FROM SCRATCH

Automate your financial and investment decisions

Build a Robo Advisor with Python

Rob Reider
Alex Michalka

MEAP

MANNING

Automate your financial and investment decisions

FROM SCRATCH

Build a Robo Advisor with Python

Rob Reider
Alex Michalka

MEAP

MANNING

Build a Robo Advisor with Python (From Scratch) MEAP V01

1. Copyright_2023_Manning_Publications
2. welcome
3. 1_The_Rise_of_Robo-Advisors
4. 2_An_Introduction_to_Portfolio_Construction

MEAP Edition

Manning Early Access Program

Build a Robo Advisor with Python (From Scratch)

Automate your financial and investment decisions

Version 1

Copyright 2023 Manning Publications

proofreaders.

https://livebook.manning.com/#!/book/build-a-robo-advisor-with-python-from-scratch/discussion

For more information on this and other Manning titles go to

manning.com

welcome

Thank you for purchasing the MEAP version of our book, *Build a Robo Advisor with Python (From Scratch)*. We've enjoyed writing it, and we hope you'll enjoy - and benefit from - reading it. We look forward to hearing your feedback and suggestions for making it even better.

We hope that readers will learn about both finance and Python by reading the book. It *isn't* intended to teach either of those topics from the ground up - we expect that readers will have a basic understanding of both financial concepts and Python programming - but accessibility is important to us. If there are places where you feel we are assuming too much knowledge of either, please let us know. We also hope you'll let us know if you feel like any chapters are too basic. We recognize that not all readers will benefit from every chapter, but we hope that every chapter is useful to *someone*.

The balance of theory and implementation varies by chapter. Some chapters are very financially-focused, and the Python content is limited to showing how a few functions from existing libraries can be used to perform certain calculations or accomplish desired tasks. In other chapters, there aren't any existing Python libraries that we can use. These chapters are much more code-heavy, and essentially build new Python libraries implementing the concepts they cover. We know reading code isn't always easy, so we try to show example usages of new code whenever possible to aid understanding.

Because your feedback is essential to creating the best book possible, we invite you to leave comments in the LiveBook Discussion forum. We appreciate your interest and produce the best book possible for you.

Thank you again for your interest and feedback.

—Rob and Alex

In this book

Copyright 2023 Manning Publications welcome brief contents 1 The Rise of Robo-Advisors 2 An Introduction to Portfolio Construction

1 The Rise of Robo-Advisors

This chapter covers

- The increasing popularity of robo-advisors
- Key features and a comparison of popular robo-advisors
- Python in the context of robo-advising

1.1 What are Robo-Advisors?

Robo-Advisors have become a popular alternative to human financial advisors. Historically, financial advisors would meet with clients, discuss their goals, create a financial plan, and then manage their clients' money over time. In exchange for this personal attention, they would charge clients fees, often in excess of 1% per year of their assets under management. Numerous companies have been trying to disrupt this business through online platforms that provide automated, algorithmic investment services similar to those of a financial advisor. Some of these automated systems "advise" clients through algorithmic implementations of modern portfolio theory, based on the Nobel Prize winning work of Harry Markowitz in the 1950's, while others use machine learning and optimization techniques borrowed from other disciplines. These companies have collectively become known as "robo-advisors".

In this book, we show how anyone with a basic understanding of Python can build their own robo-advisor. We hope this will be useful for anyone who wants to work in this area or wants to apply these algorithms for their own portfolio or advising others.

1.1.1 Key Features of Robo-Advisors

The most basic feature provided by robo-advisors is a personalized asset allocation - a portfolio of investments designed to match the level of riskiness suitable for the client. The core asset allocations offered by different robo-

advisors vary in their choice of asset classes, but usually contain a diversified mix of stocks and bonds, but may vary in the precise choice of asset classes. Some advisors may choose to include more non-US assets in their allocations, or an allocation that's more heavily weighted to growth or value stocks, but in essentially all cases, the instruments that robo-advisors invest in are liquid and traded on exchanges, and not things like venture capital, private equity, real estate.

Aside from the core asset allocation, there are a handful of features that robo-advisors provide. We will cover most of these in more detail later in the book; here we only provide a high-level description of each one.

- **Rebalancing** A client's portfolio will start with weights close to the target allocation at the time of the initial investment, differences in returns will cause the portfolio to draft away from the target over time. Rebalancing may be *drift-based*, meaning the portfolio is rebalanced whenever it deviates from the targets by a pre-specified amount, or *time-based*, meaning that rebalancing occurs on a fixed schedule. Additionally, rebalancing may or may not be *tax-aware*. In tax-aware rebalancing, some appreciated positions may not be sold if doing so would involve a high expected tax cost. In some cases, the robo-advisor may be able to keep the portfolio "on track" simply by intelligently directing dividends and deposits towards assets that have drifted below their target weights.
- **Financial Projections** A key question for most individual investors is "When can I retire?". Robo-advisors may offer tools which show projections of the client's net worth through time, based on assumptions for income, spending, inflation, and investment returns. By varying these assumptions, clients can assess the feasibility of retirement at different ages.
- **Tax-Loss Harvesting** The basic idea of *tax-loss harvesting* is to reduce the investor's current tax burden by opportunistically realizing losses in assets that have declined in value. The realized losses can be used to offset realized gains or some income. Tax-loss harvesting is a common feature offered by robo-advisors, but one that is tricky to understand properly. We will discuss the true economic benefit, as well as the implementation of tax-loss harvesting strategies, later.

- **Glide Paths** As clients age and approach retirement, the amount of risk that they should take in their investments decreases. A *glide path* is a series of portfolios or asset allocations that gradually decreases in riskiness through time. By the time a client reaches retirement and gives up their employment income, the glide path will assign a low-risk portfolio. While some robo-advisors use glide paths in their investment process, others let the client control how much risk they are willing to take throughout the lifetime of their investments.

Aside from these important features, robo-advisors vary in two additional dimensions - their management fee, and the minimum account size. While there is some dispersion in these values, both are usually quite low, making most robo-advisors accessible to clients in the earliest stages of their careers.

1.1.2 Comparison of Robo-Advisors

As of 2020, there were over 200 robo-advisors based in the US alone. The table below compares the feature sets of the some largest, as reported by Investopedia in March of 2021.

Name	Advisory Fee	Minimum Account Size	Tax-Aware Rebalancing	Tax-Loss Harvesting
Betterment	0.25%-0.40%	$0	Yes	Yes
Wealthfront	0.25%	$500	Yes	Yes
Personal Capital	0.49%-0.89%	$100,000	Yes	Yes
Blooom	$45-$250 Annually	$0	No	No

Acorns	$1-5 monthly	$0	No	No
SigFig	0.25%	$2,000	No	Yes
Axos Invest	0.24%	$0	Yes	Yes
Ally Invest	Free	$100	No	No
Vanguard	~0.15%	$50,000	No	No
Charles Schwab	Free	$5,000	No	Available for accounts above $50,000
Fidelity	0.35%	$5,000	No	No
E*Trade	0.30%	$5,000	No	No

1.1.3 Things Robo-Advisors Don’t Do

Software is useful for performing simple, repeatable tasks very quickly. Even the most complex-looking algorithms are just sequences of simple conditions and steps. The examples given above are ones which are naturally suited to be accomplished using software. However, there are some services performed by traditional advisors that software can’t replicate. These are generally infrequent or one-time events that may require detailed personal information. Examples of services which are (for now) best accomplished by human

advisors include estate planning, management of non-traditional assets like art or real estate, and specialized tax advice for things like stock options.

This is not to say that robo-advisors can't expand into areas that have traditionally been the domain of human advisors. For example, financial advisors are often called upon to help retirees with defined-benefit pension plans on whether to take a lump sum or monthly payments for life. The same Python programs used to analyze Social Security can be adapted to analyze pensions. Of course, there are some things that robo-advisors will never be able to do - software will never get you basketball tickets (no matter how large your account gets), or treat you to dinner.

1.2 Advantages of Using a Robo-Advisor

There are several advantages of using a robo-advisor over either using a human advisor or do-it-yourself investing. We highlight three of those advantages here.

1.2.1 Low Fees

Surveys show that about 30% of Americans use a financial advisor of some kind. Fee-based financial advisors charge fees based on a percentage of assets under management (AUM), an annual retainer, or an hourly charge, and sometimes a combination of these. In addition to fee-based financial advisors, some advisors instead follow a commission-based model, where they are compensated by charging commissions on financial transactions and products like life insurance or annuities. For those that charge an AUM fee, which is the largest category, the average fee is about 1% per year. Robo-advisor fees are a fraction of that (see table above). In addition, robo-advisors usually have much lower minimum account size than financial advisors.

Even small savings, when accumulated over decades, can make a big difference. A 1% annual fee charged by a traditional human advisor may not seem like much, but the cost compounds over time. Imagine starting out with a $100 investment which earns a 7% return each year, before fees. The 1% fee charged by a traditional advisor reduces the return to 6% annually. This means that after 30 years, the $100 investment would grow to about $574.

