing in cryptocurrency, institutional investors have been slower to embrace this asset class.

One reason for this hesitation is the high volatility and unpredictability of cryptocurrency prices. Cryptocurrencies can experience significant price movements in a short period of time, making it difficult for investors to accurately value them and to develop reliable investment strategies. In addition, the speculative nature of cryptocurrency markets can make them more risky for some investors.

Despite these challenges, some studies have suggested that including cryptocurrency in a portfolio can have significant benefits. For example, cryptocurrencies may provide diversification benefits, as they tend to have low correlations with traditional asset classes such as stocks and bonds. This means that they may perform differently during different market conditions, which can help to mitigate the overall risk of a portfolio. Additionally, the potential for high returns in the cryptocurrency market may make it an attractive investment opportunity for some investors.

Overall, the decision to include cryptocurrency in a portfolio is a complex one that requires careful consideration of the potential risks and rewards. Investors should carefully assess their risk tolerance and investment goals before making a decision, and should also be aware of the regulatory landscape and potential risks associated with investing in cryptocurrency.

2 Literature Review

Harry Markowitz's portfolio selection model, published in 1952, is a widely used method for optimizing the allocation of assets in a portfolio [Mar52]. This approach involves selecting a portfolio that minimizes the variance of returns while still achieving a desired level of expected return. We model the rate of return on assets as random variables, and select our portfolio weighting factors optimally in order to achieve an acceptable baseline expected rate of return with minimal volatility. The result of this optimization process is the efficient frontier, which represents the set of optimal portfolios for a given level of expected return.

Markowitz's portfolio selection model was a significant development in the field of finance, as it provided a systematic way to optimize portfolio allocation based on expected returns and risks. Prior to this model, portfolio selection was largely based on subjective judgment and ad-hoc decision making. The model introduced the concept of diversification, which involves spreading investment across a range of assets in order to reduce risk, as well as the use of statistical analysis to quantify risk and return.

However, the Markowitz portfolio selection model has several limitations. One limitation is that it assumes that markets are perfectly efficient, with no taxes or transaction costs. This assumption is unrealistic, as taxes and transaction costs can significantly impact the performance of a portfolio. Additionally, the model assumes that all investors are risk averse, meaning that they prefer to avoid risk whenever possible. This assumption does not account for the fact that different investors have different risk tolerances.

Another limitation of the Markowitz model is that it assumes that investment decisions are based solely on expected returns and risks, without taking into account additional information such as economic or market conditions. In reality, investors often consider a wide range of factors when making investment decisions, and the inclusion of additional information can significantly impact the performance of a portfolio.

Despite these limitations, the Markowitz portfolio selection model has had a lasting impact on the field of finance, and it remains an important tool for portfolio optimization. Many variations of the model have been developed over the years in order to address some of its limitations, and it continues to be a subject of active research and development.

3 Classical Portfolio Optimization

We first establish vocabulary for the important variables involved in an investor's decision problem.

A security is a legal contract that allows the investor to receive economic benefits under specific conditions. Common stocks or equities are a type of security that give the investor the right to share in the profits of the company.

A crucial component in investors' decision making problem is the expected utility they can derive from making the investment. This is most accurately reflected by the expected returns they can make from holding the security. Let r_{ij} be the return of a security i during a time period j. We define the expected return $E(r_i)$ of the security i over a series of M future time periods as follows, with all time periods being equally important:

$$E(r_i) = \sum_{j=1}^{M} \frac{r_{ij}}{M}.$$

The most common metric of a security's risk is the variance of returns. Let r_{ij} be the return of a security i during a time period j and let $E(r_i)$ be the expected return of the security. The variance of returns, σ_i^2, of the security i, concerning a series of M equally important future time periods, is defined as follows:

$$\sigma_i^2 = \sum_{j=1}^{M} \frac{(r_{ij} - E(r_i))^2}{M}.$$

We briefly outline the classical portfolio construction problem described in the classical Harry Markowitz (1952)'s literature. The term "portfolio" refers to a collection of financial assets, such as stocks, bonds, and cash. By the classical definition that Markowitz posits, a portfolio P is efficient if and only if there is no other portfolio P' such that

$$E(r_{P'}) \geq E(r_P)$$

and

$$\sigma_{P'} \leq \sigma_P.$$

We require at least one inequality to be strict. Thus, a portfolio P is efficient if and only if there is no other portfolio P' which outweighs P either concerning return or risk. The set including all the efficient portfolios is called the efficient frontier.

In Figure 1, we examine the set of all portfolios, considering those that are efficient or feasible. A feasible portfolio is any portfolio with proportions summing to one. All portfolios

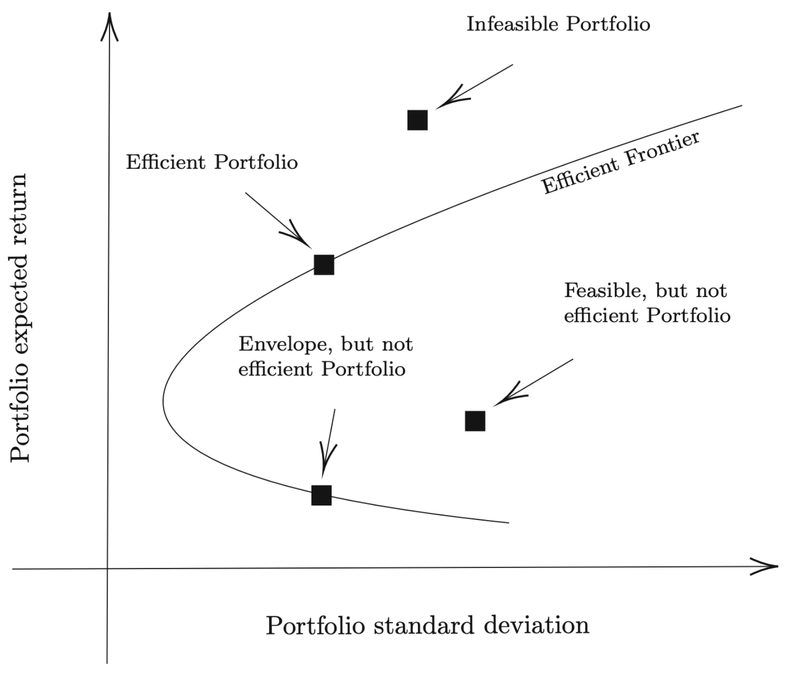

Infeasible Portfolio
Efficient Portfolio
Efficient Frontier
Feasible, but not efficient Portfolio
Envelope, but not efficient Portfolio
Portfolio expected return
Portfolio standard deviation

Figure 1: Efficient frontier.

to the right of the curved line in Figure 1 are a part of the feasible set. All portfolios which have minimum variance for a given mean return are called envelope portfolios, and they are depicted on the envelope of the feasible set. Finally, all portfolios which have maximum return given the portfolio variance are called efficient portfolios and are depicted by the curve in Figure 1.

A portfolio that falls to the right of the efficient frontier has greater risk relative to its predicted return, while a portfolio that falls beneath the slope of the efficient frontier offers a lower level of return relative to risk. Individual investors with varying degrees of risk tolerance may prefer different portfolios. For instance, a risk-averse investor would prefer to hold a portfolio on the left half of the efficient frontier, and a risk-seeking individual would seek to hold a portfolio to the right of the frontier.

In Figure 1, rational investors would hold portfolios that are on the upper half of the depicted curve. It would not make sense for an investor to choose the *Envelope, but not efficient Portfolio*, as there exists a portfolio that exactly replicates its risk (has the same standard deviation), but has higher expected return (labelled as the *Efficient Portfolio* in the diagram). Likewise, no rational investor would choose to hold the *Feasible, but not efficient Portfolio* as there exists a replicating portfolio with less standard deviation and the same return.

The efficient frontier indicates the combination of investments that will provide the highest return for the lowest level of risk. The efficient frontier is an illustration of all theoretically optimal ways to combine risky assets in a way that manifest specific portfolio risk-return characteristics.

4 Portfolio Optimization Methods

First, to demonstrate how the Markowitz theory can be applied, we solve the Markowitz portfolio construction problem and its many sub-problems with our existing cryptocurrency data. The focus of this section is not on predicting signal with machine learning methods. Instead, we make the simplifying (and unrealistic) assumption that investors have perfect information on crypto returns: they know exactly how future prices will evolve, and construct their portfolio based on this information.

In this section, we first focus on delineating original examples and solutions of example optimization problems. We respectively describe and solve the constraint optimization problem for variance minimization, Sharpe Ratio maximization, and efficient frontier construction. The variance minimization and efficient frontier sub-problems are easier solved via the direct application of linear algebra. The maximum Sharpe Ratio sub-problem gives rise to a more complex objective function, and may be more easily solved using gradient descent.

4.1 Sub-Problem 1: Global Minimum Variance Portfolio

Consider a hypothetical investor who is extremely risk-averse. Such an investor wants to minimize portfolio variance at all cost, to the extent that they do not care about maximizing returns at all. What composition of stocks would such an investor want to hold ?

Such a portfolio of stocks is called the global minimum variance portfolio. We are solving for a vector of weights of assets that uniquely minimizes the portfolio variance

$$\mathbf{m} = (m_1, m_2, m_3, ..., m_n)'.$$

To better illustrate this concept, we first examine the solution to a simplified two asset constrained minimization problem, and extend the solution to an n assets case. When we have two assets, the asset allocation problem is to find the weight vector of:

$$\mathbf{m} = (m_A, m_B)'.$$

The variance of the portfolio, σ_p^2, is expressed as follows:

$$\sigma_p^2 = \begin{bmatrix} m_A & m_B \end{bmatrix} \begin{bmatrix} \sigma_A^2 & \sigma_{A,B} \\ \sigma_{A,B} & \sigma_B^2 \end{bmatrix} \begin{bmatrix} m_A \\ m_B \end{bmatrix} = \begin{bmatrix} m_A\sigma_A^2 + m_B\sigma_{A,B}, m_A\sigma_{A,B} + m_B\sigma_B^2 \end{bmatrix} \begin{bmatrix} m_A \\ m_B \end{bmatrix}$$
$$= m_A^2\sigma_A^2 + 2m_Am_B\sigma_{A,B}^2 + m_B^2\sigma_B^2.$$

Our minimization problem is thus:

$$\min_{m_A, m_B,} \sigma_{p,m}^2 = m_A^2\sigma_A^2 + m_B^2\sigma_B^2 + 2m_Am_B\sigma_{AB}$$
$$\text{s.t. } m_A + m_B = 1.$$

The Lagrangian for this problem is

$$L(m_A, m_B, \lambda) = m_A^2\sigma_A^2 + m_B^2\sigma_B^2 + 2m_Am_B\sigma_{AB} + \lambda(m_A + m_B - 1),$$

and the first order conditions (FOCs) for a minimum are

$$0 = \frac{\partial L}{\partial m_A} = 2m_A\sigma_A^2 + 2m_B\sigma_{AB} + \lambda,$$
$$0 = \frac{\partial L}{\partial m_B} = 2m_B\sigma_B^2 + 2m_A\sigma_{AB} + \lambda,$$
$$0 = \frac{\partial L}{\partial \lambda} = m_A + m_B - 1.$$

The three linear equations describing the first order conditions has the matrix representation

$$\begin{pmatrix} 2\sigma_A^2 & 2\sigma_{AB} & 1 \\ 2\sigma_{AB} & 2\sigma_B^2 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} m_A \\ m_B \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

This problem can be easily solved as there are only 3 equations and 3 unknowns. We are able to find the closed form solution:

$$\begin{aligned} m_A\sigma_A^2 + m_B\sigma_{AB} &= m_B{\sigma_B}^2 + m_A\sigma_{AB} \\ m_A\sigma_A^2 + (1-m_A)\,\sigma_{AB} &= (1-m_A)\,{\sigma_B}^2 + m_A\sigma_{AB} \\ m_A &= \frac{\sigma_{B^2} - \sigma_{AB}}{\sigma_A^2 - 2\sigma_{AB} + \sigma_B^2}. \end{aligned}$$

Likewise,

$$\begin{aligned} (1-m_B)\,\sigma_A^2 + m_B\sigma_{AB} &= m_B\sigma_B^2 + (1-m_B)\,\sigma_{AB} \\ {\sigma_A}^2 - \sigma_{AB} &= m_B\left(\sigma_B^2 - 2\sigma_{AB} + {\sigma_A}^2\right) \\ m_B &= \frac{\sigma_A^2 - \sigma_{AB}}{\sigma_B^2 - 2\sigma_{AB} + \sigma_A^2}. \end{aligned}$$

In an allocation problem with more assets, the matrix representation of the set of linear equations has the following form:

$$\begin{pmatrix} 2\mathbf{\Sigma} & \mathbf{1} \\ \mathbf{1}' & 0 \end{pmatrix} \begin{pmatrix} \mathbf{m} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix}$$

Here, Σ is the covariance matrix of individual assets. The system is of the form $\mathbf{A}_m\mathbf{z}_m = \mathbf{b}$, where:

$$\mathbf{A}_m = \begin{pmatrix} 2\Sigma & \mathbf{1} \\ \mathbf{1}' & 0 \end{pmatrix}, \mathbf{z}_m = \begin{pmatrix} \mathbf{m} \\ \lambda \end{pmatrix} \text{ and } \mathbf{b} = \begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix}.$$

The FOCs give rise to $N+1$ linear equations and $N+1$ unknowns. Provided $\mathbf{A}_m$ is invertible, the solution for $\mathbf{z}_m$ is:

$$\mathbf{z}_m = \mathbf{A}_m^{-1}\mathbf{b}.$$

The first N elements of $\mathbf{z}_m$ comprise the portfolio weight vector $\mathbf{m}$ for the global minimum variance portfolio.

4.2 Sub-Problem 2: Maximum Sharpe Ratio Portfolio

The Sharpe Ratio, developed by Nobel Laureate William F. Sharpe, is a measure of calculating risk adjusted return. Intuitively, it is an expression for the returns on investors' investments relative to the risks they hold [Sha64].

