Optimization with Python: How to make the most amount of money with the least amount of risk?

We show how to apply a Nobel-prize winning economic theory to the stock market and solve the resulting optimization problem using simple Python programming.

Introduction

One of the major goals of the modern enterprise of data science and analytics is to solve complex optimization problems for business and technology companies to maximize their profit.

In my article “Linear Programming and Discrete Optimization with Python”, we touched on basic discrete optimization concepts and introduced a Python library PuLP for solving such problems.

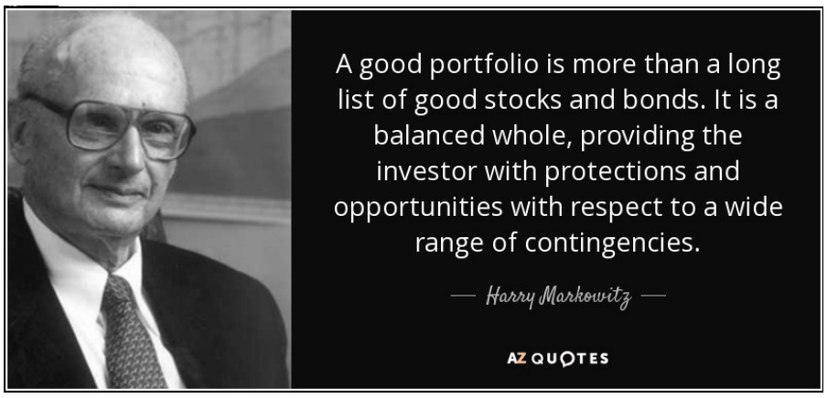
Although a linear programming (LP) problem is defined only by linear objective function and constraints, it can be applied to a surprisingly wide variety of problems in diverse domains ranging from healthcare to economics, business to military.

In this article, we show one such amazing application of LP using Python programming in the area of economic planning — ***maximizing the expected profit from a stock market investment portfolio while minimizing the risk associated with it***.

Sounds interesting? Please read on.

How to maximize profit and minimize risk in the stock market?

The 1990 Nobel prize in Economics went to Harry Markowitz, acknowledged for his famous Modern Portfolio Theory (MPT), as it is known in the parlance of financial markets. The original paper was published long back in 1952.



Source: AZ Quotes

The key word here is **Balanced**.

A good, balanced portfolio must offer both **protections** (minimizing the risk) and **opportunities** (maximizing profit).

And, when concepts such as minimization and maximization are involved, it is natural to cast the problem in terms of **mathematical optimization theory**.

The fundamental idea is rather simple and is rooted in the innate human nature of risk aversion.

In general, stock market statistics show that higher risk is associated with a greater probability of higher return and lower risk with a greater probability of smaller return.

MPT assumes that investors are risk-averse, **meaning that given two portfolios that offer the same expected return, investors will prefer the less risky one**. Think about it. You will collect high-risk stocks only if they carry a high probability of large return percentage.

But how to quantify the risk? It is a murky concept for sure and can mean different things to different people. However, in the generally accepted economic theory, the variability (volatility) of a stock price (defined over a fixed time horizon) is equated with risk.

Therefore, the central optimization problem is to minimize the risk while ensuring a certain amount of return in profits. Or, maximizing the profit while keeping the risk below a certain threshold.

An example problem

In this article, we will show a very simplified version of the portfolio optimization problem, which can be cast into an LP framework and solved efficiently using simple Python scripting.

The goal is to illustrate the power and possibility of such optimization solvers for tackling complex real-life problems.

We work with 24 months stock price (monthly average) for three stocks — Microsoft, Visa, Walmart. These are older data but they demonstrate the process flawlessly.

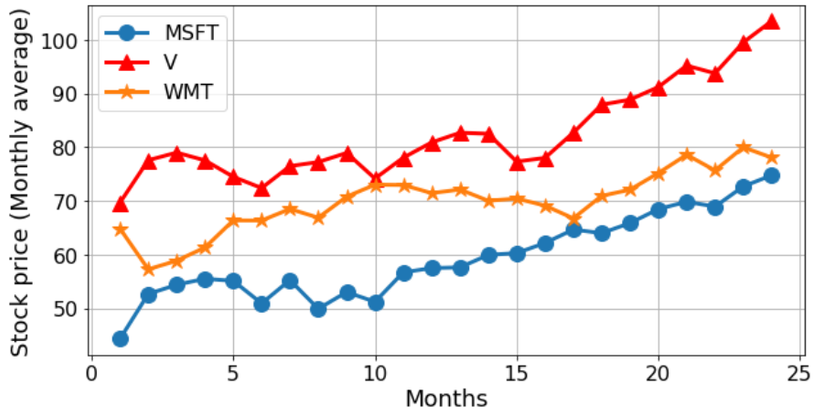


Fig: Monthly stock price of three companies over a certain 24-month period.