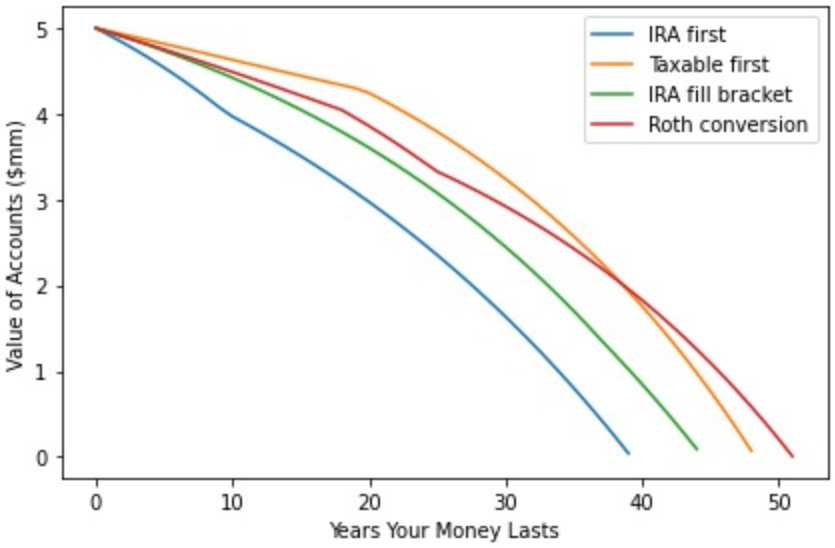
Not bad, but let's compare this to a typical robo-advisor charging 0.25% per year. With the robo-advisor, the investment would grow to about $710 - almost 25% more!

1.2.2 Tax Savings

In the last section of this book, we describe several tax-saving strategies that could be automated by a robo-advisor. How much money can people save through tax strategies with robo-advisors? It's very difficult to give an exact number, because it's different for each individual depending on their circumstances. Someone that has only taxable accounts and no retirement accounts like an IRA or 401(k) would not benefit from several of the automated robo-advisor functions that we talk about in later chapters. Indeed, with the tools covered in this book, you will be able to estimate, through Monte Carlo simulations, the amount of savings for a specific set of circumstances. But the savings can be significant.

To give one example, consider a topic we will cover in detail in the last chapter: optimal sequencing of withdrawals. As just a brief introduction, and we will cover this in excruciating detail later in the book, during the retirement stage of life when people are "decumulating" assets as opposed to accumulating, there are numerous options available for withdrawing assets. For example, you can take money out of your taxable accounts first, and when that is depleted, start taking money out of retirement assets ("Taxable first"). You can also switch the order and take money out of retirement accounts first ("IRA first"). A third strategy is to take money out of retirement accounts each year up to the top of the nearest tax bracket, so that future IRA withdrawals don't push you into a higher tax bracket ("IRA fill bracket"). Finally, the fourth strategy is similar to the third but you convert your IRA distribution into a Roth IRA ("Roth conversion").

To illustrate how consequential those decisions can be, and how much you can save by employing the best strategy for a given set of circumstances, the figure below shows how long your money will last for those four strategies. The specific set of assumptions and the details of the strategy will be covered later, but the point is that it makes a big difference. The optimal strategy can extend your assets by many *years*.



1.2.3 Avoid Behavioral Biases

It is well documented that investors are subject to numerous behavioral biases, many of which can be avoided by algorithm-based automated trading.

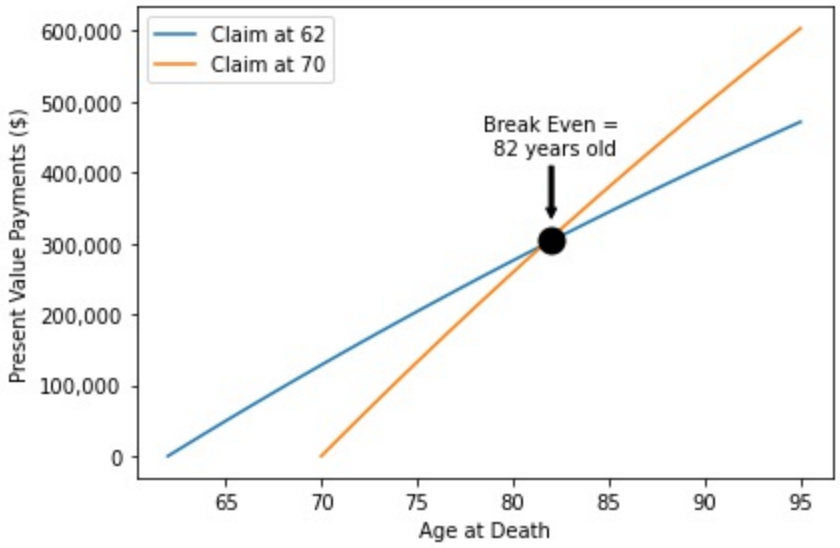
- **Disposition effect** Studies have shown that investors tend to hold onto losing stocks hoping to get back to break-even. However, robo-advisors realize it's often useful to sell losing stocks to harvest tax losses.
- **Herd behavior** Investors tend to be influenced by others, which explains why they dive into stocks during bubbles and panic during crashes. Herding might have conferred benefits when fleeing predators in prehistoric times, but it is not a great investment strategy. Robo-advisors, on the other hand, unemotionally rebalance gradually toward stocks during crashes and away from stocks during bubbles.
- **Overtrading** Several studies on overtrading have all reached the same conclusion: the more active a retail investor tends to be, the less money they make. Individual investors may incorrectly assume that if they are not paying brokerage commissions, there is no cost to frequent trading. However, commissions are only one cost. The bid/ask spread, the difference between the price you can buy a stock and sell a stock on an exchange, is a significant component of trading costs. And frequent trading leads to short term capital gains, which is not tax efficient. Robo-advisors methodically factor these costs into account when they make trades.

1.2.4 Saving Time

By automating simple tasks associated with investing, a robo-advisor can save investors huge amounts of time compared to a “do it yourself” approach. Monitoring the portfolio for drift away from its targets or for tax-loss harvesting opportunities, and placing trades to rebalance, harvest losses, or invest deposits aren’t especially difficult, but aren’t especially fun either. Robo-advisors automate these tasks, leaving investors more time for more enjoyable pursuits.

1.3 Example Application: Social Security Benefits

One thing that gets us excited about this topic is that robo-advising is still in its infancy. Consider one example: the important decision on when to claim Social Security benefits. For many Americans, their Social Security check can be considered one of their most valuable assets, in many cases worth more than a half a million dollars on a present value basis. People must elect when they want to start receiving, or claiming, their Social Security payments, which is anytime between 62 and 70, and the longer you wait, the larger your payment will be. It is one of many important retirement decisions that are consequential and mostly irreversible. So what kind advice is out there on when to claim Social Security? If you do a google search on when to claim Social Security, you’ll find numerous calculators that mostly all do the same thing. They usually have a simple break-even analysis like the one we created below. To see a similar one, see for example the one offered by Charles Schwab (schwab.com/resource-center/insights/content/7-questions-about-social-security).



In this break-even analysis, the x-axis represents the age you expect to die, and the y-axis represents the present value of your Social Security payments up until the time of the death. As you can see from the graph, in this simple analysis that compares claiming at 66 vs. 70, if you expect to live past 82, you are better off waiting until 70, and if you expect to die before 82, you are better off claiming at 66. As the chart shows, if you live to 90 and you claim at 70, the present value of all payments is about $500k.

What is this analysis missing, and how can Python be used to improve the analysis? First of all, the analysis gets exponentially more complicated when spouses are considered, and the Social Security rules can get very complicated. Although the break-even analysis above can be done in a spreadsheet, taking into account couples and all the Social Security rules would overwhelm a spreadsheet and must be coded in a computer language like Python.

In order to accurately predict Social Security benefits, an estimate must be made for the trajectory of future income up to retirement. Python has several libraries for forecasting a time-series like income. In this book, we will use a library called Prophet, which was developed by Facebook and is particularly good at modeling non-linear trends that you would find in a person's lifetime income. Prophet requires income data for the investor, which can be downloaded from Social Securities website, and Python can be used for scraping the file.

The break-even analysis above is not customized in any way, and Python can help here too. Since Social Security benefits are similar to an asset like a bond, it should be incorporated into an investor's asset allocation - larger Social Security payments would mean that even a conservator investor could hold fewer bonds. The break-even analysis also ignores taxes on Social Security income. And it ignores the Social Security benefits that may be withheld when someone's income exceeds a threshold while they are collecting benefits. The Social Security analysis claiming analysis can therefore be extended to include an investor's particular tax situation.

