Abstract

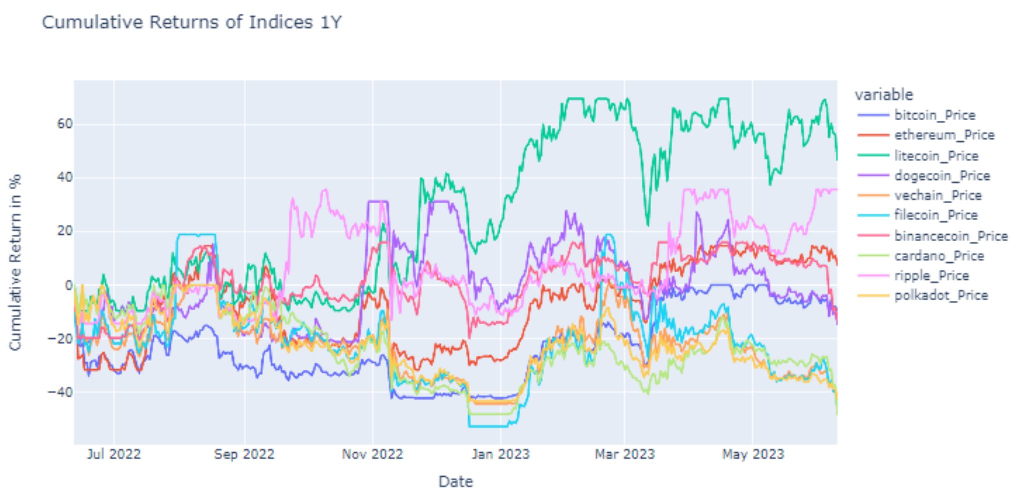
The goal of the case is to provide a framework for downloading historical cryptocurrency price data, preprocessing the data, optimizing portfolios using different models, and evaluating portfolio performance based on rebalancing frequency. It also includes an additional part of the implementation of the Q-learning algorithm for automatic stock trading.

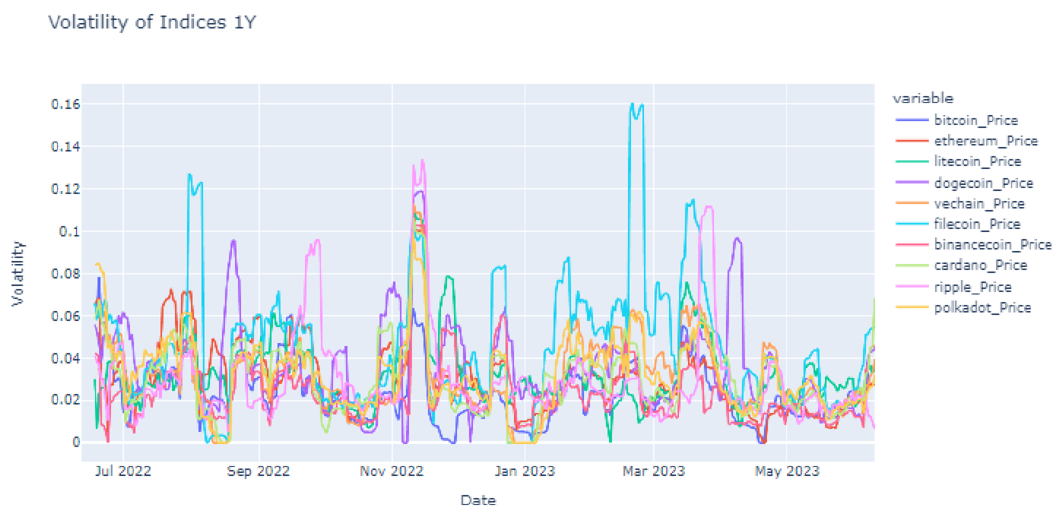
1 Data Retrieval

First, historical data for a given set of 10 cryptocurrency (bitcoin, ethereum, litecoin, dogecoin, vechain, filecoin, binancecoin, cardano, ripple, polkadot) from the CoinGecko API is downloaded for the last 365 days. It specifies a list of cryptocurrency IDs and the number of days of historical data to download (days). The code then iterates through the list of crypto IDs and downloaded data is stored in a list of DataFrames, where each DataFrame represents the historical price data for a specific cryptocurrency.