How to define the return? We can simply compute a rolling monthly return by subtracting the previous month's average stock price from the current month and dividing by the previous month's price.

Monthly return:

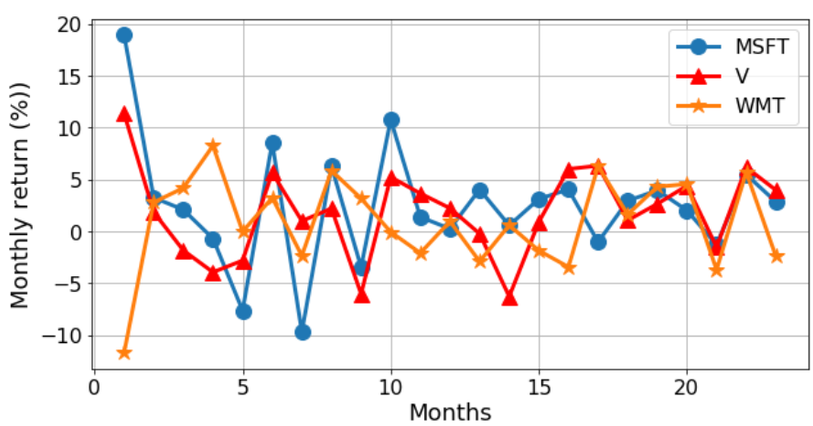
$$r_{jt} = \frac{p_{jt} - p_{jt-1}}{p_{jt-1}}$$

where

r_{jt} = return of stock j in month t

p_{jt} = price of stock j in month t

The return is shown in the following figure,



The optimization model

The return on a stock is an uncertain quantity. We can model it as a **random vector**.

$$\tilde{\mathbf{r}} = \begin{bmatrix} \tilde{r}_1 \\ \tilde{r}_2 \\ \tilde{r}_3 \end{bmatrix} \text{ where } \tilde{r}_j \text{ is the random return of stock } j$$

$$\text{Here } j = 1 : \text{MSFT}, j = 2\text{: V, and } j = 3 : \text{WMT}$$

The portfolio can also be modeled as a vector.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ where } x_j \text{ is the \$ invested in stock } j$$

Therefore, the return on a certain portfolio is given by an inner product of these vectors and it is a random variable. The million-dollar question is:

How can we compare random variables (corresponding to different portfolios) to select a "best" portfolio?

Following the Markowitz model, we can formulate our problem as,

Given a fixed quantity of money (say $1000), how much should we invest in each of the three stocks so as to (a) have a one month expected return of at least a given threshold, and (b) minimize the risk (variance) of the portfolio return.

We cannot invest a negative quantity. This is the **non-negativity constraint**,

$$x_i \geq 0 \; \forall \; i = 1, 2, 3 \quad \text{or} \quad \mathbf{x} \geq \mathbf{0}$$

Assuming no transaction cost, the total investment is restricted by the fund at hand,

$$\sum_{i=1}^{3} x_i \leq 1000$$

Return on the investment,

$$\sum_{i=1}^{3} \tilde{r}_i x_i \quad \text{or} \quad \tilde{\mathbf{r}}^\top \mathbf{x}$$

But this is a **random variable**. So, we have to work with the **expected quantities**,

$$\mathbb{E}\left[\sum_{i=1}^{3} \tilde{r}_i x_i\right] = \sum_{i=1}^{3} \mathbb{E}[\tilde{r}_i] x_i = \sum_{i=1}^{3} \bar{r}_i x_i = \bar{\mathbf{r}}^\top \mathbf{x}$$
$$\text{where } \bar{r}_i \text{ is expected return of stock } i$$

Supposed we want a **minimum expected return**. Therefore,

$$\sum_{i=1}^{3} \bar{r}_i x_i \geq r_{\min} \quad \text{or} \quad \bar{\mathbf{r}}^\top \mathbf{x} \geq r_{\min}$$

Now, to model the risk we have to compute the **variance**,

$$\begin{aligned}
\text{Var}[\sum_{i=1}^{3} \tilde{r}_i x_i] &= \mathbb{E}\left[\left(\sum_{i=1}^{3} \tilde{r}_i x_i - \sum_{i=1}^{3} \bar{r}_i x_i\right)^2\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{3} (\tilde{r}_i - \bar{r}_i) x_i\right)\left(\sum_{j=1}^{3} (\tilde{r}_j - \bar{r}_j) x_j\right)\right] \\
&= \sum_{i=1}^{3}\sum_{j=1}^{3} x_i x_j \mathbb{E}[(\tilde{r}_i - \bar{r}_i)(\tilde{r}_j - \bar{r}_j)] = \sum_{i=1}^{3}\sum_{j=1}^{3} x_i x_j \sigma_{ij} \\
&= \mathbf{x}^\top Q \mathbf{x}
\end{aligned}$$