Owing to its beauty and simplicity, the Sharpe Ratio is often used to assess and benchmark the performance of asset managers. It is an extremely effective assessment of how well the return of an asset compensates the investor for the risk taken. When comparing several different assets, the asset with a higher Sharpe Ratio appears to provide better return for the same risk, which is usually attractive to investors.

The annualized Sharpe Ratio is computed as follows:

$$SharpeRatio = \sqrt{252}\frac{R_p - R_f}{\sigma_p}$$

where R_p is the average daily return, R_f is the risk-free rate of return, and σ_p is the standard deviation of the return, (i.e., the volatility or the risk).

Not only does the Sharpe Ratio account for the returns and risk of the existing portfolio, it also factors in the opportunity cost (cost that is forgone) by holding the risky portfolio. In finance, the risk-free rate is usually proxied by the returns of a risk-free security (e.g. US T-bills). A crucial assumption we make in the Markowitz model is that there are zero transaction costs, and that unlimited capital can be borrowed or invested at the risk-free rate.

The higher the Sharpe Ratio of an investment, the more excess returns should be achieved in comparison to holding a risk-free investment, relative to the increased volatility that the investment is exposed to. A Sharpe Ratio of 0 denotes that the investment is risk-free or that the investment does not yield any excess returns. In practice, while a Sharpe Ratio of 1 is a sign that the investment is acceptable or good for investors, a value less than 1 grades the investment as suboptimal, and values greater than 1 and moving towards 2 or 3 grade the investment as highly superior.

To optimize for the highest Sharpe Ratio, we again want to evaluate

$$\mathbf{m} = (m_1, m_2, m_3, ..., m_n)'.$$

Let the returns of the individual securities of our portfolio be represented by

$$\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3, ..., \mu_n).$$

In this case, our maximization problem goes as follows:

$$\max_{\mathbf{m}} \frac{\sum_{i=1}^{N} m_i \cdot \mu_i - R_f}{\sqrt{\mathbf{m}'\boldsymbol{\Sigma}\mathbf{m}}}$$

subject to

$$\mathbf{m}'\mathbf{1} = 1.$$

The numerator of the objective function denotes the excess returns of the investment over those of a risk-free asset; the denominator denotes the risk of the investment. The objective is to maximize the Sharpe Ratio. The basic constraints indicate that the investor wishes to have a fully invested portfolio.

To solve the above numerically, we have several options: for example, we can use steepest descent or gradient descent, as described in Algorithm 4.1.

Algorithm 4.1 Gradient Descent to find Sharpe Ratio (for illustrative purposes)

```
1: Initialize m such that it randomizes the weights of all selected stocks
2: Set learning rate as $t_0$
3: Set max Sharpe Ratio as 0
4: while stopping criteria not met do
5:     $\mathbf{m_{k+1}} = \mathbf{m_k} + t_k \frac{\partial SR}{\partial \mathbf{m_k}}$ and compute the new Sharpe Ratio using the new value of $\mathbf{m_{k+1}}$
6:     if max Sharpe Ratio is less than new Sharpe Ratio, update max Sharpe Ratio
7: end while
```

Let SR be the Sharpe Ratio expressed as a function of vectors $\mathbf{m}$ and $\boldsymbol{\mu}$:

$$SR = \frac{\sum_{i=1}^{N} m_i \cdot \mu_i - R_f}{\sqrt{\mathbf{m}'\boldsymbol{\Sigma}\mathbf{m}}}.$$

We go on to construct the efficient frontier.

4.3 Sub-Problem 3: Constructing the Efficient Frontier

Recall that the efficient frontier is the set of optimal portfolios that offer the highest expected return for a defined level of risk or the lowest risk for a given level of expected return. How do we model the efficient frontier as a function?

Let $\sigma^2_{p,0}$ denote a target level of risk. As by Harry Markowitz's characterization, the constrained maximization problem to find an efficient portfolio is

$$\max_{\mathbf{m}} \mu_p = \mathbf{m}'\boldsymbol{\mu}$$

subject to

$$\sigma_p^2 = \mathbf{m}'\boldsymbol{\Sigma}\mathbf{m} = \sigma^2_{p,0} \text{ and } \mathbf{m}'\mathbf{1} = 1.$$

Markowitz showed that the investor's problem of maximizing portfolio expected return subject to a target level of risk has an equivalent dual representation, in which the investor minimizes the risk of the portfolio (as measured by portfolio variance) subject to a target expected return level. Let $\mu_{p,0}$ denote a target expected return level. Then the dual problem is the constrained minimization problem

$$\min_{\mathbf{m}} \sigma^2_{p,m} = \mathbf{m}'\boldsymbol{\Sigma}\mathbf{m}$$

subject to

$$\mu_p = \mathbf{m}'\boldsymbol{\mu} = \mu_{p,0}, \text{ and } \mathbf{m}'\mathbf{1} = 1.$$

To find efficient portfolios of risky assets in practice, the latter *dual* problem is most often solved due to its computational convenience. In addition, the dual problem is used because investors are more willing to specify target expected returns rather than target risk levels.

To solve the constrained minimization problem, we first form the Lagrangian function

$$L(m, \lambda_1, \lambda_2) = \mathbf{m}'\boldsymbol{\Sigma}\mathbf{m} + \lambda_1\left(\mathbf{m}'\boldsymbol{\mu} - \mu_{p,0}\right) + \lambda_2\left(\mathbf{m}'\mathbf{1} - 1\right).$$

Because there are two constraints ($\mathbf{m}'\boldsymbol{\mu} = \mu_{p,0}$ and $\mathbf{m}'\mathbf{1} = 1$), there are two Lagrange multipliers λ_1 and λ_2. The FOCs for a minimum are the linear equations

$$\begin{aligned}
\frac{\partial L(\mathbf{m}, \lambda_1, \lambda_2)}{\partial \mathbf{m}} &= 2\boldsymbol{\Sigma}\mathbf{m} + \lambda_1\boldsymbol{\mu} + \lambda_2\mathbf{1} = \mathbf{0}, \\
\frac{\partial L(\mathbf{m}, \lambda_1, \lambda_2)}{\partial \lambda_1} &= \mathbf{m}'\boldsymbol{\mu} - \mu_{p,0} = 0 \\
\frac{\partial L(\mathbf{m}, \lambda_1, \lambda_2)}{\partial \lambda_2} &= \mathbf{m}'\mathbf{1} - 1 = 0.
\end{aligned}$$

These FOCs consist of five linear equations in five unknowns $(m_A, m_B, m_C, \lambda_1, \lambda_2)$. We can represent the system of linear equations using matrix algebra as

$$\begin{pmatrix} 2\boldsymbol{\Sigma} & \boldsymbol{\mu} & \mathbf{1} \\ \boldsymbol{\mu}' & 0 & 0 \\ \mathbf{1}' & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mu_{p,0} \\ 1 \end{pmatrix},$$

or

$$\mathbf{A}\mathbf{z}_m = \mathbf{b}_0,$$

where

$$\mathbf{A} = \begin{pmatrix} 2\boldsymbol{\Sigma} & \boldsymbol{\mu} & \mathbf{1} \\ \boldsymbol{\mu}' & 0 & 0 \\ \mathbf{1}' & 0 & 0 \end{pmatrix}, \mathbf{z}_m = \begin{pmatrix} \mathbf{m} \\ \lambda_1 \\ \lambda_2 \end{pmatrix} \text{ and } \mathbf{b}_0 = \begin{pmatrix} \mathbf{0} \\ \mu_{p,0} \\ 1 \end{pmatrix}.$$

The solution for $\mathbf{z}_m$ is then

$$\mathbf{z}_m = \mathbf{A}^{-1}\mathbf{b}_0.$$

The first three elements of $\mathbf{z}_m$ are the portfolio weights $\mathbf{m} = (m_A, x_B, x_C)'$ for the minimum variance portfolio with expected return $\mu_{p,m} = \mu_{p,0}$. If $\mu_{p,0}$ is greater than or equal to the expected return on the global minimum variance portfolio, then $\mathbf{m}$ is an efficient portfolio.

4.4 Methods and Results: Portfolio Construction Using Cryptocurrencies Data

We utilize the *top-50-cryptocurrencies-historical-prices* dataset from Kaggle to construct long-only portfolios. Long-only means that we require all allocation weights to be positive and sum to 1. The dataset contains the historical prices for 50 of the top cryptocurrencies by market cap. In this section, we use data from the time period of May 31, 2021 to July 31, 2021 where we have price information for most cryptocurrencies.

To optimize our portfolio, we return to the three aforementioned optimization problems to configure the optimal weight assignments that an investor should allocate between May 31 and July 31, where the objective functions are respectively given by:

1. Construct a long only portfolio that uniquely minimizes the portfolio variance
2. Construct a long only portfolio that uniquely maximizes the Sharpe Ratio
3. Construct the efficient frontier

First, using prices of cryptocurrencies from 2021-05-12 to 2021-08-19, we compute the returns array:

$$R_i = \frac{Pi,t+1 - Pi,t}{Pi,t}.$$

Our covariance matrix is populated by the covariances of the returns. As an example, if we have 3 securities in our portfolio, we have the covariance matrix as computed by:

$$\begin{aligned} \operatorname{var}(\mathbf{R}) &= \begin{pmatrix} \operatorname{var}(R_A) & \operatorname{cov}(R_A, R_B) & \operatorname{cov}(R_A, R_C) \\ \operatorname{cov}(R_B, R_A) & \operatorname{var}(R_B) & \operatorname{cov}(R_B, R_C) \\ \operatorname{cov}(R_C, R_A) & \operatorname{cov}(R_C, R_B) & \operatorname{var}(R_C) \end{pmatrix} \\ &= \begin{pmatrix} \sigma_A^2 & \sigma_{AB} & \sigma_{AC} \\ \sigma_{AB} & \sigma_B^2 & \sigma_{BC} \\ \sigma_{AC} & \sigma_{BC} & \sigma_C^2 \end{pmatrix} = \boldsymbol{\Sigma} \end{aligned}$$

Subsequently, as described in Algorithm 4.2, we perform Monte Carlo simulations to produce 25,000 possible allocations of portfolios, generating the initial weight of each stock in a random fashion.

Algorithm 4.2 Using Monte Carlo Simulations to Generate Example Portfolio

```
for i in Range(0, 25,000) do
    Initialize the $\boldsymbol{m}$ array to store allocation weights
    for each security j in the portfolio do
        Generate $m_j$ from U(0,1) distribution
    end for
    Normalize the $\boldsymbol{m}$ array by dividing each of its elements by the sum of the weights $m_j$
    Compute the annualized volatility as $\sigma_p^2 = \sqrt{252}\mathbf{m'\Sigma m}$
    Compute the annualized return $252\boldsymbol{\mu}\mathbf{m}$
    Plot annualized returns against volatility
end for
```

We visualize these 25,000 portfolios and plot the solutions obtained from solving our constrained optimization problem. The solutions for our three sub-problems are computed following the structure outlined by the optimization problems illustrated in the previous 3 sections. While the variance minimization and efficient frontier problems have well formed solutions in linear algebra, for consistency and performance reasons, we solve the optimization problems using a gradient based optimizer (similar to the pseudocode outlined in Section 4.2, Maximum Sharpe Ratio Portfolio), using the scipy optimization library.

1. Minimize $\sigma_p^2 = \sqrt{252}\mathbf{m'\Sigma m}$, shown as the green point on the plot.
2. Maximize $SharpeRatio = \sqrt{252}\frac{R_p - R_f}{\sigma_p}$, shown as the red point on the plot.
3. Construct the efficient frontier: in constructing the optimal frontier, we solve the efficient frontier optimization problem repeatedly for 50 different values of annualized returns equally spaced between the minimum and maximum returns generated from the Monte Carlo Simulations (resulting in the dotted line curve).

Below are plots showing the visualizations of the three sub-problems for different combinations of securities.

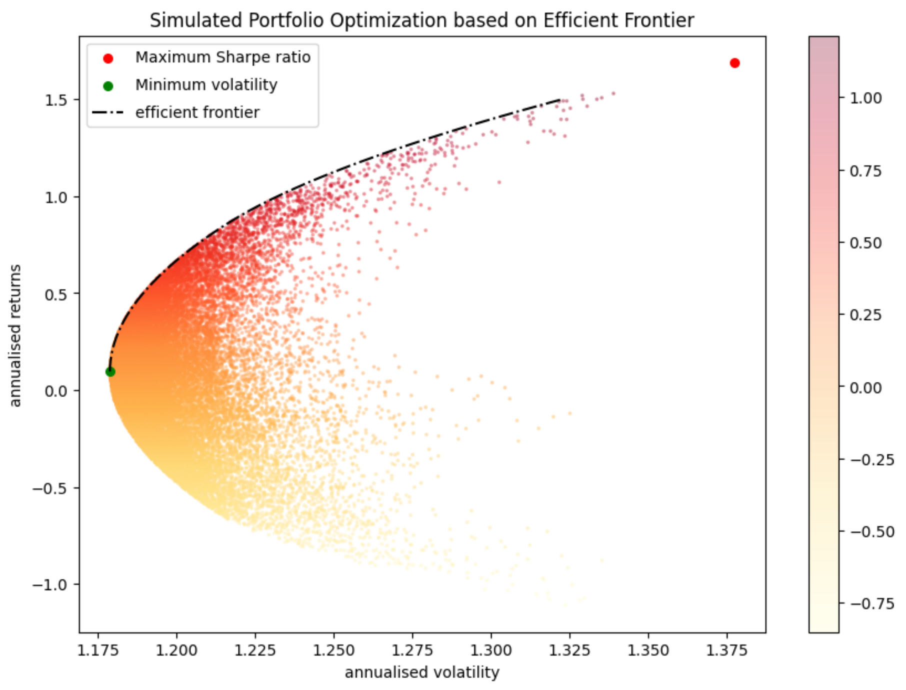


Figure 2: Simulated Portfolio Construction based on: Avalanche, Filecoin, Dogecoin, SHIBA INU