2 Pre-processing

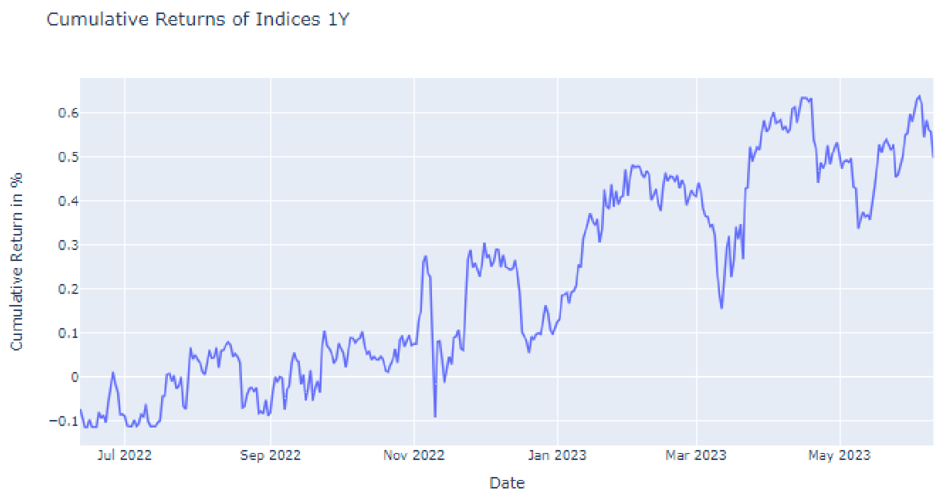
The code checks for missing values in the dataset using the and confirms that there are no missing values. Outliers in the dataset are handled by winsorizing the data at the 5th and 95th percentiles. The code calculates the daily and cumulative returns as well as the rolling volatility for each asset:



