VSB – TECHNICAL UNIVERSITY OF OSTRAVA
FACULTY OF ECONOMICS

DEPARTMENT OF FINANCE

Application of Python in Portfolio Optimization

Využití Pythonu při optimalizaci portfolia

Student: Yize Li

Supervisor of diploma thesis: doc. Ing. Aleš Kresta, Ph.D.

Ostrava 2022

VSB – Technical University of Ostrava
Faculty of Economics
Department of Finance

Diploma Thesis Assignment

Student: **Bc. Yize Li**

Study Programme: N0412A050005 Finance

Title: Application of Python in Portfolio Optimization
Využití Pythonu při optimalizaci portfolia

The thesis language: English

Description:

1. Introduction
2. Description of Python
3. Description of Portfolio Optimization Methodology
4. Application of Portfolio Optimization Models in Python
5. Conclusion

Bibliography
List of Abbreviations
List of Annexes
Annexes

References:

BRUGIÈRE, Pierre. *Quantitative portfolio management: with applications in Python*. Cham, Switzerland: Springer, 2020. ISBN 978-3-030-37739-7.
HILPISCH, Yves J. *Python for finance: mastering data-driven finance*. Second edition. Sebastopol, CA: O'Reilly, 2018. ISBN 978-1-492-02433-0.
REILLY, Frank K., Keith C. BROWN and Sanford J. LEEDS. *Investment analysis and portfolio management*. 11th ed. Boston: Cengage, 2019. ISBN 978-1-305-26299-7.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **doc. Ing. Aleš Kresta, Ph.D.**

Date of issue: 19.11.2021
Date of submission: 22.04.2022

Ing. Petr Gurný, Ph.D.
Head of Department

doc. Ing. Vojtěch Spáčil, CSc.
Dean

Contents

1 Introduction

The portfolio optimization is the approach to selecting the optimal portfolio that provides the most profitable rate of return for each unit of risk taken by an investor. An investment portfolio is the distribution of an investor's assets, alternatively, it is the selection pool of an investor's investments. Modern portfolio theory assumes that an investor or asset manager seeks to maximize the return of a portfolio within a specific level of risk in a strategic manner. In this particular context, risk is defined as the standard deviation of portfolio returns. The optimal portfolio for an investor depends on a variety of choices such as risk preference, expected rate of return, and others.

The objective of this thesis is to validate and compare the out-of-sample performance of the following strategies: naive strategy, minimum variance strategy and maximum Sharpe ratio strategy. Therefore, we chose thirty stocks that have been listed on the NASDAQ Composite Index during the past ten years. The top 30 companies in the NASDAQ Composite Index ranked by market capitalization. The selected stock data is collected from 1/1/2011 to 26/12/2020 in the form of weekly data of the adjusted closing prices of stocks on Yahoo Finance. These stock prices are divided into two parts, including the in-sample period (1/1/2011-26/12/2015) and the out-of-sample period (2/1/2016-26/12/2020).

This thesis is divided into five parts. Chapter 1 is the introduction. Chapter 2 introduces the description of Python, including basic introduction of Python, brief history of Python, types of operators, Python keywords, variable in python, the JupyterLab interface and plotting in Python.

Chapter 3 describes the methodology for portfolio optimization. This chapter is divided into five parts to introduce. The second part is a description of the basic model input data, including expected returns and standard deviations. The second part is the effective frontier. The third part is backtesting analysis of portfolio optimization. The fourth part describes the naive strategy, the minimum variance strategy, and the maximum Sharpe ratio strategy. The last part is performance measurements, including the Sharpe ratio, Roy’s safety-first ratio, and maximum drawdown.

Chapter 4 is the application part. This chapter is divided into four parts to introduce. The first part is the description of data, where we provide a brief description of the

collected stock data. The second and third parts are used to analyze the performance of the naive strategy, the minimum variance strategy and the maximum Sharpe ratio strategy during the in-sample period and the out-of-sample period by Python. The last part is comparison of results, which includes comparison of the performances in in-sample and out-of-sample and comparison of the strategies based on out-of-sample performance

Chapter 5 discusses the results of this thesis and concludes with conclusions.

2 Description of Python

"Python is a high-level, multipurpose programming language that is used in a wide range of domains and technical fields." (Hilpish, 2018, p. 3). In this chapter, we introduce the description of Python, brief history of Python, types of operators, Python keywords, variables in python, the JupyterLab interface and plotting in Python.

2.1 Basic introduction of Python

"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed." (Hilpish, 2018, p. 3)

The Python is currently used by schools, universities, web companies, large corporations and financial institutions, as well as by junior programmers and highly skilled expert developers in any scientific field. The Python language includes the following features: open source, interpreted, multi-paradigm, cross-platform, cross-platform, dynamically typed, indentation aware, and garbage collecting.

Open source

Python and most of the available support libraries and tools are open source and generally have fairly flexible and open licenses.

Interpreted

The reference CPython implementation is the language's interpreter, which translates Python code into executable bytecode at runtime.

Multiparadigm

Python supports a number of different programming and implementation paradigms, including object-oriented and imperative, functional and procedural programming.

Multipurpose

Python provides the capability to be applied for high-speed, interactive code development or for building large applications. Python can be used for low-level system operations or high-level analysis tasks.

Cross-platform.

Python operates on the most critical operating systems, like Windows, Linux, and macOS. It is used to build desktop and web applications, on the largest clusters and most powerful servers, and for high-level analysis tasks.

Dynamically typed

Types in Python are typically inferred at runtime, rather than declared statically as in most compiled languages.

Indentation aware

Python uses indentation to mark blocks of code, unlike most other programming languages, rather than parentheses, curly braces, or semicolons.

Garbage collecting

Python features automatic garbage collection and no programmer is required to manage memory.

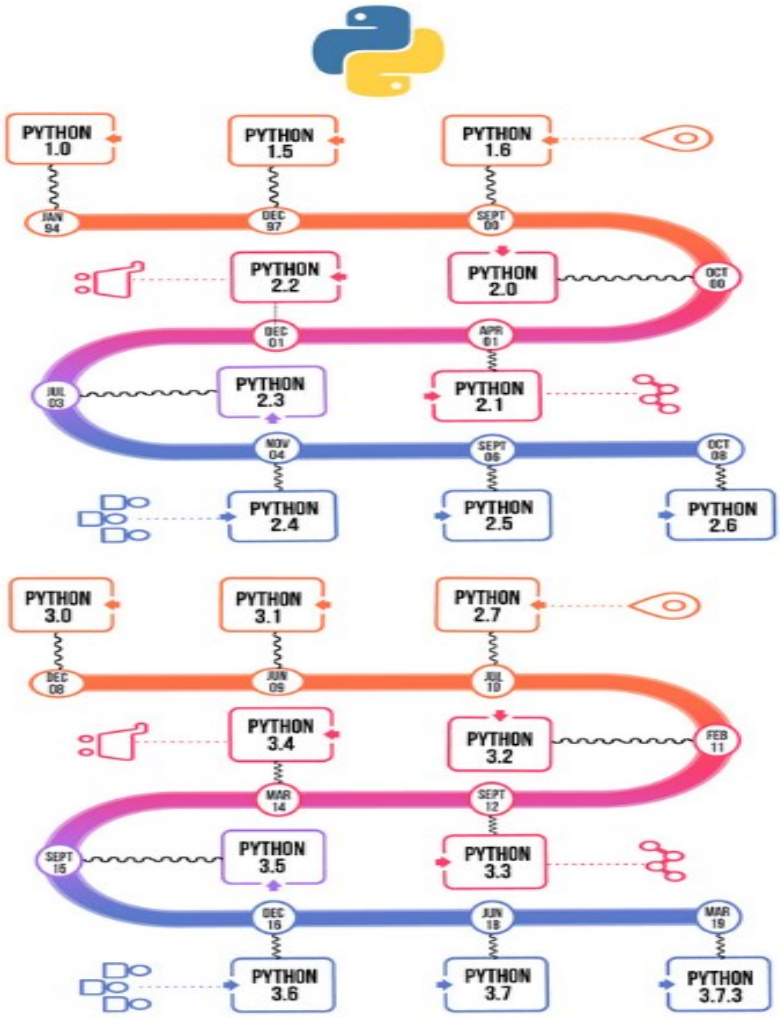
2.2 Brief history of Python

The Python as a high-level programming language is universal and can be widely used. It is object oriented and provides powerful visual programming capabilities and a good user interface, with good portability and extensibility. It is widely used in various applications on the Internet.

"*In fact, development efforts began in the 1980s by Guido van Rossum from the Netherlands. He is still active in Python development and has been awarded the title of Benevolent Dictator for Life by the Python community. In July 2018, van Rossum stepped down from this position after decades of being an active driver of the Python core development."* (Hilpish, 2018, p. 5)

When python is released, we compare it to Java, C++, C etc. Python is quite well designed with the concept of representing concepts in less code. Its primary purpose is to provide readability to code and productivity for advanced developers. Since it uses an object-oriented approach to programming, it brings a lot of convenience to people from different fields when developing a new system. The release is fully capable of providing inherited classes, several core data type exception handling and functionality. The illustrations and timelines for different versions of Python are shown in Figure 2.1

Figure 2.1 The illustrations and timelines for different versions of Python



Source: https://www.geeksforgeeks.org/history-of-python/

Python 2.0 was released on October 16, 2000, with many major new features, including a loop-detection garbage collector for memory management (in addition to reference counting) and support for Unicode. However, the most important change is the

development process itself, shifting to a more transparency and community supported process.

Python 3.0 is a major backwards compatible version which was released on December 3, 2008 after a long period of testing. It also has many of the main features backported to the backward-compatible Python 2.6 and 2.7, which are no longer supported though.

Python is the constant source of inspiration for numerous other coding languages, such as Ruby, Cobra, Boo, Coffee Script ECMAScript, Groovy, Swift Go, OCaml, Julia, and many others. The language Python is in use for various purposes such as development, scripting, generation and software testing. For its elegance and simplicity, top technology organizations like Dropbox, Google, Quora, Mozilla, Hewlett-Packard, Qualcomm, IBM, and Cisco have implemented Python.