Maximum Sharpe Ratio Portfolio Allocation

- Annualised Return: 1.61
- Annualised Volatility: 1.36

Avalanche	**Filecoin**	**Dogecoin**	**SHIBA INU**
90.26	7.07	0.44	2.23

Minimum Volatility Portfolio Allocation

- Annualised Return: 0.19
- Annualised Volatility: 1.18

Avalanche	**Filecoin**	**Dogecoin**	**SHIBA INU**
24.68	39.28	16.34	19.7

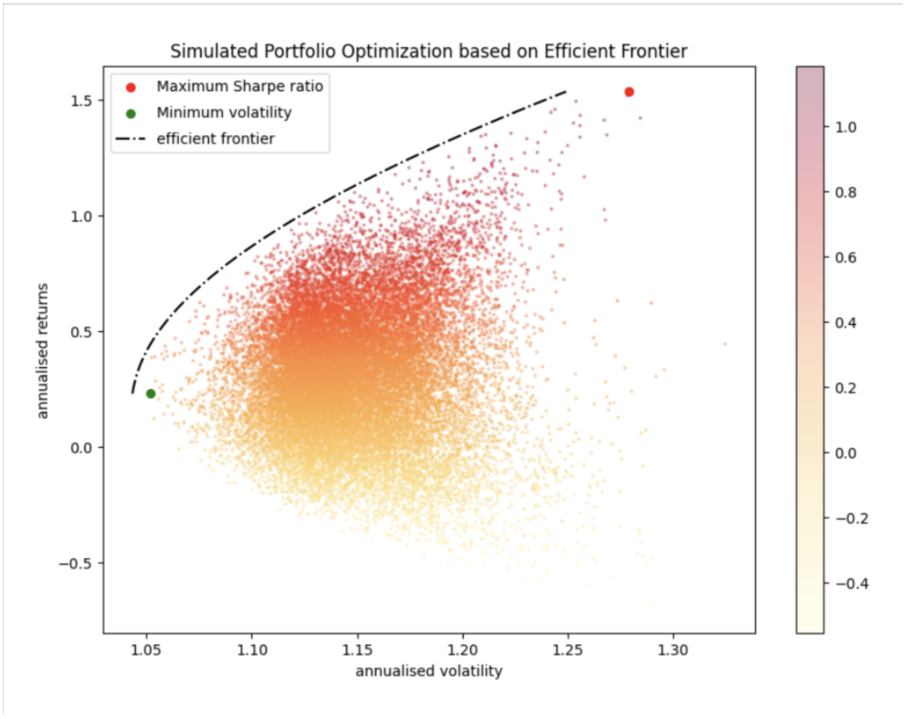


Figure 3: Simulated Portfolio Construction based on: Avalanche, Filecoin, Dogecoin, SHIBA INU, Ethereum

Maximum Sharpe Ratio Portfolio Allocation

- Annualised Return: 1.5
- Annualised Volatility: 1.3

Avalanche	**Filecoin**	**Dogecoin**	**SHIBA INU**	**Ethereum**
73.22	5.53	15.81	2.88	2.56

Minimum Volatility Portfolio Allocation:

- Annualised Return: 0.20
- Annualised Volatility: 1.05

Avalanche	**Filecoin**	**Dogecoin**	**SHIBA INU**	**Ethereum**
2.06	3.72	13.35	4.07	76.81

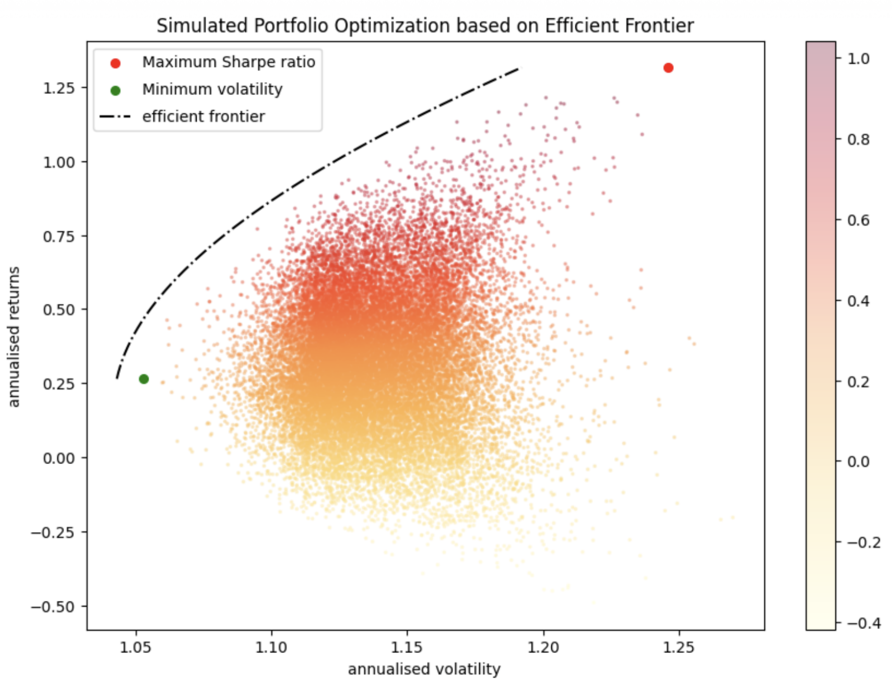


Figure 4: Simulated Portfolio Construction based on: Avalanche, Filecoin, Dogecoin, SHIBA INU, Ethereum, Monero

Maximum Sharpe Ratio Portfolio Allocation

- Annualised Return: 1.65
- Annualised Volatility: 1.25

Avalanche	**Filecoin**	**Dogecoin**	**SHIBA INU**	**Ethereum**	**Monero**
67.59	9.12	1.29	9.18	8.0	4.82

Minimum Volatility Portfolio Allocation:

- Annualised Return: 0.26
- Annualised Volatility: 1.05

Avalanche	**Filecoin**	**Dogecoin**	**SHIBA INU**	**Ethereum**	**Monero**
4.57	10.72	11.26	1.46	71.27	0.72

From our results, we make two observations as the number of assets in our portfolio increases:

1. From the visualizations, we observe that our Sharpe Ratio increases and volatility decreases for the same objective function. We are able to achieve a more desirable optimization outcome. This illustrates the importance of having a sufficiently large universe in which optimization can be conducted.

2. As the number of assets increases, results generated from the Monte Carlo simulations stray further away from the efficient frontier. This shows us that for a high dimensional problem with many optimization parameters, solving optimization problems with only a Monte Carlo-based strategy, as we have done here, may not yield a desirable result.

5 Machine Learning Optimization Methods

5.1 Neural Networks

Similar to other machine learning models, neural networks are, at a high level, approximators that map data to certain information. What makes neural nets distinct from other types of machine learning is the "perceptron," commonly referred to as a neuron. Neural networks, in their simplest form, consist of n inputs, m neurons, and one output, where n is the number of features in the dataset. Figure 5 shows an example of a Neural Network with 5 inputs and 1 hidden layer of size 6.

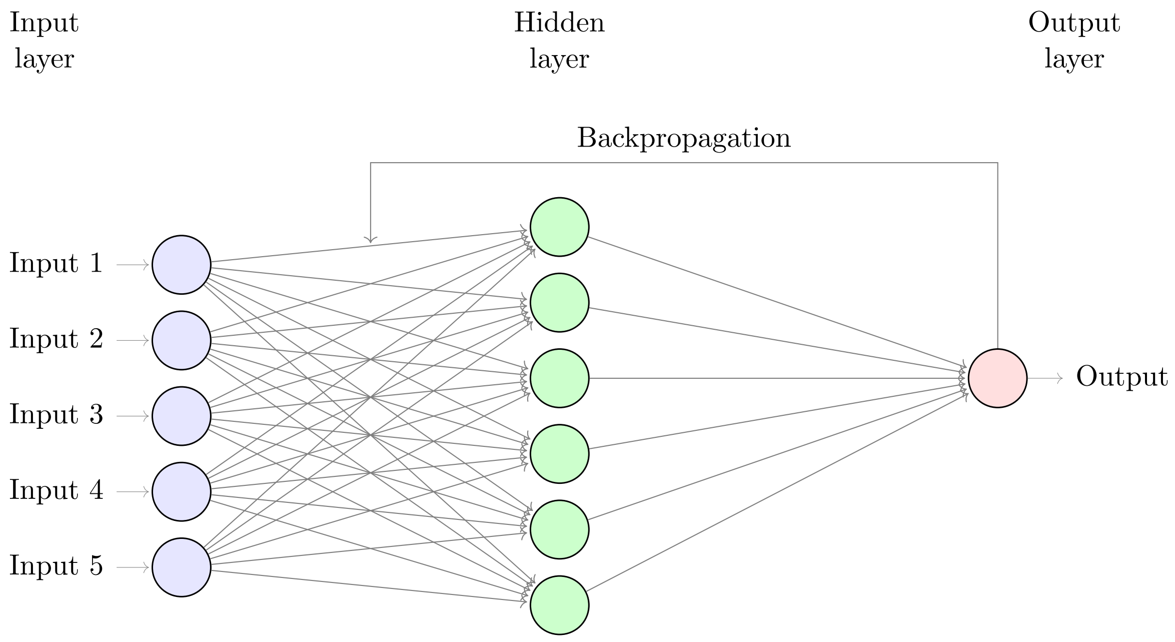


Figure 5: Simple Feed-Forward Neural Network with 1 hidden layer

The process of passing the input through this network is known as forward propagation [Dhi]. The process begins with the input layer, where each neuron receives a value from the input data. These values are then passed through the network, layer by layer, until they reach the output layer. At each layer, the values are multiplied by the weights of the connections between the neurons, and then passed through an activation function, which applies a non-linear transformation to the values. Once the values reach the output layer, the final predictions are made.

Each intermediate neuron value is calculated from the previous layer, along with the weights and bias associated with the edges connecting the neurons. In addition, each neuron in the hidden layers goes through an activation function, σ, that normalizes the values across the neurons in a layer.

Given Z which represents the neuron values and A which represents the value after the activation function is applied, we can formulate Figure 5 mathematically as follows:

$$Z^{(1)} = W^{(1)}X + B^{(1)}$$

$$\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix} = \begin{pmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,2} & w_{m,2} & \dots & w_{m,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

$$A^{(1)} = \sigma(Z^{(1)})$$

$$\begin{pmatrix} a_1^{(1)} \\ a_2 \\ \vdots \\ a_m \end{pmatrix} = \sigma \left[\begin{pmatrix} z_1^{(1)} \\ z_2 \\ \vdots \\ z_m \end{pmatrix} \right]$$

When evaluating the final output neuron, we simply have edges connecting to one neuron.

$$Z^{(2)} = W^{(2)} A^{(1)} + B^{(1)}$$

$$\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix} = \begin{pmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \end{pmatrix}$$

$$A^{(2)} = \sigma(Z^{(2)})$$

$$\begin{pmatrix} a_1^{(2)} \\ a_2 \\ \vdots \\ a_m \end{pmatrix} = \sigma \left[\begin{pmatrix} z_1^{(2)} \\ z_2 \\ \vdots \\ z_m \end{pmatrix} \right]$$

The key feature of Neural Networks is backpropagation [RHW86] [GBC16]. Backpropagation is the process by which the weights and biases are adjusted by propagating the loss back to the neurons. It is a way of efficiently calculating the gradient, or derivative, of the loss function with respect to the weights of the network, which can then be used to update the weights in order to minimize the loss. Backpropagation reduces the loss of the output, where the loss is the error of the final output compared to the ground-truth value.

Here are the steps of the backpropagation algorithm:

1. Forward propagation: In the forward propagation step, the input data is passed through the neural network, and the output is calculated.

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = \sigma(z^{(l)})$$

   where $W^{(l)}$ and $b^{(l)}$ are the weights and biases of the lth layer, $a^{(l-1)}$ is the output of the previous layer, and σ is the activation function.

2. Loss calculation: The loss is calculated by comparing the predicted output to the true output.

$$L = \frac{1}{2}(y - \hat{y})^2$$

   where y is the true output and $\hat{y}$ is the predicted output.

3. Backward propagation: In the backward propagation step, the gradient of the loss with respect to the weights and biases is calculated using the chain rule.

$$\frac{\partial L}{\partial w_{i,j}^{(l)}} = a_j^{(l-1)} \frac{\partial L}{\partial z_i^{(l)}}$$

$$\frac{\partial L}{\partial b_i^{(l)}} = \frac{\partial L}{\partial z_i^{(l)}}$$

$$\frac{\partial L}{\partial z_i^{(l)}} = \frac{\partial L}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}}$$

where $w_{i,j}^{(l)}$ is the weight connecting the ith neuron in the lth layer to the jth neuron in the $(l-1)$th layer, and $b_i^{(l)}$ is the bias of the ith neuron in the lth layer.

4. Weight and bias update: The weights and biases are updated using the gradient of the loss and a learning rate, which determines the size of the update.

$$w_{i,j}^{(l)} = w_{i,j}^{(l)} - \alpha \frac{\partial L}{\partial w_{i,j}^{(l)}}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial L}{\partial b_i^{(l)}}$$

While vanilla neural networks are sufficient for many tasks, there are limitations when we work with sequential data. Cryptocurrency data is sequential: there is no clear way to represent the past closing price as input. Though we could theoretically treat each price as an input neuron, this would neglect the sequencing of the data, and cause issues with the size of the inputs to the model.

5.2 Recurrent Neural Network

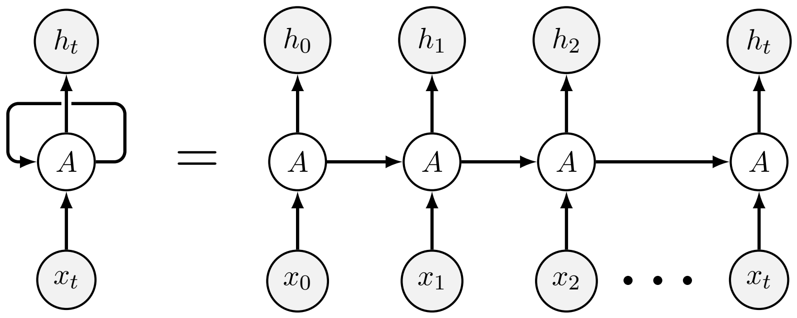


Figure 6: Recurrent Neural Network Diagram

A recurrent neural network (RNN) is a type of neural network designed to process sequential data, such as time series or natural language data. It is called a recurrent network because it uses the same set of weights for multiple inputs, allowing it to learn from the relationships between the elements in the sequence [She19].