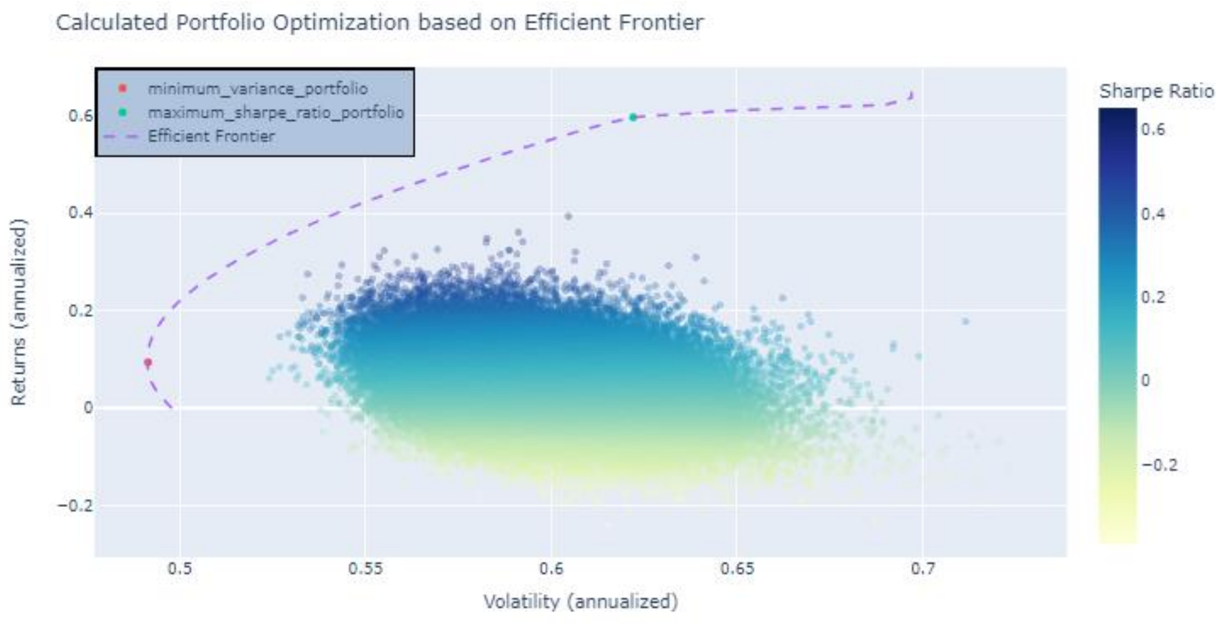


3 Portfolio Optimization and Evaluation

The code defines functions that find the portfolio weights that maximize the Sharpe ratio, and function finds the weights that minimize portfolio variance. It assumes that market risk-free rate is equal to 0. The code also defines function to calculate portfolio performance metrics and cumulative returns based on the returns and weights of the portfolio. The code iterates through different rebalancing frequencies (daily, weekly, and monthly) and optimization models to calculate the portfolio weights using the respective optimization model and evaluates the portfolio performance. The highest cumulative return is given by portfolio obtained using maximum Sharpe ration optimization using daily rebalancing rule with weights of 62.8% litecoin, 37.2% ripple:



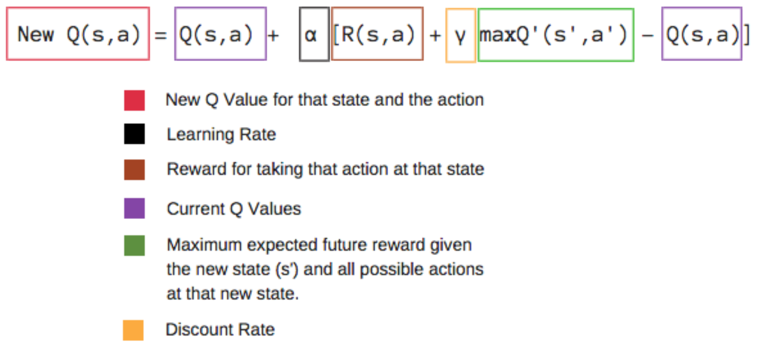
To plot the efficient frontier, 100000 random portfolios are generated by assigning random weights to the assets. Each portfolio's weights are normalized to ensure they sum up to 1. To calculate efficient frontier Calculation the weights for an efficient portfolio are determined. The efficient portfolio is one that achieves the specified target return. It uses an optimization algorithm to find the optimal weights that minimize the portfolio's volatility while satisfying the target return constraint. It means that the efficient frontier represents the set of portfolios that offer the maximum expected return for a given level of risk (volatility):



4 Additional part – Q-learning

Reinforcement learning is a branch of machine learning that deals with how an agent can learn to make sequential decisions in an environment to maximize its cumulative reward. It is inspired by the way humans and animals learn through trial and error. In reinforcement learning, an agent interacts with an environment by taking actions and receiving feedback in the form of rewards or penalties. The agent's objective is to learn a policy, which is a mapping from states to actions, that maximizes the long-term expected reward. Reinforcement learning has been successfully applied to various domains, including robotics, game playing, autonomous systems, recommendation systems, and finance, among others. It allows agents to learn complex behaviors and make decisions in dynamic and uncertain environments without explicit supervision.