Perhaps one of the most important and overlooked shortcomings of the break-even approach is that it doesn't take risk into account. In this case, the risk is longevity risk - the chance that you survive beyond your life expectancy and outlive your money. Social Security benefits, as well as defined-benefit pensions and annuities, reduce that risk by guaranteeing lifetime benefits. In later chapters, we show how to take risk into account with examples in Python.

1.4 Python and Robo-Advising

The designers of Python programming intended the language to be fast, powerful, and fun. We think that they have succeeded. Python is easy to get started with - the classic "Hello World" program is not much harder than just typing the words "Hello World" - and easy to learn. Python's documentation is extensive, and its wide adoption means it's usually quite easy to find solutions to tricky problems - chances are, someone has run into them already. Finally, Python is flexible - while it *supports* some more sophisticated features of lower-level languages like Java, Python doesn't *require* them. If you want to use Python in an object-oriented way, you can - but you don't have to. Likewise, you can add "hints" for data types in Python functions, but they aren't required. Python lets you choose.

All of these qualities make Python easy to learn and work with for any programmer. But what makes Python a good language for robo-advising? As Python's popularity has grown, the number of mathematical and statistical packages has grown as well. Applications in this book will lean heavily on packages including:

- numpy: A general-purpose package for numerical computing, numpy provides tools ranging from the basics (basic vector and matrix operations or least-squares solutions to linear systems) to the complex (such as random number generation or matrix factorizations). The numpy package achieves high speed by implementing many numerical subroutines in C.
- pandas: pandas was developed at AQR Capital Management starting in 2008, and was made open-source in 2009. It adds a level of abstraction on top of arrays and matrices to make data manipulation effortless.
- cvxpy: cvxpy is a mathematical modeling language allowing users to formulate and solve convex optimization problems using a natural and easy-to-read syntax.

We should also talk about what Python *isn't.*Python's interpreted nature means it will never be as fast as low-level compiled languages like C or Fortran. If your application requires top speed, a compiled language may be a better choice. This also means that bugs only show up when code is run, and won't be found during compilation. Overall, we still think that despite these limitations, Python is a great choice for this book. Robo-advising doesn't require lightning speed, we think that the ease of use and extensive libraries and documentation outweigh any disadvantages in development speed.

This book won't assume that readers are Python experts, but will assume a basic familiarity. For an introduction to programming in Python, we recommend "The Quick Python Book" by Naomi Ceder as a great place to start.

1.5 Who Might be Interested in Learning About Robo-Advising?

There are several groups of people who might be interested in the topics covered in this book.

- *You want to better understand personal finance to help you with your own finances.* There is no shortage of books on personal finance for do-it-yourself investors, but this book focuses on topics that can clearly save you money and goes into them in depth. You won't see chapters

found in other personal finance books like “Live within your means” or “Don’t buy complex financial products”. Even if you have no interest in applying these techniques in Python, the book is written so that you can skip the Python examples and still understand the principles behind what the algorithms do.

- *You are interested in working for a financial advisor or wealth manager.* As we mentioned, the number of robo-advisors is growing, and the incumbents are also getting into robo-advising. And traditional wealth managers are using the same techniques for their clients. A quick search on indeed.com for jobs as a “financial advisor” currently lists over 20,000 jobs. This book provides relevant skills that are used in the industry.
- *You are a Financial Advisor and would like to provide your clients with a larger set of tools.* According to the Bureau of Labor Statistics, as of 2020 there were 218,000 financial advisors in the United States. The financial advisory business is obviously very competitive, and providing sophisticated services gives a firm a competitive advantage. An advisor can differentiate themselves in this crowded field and create a competitive advantage by offering more sophisticated services, like the ones described in this book. Also, a financial advisor that can automate advice can service many more clients, while still providing customized advice.
- *You are interested in useful, practical applications of Python.* There is no better way to learn Python than applying it to interesting, practical problems and observing intuitive results. In this book, we will use numerous Python libraries to solve wealth management problems. We will use a convex optimization library and a hierarchical tree clustering library to perform asset allocation, a statistical and random number library for Monte Carlo simulations, a root finding library for measuring portfolio performance, and a reinforcement learning library to estimate a glidepath.

1.6 Summary

- Robo-advisors use algorithms to automate some of the functions of human financial advisors.
- Robo-advisors have several advantages over human advisors: they have

lower fees, they can save a considerable amount of money using tax strategies, and they can help investors avoid some well-documented behavioral biases that detract from their performance.

- Python, with its extensive libraries, can be used to implement many of the functions of a robo-advisor, from asset allocation to tax loss optimization to Monte Carlo simulations for financial planning.

2 An Introduction to Portfolio Construction

This chapter covers

- Creating risk-reward plots
- Using matrix operations to compute portfolio returns and volatilities
- Calculating, plotting, and deriving the math behind the efficient frontier
- Introducing a risk-free asset and the idea of the Capital Allocation Line
- Gauging an investor’s risk tolerance through questionnaires
- Using slider widgets to help an investor visualize the risk-reward tradeoff

One of the primary functions of a robo-advisor is to construct a well-diversified portfolio of assets. In this chapter and the two chapters that follow, we will provide the theoretical foundation for portfolio construction and all the building blocks that will be needed. We will also start using Python in our examples. Later in the book, we will delve deeper into the topic of portfolio construction including some problems with the traditional methods that are covered in this chapter.

Let's start off with a simple example with only three assets, First Energy (FE), Walmart (WMT), and Apple (AAPL). The analysis that follows can always be generalized to include as many assets as you would like. And in this example, the assets are individual stocks, but they could easily be bonds, ETFs, commodities, cryptocurrencies, or even hedge funds. The problem we want to address is what are the optimal weights to assign to these three assets. We will start off by taking the approach of the Nobel Prize-winning work of Harry Markowitz.

Suppose for these three assets that we knew their expected returns and variances, and their correlations to each other. Of course, we do not know things like the expected return of a stock, but for now, let's suspend disbelief for the purposes of this discussion. Let's say the annual expected returns for

FE, WMT, and AAPL are 4%, 9%, and 12% respectively. We can represent this as a vector:

$$\mu = \begin{bmatrix} 0.04 \\ 0.09 \\ 0.12 \end{bmatrix}$$

Let's say the annual standard deviation of returns, or volatility, for FE, WMT, and AAPL are 15%, 20%, and 35% respectively. We can represent this as a vector:

$$\sigma = \begin{bmatrix} 0.15 \\ 0.20 \\ 0.35 \end{bmatrix}$$

We can plot these three stocks in mean-standard deviation space, which some call a "risk-reward" plot, using the code in Listing 2.1:

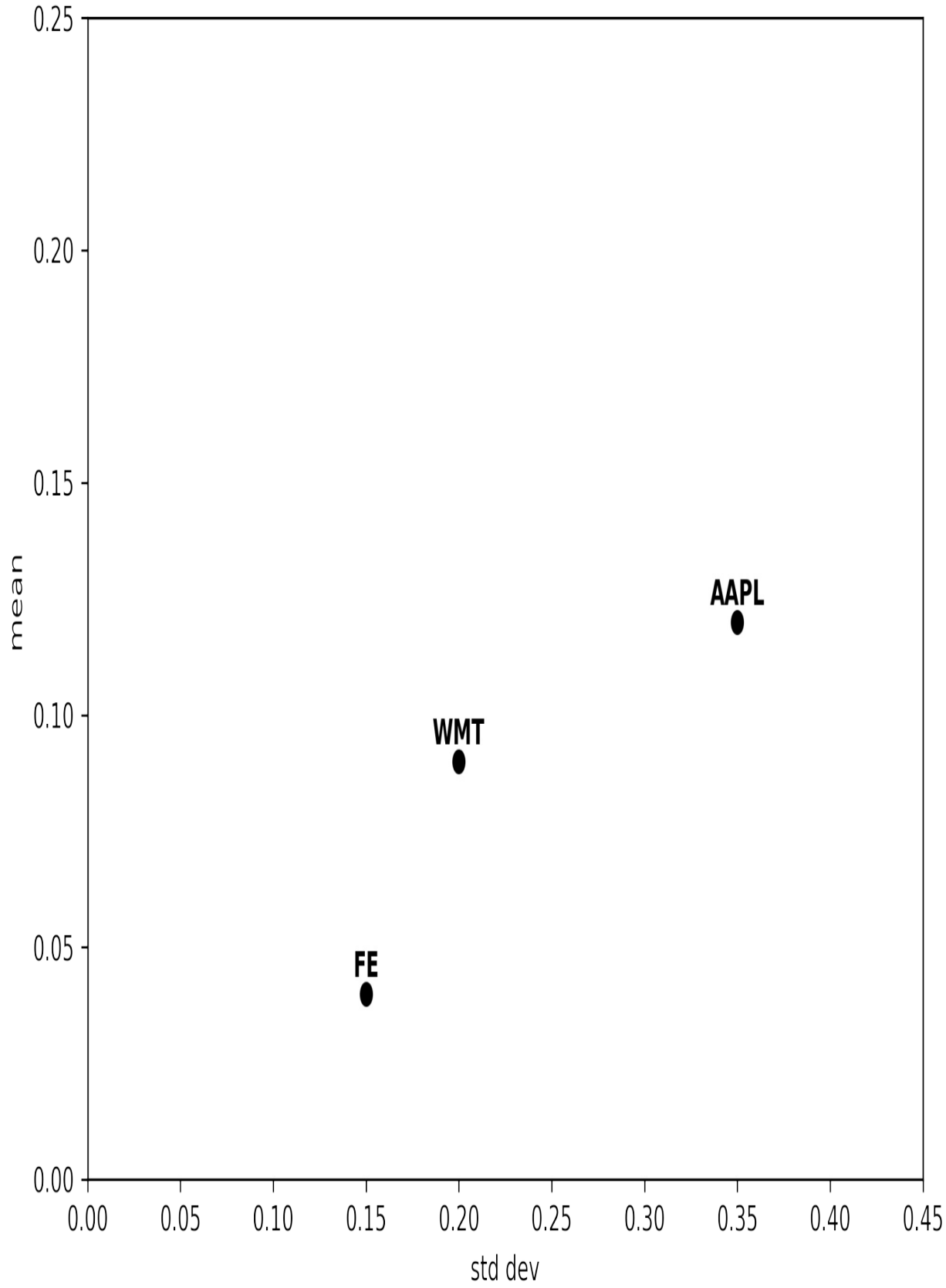
Listing 2.1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