The key feature of RNNs is the use of hidden states, which are additional inputs that are carried over from one element in the sequence to the next. These hidden states allow the network to remember important information from previous elements in the sequence and use it to inform its predictions for the current element.

The basic structure of an RNN consists of a series of repeating units, called cells, which are connected in a specific way. Each cell receives input from the previous cell, as well as from the current element in the sequence. The cell processes this input using a set of weights and produces an output and a new hidden state. This hidden state is then carried over to the next cell and used as input along with the next element in the sequence. The output of the last cell in the sequence is typically used as the final output of the network. In a language model, for example, the output of the last cell might be a probability distribution over the possible next words in the sentence.

To train an RNN, we use a variant of the backpropagation algorithm called backpropagation through time (BPTT). This algorithm works by unrolling the RNN into a feed-forward network and then applying the standard backpropagation algorithm to this unrolled network. This allows us to calculate the gradient of the loss function with respect to the weights of the network, and use this gradient to update the weights in order to minimize the loss.

- Forward propagation: At each time step, the input to the RNN is a vector x_t and the hidden state is a vector h_t. The output at each time step is a vector y_t. The hidden state and output are calculated using the following equations:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b_h)$$

$$y_t = \sigma(W_y h_t + b_y)$$

  where W_h, W_x, and W_y are the weights that connect the hidden state at the previous time step, the input at the current time step, and the hidden state at the current time step to the output, respectively, and b_h and b_y are the biases. σ is the activation function.

- Loss calculation: The loss is calculated by comparing the predicted output at each time step to the true output. The loss at each time step is given by:

$$L_t = \frac{1}{2}(y_t - \hat{y}_t)^2$$

  where y_t is the true output and $\hat{y}_t$ is the predicted output. The total loss is then given by:

$$L = \sum_{t=1}^{T} L_t$$

  where T is the number of time steps in the input sequence.

- Backward propagation: The gradient of the loss with respect to the weights and biases is calculated using the chain rule. The gradient of the loss with respect to the weights and biases at each time step is given by:

$$\frac{\partial L_t}{\partial W_h} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial W_h}$$

$$\frac{\partial L_t}{\partial b_h} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial b_h}$$

$$\frac{\partial L_t}{\partial W_x} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial W_x}$$

$$\frac{\partial L_t}{\partial W_y} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial W_y}$$

$$\frac{\partial L_t}{\partial b_y} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial b_y}$$

where $\frac{\partial L_t}{\partial y_t}$, $\frac{\partial y_t}{\partial h_t}$, and $\frac{\partial h_t}{\partial W_h}$ are the partial derivatives of the loss, the output, and the hidden state with respect to the weights and biases, respectively. These partial derivatives are calculated using the chain rule.

- Weight and bias update: The weights and biases are updated using the gradient of the loss and a learning rate, which determines the size of the update. The update equations are as follows:

$$W_h = W_h - \alpha\frac{\partial L}{\partial W_h}$$

$$b_h = b_h - \alpha\frac{\partial L}{\partial b_h}$$

$$W_x = W_x - \alpha\frac{\partial L}{\partial W_x}$$

$$W_y = W_y - \alpha\frac{\partial L}{\partial W_y}$$

$$b_y = b_y - \alpha\frac{\partial L}{\partial b_y}$$

where α is the learning rate.

RNNs are powerful models for processing sequential data and have been used in many different applications, such as language modeling, machine translation, and speech recognition. They are particularly useful for tasks that require the network to remember long-term dependencies in the data, such as in natural language processing.

However, RNNs are not without their limitations. Firstly, RNNs have a limited memory: RNNs have a limited ability to remember information from long sequences. This can make it difficult for them to accurately process very long sequences or sequences with long-term dependencies. Also, when training RNNs, the gradients of the parameters can become very small, which can make it difficult for the network to learn. This is known as the vanishing gradient problem. On the other hand, the gradients of the parameters can also become very large, which can lead to instability in the network and make it difficult to train. This is known as the exploding gradient problem.

5.3 Long Short-Term Memory

Long Short-Term Memory Network (LSTM) is a variant of RNN that can better retain relevant information from the past sequence [Sta19]. The LSTM network is specifically designed to overcome the long-distance dependency issues present in other RNN networks, thus solving tasks that are difficult to solve by RNN. The main difference between LSTM and the vanilla RNN architecture is the presence of gates. LSTM networks have a special type of memory cell called a “forget gate” that can selectively retain or forget information. This allows LSTM networks to remember important information for long periods of time, while also forgetting irrelevant information. LSTM networks also have “input” and “output” gates that control how information flows into and out of the memory cells. This allows LSTM networks to process data in a highly efficient and effective manner. These gates enable the LSTM model to have long-term temporal dependencies.

At a high level, LSTM networks use matrix operations to update the values in the memory cells and control gates. These operations typically include dot products, element-wise multiplication, and element-wise addition. LSTM networks also use sigmoid and hyperbolic tangent (tanh) functions to squash the values in the memory cells and control gates between a range of 0 and 1, which allows the network to learn and make decisions based on these values.

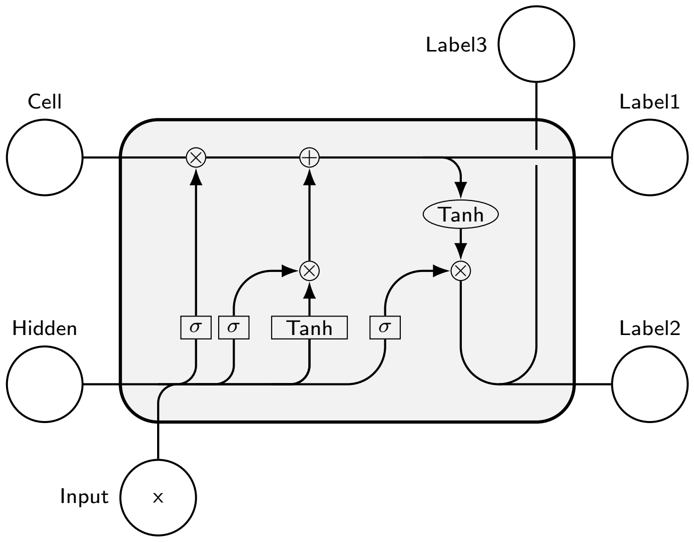


Figure 7: LSTM cell architecture

The basic design of an LSTM cell consists of four main components:

1. The input gate: This gate controls the flow of information from the input into the cell. It uses a sigmoid activation function to decide which part of the input should be passed on to the cell, and which should be discarded.

2. The forget gate: This gate controls the flow of information from the cell’s previous state (its “memory”) into the current state. It uses a sigmoid activation function to decide which part of the previous state should be kept, and which should be forgotten.

3. The cell state: This is the core of the LSTM cell, where information is stored and processed. It is a vector of values that are updated based on the input and the previous state.

4. The output gate: This gate controls the flow of information from the cell state to the output of the LSTM cell. It uses a sigmoid activation function to decide which part of the cell state should be passed on to the output, and which should be discarded.

These four components are connected together in a specific way, with the output of one component serving as the input to the next. The input gate and forget gate each receive input from the current input and the previous cell state, and use these inputs to decide which information to pass on to the cell state. The cell state then combines this information with the current input to update its own values. Finally, the output gate receives input from the cell state and uses this input to produce the output of the LSTM cell.

This design allows the LSTM cell to process long sequences of data and retain important information for long periods of time.
Steps for training the LSTM model are as follows:

- First, we need forward propagation:
  At each time step, the input to the LSTM is a vector x_t and the hidden state is a vector h_t. The output at each time step is a vector y_t. The hidden state and output are calculated using the following equations:

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i)$$

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f)$$

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o)$$

$$g_t = \tanh(W_{gx}x_t + W_{gh}h_{t-1} + b_g)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$

$$y_t = \sigma(W_{yx}x_t + W_{yh}h_t + b_y)$$

  where W_{ix}, W_{fx}, W_{ox}, and W_{gx} are the weights that connect the input at the current time step to the input, forget, output, and cell input gates, respectively, W_{ih}, W_{fh}, W_{oh}, and W_{gh} are the weights that connect the hidden state at the previous time step to the input, forget, output, and cell input gates, respectively, and b_i, b_f, b_o, and b_g are the biases of the input, forget, output, and cell input gates, respectively. σ is the sigmoid activation function and $\odot$ is the element-wise multiplication operator.

- Then we calculate the loss:
  The loss is calculated by comparing the predicted output at each time step to the true output. The loss at each time step is given by:

$$L_t = \frac{1}{2}(y_t - \hat{y}_t)^2$$

where y_t is the true output and $\hat{y}_t$ is the predicted output. The total loss is then given by:

$$L = \sum_{t=1}^{T} L_t$$

where T is the number of time steps in the input sequence.

- Backward propagation with the loss:
  The gradient of the loss with respect to the weights and biases is calculated using the chain rule. The gradient of the loss with respect to the weights and biases at each time step is given by:

$$\frac{\partial L_t}{\partial W_{ix}} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial i_t}\frac{\partial i_t}{\partial W_{ix}}$$

$$\frac{\partial L_t}{\partial W_{fx}} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial f_t}\frac{\partial f_t}{\partial W_{fx}}$$

$$\frac{\partial L_t}{\partial W_{ox}} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial W_{ox}}$$

$$\frac{\partial L_t}{\partial W_{gx}} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial g_t}\frac{\partial g_t}{\partial W_{gx}}$$

$$\frac{\partial L_t}{\partial W_{yx}} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial W_{yx}}$$

$$\frac{\partial L_t}{\partial W_{ih}} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial i_t}\frac{\partial i_t}{\partial W_{ih}}$$

$$\frac{\partial L_t}{\partial W_{fh}} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial f_t}\frac{\partial f_t}{\partial W_{fh}}$$

$$\frac{\partial L_t}{\partial W_{oh}} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial W_{oh}}$$

$$\frac{\partial L_t}{\partial W_{gh}} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial g_t}\frac{\partial g_t}{\partial W_{gh}}$$

$$\frac{\partial L_t}{\partial W_{yh}} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial W_{yh}}$$

$$\frac{\partial L_t}{\partial b_i} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial i_t}\frac{\partial i_t}{\partial b_i}$$

$$\frac{\partial L_t}{\partial b_f} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial f_t}\frac{\partial f_t}{\partial b_f}$$

$$\frac{\partial L_t}{\partial b_o} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial b_o}$$

$$\frac{\partial L_t}{\partial b_g} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial g_t}\frac{\partial g_t}{\partial b_g}$$

$$\frac{\partial L_t}{\partial b_y} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial b_y}$$

where $\frac{\partial L_t}{\partial y_t}$, $\frac{\partial y_t}{\partial h_t}$, are the partial derivatives of the loss and the output, and $\frac{\partial h_t}{\partial c_t}$, $\frac{\partial c_t}{\partial i_t}$, are the partial derivatives of the hidden state and the cell state, and $\frac{\partial i_t}{\partial W_{ix}}$, $\frac{\partial f_t}{\partial W_{fx}}$, $\frac{\partial o_t}{\partial W_{ox}}$, are the partial derivatives of the input gate, the forget gate, and the output gate, and $\frac{\partial g_t}{\partial W_{gx}}$, $\frac{\partial y_t}{\partial W_{yx}}$, are the partial derivatives of the cell input and the output, and $\frac{\partial i_t}{\partial W_{ih}}$, $\frac{\partial f_t}{\partial W_{fh}}$, $\frac{\partial o_t}{\partial W_{oh}}$, $\frac{\partial g_t}{\partial W_{gh}}$, $\frac{\partial y_t}{\partial W_{yh}}$, are the partial derivatives of the input gate, the forget gate, the output gate, the cell input, the output with respect to the weights, where $\frac{\partial i_t}{\partial b_i}$, $\frac{\partial f_t}{\partial b_f}$, $\frac{\partial o_t}{\partial b_o}$, and $\frac{\partial g_t}{\partial b_g}$ are the partial derivatives of the input gate, the forget gate, the output gate, the cell input, and the output and biases. All of these partial derivatives are calculated using the chain rule.

- Update weights and biases:
  The weights and biases are updated using the gradient of the loss and a learning rate, which determines the size of the update. The update equations are as follows:

$$W_{ix} = W_{ix} - \alpha\frac{\partial L}{\partial W_{ix}}$$

$$W_{fx} = W_{fx} - \alpha\frac{\partial L}{\partial W_{fx}}$$

$$W_{ox} = W_{ox} - \alpha\frac{\partial L}{\partial W_{ox}}$$

$$W_{gx} = W_{gx} - \alpha\frac{\partial L}{\partial W_{gx}}$$

$$W_{yx} = W_{yx} - \alpha\frac{\partial L}{\partial W_{yx}}$$

$$W_{ih} = W_{ih} - \alpha\frac{\partial L}{\partial W_{ih}}$$

$$W_{fh} = W_{fh} - \alpha\frac{\partial L}{\partial W_{fh}}$$

$$W_{oh} = W_{oh} - \alpha\frac{\partial L}{\partial W_{oh}}$$

$$W_{gh} = W_{gh} - \alpha\frac{\partial L}{\partial W_{gh}}$$

$$W_{yh} = W_{yh} - \alpha\frac{\partial L}{\partial W_{yh}}$$

$$b_i = b_i - \alpha \frac{\partial L}{\partial b_i}$$

$$b_f = b_f - \alpha \frac{\partial L}{\partial b_f}$$

$$b_o = b_o - \alpha \frac{\partial L}{\partial b_o}$$

$$b_g = b_g - \alpha \frac{\partial L}{\partial b_g}$$

$$b_y = b_y - \alpha \frac{\partial L}{\partial b_y}$$

where α is the learning rate.

Overall, LSTM recurrent neural networks can be a good model for predicting cryptocurrency prices or other financial time series data. This is because LSTM networks are particularly well-suited to modeling sequential data, such as time series data, where there may be dependencies between observations that occur at different points in time.