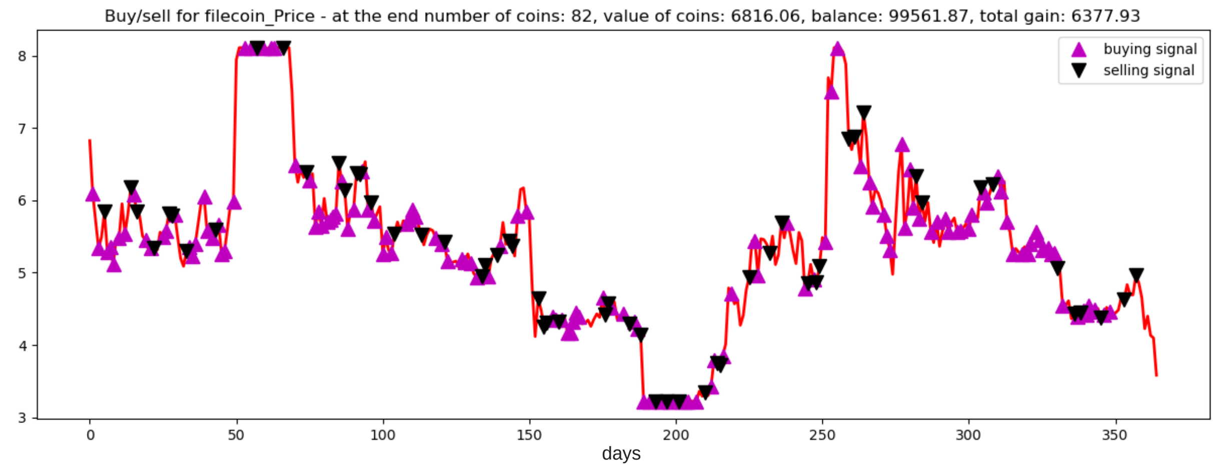
I n our case, we will use Q-learning model-free reinforcement learning, which is a type of reinforcement learning algorithm that learns an optimal policy directly from interactions with an environment without explicitly modeling the dynamics of the environment. Q-learning is a value-based algorithm that aims to learn the optimal action-value function (also known as the Q-function) for an agent in a Markov Decision Process (MDP). The Q-function represents the expected cumulative reward an agent will receive by taking a particular action in a specific state and following a certain policy. The basic idea behind Q-learning is to iteratively update the Q-values based on the observed rewards and the agent's experiences. The Q-values are initially initialized arbitrarily or to some predefined values. As the agent interacts with the environment, it updates the Q-values using the following update rule:


New Q(s,a) = Q(s,a) + α [R(s,a) + γ maxQ'(s',a') − Q(s,a)]
🟥 New Q Value for that state and the action
⬛ Learning Rate
🟫 Reward for taking that action at that state
🟪 Current Q Values
🟩 Maximum expected future reward given the new state (s') and all possible actions at that new state.
🟧 Discount Rate

Q-learning continues to update the Q-values until it converges to the optimal Q-function. Once the optimal Q-function is learned, the agent can determine the best action (Buy, Sell or Hold) to take in any state by selecting the action with the highest Q-value. Q-learning is a model-free algorithm because it does not require prior knowledge or explicit modeling of the environment's dynamics. It learns directly from the observed rewards and the transitions between states. Q-learning can be combined with function approximation techniques, such as deep neural networks, to handle large and continuous state spaces, leading to the development of deep Q-networks (DQN) and deep reinforcement learning algorithms.

The code utilizes TensorFlow to define and train a neural network that approximates the Q-function. The network consists of two fully connected layers with ReLU activation. The mean squared error is used as the cost function, and gradient descent is used to optimize the network parameters. The agent uses an epsilon-greedy exploration strategy and applies a discount factor for future rewards (gamma). The epsilon value decays over time to encourage exploitation. The main goal of this code seems to be to train the agent to make buy and sell decisions in a financial trading scenario based on selected cryptocurrency data.

Litecoin, vechain and filecoin were selected to optimize its process of buying and selling. The best gain using trained Q-learning model was achieved by filecoin:


Buy/sell for filecoin_Price - at the end number of coins: 82, value of coins: 6816.06, balance: 99561.87, total gain: 6377.93
buying signal
selling signal
days