stocks = ['FE', 'WMT', 'AAPL']
mu = [.04, 0.09, .12]
sigma = [.15, .20, .35]

def plot_points(mu, sigma, stocks):
    plt.figure(figsize=(8,6))
    plt.scatter(sigma, mu, c='black') #A
    plt.xlim(0,.45)
    plt.ylim(0,.25)
    plt.ylabel('mean')
    plt.xlabel('std dev')
    for i, stock in enumerate(stocks):
        plt.annotate(stock, (sigma[i], mu[i]), ha='center', va='b

plot_points(mu, sigma, stocks)
```

```
plt.show()
```

Figure 2.1 shows the risk-reward plot when the code in Listing 2.1 is run.

Figure 2.1 Example of risk-reward plot for three stocks


0.25
0.20
0.15
0.10
0.05
0.00
mean
AAPL
WMT
FE
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
std dev

2.1 Computing the Expected Return and Standard Deviation of a Portfolio

Our goal is to find the most efficient way to weight these assets by constructing portfolios that have the highest expected return for a given standard deviation. If we represent the weights for FE, WMT, and AAPL as w_1, w_2, and w_3 respectively, we can again write this in vector notation:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

The return on this portfolio is:

$$r_p = w_1 r_1 + w_2 r_2 + w_3 r_3$$

and the expected return on the portfolio is just the weighted average of the expected returns on the three assets that make up the portfolio:

$$\mu_p = E\left[r_P\right] = w_1 E\left[r_1\right] + w_2 E\left[r_2\right] + w_3 E\left[r_3\right]$$

We can write this in vector notation as the dot product of **w** and **μ**:

$$\mu_p = \mathbf{w}^T \mu$$

Unfortunately, the standard deviation of the portfolio is not simply the weighted average of the individual stock standard deviations. The pairwise correlations between the assets play an important role. To illustrate with just two assets, the standard deviation of the portfolio is

$$\sigma_p = \left(w_1^2\sigma_1^2 + w_2^2\sigma_2^2 + 2w_1w_2\sigma_1\sigma_2\mathrm{Corr}(r_1, r_2)\right)^{1/2}$$

Instead of dealing with the correlations and individual stock volatilities, it is often more convenient to deal with covariances, which combine correlations and volatilities. The covariance between assets *i* and *j* is

$$\mathrm{Cov}(r_i, r_j) = \sigma_i\sigma_j\mathrm{Corr}(r_i, r_j)$$

and if *i* and *j* are the same, it's just the variance of the asset:

$$\mathrm{Cov}(r_i, r_i) = \mathrm{Var}(r_i)$$

$$= \sigma_i\sigma_i\mathrm{Corr}(r_i, r_i)$$

$$= \sigma_i^2$$

For *N* assets, and writing each term in terms of covariances, we can generalize the volatility of a portfolio as a double summation

$$\sigma_p = \left(\sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j \mathrm{Cov}(r_i, r_j) \right)^{1/2}$$

or using matrix notation

$$\sigma_p = \left(\mathbf{w}^T \Sigma \mathbf{w} \right)^{1/2}$$

Where Σ is called the variance-covariance matrix, or covariance matrix for short. The covariance matrix is a square, symmetrical matrix with the variances along the diagonal and covariances on the off-diagonal elements.

Using Python and NumPy, there are several ways to do these calculations, which involve matrix multiplication, but it can be a bit confusing. Some of the confusion arises because the multiplication operator, `*`, when applied to one-dimensional or two-dimensional arrays, does an element-by-element multiplication rather than a matrix multiplication (multiplying an entire row by an entire column). In Python 3.5 and later, the operator `@` performs a true matrix multiplication. And to make things even more confusing, if you convert a NumPy array to a NumPy matrix, which is another data type, then the multiplication operator `*` actually does perform a matrix multiplication.

To compute the expected return of a portfolio, $\mu_p = w^T\mu = \mu^T w$, you can use NumPy's dot product or matrix multiplication. Both of these methods are compared in Listing 2.2 below, and of course, they give the same answer. For the chapters that follow, we'll tend to use the matrix multiplication operator (@), as the syntax is more concise and helps avoid confusion.

Listing 2.2

```
w = np.array([0.2, 0.3, 0.5])

mu_p = np.dot(w, mu) #A
print('Expected portfolio return, Method #1: ', mu_p)

mu_p = mu @ w.T #B
print('Expected portfolio return, Method #2: ', mu_p)
```

Note that in the second method when using matrix multiplication of two vectors, we transpose the second vector of weights, not the first vector of means. Whereas in mathematics we assume vectors are column vectors, in Python when you create a list or a one-dimensional numpy array, it is assumed to be a row vector. Also note that in order to take the transpose of a vector, it has to be a numpy array and not a list, so you would have to create `w` with, for example, `w = nd.array([0.2 0.3, 0.5])` rather than using a list `w = [0.2, 0.3, 0.5]`.

To convert the correlation matrix and a vector of individual volatilities into a covariance matrix using the formula we gave above, $Cov(r_i,r_j) = \sigma_i \sigma_j Corr(r_i,r_j)$, we can actually use the element-by-element matrix multiplication operator `*`. Of course, like everything else in Python, there are multiple ways to do the same thing, and in the second method, we construct a diagonal matrix from the vector of volatilities, **σ**, and use the matrix multiplication operator `@`. These two methods for computing the covariance matrix are shown in Listing 2.3.

Listing 2.3

```
Corr = [[ 1.  ,  0.1 ,  0.17],
        [ 0.1 ,  1.  ,  0.26],
        [ 0.17,  0.26,  1.  ]]

Cov = np.outer(sigma, sigma) * Corr #A
print('Covariance matrix, Method #1: \n', Cov)

Cov = np.diag(sigma) @ Corr @ np.diag(sigma) #B
print('Covariance matrix, Method #2: \n', Cov)
```

To compute the standard deviation of a portfolio,

$$\sigma_p = \left(\mathbf{w}^T \Sigma \mathbf{w}\right)^{1/2}$$

you can use the `@` operator as above or convert arrays or lists into numpy matrices and use the `*` operator, which is shown in Listing 2.4.

Listing 2.4

```
sigma_p = (w @ Cov @ w.T) ** 0.5 #A
print('Portfolio standard deviation, Method #1: ', sigma_p)

w_matrix = np.asmatrix(w)
sigma_p = (w_matrix * Cov * w_matrix.T).item() ** 0.5 #B
print('Portfolio standard deviation, Method #2: ', sigma_p)
```

Note that when σ^2_p is computed with NumPy matrices in the second method, the end result is a 1x1 matrix rather than a scalar, so we need to use the method `.item()` to convert to a scalar.

Now that we have gone through the code for finding the mean and standard deviation of a portfolio, we can use Python to gain some further intuition on the portfolio construction process.

2.2 An Illustration With Random Weights

In later chapters, we will show how to use a Python library to perform numerous optimizations to find the best weights under various assumptions, objectives, and constraints. But just to illustrate how some weights are better than others, we can generate in Python a large number of completely random weights and plot the results. The function in Listing 2.5 below uses NumPy's standardized normal random number generator to generate random weights for each asset. We then normalize the weights by dividing by its sum, which guarantees that the weights sum to one. For now, the random weights can be negative (in practice, a negative position is referred to as a "short"). Later, we will constrain all the weights to be positive.

Listing 2.5