2.3 Types of operators

Operator is a symbol which performs a mathematical calculation on a variable or a value. The operator performs an operation on an operand (value) and then returns a result. The following types of operators are supported in the Python language: arithmetic operators, comparison operators, assignment operators, logical operators, bitwise operators, membership operators and identity operators.

Arithmetic operators

Arithmetic operators are used for performing mathematical operations, like addition, subtraction, multiplication, and division. Python has seven arithmetic operators for different mathematical operations. There are some well-known operators as addition, subtraction, but also ones used to find the modulus, power, etc. The addition operator "+" adds two operands and gives their sum. The subtraction operator "-" subtracts second operand from first and gives their difference. The multiplication operator "*" multiplies two operands and gives their product. The division operator "/" divides the first operand by second and gives their quotient. The modulus operator "%" divides the first operand by second and gives their remainder. The power operator "**"raises the first raised to power of second. The list of applicable operators is shown in Table 2.1.

Table 2.1 Arithmetic operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division(float)
//	Division(floor)
%	Modulus
**	Power

Source: https://www.geeksforgeeks.org/python-operators/

Comparison operators

Comparison operators compare the two values on either side of them and decide how they are related to each other. They are also called relational operators. Python has 6 relational operators: The greater than operator “>” returns True if the first operand is greater than the second. The less than operator “<” returns True if the first operand is less than the second. The equal to operator “==” returns True if the first operand is equal to the second. The not equal to operator “! =” returns True if the first operand is not equal to the second. The greater than or equal to operator “>=” returns True if the first operand is greater than or equal to the second. The less than or equal to operator “<=” returns True if the first operand is smaller than or equal to the second. The list of comparison operators is shown in Table 2.2.

Table 2.2 Comparison operators

Operator	Description
>	Greater than
<	Less than
==	Equal to
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to
is	x is the same as y
is not	x is not the same as y

Source: https://www.geeksforgeeks.org/python-operators/

Assignment operators

Assignment operators perform an operation and use to assign values to variables. There are 8 assignment operators in Python. The assign operator "=" from right side operands to left side operand. The add and assign operator "+=" adds two operands and assigns the result to the variable on left. The subtract and assign operator "-=" subtracts second operand from first and assigns to first. The add and assign operator "+=" adds two operands and assigns the result to the variable on left. The subtract and assign operator "-=" subtracts second operand from first and assigns to first. The multiply and assign operator "*=" assigns the product to the variable on left. The divide and assign operator "/=" assign the division of two operands to the first. The modulus and assign operator "%=" perform modulus on two operands and assigns to first. The exponentiation and assign operator "%=" perform exponentiation on two values and assigns to first. The list of assignment operators is shown in Table 2.3.

Table 2.3 Assignment operators.

Operator	Description
=	Assign
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulus and assign
**=	Exponentiation and assign
//=	Floor-divide and assign

Source: https://www.tutorialspoint.com/

Logical operators

Logical operators use to combine conditional statements and perform logical sum, logical or and logical non-operations. There are 3 logical operators in Python. The logical "and" operator returns True if both operands are True. The logical "or" operator returns True if even one operand is True. The logical "not" operator returns True if the operand is false. The list of logical operators is shown in Table 2.4.

Table 2.4 Logical operators.

Operator	Description
and	Logical and
or	Logical or
not	Logical not

Source: https://www.geeksforgeeks.org/python-operators/

Bitwise operators

It is used to work on bits and perform bit-by-bit operations. There are 6 bitwise operators in Python. The bitwise and operator "&" performs logical AND on

corresponding bits in values. The bitwise or operator "|" performs logical OR on corresponding bits in values. The bitwise xor operator "^" performs logical XOR on corresponding bits in values. The bitwise 1's complement operator "~" returns the bitwise negation of a value. Each bit is inverted. The bitwise right-shift operator ">>" shifts bits for a value by given number of places right. Some bits are lost. The bitwise left-shift operator "<<" shifts bits for a value by a given number of places left. It adds 0s to new positions. The list of bitwise operators is shown in Table 2.5.

Table 2.5 Bitwise operators.

Operator	Description
&	Bitwise and
|	Bitwise or
^	Bitwise xor
~	Bitwise 1's complement operator
>>	Bitwise right shift
<<	Bitwise left shift

Source: https://www.geeksforgeeks.org/python-operators

Membership operators

Its membership operators, including in and not in, are used to test whether a value or variable is in a sequence, such as a string, list, or tuples. There are 2 membership operators in Python. The "in "operator returns True if value is found in the sequence. The "not in" operator returns Ture if value is not found in the sequence. The list of membership operators is shown in Table 2.6.

Table 2.6 Membership Operators

Operator	Description
in	True if value is found in the sequence
not in	Ture if value is not found in the sequence

Source: https://www.geeksforgeeks.org/python-operators

Identity operators.

It is used to verify that two values are in the same location in memory. If two variables are equal, it does not mean that they are the same. There are 2 identity operators in Python. The “is “operator returns True if the operands are identical. The “is not” operator returns Ture if the operands are not identical. The list of identity operators is shown in Table 2.7.

Table 2.7 Identity operators

Operator	Description
is	Ture if the operands are identical.
not	Ture if the operands are not identical.

Source: https://www.geeksforgeeks.org/python-operators

2.4 Python keywords

A keyword in Python is a reserved word. The keywords are not allowed to be used as variable names, function names, or any other identifiers. A keyword is used in Python to determine the language's syntax and structure. In Python, keywords are case-sensitive. In Python 3.7 the number of keywords is 33. This number changes slightly over time. All keywords are lowercase, with the exception of True, False, and None, which have to be written as-is. The list of all Python keywords is shown in Table 2.8.

Table 2.8 Python keywords

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
FALSE	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value

nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
TRUE	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To end a function, returns a generator

Source：https://www.w3schools.com/python/python_ref_keywords.asp

2.5 Variables in Python

Variables are used to store data, and they occupy the memory space depending on the kind of value that we assign to them. It is easy to create variables in Python, you only need to write the name of the variable on the left side of the "=" operator and the value on the right side. People don't need to explicitly mention the variable's type. Python will make inferences about their type based on the values we allocate. The variable name is called an identifier. There are several rules that must be followed while naming the variables in Python.

- The name of a variable must always begin with a letter or an underscore (_). For example. _str, str, num, _num are all valid names for variables.
- The variable's name cannot start with a number. For example: 9num is not an allowed variable name.
- The variable name cannot have special characters, such as %, $, #, etc. It is allowed to have only the alphanumeric characters and underscores (A to Z, a to z, 0-9 or _).

- In Python, the names of variables are case-sensitive, this means that num and NUM are two different variables in Python. The code to create variables in Python is shown in Figure 2.2.

Figure 2.2 Code to create variables in Python

```
num = 100       #num is of type int
str = "Chaitanya"       #str is of type string
```

Source: https://beginnersbook.com/

"Python is a dynamically typed language, which means that the Python interpreter infers the type of an object at runtime. In comparison, compiled languages like C are generally statically typed. In these cases, the type of an object has to be specified for the object before compile time." (Hilpish, 2018, p. 62) The most common types of variables are numbers, integers, floats and strings.

Numbers

Number data types are used to store numeric values. They are immutable data types, which means that changing a numeric data type will allocate a new object. Python supports 4 different types of numbers:

- Fint (signed integers) is often called integers or ints, which are positive or negative whole numbers with no decimal point.
- Long (long integers) are also called long, and they are infinite size integers that are written like integers and followed by an uppercase or lowercase L.
- Float (floating point real numbers) is also known as floating point numbers, which represent real numbers and are written by dividing the integer and decimal parts by a decimal point. Floating point numbers can also be written in the scientific notation, using E or e for powers of 10 ($2.5e^2 = 2.5 * 10^2 = 250$).
- Complex numbers (complex numbers) are of the form a + bJ, where both a and b are floating-point numbers with J (or j) representing the square root of -1 (which is an imaginary number). The real part of the numeral is a and the imaginary part is b. The complex number is not much used in Python programming.

Integers

Integers are zero, positive or negative integers without fractional parts and with infinite precision, such as -4, -3,0,5,7 etc. In Python, to declare an integer, simply write "variable Name = initial value"

"The built-in function type provides type information for all objects with standard and built-in types as well as for newly created classes and objects. In the latter case, the information provided depends on the description the programmer has stored with the class. There is a saying that "everything in Python is an object." This means, for example, that even simple objects like the int object just defined have built-in methods. One can get the number of bits needed to represent the int object in memory by calling the method bit_length ()" (Hilpish, 2018, p. 62)

Floats

Float point numbers serve to indicate the real numbers and are written by separating the integer and decimal parts with a decimal point, such as 97.98, 32.3+e18, -32.54e100 etc. Floating-point numbers in Python are expressed as 64-bit double-precision values.

The float type installs the numbers.Real abstract base class. Returns a float expression which is converted to a floating-point number. These are additional methods for float:

- float.as_integer_ratio () returns a couple of integers with a ratio exactly equal to the real floating-point number which has a positive denominator. At infinity, it gives rise to an overflow error and non-numeric value errors (NaNs).
- float.is_integer () returns True for float instances with finite integral values, otherwise, it returns False.
- float.hex () returns a hex string of a float.
- float.fromhex (s) returns a hex string representation of a floating point number.
- float.fromhex (s) returns a floating point number expressed by the hexadecimal string s. String s has leading and trailing whitespace.

Strings

"The basic data type to represent text in Python is str. The str object has a number of helpful built-in methods. In fact, Python is generally considered to be a good choice when it comes to working with texts and text files of any kind and any size. A str object is

generally defined by single or double quotation marks or by converting another object using the str () function." (Hilpisch, 2018, p. 69) The string types has also a lot of built-in function, the list of the applicable functions can be seen in Table 2.9.

Table 2.9 Lists a number of help methods of str object.