Putting together, the final optimization model is,

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{3}\sum_{j=1}^{3} x_i x_j \sigma_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^{3} x_i \leq 1000.00, \\
& \sum_{i=1}^{3} \bar{r}_i x_i \geq r_{\min}, \\
& x_i \geq 0 \quad i = 1, 2, 3.
\end{aligned}
$$

Next, we show how easy it is to formulate and solve this problem using a popular Python library.

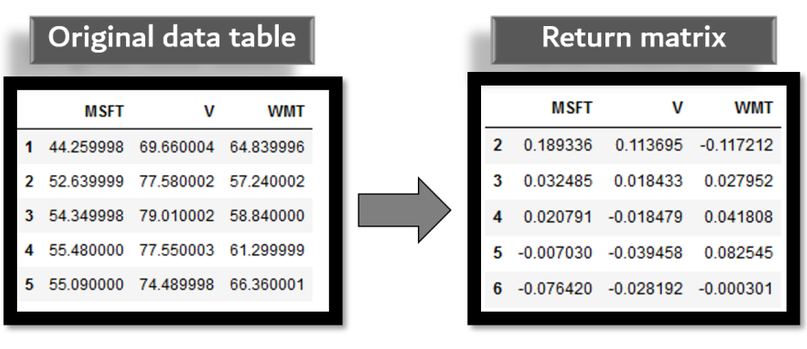
Using Python to solve the optimization: CVXPY

The library we are going to use for this problem is called **CVXPY**. It is a Python-embedded modeling language for convex optimization problems. It allows you to **express your problem in a natural way** that follows the mathematical model, rather than in the restrictive standard form required by solvers.

The **entire code is given in this Jupyter notebook**. Here, I just show the core code snippets.

To set up the necessary data, the key is to compute the return matrix from the data-table of the monthly price. The code is given below,

Now, if you view the original data table and the return table side by side, it looks like following,



	MSFT	V	WMT
1	44.259998	69.660004	64.839996
2	52.639999	77.580002	57.240002
3	54.349998	79.010002	58.840000
4	55.480000	77.550003	61.299999
5	55.090000	74.489998	66.360001

	MSFT	V	WMT
2	0.189336	0.113695	-0.117212
3	0.032485	0.018433	0.027952
4	0.020791	-0.018479	0.041808
5	-0.007030	-0.039458	0.082545
6	-0.076420	-0.028192	-0.000301

Next, we simply compute the mean (expected) return and the covariance matrix from this return matrix,

After that, CVXPY allows setting up the problem simply following the mathematical model we constructed above,

Note the use of extremely useful classes like **quad_form()** and **Problem()** from the CVXPY framework.

Voila!

We can write a simple code to solve the **Problem** and show the optimal investment quantities which ensure a minimum return of 2% while also keeping the risk at a minimum.

The final result is given by,

```
Optimal portfolio
---------------------
 Investment in MSFT : 58.28% of the portfolio
 Investment in V : 20.43% of the portfolio
 Investment in WMT : 21.29% of the portfolio
---------------------
Exp ret = 2.0%
Expected risk    = 3.83%
```

Extending the problem

Needless to say that the setup and simplifying assumptions of our model can make this problem sound simpler than what it is. But once you understand the basic logic and the mechanics of solving such an optimization problem, you can extend it to multiple scenarios,

- Hundreds of stocks, longer time horizon data
- Multiple risk/return ratio and threshold
- Minimize risk or maximize return (or both)
- Investing in a group of companies together
- Either/or scenario — invest either in Cococola or in Pepsi but not in both

You have to construct more complicated matrices and a longer list of constraints, use indicator variables to turn this into a **mixed-integer problem** - but all of these are inherently supported by packages like CVXPY.

Look at the examples page of the CVXPY package to know about the breadth of optimization problems that can be solved using the framework.

Summary

In this article, we discussed how the key concepts from a seminal economic theory can be used to formulate a simple optimization problem for stock market investment.

For illustration, we took a sample dataset of three companies' average monthly stock price and showed how a linear programming model can be set up in no time using basic Python data science libraries such as NumPy, Pandas, and an optimization framework called CVXPY.

Having a working knowledge of such flexible and powerful packages adds immense value to the skillset of upcoming data scientists because the need for solving optimization problems arise in all facets of science, technology, and business problems.

Readers are encouraged to try more complex versions of this investment problem for fun and learning.

#datascience, #programming, #statistics