```
def random_weights(n_assets):
    k = np.random.randn(n_assets)
    return k / sum(k)
print(random_weights(3))

>>>  [ 0.45991849 -0.0659656   0.60604711]
```

The next function, in Listing 2.6, takes a vector of weights, as well as the means, and variance-covariance matrix as arguments, and returns the

expected return of the portfolio, μ_p and the standard deviation of the portfolio, σ_p, using the equations above.

Listing 2.6

```
def mu_sigma_portfolio(weights, means, Cov):
    mu_p = np.dot(weights, means)
    sigma_p = (weights @ Cov @ weights.T) ** 0.5
    return mu_p, sigma_p
```

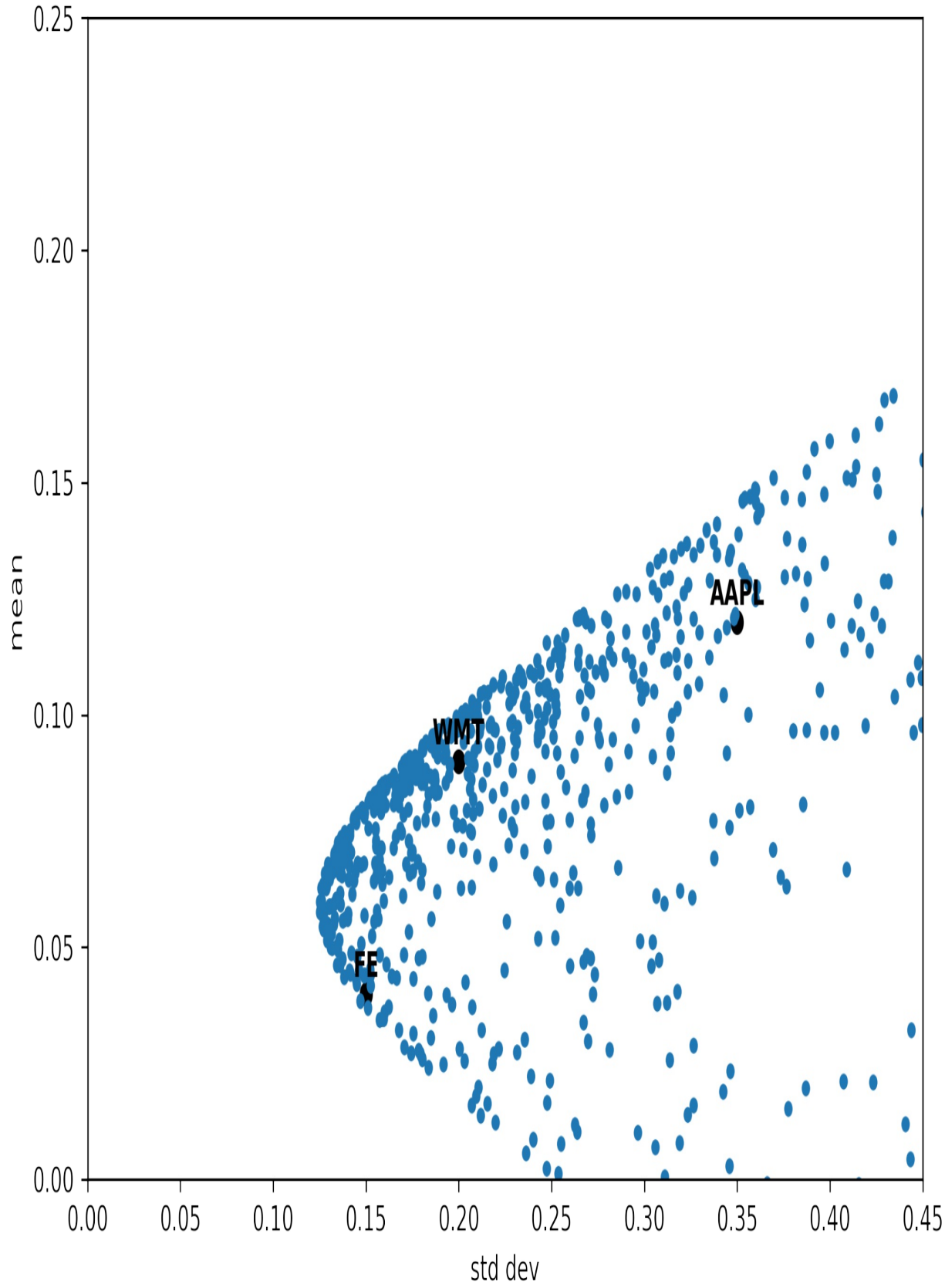
With the functions above to compute random weights and take those random weights and compute the mean and standard deviation of the portfolio associated with the random weights, we are ready to add 1000 random portfolios to the risk-reward plot, which is shown in Listing 2.7.

Listing 2.7

```
def plot_random_portfolios(n_simulations, n_assets):
    mu_p_sims = []
    sigma_p_sims = []
    for i in range(n_simulations):
        w = random_weights(n_assets) #A
        mu_p, sigma_p = mu_sigma_portfolio(w, mu, Cov) #B
        mu_p_sims.append(mu_p)
        sigma_p_sims.append(sigma_p)
    plt.scatter(sigma_p_sims, mu_p_sims, s=12)

plot_points(mu, sigma, stocks)

n_simulations = 1000
n_assets = 3
plot_random_portfolios(n_simulations, n_assets)
plt.show();
```

Figure 2.2 shows the risk-reward plot when the code in Listing 2.7 is run for 1000 randomly-weighted portfolios.

Figure 2.2 Risk-reward plot for 1000 random portfolios of three stocks


0.25
0.20
0.15
0.10
0.05
0.00
mean
AAPL
WMT
FE
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
std dev

From this plot, it is apparent that there is a pattern to the lowest possible portfolio volatility for a given level of portfolio expected return. It turns out that in the simple case we have described so far, there is a mathematical formula for the set of optimal portfolios, called the **minimum-variance frontier**. The top half of the curve is referred to as the **efficient frontier**. Any portfolio on the lower portion of the minimum-variance frontier is inefficient because there is a portfolio with the same volatility but a higher expected return on the upper side of the curve.

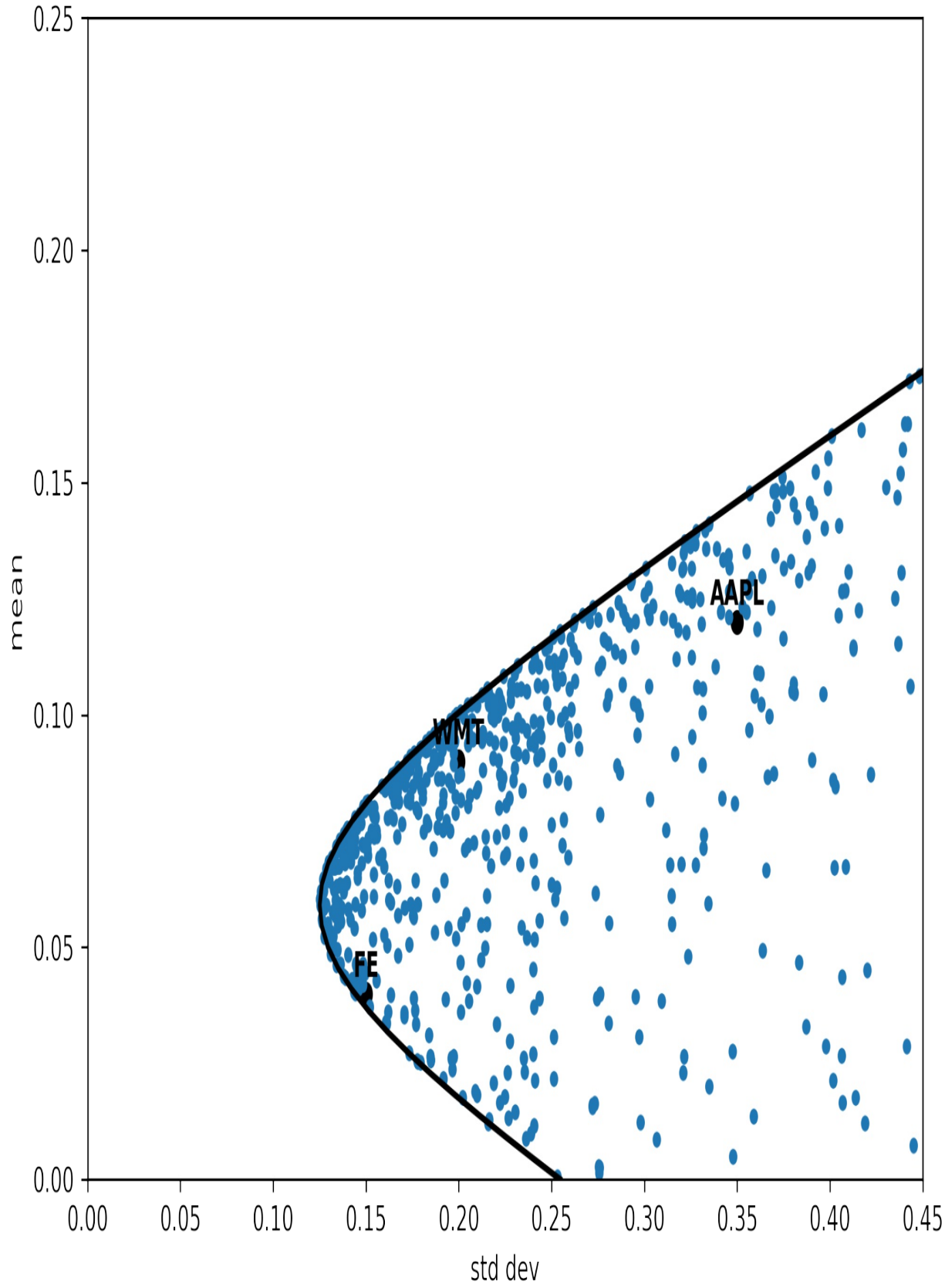
Listing 2.8 below adds the minimum-variance frontier to the risk-reward plot. For those that are interested in the math, the mathematical formula is derived in the Appendix, but it is not necessary to understand the derivation for anything we do later.

Listing 2.8