Method	Arguments	Returns/result
`capitalize`	`()`	Copy of the string with first letter capitalized
`count`	(*sub*[, *start*[, *end*]])	Count of the number of occurrences of substring
`decode`	([*encoding*[, *errors*]])	Decoded version of the string, using *encoding* (e.g., UTF-8)
`encode`	([*encoding*+[, *errors*]])	Encoded version of the string
`find`	(*sub*[, *start*[, *end*]])	(Lowest) index where substring is found
`join`	(*seq*)	Concatenation of strings in sequence *seq*
`replace`	(*old*, *new*[, *count*])	Replaces *old* by *new* the first *count* times
`split`	([*sep*[, *maxsplit*]])	List of words in string with *sep* as separator
`splitlines`	([*keepends*])	Separated lines with line ends/breaks if *keepends* is `True`
`strip`	(*chars*)	Copy of string with leading/lagging characters in *chars* removed
`upper`	`()`	Copy with all letters capitalized

Source: Hilpisch (2018, p. 69)

2.6 The JupyterLab Interface

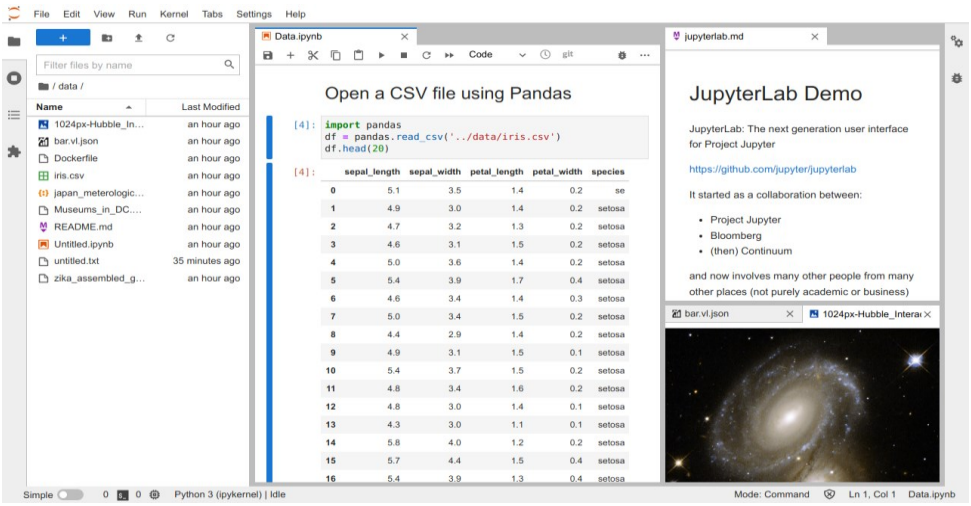
"JupyterLab provides flexible building blocks for interactive, exploratory computing. While JupyterLab has many features found in traditional integrated development environments (IDEs), it remains focused on interactive, exploratory computing.

The JupyterLab interface consists of a main work area containing tabs of documents and activities, a collapsible left sidebar, and a menu bar. The left sidebar contains a file

browser, the list of running kernels and terminals, the command palette, the notebook cell tools inspector, and the tabs list.

JupyterLab sessions always reside in a workspace. Workspaces contain the state of JupyterLab: the files that are currently open, the layout of the application areas and tabs, etc. Workspaces can be saved on the server with named workspace URLs." (Project Jupyter, 2018). The Jupyterlab launcher is shown in Figure 2.3.

Figure 2.3 Jupyterlab launcher



Source: https://jupyterlab.readthedocs.io/

A menu bar at the top of JupyterLab contains top-level menus which reveal the actions which are available in JupyterLab and the keyboard defaults to them. The JupyterLab extension allows to create a new top-level menu in the menu bar as well. The default menus for that include File, Edit, View, Run, etc.

- File: operations associated to files and directories
- Edit: operations associated to editing files and other activities
- View: operations for changing the appearance of JupyterLab
- Run: operations for running code in different activities, such as notebooks and code consoles
- Kernel: operations to manage the kernel, which is a separate process that runs the code
- Tabs: a list of documents and activities open in the Dock panel

- Settings: editor with common settings and advanced settings
- Help: a list of JupyterLab and the Kernel Help links

2.7 Plotting in Python

Graphs in Python can be plotted through the use of the Matplotlib library. Matplotlib library is mainly used in graph plotting. Matplotlib needs to be installed before you can use it to plot graphs. Matplotlib can plot simple line plots, bar graphs, histograms, and pie charts. The built-in functions in the Matplotlib library are available for plotting all types of graphs.

For example, when we want to plot a straight line in a graph, we use Matplotlib to draw a simple straight line in a graph. Here are the steps to draw a straight line.

(1) Import matplotlib.

(2) Specify the x and y coordinates of the lines.

(3) Use the .plot () function to plot the specified point with a specific function.

(4) Use the .xlabel () and .ylabel () functions to name the x- and y-axes.

(5) use the title () function to add a title to the graph (optional).

(6) Use the .show () function to display the graph.

The example for plotting straight lines in Python is shown in Figure 2.4.

Figure 2.4 Plotting the straight line

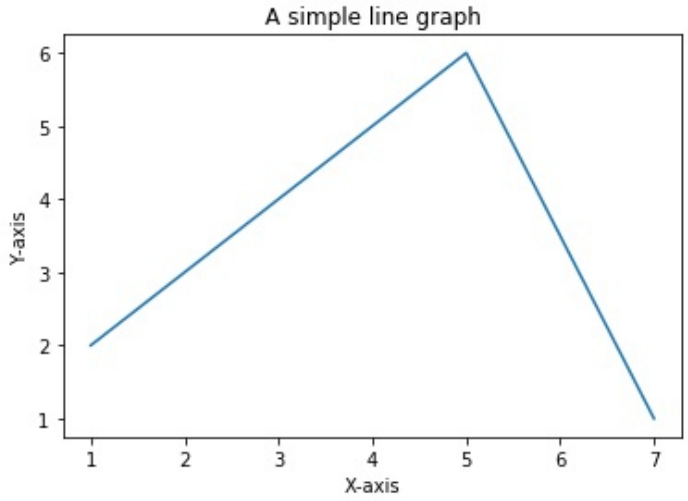
```
import matplotlib.pyplot as plt

x=[1,3,5,7]
y=[2,4,6,1]
plt.plot(x,y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title("A simple line graph")
plt.show()
```

Source: https://www.tutorialspoint.com/

The above code plots the points (1,2), (3,4), (5,6), (7,1), which are connected with straight lines and shown in Figure 2.5.

Figure 2.5 Output of line graph



Source: https://www.tutorialspoint.com/

Another type of chart we can plot in Python is a bar chart. The bar chart is a means of presenting data in rectangles of different heights at particular locations on the X-axis. The procedure to draw a bar chart is as follows:

(1) Import Matplotlib.

(2) Specify the x coordinates where the lower-left corner of the rectangle is located.

(3) Specify the height of the bars or rectangle plot.

(4) Specify the label of the bar chart.

(5) Plot the bar using the. bar () function.

(6) Label the x-axis and y-axis.

(7) Give the graph a title.

(8) Use the. show () function to display the graph.

The example for plotting bar chart in Python is shown in Figure 2.6.

Figure 2.6 Plotting bar chart

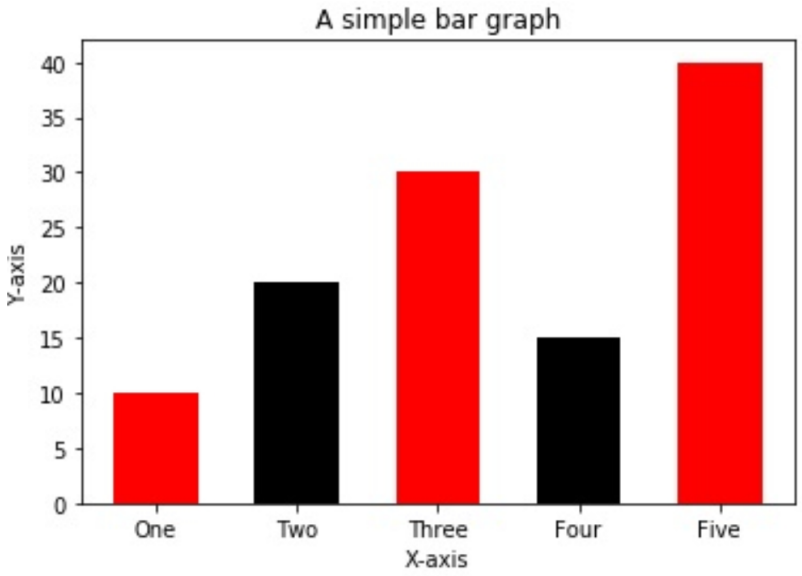
```
import matplotlib.pyplot as plt

left_coordinates=[1,2,3,4,5]
heights=[10,20,30,15,40]
bar_labels=['One','Two','Three','Four','Five']
plt.bar(left_coordinates,heights,tick_label=bar_labels
d','black'])
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title("A simple bar graph")
plt.show()
```

Source: https://www.tutorialspoint.com/

The example for plotting a bar graph chart in Python can be seen in Figure 2.7. The width parameter in Pl. Bar () describes the width of each bar. The color list specifies the color of the bar. The output of the code is shown in Figure 2.7.

Figure 2.7 Output of bar charts



Source: https://www.tutorialspoint.com/

3 Description of portfolio optimization methodology

Portfolio optimization is the process of choosing the ratio of various assets in a portfolio in a way that makes the portfolio perform better than any others depending on specific constraints.

Modern portfolio theory (Markowitz, 1952) argues generally that investors or asset managers seek to maximize the return of a portfolio within a specified level of risk in a strategic way. In this specific setting, risk is defined as the standard deviation of portfolio returns. Furthermore, the optimal or efficient portfolio perspective introduces to describe the portfolio which maximizes returns at a given level of risk. The efficient frontier permits a graphical representation of a group of optimal portfolios at a specific, well defined risk level.