Additionally, LSTM networks are capable of learning and adapting to changing patterns in the data over time, which can be useful in the fast-moving and volatile cryptocurrency market.

In terms of specific implementation details, there are a few key factors to consider when using LSTM networks for cryptocurrency prediction. One important factor is the choice of input features for the model. It is often useful to include a variety of different indicators, such as technical analysis indicators or fundamental data, as input features in order to capture different aspects of the market.

Another important factor is the length of the time series used as input to the model. In general, using a longer history of data can provide the model with more context and help it make more accurate predictions. However, it is also important to consider the trade-off between using more data and the potential for the model to overfit to the training data.

6 Methods and Results: Machine Learning

For the task of portfolio optimization, we have found that the LSTM model performs best for predicting the price of individual cryptocurrencies. This section of the paper will discuss the methods used for choosing an optimal portfolio of cryptocurrencies using ML and portfolio optimization techniques.

6.1 Dataset

To predict future crypto prices, we trained an LSTM network on the *top-50-cryptocurrencies-historical-prices* dataset.

We made predictions for the following 19 cryptocurrencies for the duration May 31, 2021 - July 31, 2021.

Aave	Binance Coin	Bitcoin
Bitcoin Cash	Cardano	Chainlink
Dai	Dash	Dogecoin
EOS	Ethereum	IOTA
Litecoin	Maker	Monero
THETA	Tezos	Tron
VeChain		

To pre-process the data, we first performed feature engineering to create a set of informative features from the original features in the dataset. We then re-scaled all features to be used for the model's predictions before applying a standard train-test-validation (85/10/5) split for the model to learn from. The training data consisted of the first 85% of time steps in the data, the validation data consisted of the next 5% of time steps, and then the testing data was the final 10% of time steps in our dataset. During the data splitting process, several tests were performed to make sure there was no data leakage between the different splits. Training data and testing data were separate to ensure there was no overfitting. Afterwards, the training data was split into sequences of 20 time steps, each with an assigned label, before being fed into the model for training.

6.2 Model Training

After the data was pre-processed and ready to be used for training, we started designing and modifying the training pipeline for our LSTM model. 1 model was trained for each respective cryptocurrency, so in total 19 LSTMs were trained to predict for our cryptocurrencies. The architecture for each model is identical and is a standard LSTM model with 28 hidden layers and 2 fully connected layers for the regressor. Additionally, a dropout layer of 0.2 was added right before the last layer to help with overfitting on the cryptocurrencies with less data. All models were trained on P100 GPUs for a maximum of 100 epochs (early stopping was implemented). The models were trained using standard mini-batch gradient descent with an initial learning rate of .001, a batch size of 64, and the learning rate was divided by 10 for every 2 epochs of training where model performance had not improved. Since this is a regression task, the loss function we used was mean squared error and the optimizer was Adaptive Moment Estimation (Adam).

See A.1 for the neural network performance for each of our models on the holdout test set that was employed.

6.3 Portfolio Construction using Forecasted Prices

Using our available price predictions, we construct returns for the individual cryptocurrencies and their corresponding covariance matrix over the period of time. Assuming the point of view of an investor who could assume a long position over any compositions of the 19 cryptocurrencies in our portfolio during the period May 31, 2021 - July 31, 2021, we follow the methodologies illustrated in the portfolio construction section to derive the vector of allocation, **m**, as well as the vector of returns, and solve the Sharpe Ratio maximization problem.

Using the prices we generate from our LSTM, we are able to derive an expression for expected returns and the covariance matrix for the expected return for each crypto in our portfolio:

$$E[\mu_i] = \frac{E[Pricei, t+1] - E[Pricei, t]}{E[Pricei, t]}$$

Now, $\mathbf{\Sigma}$ becomes the covariance of expected returns. We substitute the expected returns and the covariance matrix into our optimization problem:

$$\max_{\mathbf{m}} \frac{\sum_{i=1}^{N} m_i \cdot E[\mu_i] - R_f}{\sqrt{\mathbf{m}'\mathbf{\Sigma}\mathbf{m}}}$$

subject to

$$\mathbf{m}'\mathbf{1} = 1$$

The allocation we obtain is given as:

Cryptocurrency	Allocation (percentage)
Aave	1.15
Binance Coin	1.57
Bitcoin	15.87
Bitcoin Cash	16.1
Cardano	0.69
Chainlink	0.83
Dai	6.12
Dash	0.39
Dogecoin	16.25
EOS	1.21
Ethereum	3.55
IOTA	3.3
Litecoin	4.8
Maker	4.43
Monero	3.71
THETA	1.63
Tezos	10.39
Tron	2.75
VeChain	5.24

Table 1: Asset Allocation (percentile)

Using this as our vector $\mathbf{m}$, we compare the actual annualized portfolio returns generated by this portfolio with 25,000 randomly sampled portfolios with allocation weights summing up to 1 (treating this as the index portfolio). We obtain the following results:

Average Returns (percentage) from Monte Carlo Simulations	Returns (percentage) from Sharpe Optimization
2.510	2.830

Table 2: Returns comparison

The 2.510 percents in returns obtained from Monte Carlo simulations represents the expected payoff of an investor betting their stake on the portfolio in a random fashion. Compared to this benchmark, we have achieved a significant performance improvement of 12.7 percent! We have indeed been able to generate a good amount of alpha by combining our portfolio optimization and machine learning methodologies. This is contrarian to classical financial theories, which posits the impossibility to generate above market level returns using only publicly available information.

Princeton's professor Burton Malkiel claimed in his bestselling book, A Random Walk Down Wall Street, that "A blindfolded monkey throwing darts at a newspaper's financial pages could select a portfolio that would do just as well as one carefully selected by experts." The famously known efficient market hypothesis (EMH) posits that at the time of trading, security prices reflect all information. Theoretically, no above market returns can be generated from only analysing public information such as prices and volumes. In the industry, hedge funds rely on privately available information obtained from speaking to internal members of their portfolio companies to gain competitive advantage.

Given these, it is surprising that we are able to generate returns significantly above market level using limited data. Cryptocurrencies being a relatively new asset class may contribute to our success, as there may be fewer investors who are participating in the transaction of such financial products.

7 Future Work

Adaptations to our work could further explore ways to extract signals and optimize returns for a given cryptocurrency portfolio. There are other optimization methods that can be implemented to potentially improve the performance. Some methods to consider are:

- Modern Portfolio Theory (MPT): This is an extension of Markowitz Portfolio Optimization that takes into account the fact that different assets may have different levels of risk and return, and that these characteristics may change over time. MPT also assumes that investors are risk-averse and will only invest in portfolios that offer the highest expected return for a given level of risk [Sha70].

- Black-Litterman Model: This is a portfolio optimization technique that combines the principles of Markowitz Portfolio Optimization with the use of subjective views or opinions about the expected returns of different assets. It allows investors to incorporate their own views about the market into their portfolio construction process [LRR03].

- Risk Parity: This is a portfolio construction technique that aims to achieve a balance of risk across different asset classes by allocating a higher proportion of portfolio assets to those that have lower risk. The goal of risk parity is to achieve a more diversified and balanced portfolio that is less susceptible to the risk of a single asset class [EV11].

There are alternative methods of constructing a portfolio, including those that rely on no financial theory or models. One such method is with Deep Reinforcement Learning [Jia19], whereby we may use deep reinforcement learning methods to directly produce a portfolio vector, with raw market data and historic prices as the input.

8 Conclusion

In this study, we present a comprehensive solution to the challenging problem of portfolio construction for cryptocurrency, a highly innovative and rapidly evolving asset class with a unique set of properties and characteristics. To address this problem, we have adopted a two-pronged approach:

First, we have applied the classical Markowitz model, a widely recognized and widely used framework for modern portfolio theory, to our dataset of cryptocurrency assets. Specifically, we have implemented the optimization problems of minimizing portfolio variance, maximizing the Sharpe Ratio, and constructing the efficient frontier, and have analyzed the resulting asset allocation patterns. Our analysis has revealed that different objective functions can lead to significantly different outcomes in terms of asset allocation, and that a diversified portfolio comprising a wide range of cryptocurrencies is likely to yield better optimization results.

In the second part of this paper, we have replicated the full quantitative research workflow that would typically be followed by investors in a hedge fund or asset management setting. Specifically, we have forecasted expected returns by leveraging long short-term memory (LSTM) techniques to generate price predictions, and have then assumed the perspective of a long-only, Sharpe Ratio-maximizing hedge fund researcher, incorporating returns into our optimization problem.

Through the combined approach of portfolio construction and LSTM techniques, we were able to construct an optimal cryptocurrency portfolio that outperforms the expected returns of a typical, randomly generated portfolio by more than 12%. Our findings challenge the efficient market hypothesis in the context of cryptocurrency investments, and suggest that there may be significant opportunities for active portfolio management in this asset class.

References

[Dhi] Shubham Dhingra. Simplified mathematics behind neural networks. https://towardsdatascience.com/simplified-mathematics-behind-neural-networks-f2b7298f86a4.

[EV11] Claude Erb and Tadas Viskanta. Risk parity: a modern perspective. *Financial Analysts Journal*, 67(4):40–51, 2011.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[Jia19] Zhengyao Jiang. Cryptocurrency portfolio management with deep reinforcement learning. *arXiv preprint arXiv:1901.07601*, 2019.

[LRR03] Yves Lempérière, Christian P Robert, and Jean-Paul Renault. The black-litterman model: a review. *Journal of investment strategies*, 2(2):1–27, 2003.

[Mar52] Harry Markowitz. Portfolio selection. *The Journal of finance*, 7(1):77–91, 1952.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[Sha64] William F Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *The Journal of finance*, 19(3):425–442, 1964.

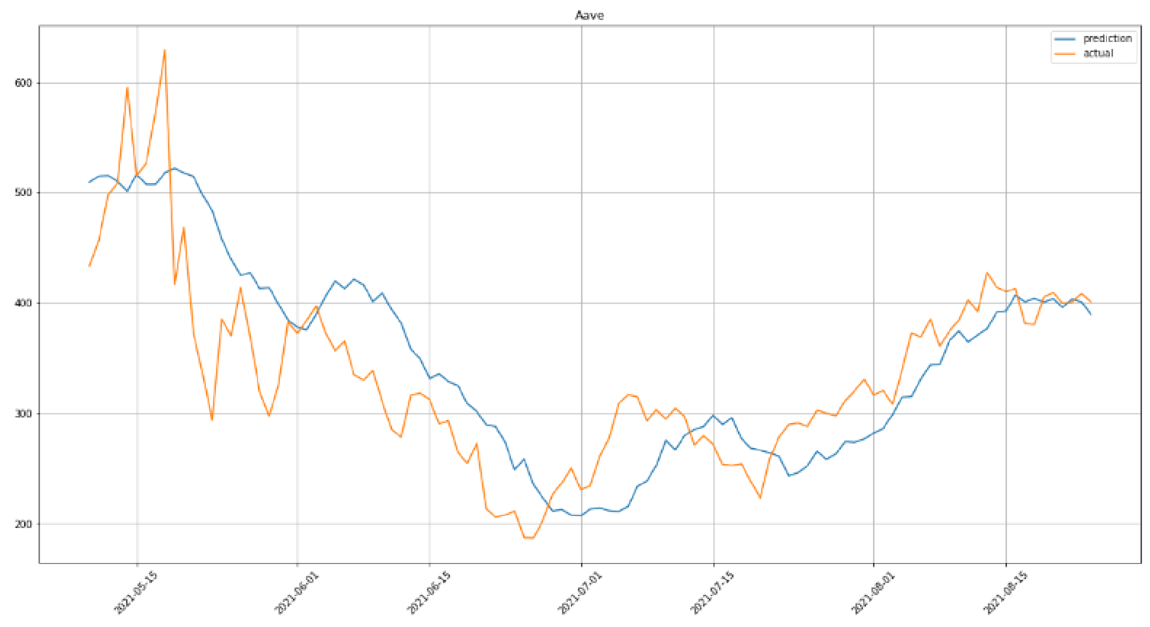
[Sha70] William F Sharpe. Mean-variance analysis in portfolio choice and capital markets. *The Journal of finance*, 25(2):383–417, 1970.

[She19] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn), 2019.

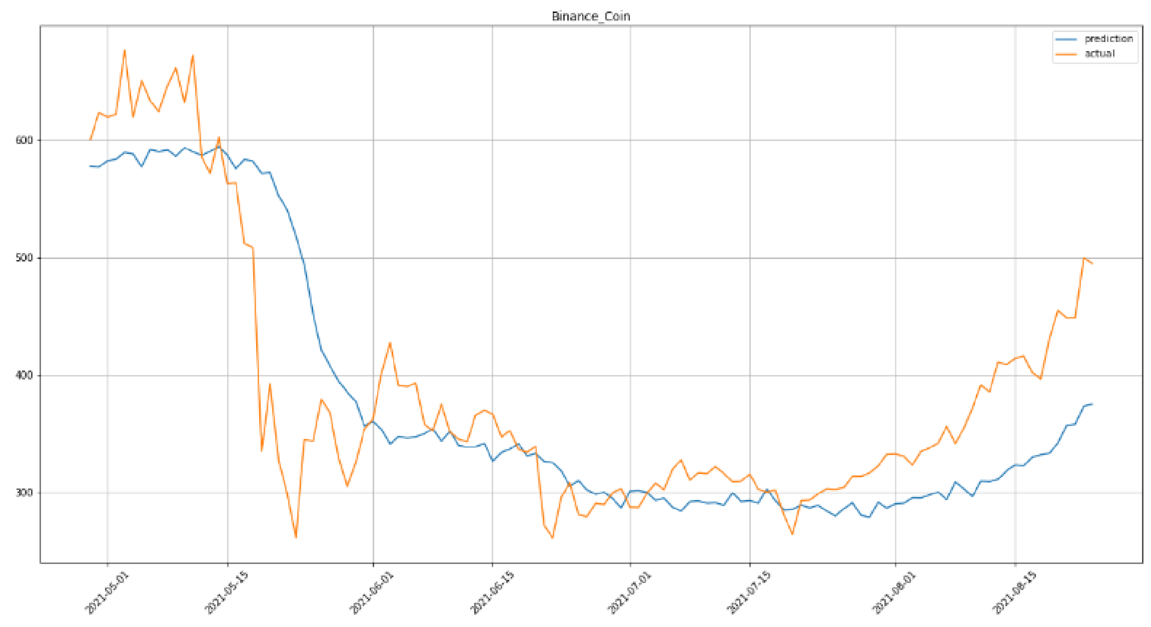
[Sta19] Ralf C. Staudemeyer. Understanding lstm: a tutorial into long short-term memory recurrent neural networks, 2019.