```
def plot_min_var_frontier(mu, Cov): #A
    A,B,C = compute_ABC(mu, Cov)
    y = np.linspace(0,.45,100)
    x = (A*y*y-2*B*y+C)/(A*C-B*B)
    x = np.sqrt(x)
    plt.plot(x,y, color='black', lw=2.0)

def compute_ABC(mu, Cov):
    Cov_inv = np.linalg.inv(Cov)
    ones = np.ones(n_assets)
    A = ones @ Cov_inv @ ones
    B = ones @ Cov_inv @ mu
    C = mu @ Cov_inv @ mu
    return A,B,C

plot_points(mu, sigma, stocks)
plot_random_portfolios(n_simulations, n_assets)
plot_min_var_frontier(mu, Cov)
plt.show()
```

Figure 2.3 shows the risk-reward plot when the code in Listing 2.8 is run, which now superimposes the minimum-variance frontier.

Figure 2.3 Risk-reward plot with minimum-variance frontier


0.25
0.20
0.15
0.10
0.05
0.00
mean
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
std dev
AAPL
WMT
FE

In the next section, we will see how the portfolio problem changes when we add a risk-free asset.

2.3 Introducing a Risk-Free Asset

Suppose now we introduce one more asset, which is a risk-free asset. And to be precise, a risk-free asset is not simply any government bond that is free of credit risk, but the maturity of the bond exactly matches the investment horizon and there are no intermediate coupons that might introduce some reinvestment risk. For example, if we had a one-month horizon, a one-month government bond (known as a one-month Treasury Bill, or T-Bill for short), would be a risk-free asset. You know the exact amount you will be receiving in a month. Therefore, this risk-free asset has zero volatility and on the mean-standard deviation plot, it would be located on the y-axis with a mean return of the risk-free rate. In contrast, if you bought a 10-year government bond and sold it in a month, there could be capital gains or losses which introduces some volatility.

With a risk-free asset, we can actually do better than the previous efficient frontier. As Figure 2.4 shows, the line that connects the risk-free asset to a "tangent portfolio" represents any feasible portfolio and lies above the efficient frontier. The line that goes from the risk-free rate to the tangent portfolio is sometimes referred to as the **Capital Allocation Line**.

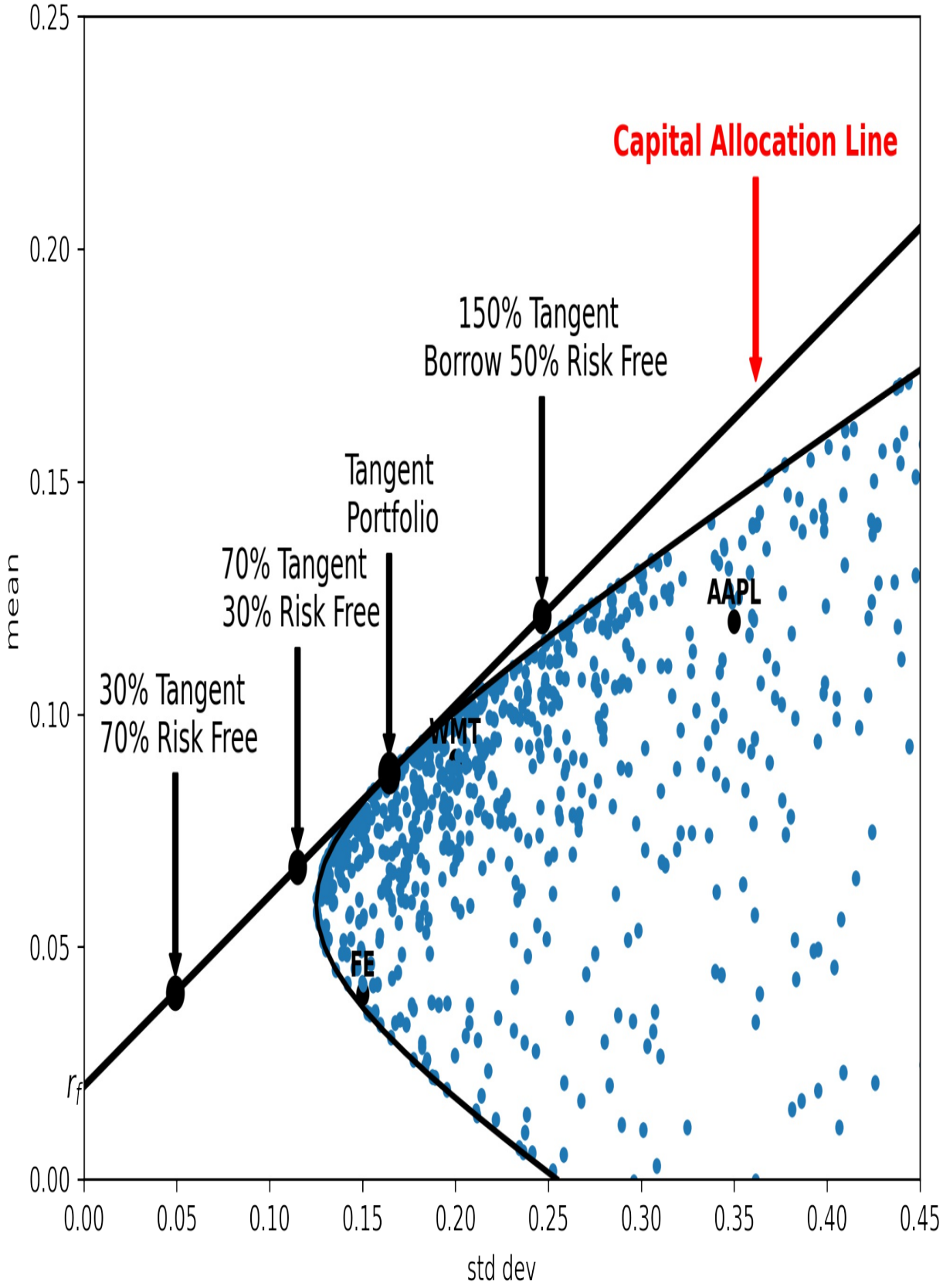
Economists take this a step further and argue that if all investors have the same information and same expectations, they will hold securities in the same weights, and in equilibrium, the tangent portfolio will be the market portfolio. In this case, the line that goes from the risk-free rate to the market portfolio is referred to as the **Capital Market Line** rather than the Capital Allocation Line. These assumptions are rather unrealistic, and for generality, we will continue to use the term Capital Allocation Line and not assume the tangent portfolio is necessarily the market portfolio.

Any point between the risk-free rate asset and the tangent portfolio can be achieved by a combination of the two assets, the tangent portfolio and the risk-free asset. A portfolio that consists of 70% of the tangent portfolio and

30% of the risk-free asset lies 70% towards the tangent portfolio. A more risk-averse investor might want 70% of the risk-free asset and 30% of the tangent portfolio.

An investor can achieve points beyond the tangent portfolio using leverage. For example, an investor with $1mm in assets can borrow $500k and invest $1.5mm in the tangent portfolio. We assume for simplicity an investor can borrow at the risk-free rate, although that assumption can easily be relaxed. If the borrowing cost were higher than the risk-free rate, the Capital Allocation Line would be kinked at the tangent point.

Figure 2.4 Risk-reward plot with Capital Allocation Line


Capital Allocation Line
150% Tangent
Borrow 50% Risk Free
Tangent
Portfolio
70% Tangent
30% Risk Free
30% Tangent
70% Risk Free
AAPL
WMT
FE
r_f
mean
std dev
0.25
0.20
0.15
0.10
0.05
0.00
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45

The code that generated the Capital Allocation Line in Figure 2.4 is shown in Listing 2.9. The mathematical derivation for the Capital Allocation Line can be found in the Appendix for those readers that are interested in the math, but is only included for completeness and is not necessary for understanding the portfolio construction process.

Listing 2.9