In this chapter, the methodology describes for portfolio optimization. This chapter is divided into four parts to introduce. The first part is the effective frontier. The second part is a description of the basic model input data, including expected returns and standard deviations. The third part is backtesting analysis of portfolio optimization. The last part describes the naive strategy, the minimum variance strategy, the maximum Sharpe ratio .

3.1 Description of basic input data

An effective portfolio consists of investments that provide the greatest return for the risk, or arguably the lowest risk for a given return. To form an effective portfolio, people need to know how to calculate the return and risk of the portfolio and how to reduce the risk through diversification. The basic description of a stock portfolio can help us to analyze and conduct a stock portfolio in a more diversified way.

As we talk about asset returns, we can distinguish between discrete and continuously compounded returns. In the discrete case, the return R_t is computed as a relative change of the asset's price P_t,

$$R_t = \frac{P_t - P_{t-1}}{P_t}. \tag{3.1}$$

After calculating asset returns, we can obtain the following formulas for discrete and continuously compound portfolio returns,

$$R_{P,t} = \sum_{i=1}^{N} w_i R_{i,t}. \tag{3.2}$$

In this equation, the return of the portfolio represented by $R_{P,t}$ is given by the sum of the products of the specified return of stock $R_{i,t}$ and the weights of each stock w_i in the portfolio.

We can calculate the expected return of the portfolio as follows,

$$E(R_P) = w^T . E(R) = \sum_{i=1}^{N} w_i . E(R_I) , \tag{3.3}$$

the variance of the portfolio return,

$$\sigma_p^2 = w^T . Q . w = \sum_{i=1}^{N} \sum_{j=1}^{N} w_i . \sigma_{i,j} . w_j , \tag{3.4}$$

and standard deviation of the portfolio return,

$$\sigma_p = \sqrt{\sigma_p^2}, \tag{3.5}$$

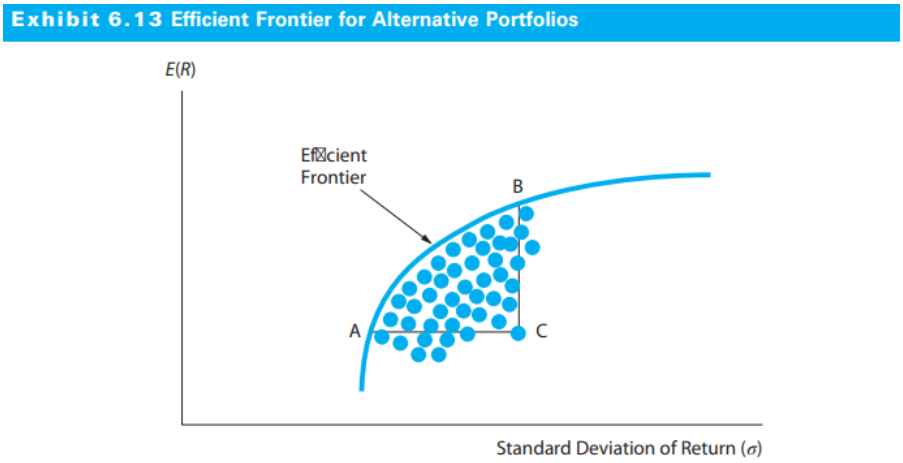
where Q represents covariance matrix, w represents the vector of weights and $E(R)$ represents the vector of expected returns.

3.2 The efficient frontier

Investors are risk averse, therefore an optimal portfolio of investments can be defined as a portfolio that generates a specific expected return objective with minimal risk. Nevertheless, there is a different portfolio of assets to represent the risk minimizing portfolio for each different expected return objective, and a larger expected return objective requires a portfolio with a greater level of risk.

The efficient frontier is the set of portfolios with the greatest return for each given level of risk or the set of portfolios that provide the best portfolio with the lowest risk for a given level of expected return. The set of portfolios that minimizes risk for each potential expected return objective is referred to as the efficient frontier. The portfolios which are below the efficient frontier are suboptimal because they do not provide sufficient returns for the given level of risk. The efficient frontier for alternative portfolios is shown in Figure 3.1

Figure 3.1 Efficient frontier for alternative portfolios



Source：Reilly et al. (2019, p. 218)

"An illustration of such a frontier is shown in Exhibit 6.13. Every portfolio that lies on the efficient frontier has either a higher rate of return for the same risk level or lower risk for an equal rate of return than some portfolio falling below the frontier. Thus, we would say that Portfolio A in Exhibit 6.13 dominates Portfolio C because it has an equal rate of return but substantially less risk. Similarly, Portfolio B dominates Portfolio C because it has equal risk but a higher expected rate of return. Because of the benefits of diversification among less-than-perfectly correlated assets, we would expect the efficient frontier to be made up of portfolios of investments rather than individual securities. Two possible exceptions arise at the end points, which represent the asset with the highest return and the asset with the lowest risk." (Reilly et al., 2019, p. 217)

3.3 Portfolio optimization strategies

The idea and goal of portfolio optimization are that investors want to build their portfolios to be able to generate the maximum possible return while maintaining the amount of risk they are willing to take. This represents the need for investors to create a balanced portfolio by spreading their investment capital across a variety of assets. This is done by balancing these assets to achieve the desired risk-return outcome. The result of portfolio optimization should be what the investor considers to be an efficient portfolio, meaning that it produces the highest possible return for given risk tolerance.

3.3.1 Naive strategy

A typical equally weighted portfolio is the naive portfolio, which has components that invest with equal weights. The portfolio's components are equally weighted for investment. The calculation of equal weights for each component in a naive portfolio is as follows,

$$w_{naive} = \frac{1}{N}, \tag{3.6}$$

where N is the number of the components in naive strategy.

"There are two reasons for using the naive rule as a benchmark. First, it is easy to implement because it does not rely either on estimation of the moments of asset returns or on optimization. Second, despite the sophisticated theoretical models developed in the last 50 years and the advances in methods for estimating the parameters of these models, investors continue to use such simple allocation rules for allocating their wealth across assets." (Demiguel et al., 2009, p. 1916)

Diversification of a portfolio by not considering or incorrectly considering the mathematical equations in the capital asset pricing model. Naive diversification is based on the assumption that by investing in enough uncorrelated assets one can adequately reduce risk and still make a profit. In addition, people may diversify naively by misapplying asset pricing models for capital and finding the wrong efficient portfolio frontier. Such diversification does not necessarily reduce the risk for a given expected return, and may in fact add risk.

3.3.2 Minimum variance strategy

The minimum variance portfolio is a part of Markowitz efficient portfolios. *" The Markowitz model is the type of the mean variance model, for which is allowed to invest only risky assets while selling the assets short is not feasible. Respecting the fact, that there occur three distinct steps, we have to formulate three types of problems. The first step is to find the portfolio with the minimal risk (portfolio A), the second step is to find the portfolio with the maximal expected return (portfolio B). Subsequent steps consist in selecting the portfolios for interior points of the efficient set (portfolios C to H)."* (Zmeškal et al., 2004, p. 84)

For the minimum variance strategy, the objective function of find the minimum risk portfolio is shown in Figure 3.2.

Figure 3.2 Find the minimum risk portfolio

Objective function

$\sigma_P \rightarrow min.$

Constraints

$\sum_i x_i = 1$ (C1)

$x_i \geq 0, for\ i = 1,2,\cdots,N,$ (C2)

$where\ \sigma_P = \sqrt{\sum_i \sum_j x_i \cdot \sigma_{ij} \cdot x_j} = \sqrt{x^{\rightarrow T} \cdot C \cdot x^{\rightarrow}}.$ (E1)

Source: Zmeškal et al. (2004, p. 84)

"The objective function expressed the minimal stand deviation of the portfolio we are looking for. The constraint (C1) states that the sum of relative shares (percentages) x_i is equal to 1. Hence, it is allowed to invest just the money amount we have held initially. The constraints (C2) exclude negativity, since short selling is not allowed there. By equation (E1) we define the calculation of the standard deviation for the optimal portfolio." (Zmeškal et al., 2004, p. 84)

3.3.3 Maximum Sharpe ratio strategy

The portfolio is optimized to provide the maximum Sharpe ratio, which is a ratio based on the historical data and compares the amount of return with the amount of risk. The returns are based on compound annual growth rate (CAGR) and the risk is based on volatility. This portfolio is well adapted to risk-averse investors with modest growth expectations.

Based on Zmeškal et al. (2004, p. 91), we identify the market portfolio M with the maximum Sharpe ratio strategy. For the maximum Sharpe ratio strategy, the objective function of set up the market portfolio M is shown in Figure 3.3.

Figure 3.3 Set up the market portfolio (M)

Objective function

$$\frac{E(R_M)-R_F}{\sigma_M} \rightarrow max.$$

Constraints

$$x_F + \sum_k x_k = 1, \quad \text{(C1)}$$

$$x_k \geq 0, for\ k = 1,2,\cdots,N, \quad \text{(C2)}$$

$$x_F = 0\,, \quad \text{(C3)}$$

$$where\ E(R_M) = \sum_{I=1}^{N+1} x_i \cdot E(R_i)\,, \quad \text{(E1)}$$

$$var(R_M) = \sum_{i=1}^{N+1}\sum_{j=1}^{N+1} x_i \cdot \sigma_{ij} \cdot x_j = x^{\rightarrow T} \cdot C \cdot x^{\rightarrow}\ , \quad \text{(E2)}$$

$$\sigma_M = \sigma(R_M) = \sqrt{var(R_M)}\ , \quad \text{(E3)}$$

Source: (Zmeškal et al., 2004, p. 90)

"Here x_k means a share of the risky asset, similarly, x_F states a share of the riskless asset in the portfolio, and $x_i(x_j)$ is used to indicate a share of any asset. The vector $x^{\rightarrow}$ and the covariance matrix C include also all types of assets and their relationship.