A Appendices

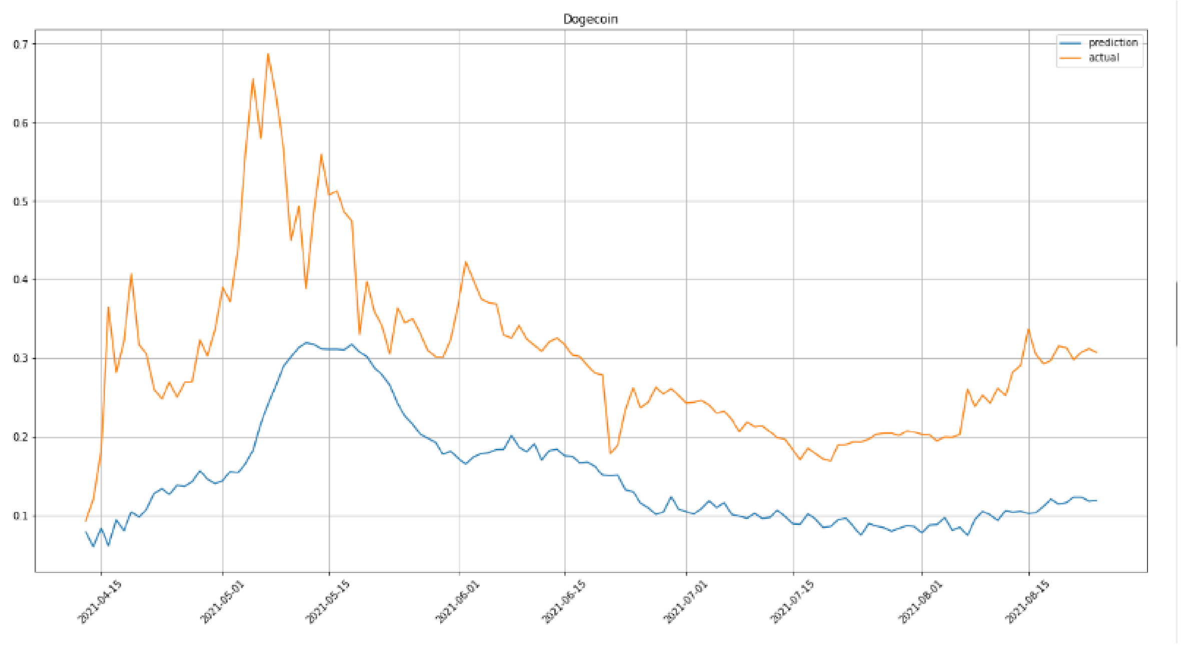
A.1 Model Performances



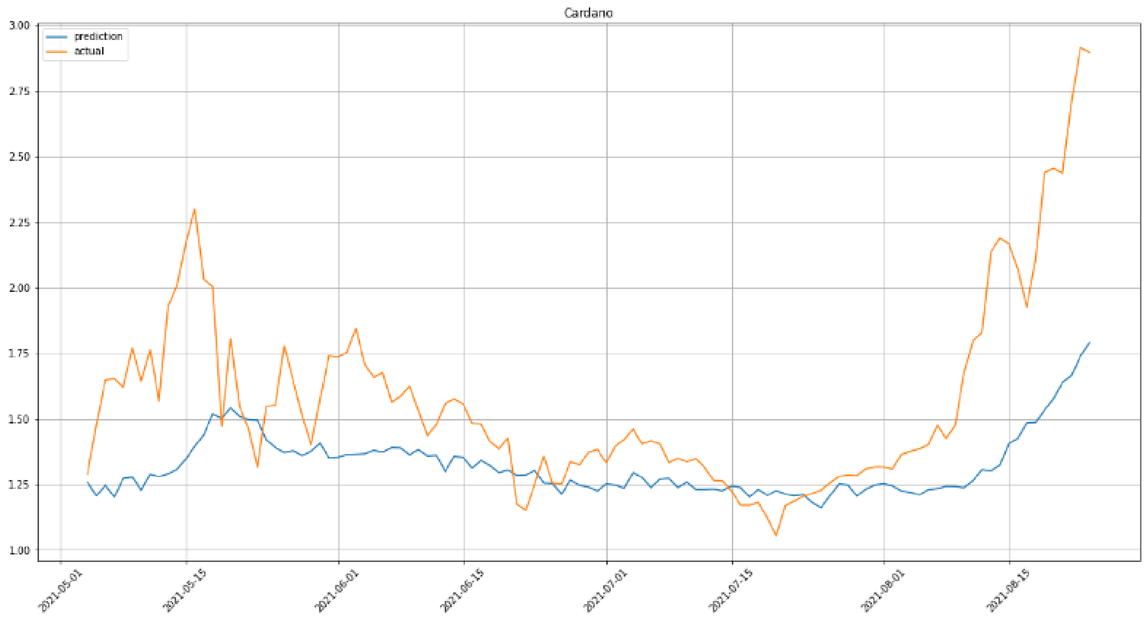
Aave



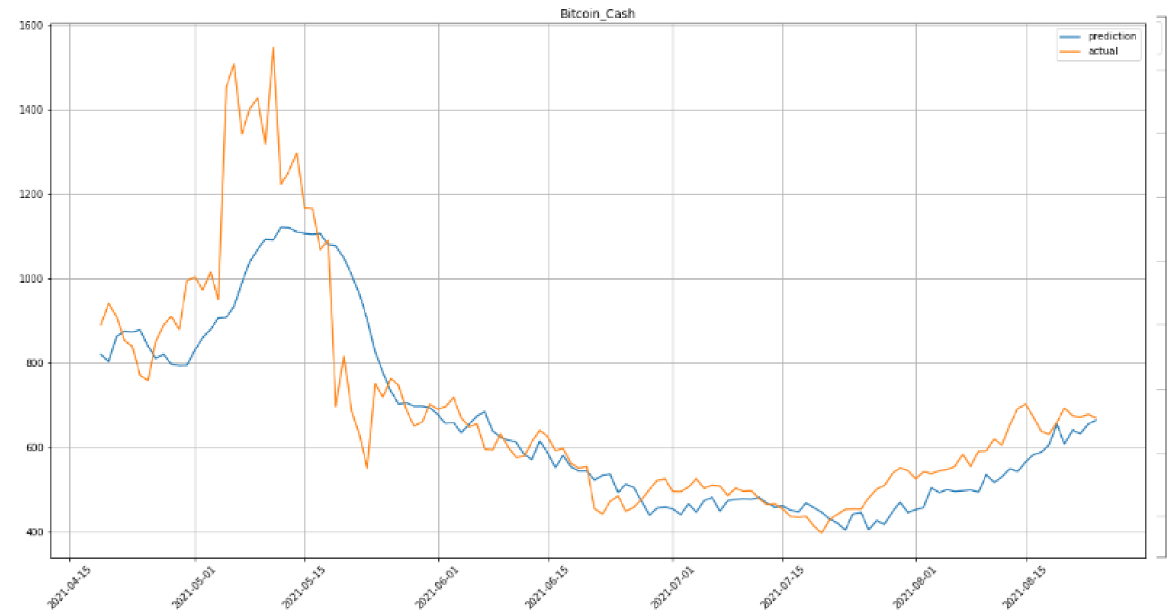
Binance Coin



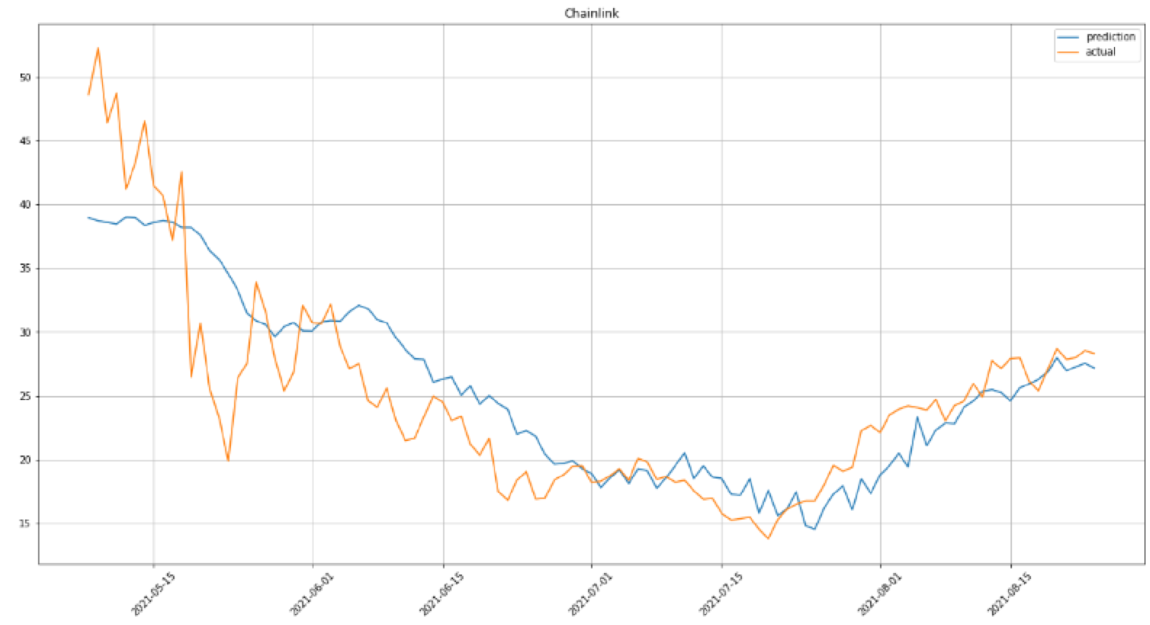
Dogecoin



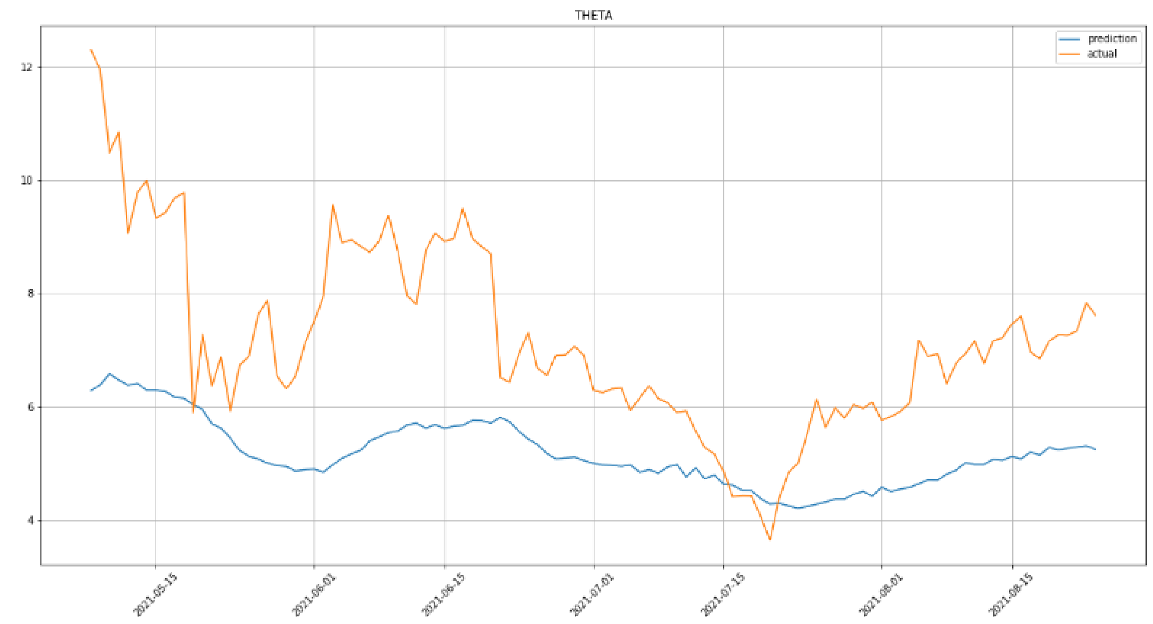
Cardano



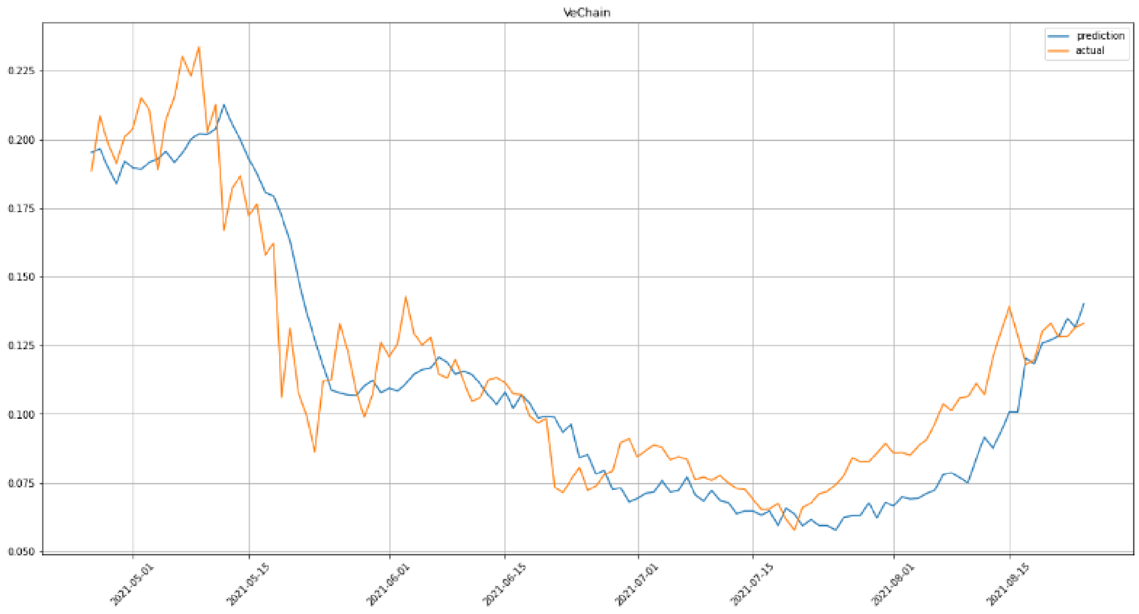
Bitcoin Cash



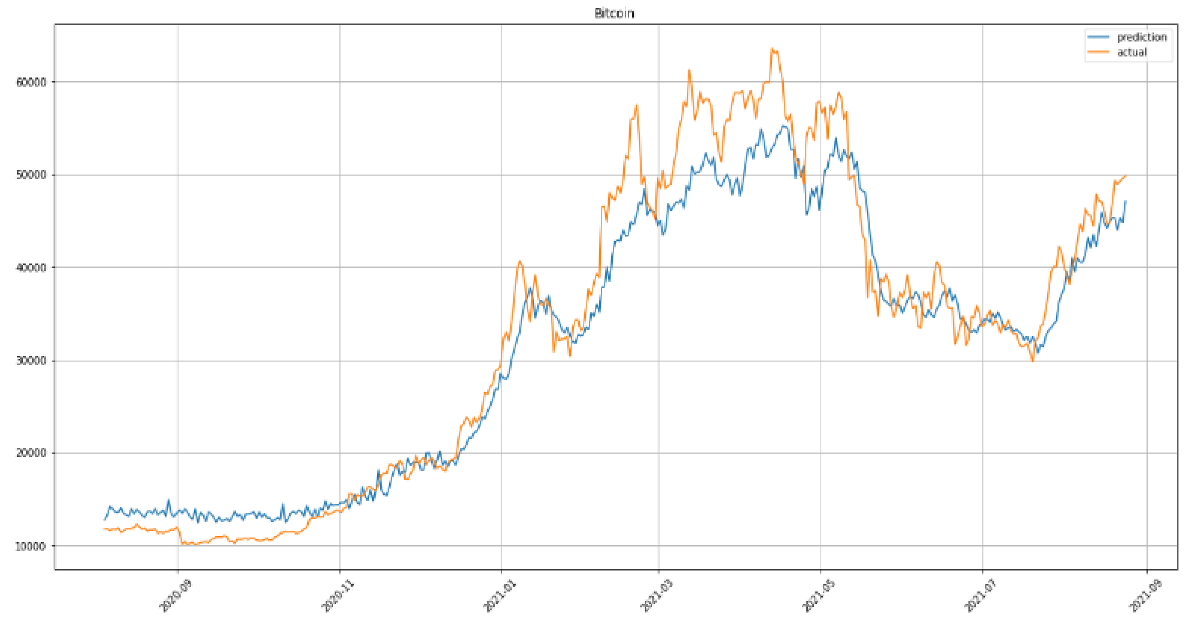
Chainlink



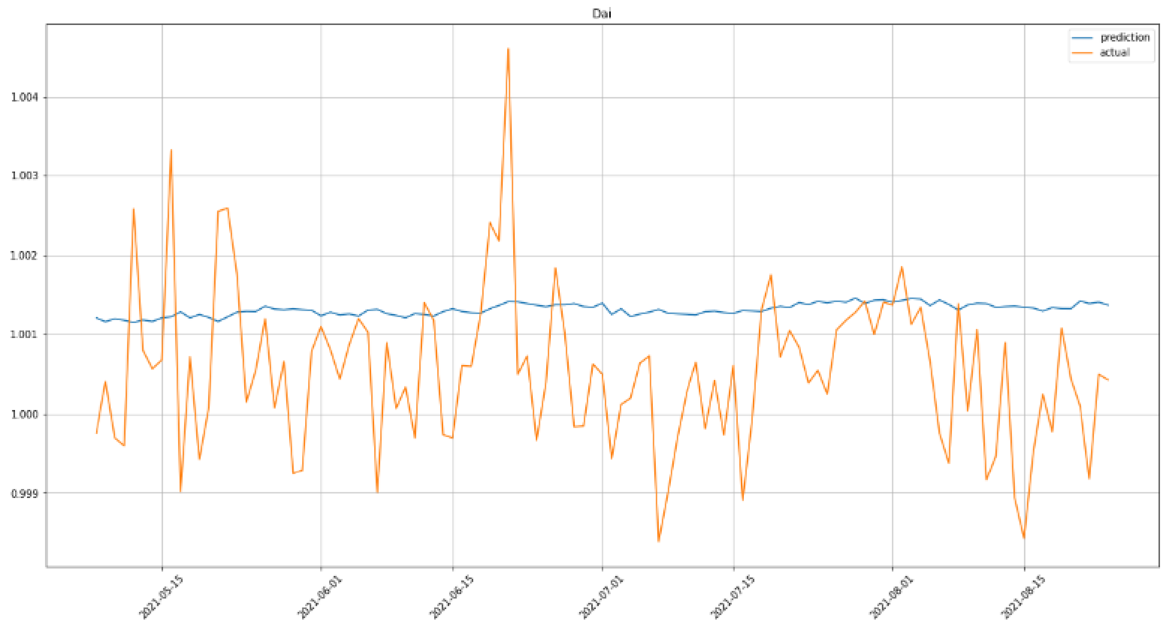
Theta



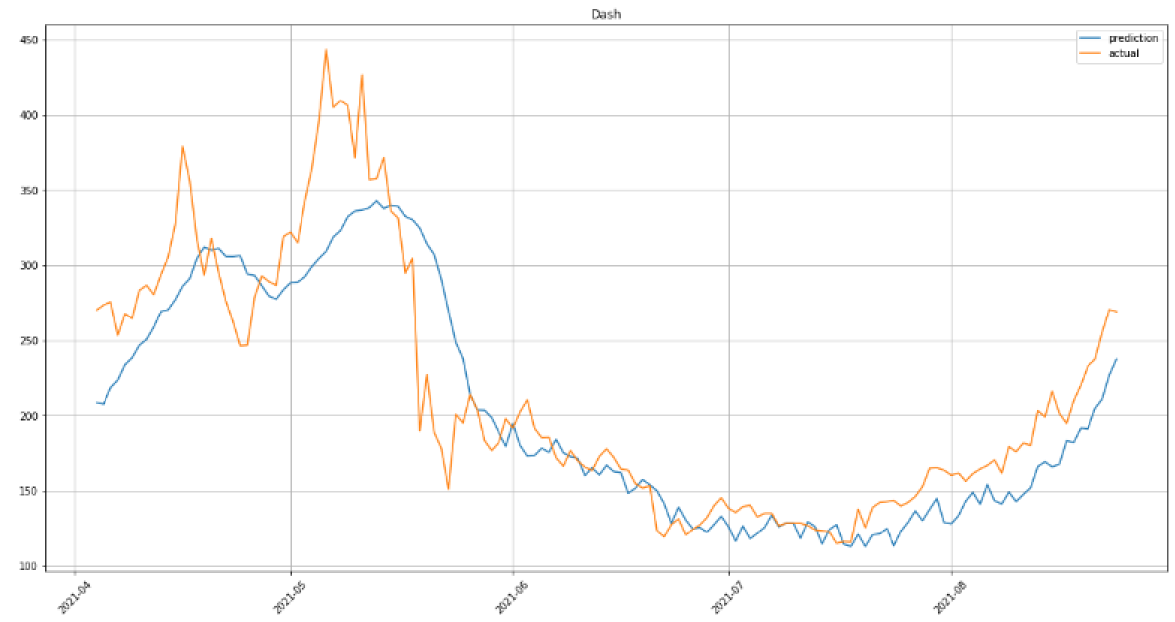