```
def plot_Capital_Allocation_Line(rf, mu, Cov): #A
    A,B,C = compute_ABC(mu, Cov)
    x = np.linspace(0,.45,100)
    y = rf + x*(C-2*B*rf+A*rf**2)**0.5
    plt.plot(x,y, color='black', lw=2.5)

plot_points(mu, sigma, stocks)
plot_random_portfolios(n_simulations, n_assets)
plot_min_var_frontier(mu, Cov)

rf = 0.02
plot_Capital_Allocation_Line(rf, mu, Cov)
plt.show()
```

As this section illustrates, we can separate the portfolio construction process into two steps:

1. First, select the optimal composition of risky assets that lies on the tangent of the efficient frontier
2. Second, decide how much to invest in the tangent portfolio versus the risk-free asset.

The first step only involves math, once the expected returns and covariance matrices have been estimated. We will talk more about estimating the parameters used in the first step in the next chapter. The second step, which arguably has a much larger impact on investment returns, depends on an investor's personal preferences about the tradeoff between risk and returns, which is the topic in the next section.

2.4 Risk Tolerance

People inherently have different tolerances for risk, and part of the task of a robo-advisor is to figure out where along the risk-reward curve an investor should be. Robo-advisors usually ask a series of about 6-12 questions to gauge an investor's tolerance for risk, and then map those answers into a stock-bond portfolio. Many of these questionnaires can be found on the internet, and they are all similar in the types of questions they ask.

As an example, consider a few of the 13-questions from the risk tolerance scale that was developed, tested, and published by Grable and Lytton, and which is widely used by financial advisors (the full questionnaire can be found at www.kitces.com/wp-content/uploads/2019/11/Grable-Lytton-Risk-Assessment.pdf):

1. In general, how would your best friend describe you as a risk taker?
   A. A real gambler
   B. Willing to take risks after completing adequate research
   C. Cautious
   D. A real risk avoider
2. You are on a TV game show and can choose one of the following; which would you take?
   A. $1,000 in cash
   B. A 50% chance at winning $5,000
   C. A 25% chance at winning $10,000
   D. A 5% chance at winning $100,000
3. You have just finished saving for a “once-in-a-lifetime” vacation. Three weeks before you plan to leave, you lose your job. You would:
   A. Cancel the vacation
   B. Take a much more modest vacation
   C. Go as scheduled, reasoning that you need the time to prepare for a job search
   D. Extend your vacation, because this might be your last chance to go first-class
4. If you unexpectedly received $20,000 to invest, what would you do?
   A. Deposit it in a bank account, money market account, or insured CD
   B. Invest it in safe high-quality bonds or bond mutual funds
   C. Invest it in stocks or stock mutual funds
5. In terms of experience, how comfortable are you investing in stocks or

stock mutual funds?
A. Not at all comfortable
B. Somewhat comfortable
C. Very Comfortable

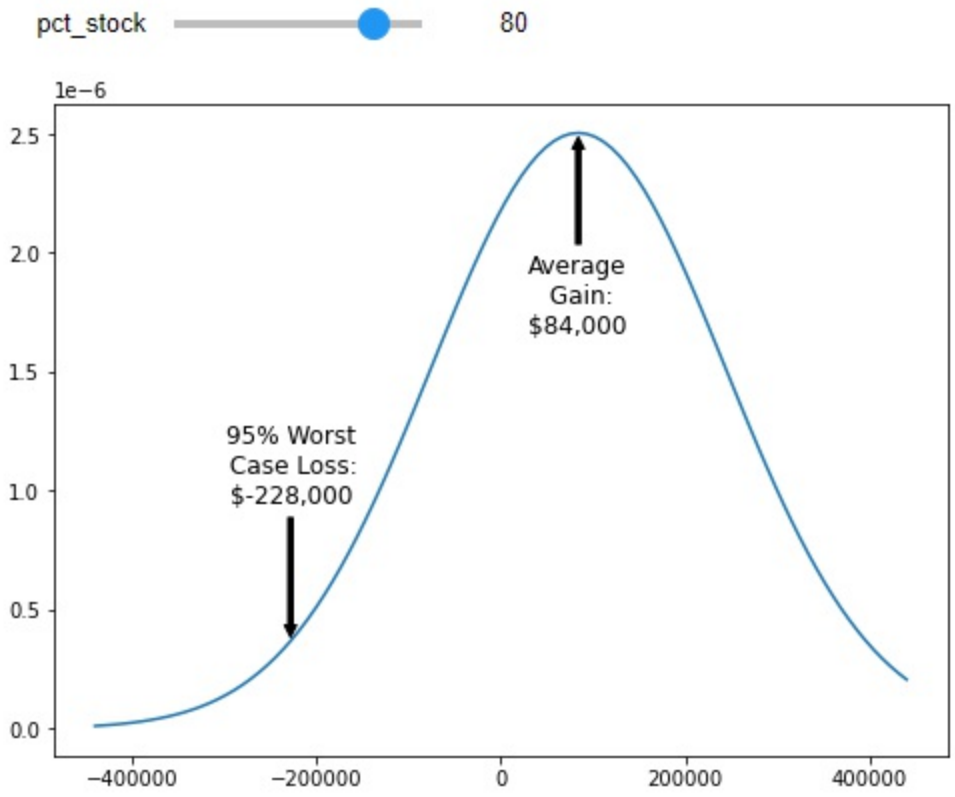
Points are assigned to each question, and the investor is categorized into one of five categories of risk tolerance (high, above average, average, below average, low) based on the aggregate score.

Some questionnaires ask the same type of question in different ways to gauge consistency. Some questionnaires ask *risk capacity* questions, in contrast to risk tolerance questions. These are questions about an investor's time horizon (an investor with a longer time horizon has a higher capacity for risk), any other assets holdings like pensions, and the need for cash. Sometimes questions about investment experience are included in risk tolerance questionnaires (see the last question above), which may seem unrelated to risk tolerance, but investors who have more experience are less likely to get flustered during market drops if they have experienced similar drops in the past.

Some online risk tolerance tools use sliders and charts to help investors visualize the risks. For example, one 401(k) provider shows a simulated path of the market value of an investor's portfolio, and the investor can see the larger drops as well as their higher average returns that are associated with riskier portfolios in order to help gauge their risk tolerance.

Python provides several widgets for making interactive graphs with sliders. Below we give a simple example of what you can do in a few lines of Python code. Figure 2.5 shows the 95% worst-case loss over a one-year horizon and the average gain, and the investor can move the slider at the top of the figure from 0% stocks to 100% stocks to gauge the level of risk they are comfortable with. We show the 95% worst-case loss since investors have a tendency to focus on losses (behavioral economists refer to this as loss aversion). We also show gains and losses in terms of dollars rather than percentages, since that is sometimes easier for investors to comprehend.

Figure 2.5 A slider used to visualize the effects of changing the stock-bond mix



The code that was used to generate Figure 2.5 is shown in Listing 2.10. The last line of the code below implements the slider. Every time the user moves the slider, the function `update()` is called and the chart is redrawn. You can set the minimum, maximum, and step size of the slider, and assign the value to a variable, which we call `pct_stock`, which represents the percentage of the portfolio in stocks, as opposed to bonds. The chart shows a normal distribution with the help of the `norm` function, which is part of the SciPy library. In the interactive plot, the mean value of the normal distribution is highlighted, which will go up as the percentage of stocks increases. But the maximum loss also goes up. We defined the maximum loss at the 95% worst-case loss, which is 1.96 standard deviations away from the average. Throughout the book, we will be using libraries that are not pre-installed with python, like the SciPy and ipywidgets libraries in Listing 2.10. These packages, as well as any other missing packages throughout the book, can be installed with `pip install`.

Listing 2.10