The objective function stands for the maximization of the slope of the CML line (effective set). The constraint (C1) represents the investment structure and the set of feasible variables. The constraint (C2) defines, that just investment in risky assets allowed. Similarly, the constraint (C3) states, that the riskless assets cannot be included in the portfolio. Equations (E1), (E2) and (E3) formulate the calculation of portfolio parameters.

Finding the effective portfolio for a given standard deviation is based on maximization of the mean value of portfolio return for a given level of the standard deviation. Here, it is allowing to invest into both, risky and riskless assets. Further, riskless assets can be also shorted." (Zmeškal et al., 2004, p. 91)

3.4 Backtesting analysis of portfolio optimization

"Within backtesting procedure, the historical data are utilized. For each observation (day) we compute the portfolio composition based on the information set known at that

moment, i.e., the weights of portfolio at time t are determined based on the returns/prices of the assets in historical window(t-m, t-1), where m determines the size of past data, which are utilized. In order to avoid look-ahead bias, it is vital to assure that algorithm utilizes only information that would have been available at the time of the portfolio rebalancing." (Kresta, 2015, p. 61)

We can calculate the ex-post portfolio returns,

$$r_{P,t} = \sum_{i=1}^{n} r_{i,t} \, . w_{i,t}, \tag{3.7}$$

where r represents the ex-post observed returns and $w_{i,t}$ represents the weights of assets in the portfolio, which are obtained by portfolio optimization based on the returns/prices of the assets in the period $(t-m, t-1)$. We can then also calculate the ex-post wealth path,

$$W_{t+1} = W_t \, . \left(1 + r_{p,t}\right). \tag{3.8}$$

3.5 Performance measurements

This section presents two performance ratios: Sharpe ratio and Roy's safety-first ratio and maximum drawdown to measure the efficiency of different asset allocation strategies.

3.5.1 Sharpe ratio

The Sharpe ratio is one of the methods most widely used to calculate risk-adjusted returns. *"The Sharpe ratio is maximal for portfolios belonging to the Capital Market Line. All the portfolios belonging to the Capital Market Line have the same Sharpe ratio, which is equal to the slope of the Capital Market Line. When investing in a single portfolio, wealth should be allocated first by determining a portfolio with the maximum Sharpe ratio and then by allocating all the wealth between this portfolio and the risk-free asset."* (Brugière, 2020, p. 95)

The formula of Sharpe ratio is as follows,

$$Sharpe \ ratio = \frac{(E(R_p) - R_f)}{\sigma_p}, \tag{3.9}$$

where the R_p represents the return of the portfolio, R_f represents risk-free rate and σ_p represents the standard deviation of the portfolio's excess return.

In the Sharpe ratio, the risk-free rate is deducted from the average rate of return, which allows the investor to better separate out the profits associated with the risk-taking activity. The risk-free rate of return is the return on a risk-free investment, meaning that it is the expected return without the investor taking any risk. In general, the higher Sharpe ratio indicates a higher reward per unit of risk and the more desirable the investment vehicle is. A low Sharpe ratio means that the fund is earning returns by taking higher risk. A high Sharpe ratio means that the fund has a higher ability to diversify and reduce unsystematic risk and that there is room for returns to rise. When the Sharpe ratio is above the CML, it indicates that the fund is outperforming the overall performance of the market.

3.5.2 Roy's safety-first ratio

Roy's safety-first criterion, also referred to SFRatio, is a method of investment decision-making that creates a minimum required return at a defined level for a given level of risk. Roe's safety-first criterion permits investors to invest in potential portfolios by comparing them according to the probability that a portfolio's return will be lower than its minimum expected return limit. It can be calculated as follows,

$$SFRatio = \frac{r_e - r_m}{\sigma_p}, \tag{3.10}$$

where r_e is expected return on the portfolio, r_m is investor's minimum required return and σ_p is the standard deviation of the portfolio. The SFRatio offers a probability of receiving the minimum required return on a given portfolio. The optimal decision for investors is to choose the investment portfolio which has the highest SFRatio. The formula can be used by investors to calculate and assess the various situations that involve different portfolio asset class weights, and various investments and otherwise influence the probability of having met their required minimum return in terms of threshold.

3.5.3 Maximum drawdown

The maximum drawdown is a specific indicator that measures the drawdown and looks for the maximum movement from a high to a low before a new peak is reached. The MDD only measures the amount of maximum loss and does not consider the efficiency of the frequency of big losses. Because an MDD measures only the maximum decline, it does not indicate the length of time it takes for an investor to recover a loss, or the investment to even recover at all.

Maximum downturn (MDD) is a measure used to evaluate the relative riskiness of a stock selection strategy to another strategy due to its focus on capital conservation, which is the primary concern of for most investors. The two selection strategies may have the same mean outperformance, tracking error and volatility, but their maximum drawdowns compared to the benchmark, is likely to be very different.

The measure of maximum percentage drawdown in period *(0, T)* can be computed as follows,

$$\mathrm{MAXDD}_{0,T} = \max_{\tau\in(0,T)} [1 - \frac{W(\tau)}{\max\limits_{t\in(0,\tau)} W(t)}]. \tag{3.11}$$

where $W(t)$ is the investment wealth at time t.

As a low maximum drawdown should be preferred because it indicates a minimal loss of investment. The maximum drawdown will be zero if an investment has never lost a penny. The worst maximum drawdown would be -100%, which means the investment is completely worthless.

4 Application of portfolio optimization models in Python

This chapter is a practical application where we apply the theoretical knowledge of portfolio optimization to real data from the financial markets. In chapters two and three, we introduced the basic functions and usage of Python, and describe several different asset portfolio allocation strategies. We select weekly data for thirty stocks listed on the NASDAQ composite index ranked by market capitalization over the past ten years, and use three different asset portfolio models for portfolio optimization, which are the naive strategy, minimum variance strategy and maximum Sharpe ratio strategy. The data are divided into two parts. There is the in-sample period (1/ 1/2012 to 26/12/2015) and the out-of-sample period (2/1/2016 to 26/12/2020). Next, we use these asset portfolio models to analyze the asset portfolios and calculate the Sharpe ratio, Roy's safety-first ratio for these models.

4.1 Data description

We chose thirty stocks that are listed on the NASDAQ composite index during the past ten years. These are the top 30 companies in the NASDAQ Composite Index ranked by market capitalization. The selected stock data were collected from 1/1/2011 to 26/12/2020 in the form of weekly data of the adjusted closing price of stocks on Yahoo Finance, so we have stock price data for 523 weeks. The stock price is shown in US dollars. The whole sample period is divided into in-sample period (1/1/2011-26/12/2015) and out -of-sample period (2/1/2016-26/12/2020). The list of stock’s name and tickers is shown in Table 4.1.

Table 4.1 List of stock's name and tickers

Name	Ticker	Name	Ticker
Apple Inc.	AAPL	Intel Corporation	INTC
Microsoft Corporation	MSFT	Adobe Inc.	ADBE
Alphabet Inc.	GOOGL	Qualcomm Incorporated	QCOM
Alphabet Inc.	GOOG	Texas Instruments Incorporated	TXN
Amazon.com, Inc.	AMZN	T-Mobile US, Inc.	TMUS
Tesla, Inc.	TSLA	Netflix, Inc.	NFLX
Nvidia Corporation	NVDA	Amgen Inc.	AMGN
ASML Holding N.V.	ASML	Sanofi	SNY
Broadcom Inc.	AVGO	Advanced Micro Devices, Inc.	AMD
Costco Wholesale Corporation	COST	Intuit Inc.	INTU
PepsiCo, Inc.	PEP	Applied Materials, Inc.	AMAT
Cisco Systems, Inc.	CSCO	Starbucks Corporation	SBUX
Comcast Corporation	CMCSA	Charter Communications, Inc.	CHTR
Verizon Communications Inc.	VZ	Automatic Data Processing, Inc.	ADP
AstraZeneca plc	AZN	Booking Holdings Inc.	BKNG

Source: https://finance.yahoo.com

The mean and standard deviation of return of chosen stocks (weekly) is shown in Table 4.2.

Table 4.2 The mean and standard deviation of return of chosen stocks (weekly)

	in-sample		out-of-sample	
Stocks	return	std	return	std
AAPL	0.30%	3.89%	0.65%	3.89%
MSFT	0.28%	3.28%	0.57%	3.28%
GOOGL	0.33%	3.67%	0.31%	3.67%
GOOG	0.32%	3.72%	0.32%	3.72%
NVDA	0.18%	4.99%	1.07%	4.99%
AMZN	0.45%	4.38%	0.60%	4.38%
TSLA	0.77%	6.62%	1.03%	6.62%
ASML	0.41%	4.27%	0.67%	4.27%
AVGO	0.61%	4.81%	0.49%	4.81%
COST	0.35%	2.28%	0.37%	2.28%
PEP	0.20%	1.76%	0.21%	1.76%
CSCO	0.11%	3.55%	0.25%	3.55%
CMCSA	0.37%	2.98%	0.27%	2.98%
VZ	0.18%	2.23%	0.18%	2.23%
AZN	0.21%	2.91%	0.22%	2.91%
INTC	0.23%	3.26%	0.19%	3.26%
ADBE	0.39%	3.22%	0.64%	3.22%
QCOM	-0.01%	3.50%	0.49%	3.50%
TXN	0.21%	3.23%	0.47%	3.23%
TMUS	0.24%	6.42%	0.47%	6.42%

NFLX	0.56%	8.75%	0.60%	8.75%
AMGN	0.41%	3.10%	0.19%	3.10%
SNY	0.16%	3.14%	0.13%	3.14%
AMD	-0.54%	7.45%	1.33%	7.45%
INTU	0.27%	2.92%	0.54%	2.92%
AMAT	0.12%	3.93%	0.61%	3.93%
SBUX	0.50%	3.08%	0.26%	3.08%
CHTR	0.57%	3.64%	0.49%	3.64%
ADP	0.29%	2.34%	0.32%	2.34%
BKNG	0.36%	4.22%	0.21%	4.22%

Source：own calculation

We chose the weekly adjusted closing prices of these stocks from 2011 to 2020, with a total of 523 weeks of data, and calculated the mean and standard deviation of their weekly returns. From the mean and standard deviation of the weekly returns, we can find that in- sample period, the mean returns of stock QCOM and AMD are negative and the mean return of stock AMD is the lowest in -0.54%. The mean return of stock TSLA is the highest for 0.77 %. The mean of the return of AVGO is the second highest return is 0.61%. For the in-sample interval, stock NFLX has the highest standard deviation of 8.75%. The higher the standard deviation, higher will be the fluctuations in the returns. The stock PEP has a minimum standard deviation of 1.76%.