Vechain


Bitcoin
prediction
actual
60000
50000
40000
30000
20000
10000
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09

Bitcoin

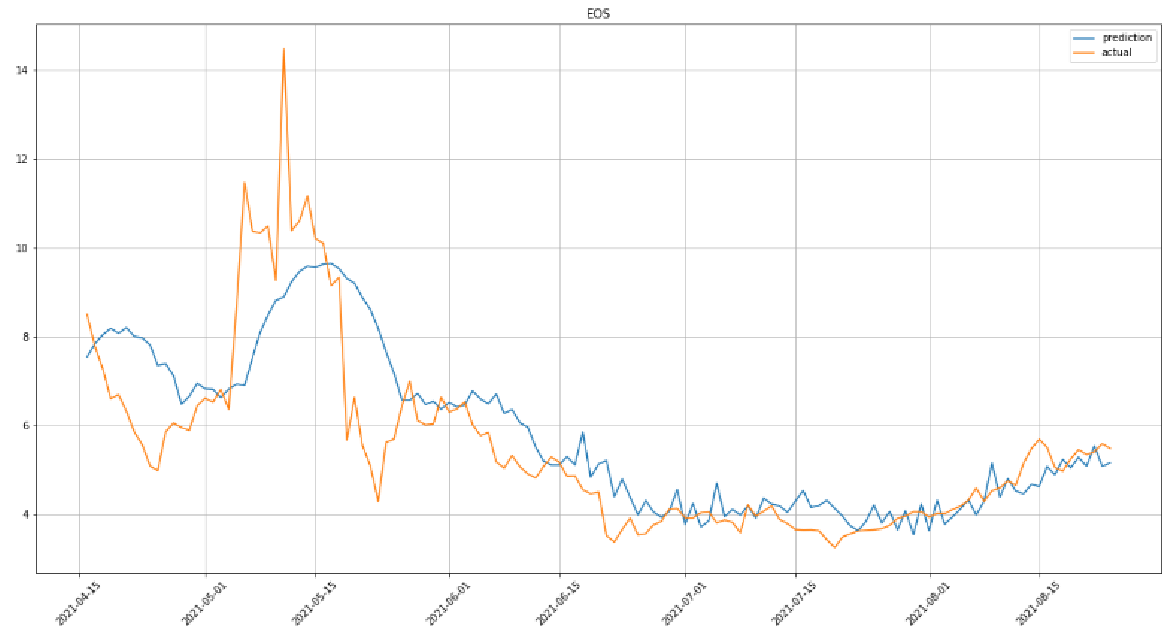

Dai
prediction
actual
1.004
1.003
1.002
1.001
1.000
0.999
2021-05-15
2021-06-01
2021-06-15
2021-07-01
2021-07-15
2021-08-01
2021-08-15

Dai



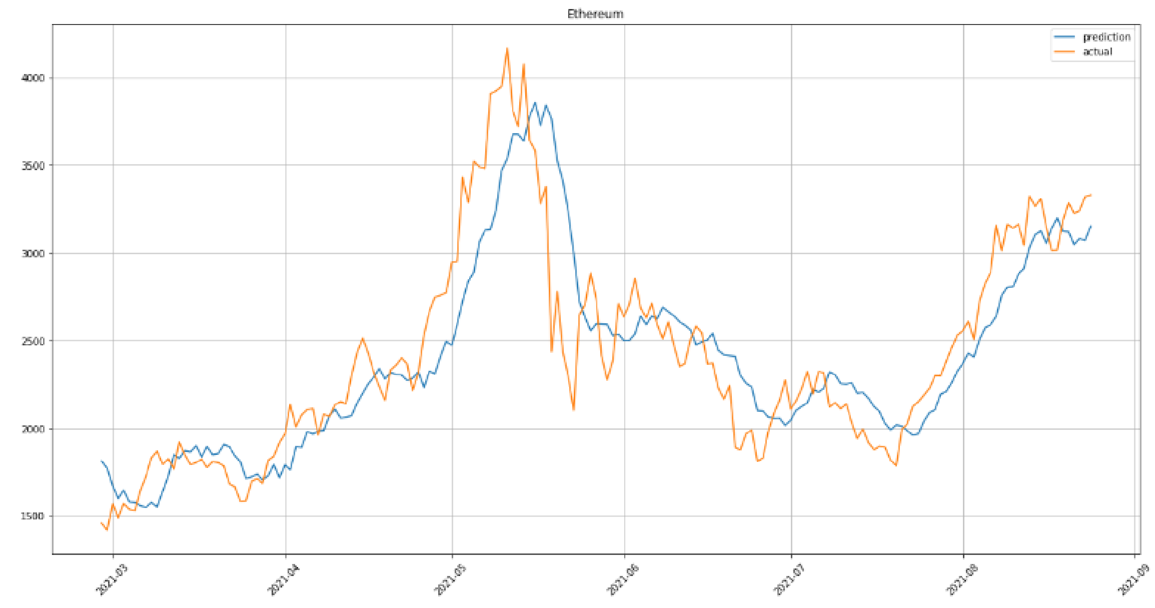
Dash
prediction
actual
450
400
350
300
250
200
150
100
2021-04
2021-05
2021-06
2021-07
2021-08