```
from scipy.stats import norm #A
from ipywidgets import *

def update(pct_stock):
    wealth = 1000000
    mu = np.array([0.02, 0.10]) * wealth #B
    sigma = np.array([0.05, 0.20]) * wealth #C
    w = np.array([1-pct_stock/100, pct_stock/100])
    Corr = [[ 1. ,-0.1], #D
            [-0.1, 1. ]]
    Cov = np.outer(sigma, sigma) * Corr
    mu_p, sigma_p = mu_sigma_portfolio(w, mu, Cov)

    x = np.linspace(-2.2*sigma[1], +2.2*sigma[1], 100)

    arrowprops = dict(arrowstyle="simple", color='k')
    xp = -1.96*sigma_p + mu_p #E
    xp = round(xp,-3)
    yp = norm.pdf(xp, mu_p, sigma_p)
    plt.figure(figsize=(8,6))
    plt.annotate('95% Worst\n Case Loss:\n${:,.0f}'.format(xp), x
             xycoords='data',
             xytext=(xp,yp*2.5),
             ha='center', va='bottom', size=12,
             arrowprops=arrowprops,
            )
    xp = mu_p #F
    xp = round(xp,-3)
    yp = norm.pdf(xp, mu_p, sigma_p)
    plt.annotate('Average\n Gain:\n${:,.0f}'.format(mu_p), xy=(xp
             xycoords='data',
             xytext=(xp,yp*.8),
             ha='center', va='top', size=12,
             arrowprops=arrowprops,
            )
    plt.plot(x, norm.pdf(x, mu_p, sigma_p))
    plt.show()

interact(update, pct_stock = widgets.IntSlider(value=50, min=0, m
```

In the slider example, as well as all the other examples in this chapter, we assumed some expected returns, standard deviations, and correlations for the various assets, which were inputs to our calculations. In the next chapter, we look at ways to estimate those inputs.

2.5 Appendix

For those who are interested, we derive the equations for the minimum-variance frontier when there is no risk-free rate and the Capital Market Line when we introduce a risk-free rate. This section is only for the mathematically inclined who are looking for a deeper understanding of the formulas presented earlier. Other readers can skip it and will still be able to understand the rest of the book.

2.5.1 No Risk-Free Rate

The problem of finding the optimal weights can be described mathematically as the solution to the following problem:

$$\min_{w} \quad \frac{1}{2} w^T \Sigma w$$

$$\text{subject to} \quad \mu^T w = \mu_p$$

$$\mathbf{1}^T w = 1$$

In this problem, we have not introduced the risk-free asset and we have not imposed short-sale constraints of the form $w_i \geq 0$.

We can use the method of Lagrangian multipliers to find the minimum of a function subject to equality constraints. The Lagrangian equation is then:

$$L = \mathbf{w}^T \Sigma \mathbf{w}/2 + \lambda(1 - \mathbf{1}^T w) + \gamma(\mu_p - \mu^T w)$$

Where λ and *γ* are Lagrange multipliers.

Taking the partial first derivatives of the Lagrangian with respect to the weights and setting the equations to zero gives:

$$\frac{\partial L}{\partial \mathbf{w}} = \Sigma \mathbf{w} - \lambda \mathbf{1} - \gamma \mu = 0$$

and we can solve for the optimal weights by premultiplying the above equation by $\mathbf{\Sigma}^{-1}$:

$$\mathbf{w}^* = \lambda \Sigma^{-1} \mathbf{1} + \gamma \Sigma^{-1} \mu$$

To find the Lagrange multipliers λ and *γ*, premultiply the optimal weights by **1** and by **μ**, and using the two constraints of the problem, we get

$$\underbrace{\mathbf{1}^T \mathbf{w}^*}_{1} = \lambda \underbrace{\mathbf{1}^T \Sigma^{-1} \mathbf{1}}_{A} + \gamma \underbrace{\mathbf{1}^T \Sigma^{-1} \mu}_{B}$$

$$\underbrace{\mu^T \mathbf{w}^*}_{\mu_p} = \lambda \underbrace{\mu^T \Sigma^{-1} \mathbf{1}}_{B} + \gamma \underbrace{\mu^T \Sigma^{-1} \mu}_{C}$$

Solving the two equations above for the two unknowns, the Lagrange Multipliers are

$$\lambda = \frac{C - B\mu_p}{AC - B^2}$$

$$\gamma = \frac{\mu_p A - B}{AC - B^2}$$

where for notational simplicity we define

$$A = \mathbf{1}^T \Sigma^{-1} \mathbf{1}$$

$$B = \mathbf{1}^T \Sigma^{-1} \mu$$

$$C = \mu^T \Sigma^{-1} \mu$$

Finally, the equation that relates the standard deviation of a portfolio to its mean using the optimal weights, which is the equation of the minimum-variance frontier, is

$$\sigma_p^2 = \mathbf{w}^{*T}\Sigma\mathbf{w}^*$$

$$= \mathbf{w}^{*T}\Sigma(\lambda\Sigma^{-1}\mathbf{1} + \gamma\Sigma^{-1}\mu)$$

$$= \lambda\mathbf{w}^{*T}\mathbf{1} + \gamma\mathbf{w}^{*T}\mu$$

$$= \frac{A\mu_p^2 - 2B\mu_p + C}{AC - B^2}$$

and

$$\sigma_p = \sqrt{\frac{A\mu_p^2 - 2B\mu_p + C}{AC - B^2}}$$

which is a sideways parabola in mean-variance space and a hyperbola in mean-standard deviation space. This is the equation used in Listing 2.8 in the chapter.

2.5.2 Adding a Risk-Free Rate

We take a very similar approach when we add a risk free rate. Let R_f be the return on the risk-free asset and let w_0 be the weight of the risk-free asset. The optimization problem is only slightly modified from earlier:

$$\min_{w,w_0} \quad \frac{1}{2} w^T \Sigma w$$

$$\text{subject to} \quad \mu^T w + R_f w_0 = \mu_p$$

$$\mathbf{1}^T w + w_0 = 1$$

We can eliminate w_0 from the second constraint:

$$w_0 = 1 - \mathbf{1}^T w$$

and then substitute into the first constraint:

$$\mu^T w + R_f(1 - \mathbf{1}^T w) = \mu_p$$

$$(\mu^T - R_f \mathbf{1}^T) w = \mu_p - R_f$$

The Lagrangian equation is:

$$L = \mathbf{w}^T \Sigma \mathbf{w}/2 + \gamma \left(\mu_p - R_f - (\mu^T - R_f \mathbf{1}^T) w\right)$$

Where *γ* is the Lagrange multiplier.

Taking the partial first derivatives of the Lagrangian with respect to the weights and setting the equations to zero gives:

$$\frac{\partial L}{\partial \mathbf{w}} = \Sigma \mathbf{w} - \gamma(\mu^T - R_f \mathbf{1}^T) = 0$$

and we can solve for the optimal weights by premultiplying the above equation by $\mathbf{\Sigma}^{-1}$:

$$\mathbf{w}^* = \gamma \Sigma^{-1}(\mu^T - R_f \mathbf{1}^T)$$

To solve for *γ*:

$$(\mu^T - R_f \mathbf{1}^T)^T \mathbf{w}^* = \gamma(\mu^T - R_f \mathbf{1}^T)^T \Sigma^{-1}(\mu^T - R_f \mathbf{1}^T)$$

$$\mu_p - R_f = \gamma(\mu^T \Sigma^{-1} \mu - 2R_f \mu^T \Sigma^{-1} \mathbf{1} + R_f^2 + \mathbf{1}^T \Sigma^{-1} \mathbf{1}^T$$

$$= \gamma(C - 2R_f B + R_f^2 A)$$

and

$$\gamma = \frac{\mu_p - R_f}{C - 2R_f B + R_f^2 A}$$

The equation for the standard deviation of a portfolio using the optimal weights is

$$\sigma_p^2 = \mathbf{w}^{*T}\Sigma\mathbf{w}^*$$

$$= \mathbf{w}^{*T}\Sigma\gamma\Sigma^{-1}(\mu - R_f\mathbf{1})$$

$$= \gamma\mathbf{w}^{*T}(\mu - R_f\mathbf{1})$$

$$= \gamma(\mu_p - R_f)$$

$$= \frac{(\mu_p - R_r)^2}{C - 2R_fB + R_f^2A}$$

and

$$\sigma_p = \frac{\mu_p - R_f}{(C - 2R_fB + R_f^2A)^{1/2}}$$

and the equation for the tangent line, which was plotted in Listing 2.9 in the chapter, is

$$\mu_p = R_f + \sigma_p \underbrace{(C - 2R_fB + R_f^2A)^{1/2}}_{\text{slope}}$$

2.6 Summary

In this chapter we have learned:

- Python has both matrix multiplication operators and element-by-element operators, and both are useful for various portfolio calculations.
- The minimum-variance frontier is a curve that represents the minimum variance achievable for a given level of expected return.
- Generating random portfolio weights is an intuitive way to visualize the minimum-variance frontier.
- The mathematical formula for the minimum-variance frontier is a hyperbola in mean-standard deviation space (and a sideways parabola in mean-variance space).
- The portfolio construction process can be divided into two steps: choosing a mix between a portfolio of risky assets and a risk-free asset based on an investor's risk tolerance, and computing the optimal weights for the portfolio of risky assets.
- Questionnaires are often used to gauge risk tolerance, where answers are assigned points and those points are mapped to an appropriate mix of stocks and bonds.
- With a single line of code, a slider can be added to help investors visualize the effects of changing the allocation between stocks and bonds