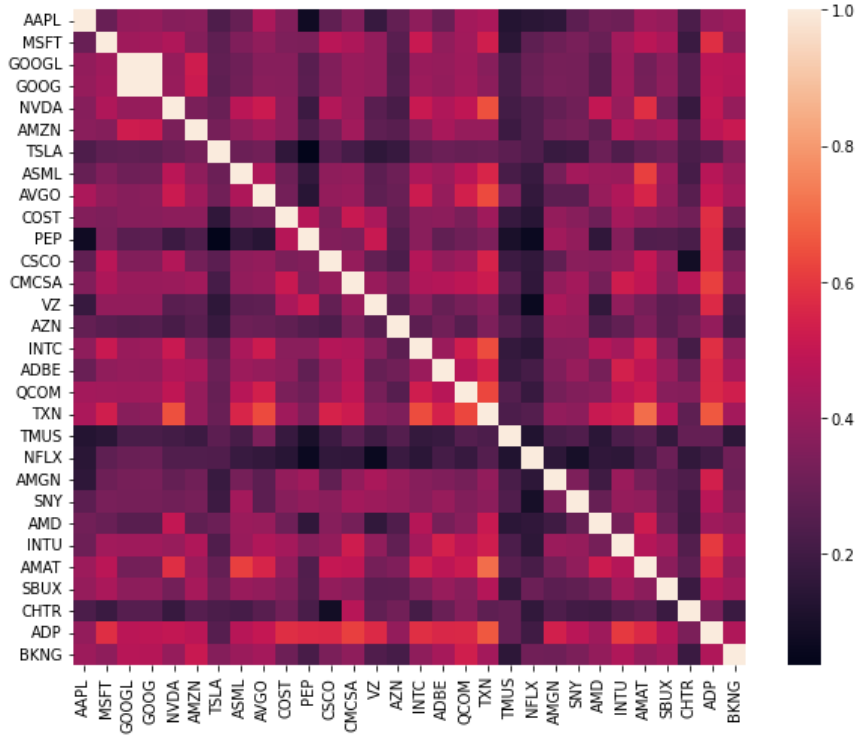
In the out-of-sample period, the lowest stock return is SNY at 0.13%, followed by stock VZ at 0.18%. The mean return of stock AMD is the highest for 1.33%. The mean of the return of NVDA is the second highest return is 1.07%. The returns of stocks AMD and NVDA increased considerably, with the mean return of AMD increasing from -0.54% in-sample to 1.33% out-of-sample. The stock SBUX had the largest decrease in returns, from 0.50% to 0.26%.

During the in-sample period, the stock with the largest standard deviation is NFLX, while the stock with the largest return is AVGO. During the out-of-sample period, the

stock with the largest standard deviation is NFLX, while the stock with the largest return is TSLA.

The heatmap of the correlation matrix for these 30 stocks in the in-sample period is shown in Figure 4.1. The correlation heatmap is a graph that visually displays the strength of the relationship between numerical variables. The correlation matrix is also summarized in Annex B.

Figure 4.1 Heatmap of correlation matrix



Source: own calculation

The correlation coefficient value takes the any value from -1 to 1. With a value of 1, it indicates there is a positive correlation between the two variables. It means that when one variable increases, the other variable also increases. With a value of -1, it indicates there is a negative correlation between the two variables. It means that when one variable increases, the other variable decreases. With a value of 0, two variables are not correlated with each other. It means the variables change in a random way between them.

The heatmap in Figure 4.1 shows that the stocks PEP&AAPL, PEP&TSLA, TMUS& NFLX, CSCO&CHTR have weak positive correlations. The stocks NVDA&TXN, AMAT&TXN, TXN&INTC have strong positive correlations.

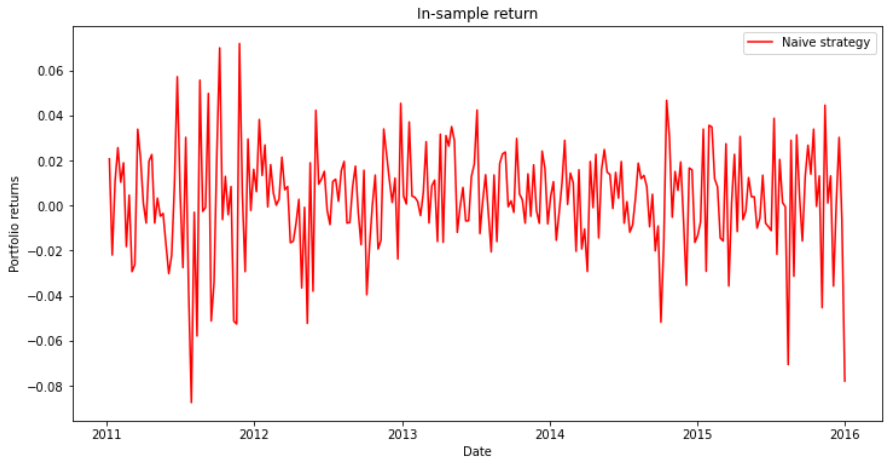
4.2 The Application of Python for portfolio optimization in the in-sample period

In this section, we use data from the in-sample period to test separately naive strategy, minimum variance strategy and maximum Sharpe ratio strategy.

4.2.1 Naive strategy

Naive strategy is when investors allocate the same amount of money in each stock so it refers if you have *n* stocks and one dollar, the investor should allocate one over *n* in every stock. We assume that the investor weighs each stock equally, with a weighting factor of $\frac{1}{30}$ for each asset in the portfolio. The in-sample period of return for naive strategy portfolio is shown in Figure 4.2.

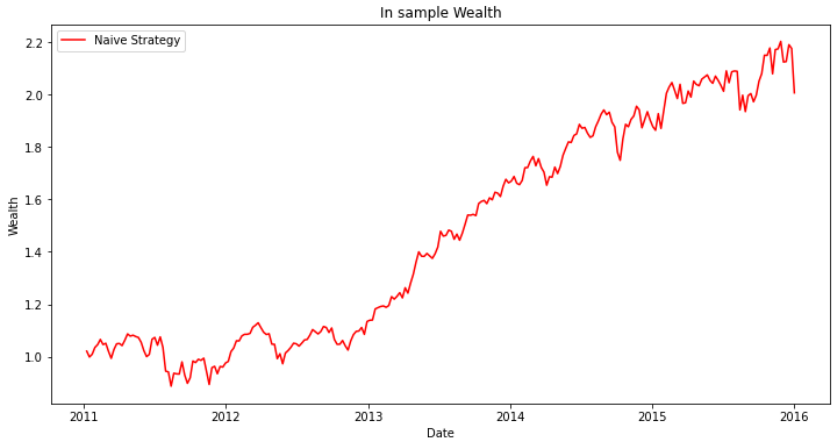
Figure 4. 2 The in-sample period of return for the naive strategy



Source: own calculation

The in-sample period of wealth for naive strategy is shown in Figure 4.3.

Figure 4.3 The in-sample period of wealth path for the naive strategy



Source: own calculation

According to Figure 4.3, the evolution of the wealth during the in-sample period under the naive strategy, the wealth of the portfolio is generally increase, increase from an initial wealth of $1 to $2.0073 at the end of the period. with the wealth reach the highest value in the sample reach $2.2038 on 21/11/2015. The in-sample period of results of the results of naive strategy is shown in Table 4.3.

Table 4.3 Results of naive strategy

Mean annualized return	15.31%
Annualized standard deviation	16.77%
Final wealth	2.0073
Sharpe ratio	0.7448
SF ratio	0.8396

Source: own calculation

We calculate two performance indicators in our naive strategy. They are the Sharpe Ratio and Roy's safety-first ratio. For the Sharpe ratio, we first calculate the portfolio's weekly return and standard deviation and annualize it. The portfolio's annual return is 15.31%, and the annual standard deviation is 16.76%. Based on the U.S. 10-year government bond rate, we get that the risk-free annual rate is 2.82%[1] and the Sharpe ratio

[1] https://finance.yahoo.com/bonds

under the naive strategy is 0.7448. We need to determine the minimum acceptable return (MAR) for Roy's safety-first ratio. According to the IMF, the inflation rate we know for the U.S. in 2020, and we set MAR at 1.23%[2]. Under a naive strategy, Roy's safe first ratio is 0.8396.

4.2.2 Minimum variance strategy

The minimum variance strategy refers to the allocating of a given budget among *n* financial assets that minimize the risk of the expected portfolio return. For minimum variance strategy, we solve the problem in Figure 3.2. The weights of the minimum variance portfolio are shown in Table 4.4.