Dash

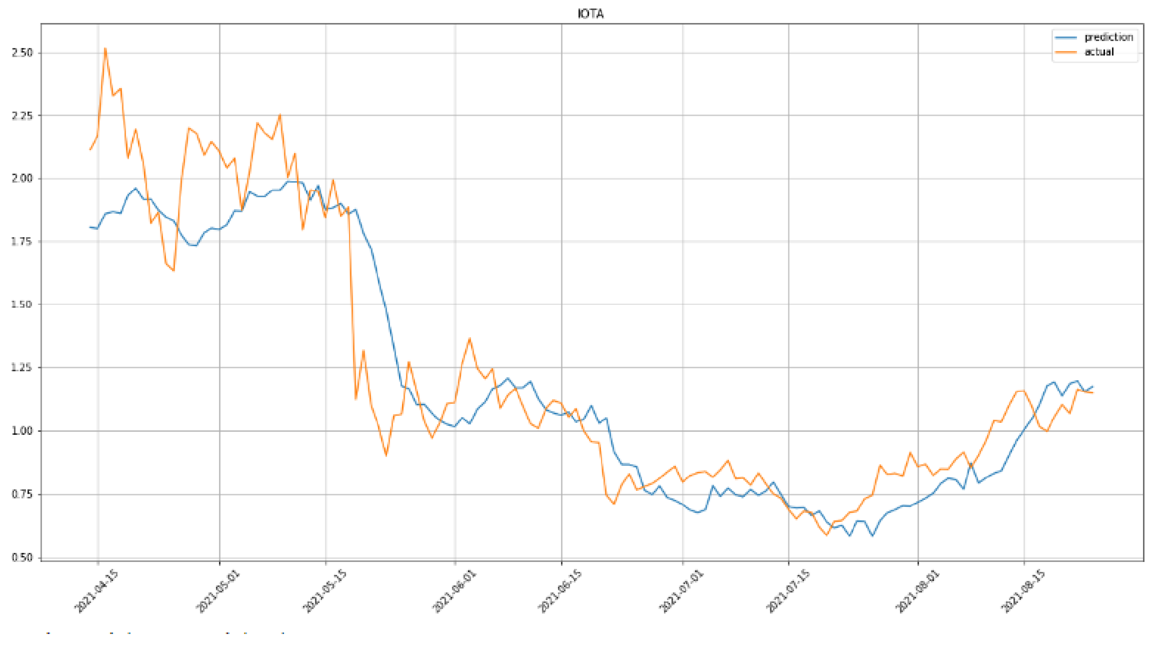


EOS
prediction
actual
14
12
10
8
6
4
2021-04-15
2021-05-01
2021-05-15
2021-06-01
2021-06-15
2021-07-01
2021-07-15
2021-08-01
2021-08-15

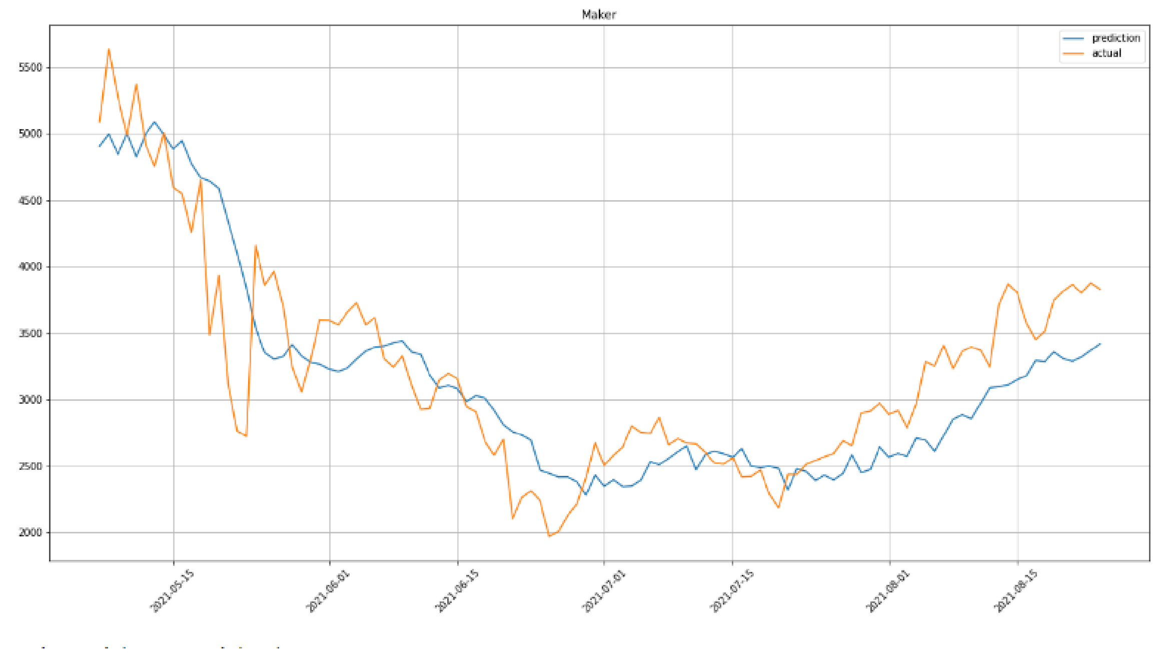
EOS



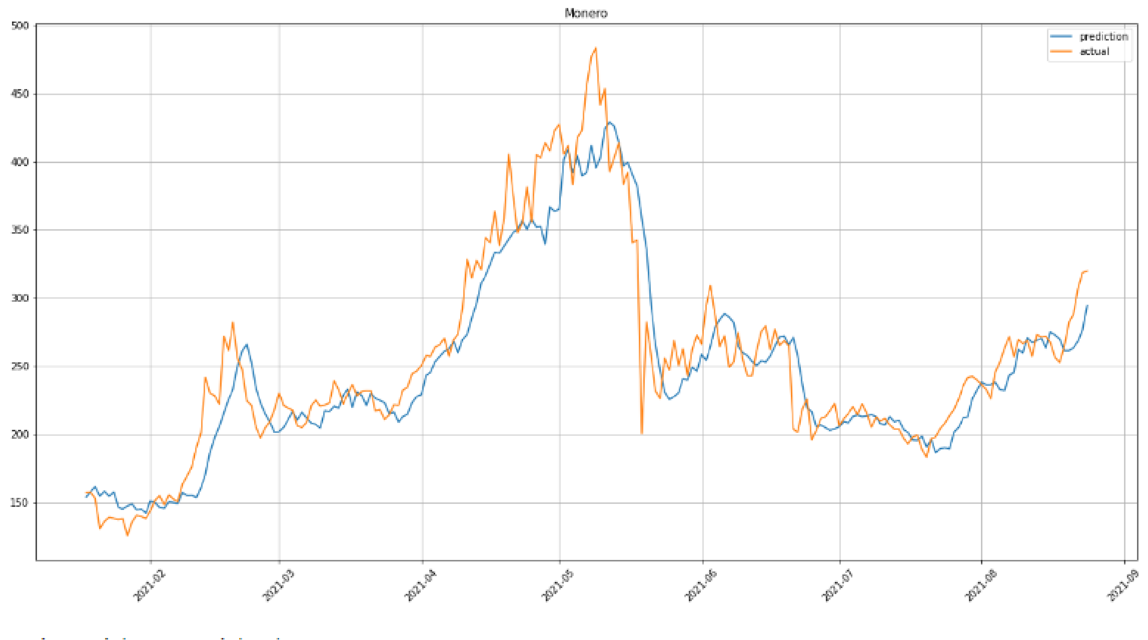
Ethereum



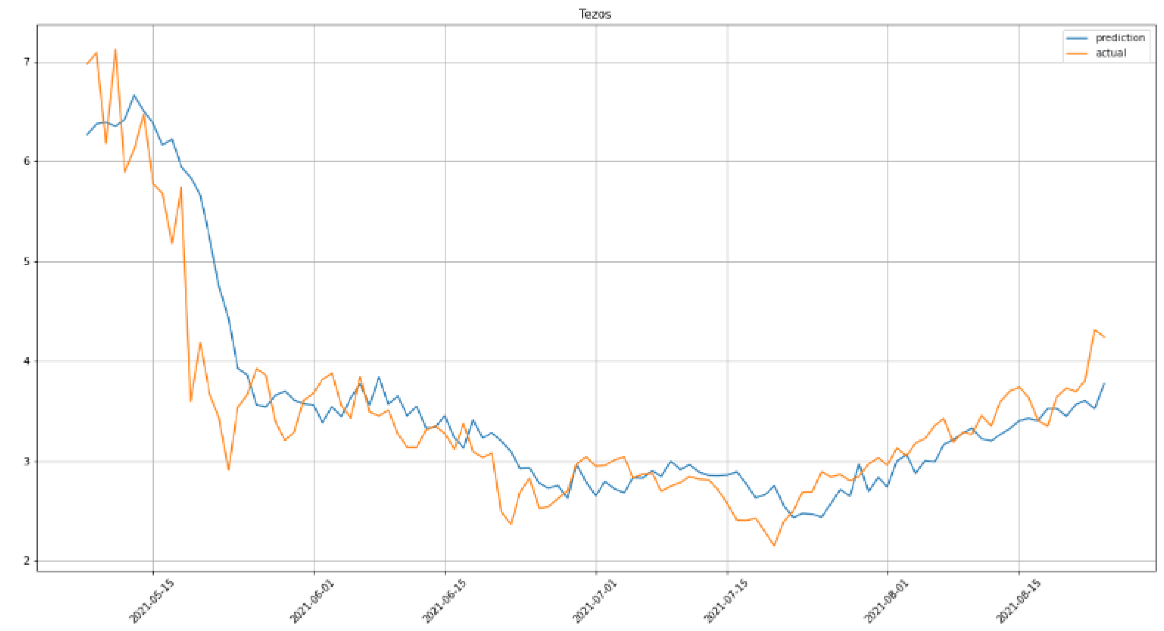
IOTA



Maker

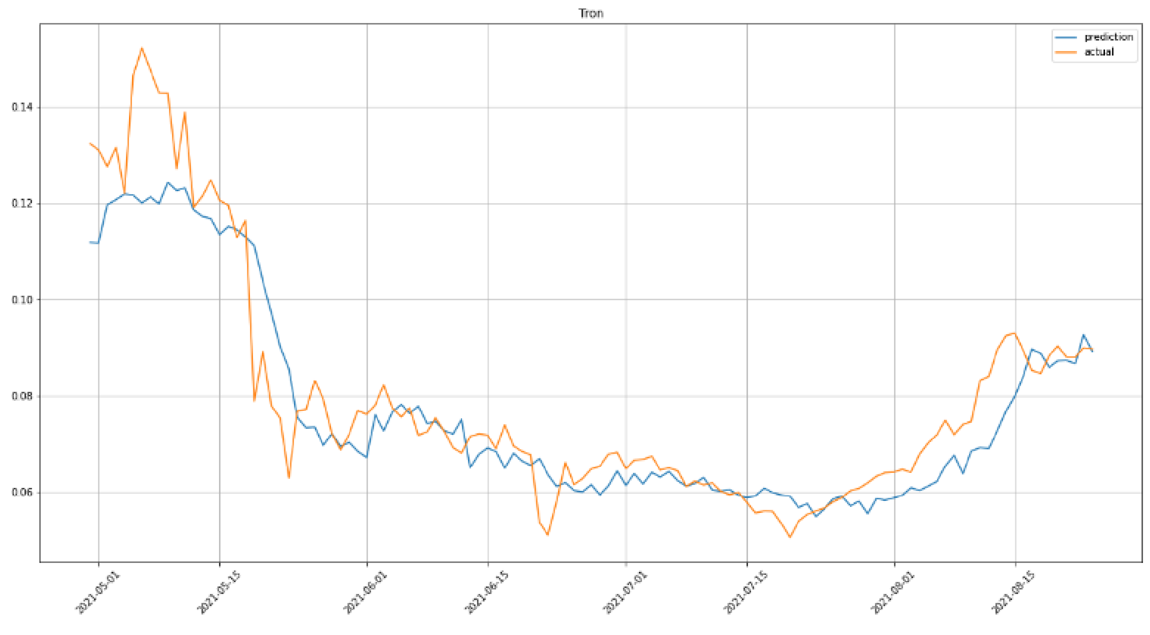


Monero



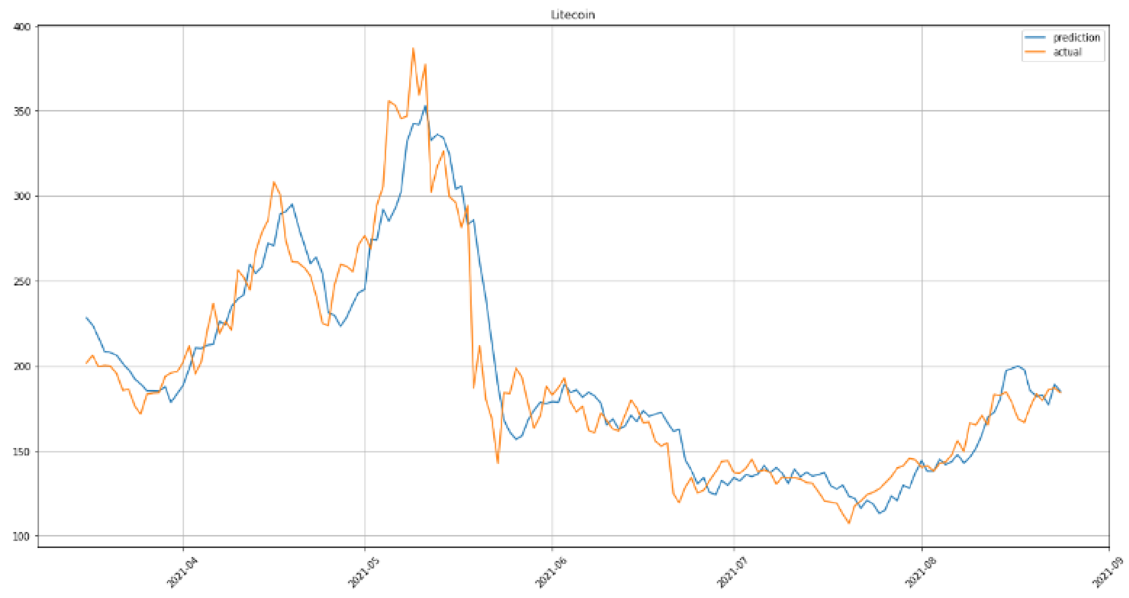
Tezos
prediction
actual
7
6
5
4
3
2
2021-05-15
2021-06-01
2021-06-15
2021-07-01
2021-07-15
2021-08-01
2021-08-15

Tezos



Tron
prediction
actual
0.14
0.12
0.10
0.08
0.06
2021-05-01
2021-05-15
2021-06-01
2021-06-15
2021-07-01
2021-07-15
2021-08-01
2021-08-15

Tron


Litecoin
prediction
actual
400
350
300
250
200
150
100
2021-04
2021-05
2021-06
2021-07
2021-08
2021-09

Litecoin