Table 4.4 Weights of the minimum variance portfolio

AAPL	0.0553	INTC	0
MSFT	0	ADBE	0.0129
GOOGL	0	QCOM	0
GOOG	0	TXN	0
NVDA	0	TMUS	0.0003
AMZN	0	NFLX	0
TSLA	0.0081	AMGN	0
ASML	0	SNY	0
AVGO	0	AMD	0
COST	0.0871	INTU	0
PEP	0.5267	AMAT	0
CSCO	0	SBUX	0.0566
CMCSA	0	CHTR	0.0284
VZ	0.1268	ADP	0
AZN	0.0977	BKNG	0

Source: own calculation

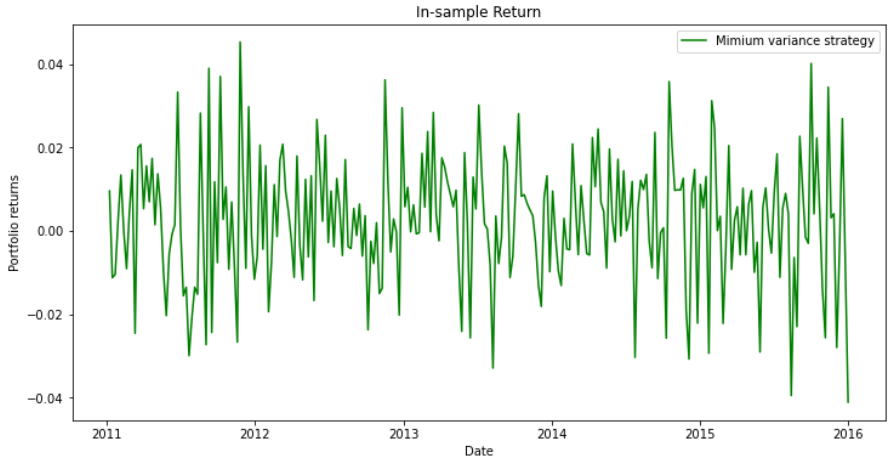
[2] https://www.imf.org/en/Countries/USA#countrydata

We can see in Table 4.4 the optimized weights of the minimum variance strategy during the in-sample period. The highest portfolio weight belongs to PEP (PepsiCo, Inc.) at 52.67%, which means that the investor should allocate half of his money to this stock. The second highest portfolio weight is the stock VZ (Verizon Communications Inc.) with 12.68%.

The idea behind calculating the optimized weights for the minimum variance strategy is that we first need to get the number of assets acquired and estimate the weight allocation for the different stocks. In the second step, we need to transform the mean weekly returns and standard deviation within the in-sample into annual mean returns and standard deviation and calculate the covariance within the in-sample period. We assume that each stock is initialized with equal weights, and for each stock's asset, we give the qualification that the weight is greater than or equal to 0 less than 1, that all the stocks' weights add up to 1, and that stocks are not allowed to be sold short.

The in-sample period of return for minimum variance strategy is shown in Figure 4.4.

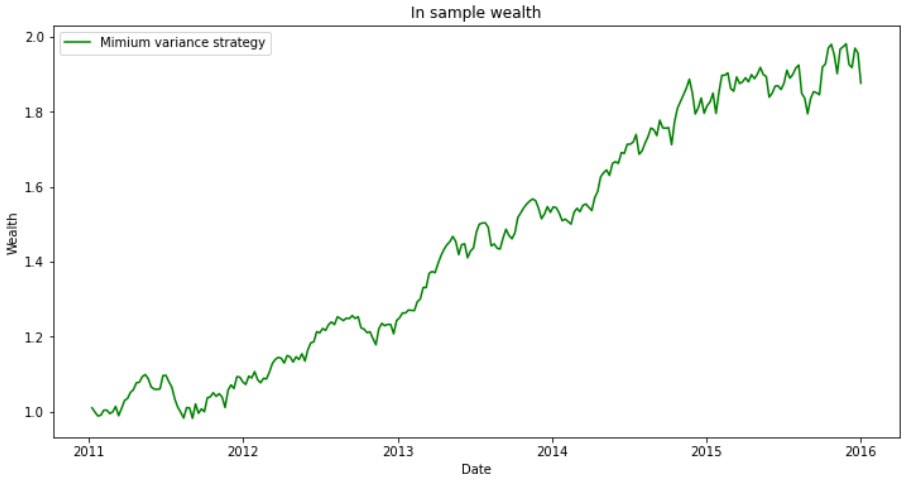
Figure 4. 4 The in-sample period of return for minimum variance strategy



Source：own calculation

The in-sample period of wealth for minimum variance strategy is shown in Figure 4.5.

Figure 4.5 The in-sample period of wealth for minimum variance strategy



Source：own calculation

According to Figure 4.5, the evolution of the wealth during the in-sample period under the minimum variance strategy, the wealth of the portfolio generally increases, increasing from an initial wealth of $1 to $1.8762 at the end of the period. The wealth reaches the highest value in the sample period, reaching $1.9811 on 21/11/2015.

The in-sample period of the results of the minimum variance strategy is shown in Table 4.5.

Table 4.5 Results of minimum variance strategy

Mean annualized return	13.16%
Annualized standard deviation	11.04%
Final wealth	1.8762
Sharpe ratio	0.9363
RoySF ratio	1.0803

Source: own calculation

We calculate two performance indicators in a minimum variance strategy. They are the Sharpe Ratio and Roy's safety-first Ratio. For the Sharpe ratio, we first calculate the portfolio's weekly return and standard deviation and annualize it. The annual return of the portfolio is 13.16%, and the annual standard deviation is 11.04%. Based on the U.S. 10-

year government bond rate, the annual risk-free rate is 2.82%, and the Sharpe ratio under the naive strategy is 0.9363. We need to determine the minimum acceptable return (MAR) for Roy's safety-first ratio. According to the IMF, the inflation rate we know for the U.S. in 2020 and set to MAR at 1.23%. Under a naive strategy, Roy's safe first ratio is 1.0803.

4.2.3 Maximum Sharpe ratio strategy

A higher Sharpe ratio essentially signifies a more risk-efficient portfolio. For maximum Sharpe ratio strategy, we solve the problem in Figure 3.3. The weights of the maximum Sharpe ratio portfolio are shown in Table 4.6.

Table 4.6 Weights of the maximum Sharpe ratio portfolio

AAPL	0	INTC	0
MSFT	0	ADBE	0
GOOGL	0	QCOM	0
GOOG	0	TXN	0
NVDA	0	TMUS	0
AMZN	0	NFLX	0
TSLA	0.057	AMGN	0.111
ASML	0	SNY	0
AVGO	0.0424	AMD	0
COST	0.1975	INTU	0
PEP	0.1286	AMAT	0
CSCO	0	SBUX	0.2426
CMCSA	0	CHTR	0.221
VZ	0	ADP	0
AZN	0	BKNG	0

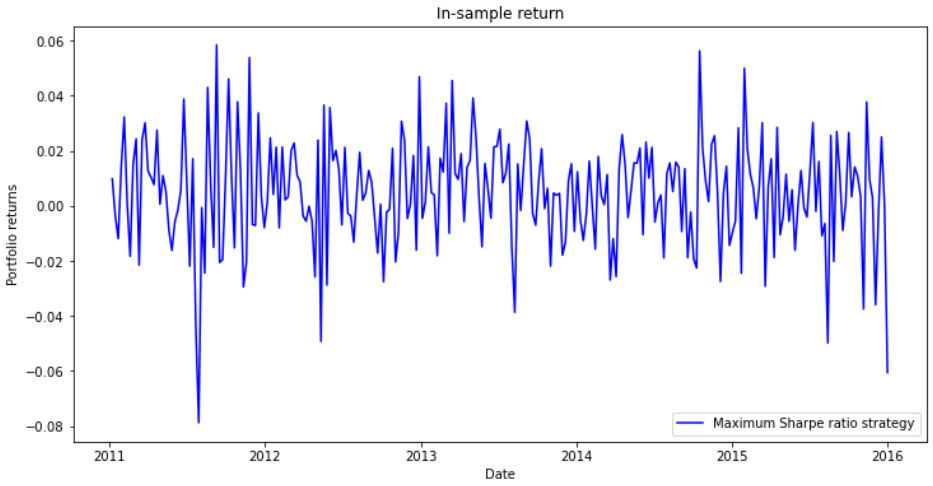
Source: own calculation

Table 4.6 illustrates the calculation of the optimal allocation weights for each stock in a portfolio by maximizing the Sharpe ratio. The highest portfolio weight is COST (Costco

Wholesale Corporation) at 19.75%, the second highest portfolio weight is the stock PEP (PepsiCo, Inc.) with 12.86%.

To calculate the optimal weights for maximizing Sharpe, the way to proceed is that we start by getting the number of assets acquired to calculate what the weight assignments should be for the different stocks. In the next step, we need to transform the weekly average returns and standard deviations within the sample into annual mean returns and standard deviations and calculate the covariance within the sample. We assume equal initialized weights for each stock, and for each stock's assets, the restrictions we give are that the weight is greater than or equal to 0 less than 1, that the weights of all stocks add up to 1, and that no short selling of stocks is allowed. The in-sample period of return for the maximum Sharpe ratio strategy is shown in Figure 4.6.

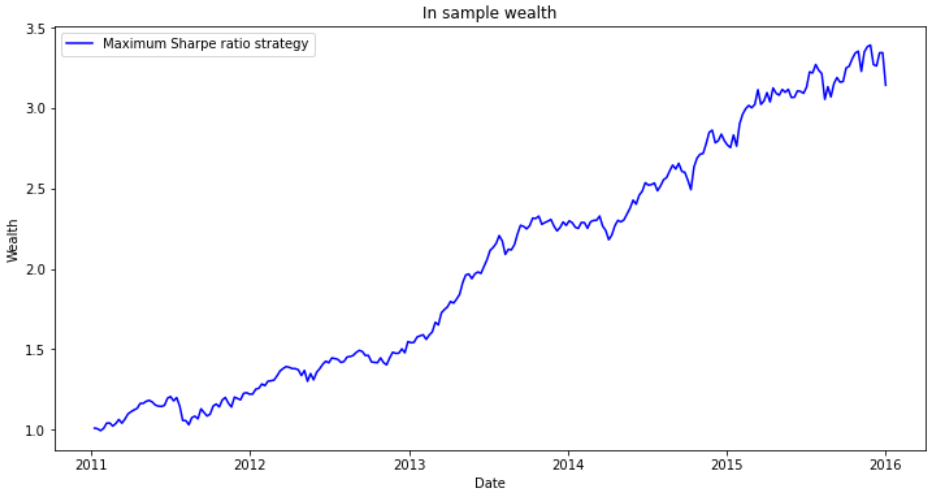
Figure 4.6 The in-sample period of return for the maximum Sharpe ratio strategy



Source：own calculation

The in-sample period of wealth for maximum Sharpe ratio strategy is shown in Figure 4.7.

Figure 4.7 The in-sample period of wealth for maximum Sharpe ratio strategy



Source：own calculation

According to Figure 4.7, the evolution of the wealth during the in-sample period under the maximum Sharpe ratio strategy, the wealth of the portfolio is generally increase, increase from an initial wealth of $1 to $3.1419 at the end of the period. with the wealth reach the highest value in the sample reach $3.3918 on 21/11/2015.

The in-sample period of results of the maximum Sharpe ratio strategy is shown in Table 4.7.

Table 4.7 Results of maximum Sharpe ratio strategy

Mean annualized return	23.88%
Annualized standard deviation	14.31%
Final wealth	3.1419
Sharpe ratio	1.4718
RoySF ratio	1.5829

Source: own calculation

Two performance metrics are calculated in our maximized Sharpe ratio strategy. They are the Sharpe Ratio and Roy's safety-first ratio. For the Sharpe ratio, we first calculate the portfolio's weekly return and standard deviation and annualize them. The portfolio had an annualized return of 23.88% and an annualized standard deviation of 14.31%.

Based on the U.S. 10-year government bond rate, we obtained an annual risk-free rate of 2.82% and a Sharpe ratio of 1.4718 under the maximum Sharpe ratio strategy. For Roy's safety-first ratio, we need to determine the minimum acceptable rate of return (MAR). According to the International Monetary Fund, we know the inflation rate for the U.S. in 2020, which we set to a MAR of 1.23%. Under a naive strategy, Roy's safe first ratio is 1.5829.

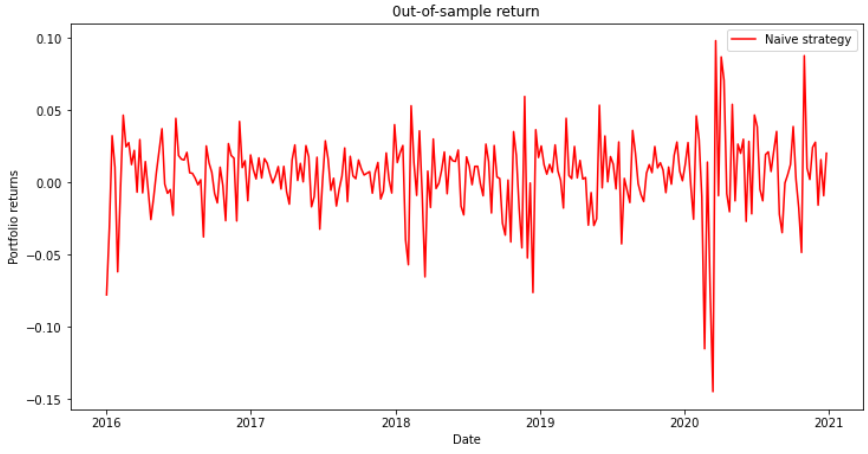
4.3 The Application of Python for portfolio performance measurement in the out-of-sample period

In this section, we use data from the out-of-sample period to test separately naive strategy, minimum variance strategy, and maximum Sharpe ratio strategy.

4.3.1 Naive strategy

We suppose that investors have equal weights for each stock and that each asset in the portfolio has a weighting factor of $\frac{1}{30}$. The out-of-sample of return for the naive strategy is shown in Figure 4.8.

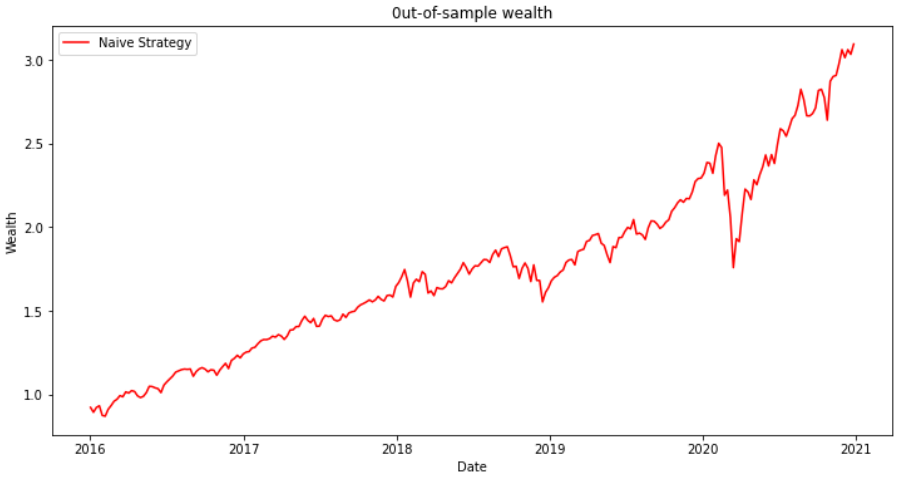
Figure 4.8 The out-of-sample period of return for the naive strategy



Source：own calculation

The out-of-sample period of wealth for the naive strategy is shown in Figure 4.9.

Figure 4.9 The out-of-sample period of wealth path for the naive strategy



Source: own calculation

According to Figure 4.9, the evolution of the wealth during the out-of-sample period under the naive strategy, the wealth of the portfolio generally increases. Still, from 24/10/2020-7/11/2020, wealth drops significantly. The increase from an initial wealth of $1 to $3.0945 at the end of the period. The wealth reaches the highest value out-of-sample of $3.0945 on 26/12/2020.

The out-of-sample period of results of the naive strategy is shown in Table 4.8.

Table 4.8 Results of naive strategy

Mean annualized return	24.55%
Annualized standard deviation	19.86%
Final wealth	3.0945
Sharpe ratio	1.0924
RoySF ratio	1.1743

Source: own calculation

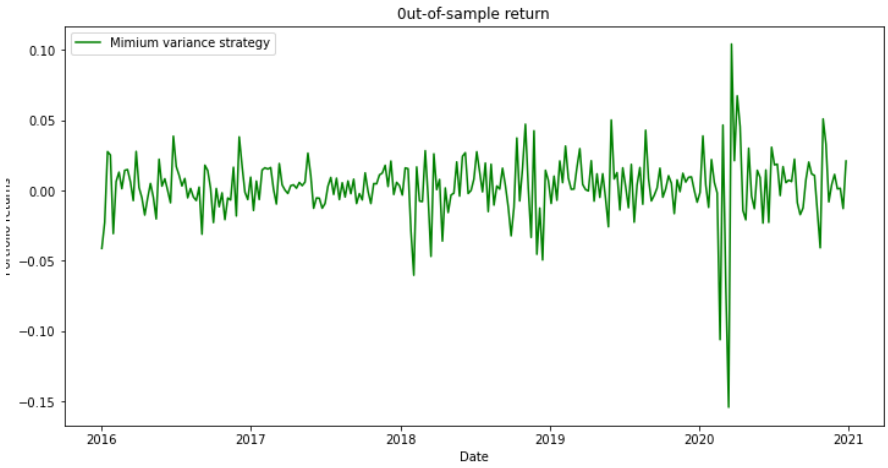
In the naive strategy, the mean annualized return is 24.55%, and the annualized standard deviation is 19.86 %. We obtain an annual risk-free rate of 2.82% and a Sharpe ratio of 1.0924 under a naive strategy based on the U.S. 10-year government bond rate. With Roy's safety-first ratio, the minimum acceptable rate of return (MAR) is what we

need to determine. Based on IMF data, we know the inflation rate for the U.S. in 2020, which we set to a MAR of 1.23%. Under the naive strategy, Roy's safe first ratio is 1.1743.

4.3.2 Minimum variance strategy

In this subchapter we analyze the out-of-sample performance of the minimum variance portfolio given in Table 4.4.

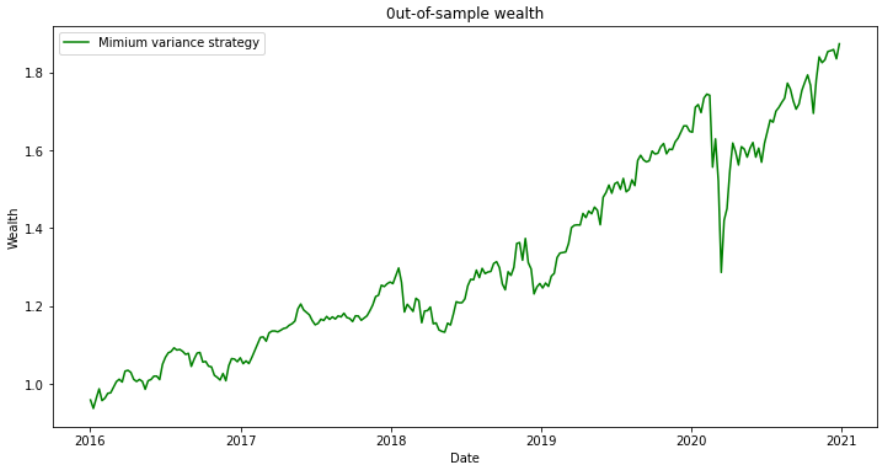
Figure 4.10 The out-of-ample period of return for minimum variance strategy



Source：own calculation

The out-of-sample period of wealth for the minimum variance strategy is shown in Figure 4.11.

Figure 4.11 The out-of-sample period of wealth for minimum variance strategy



Source：own calculation

According to Figure 4.11, the evolution of the wealth during the out-of-sample period under the minimum variance strategy generally increases the wealth of the portfolio. However, from 24/10/2020-7/11/2020, wealth drops significantly. The increase from an initial wealth of $1 to $1.837 at the end of the period. The wealth reaches its highest value $1.837 on 26/12/2020. The results of the minimum variance strategy in the out-of-sample period are shown in Table 4.9.

Table 4.9 Results of minimum variance strategy

Mean annualized return	13.88%
Annualized standard deviation	16.32%
Final wealth	1.8738
Sharpe ratio	0.6778
RoySF ratio	0.7752

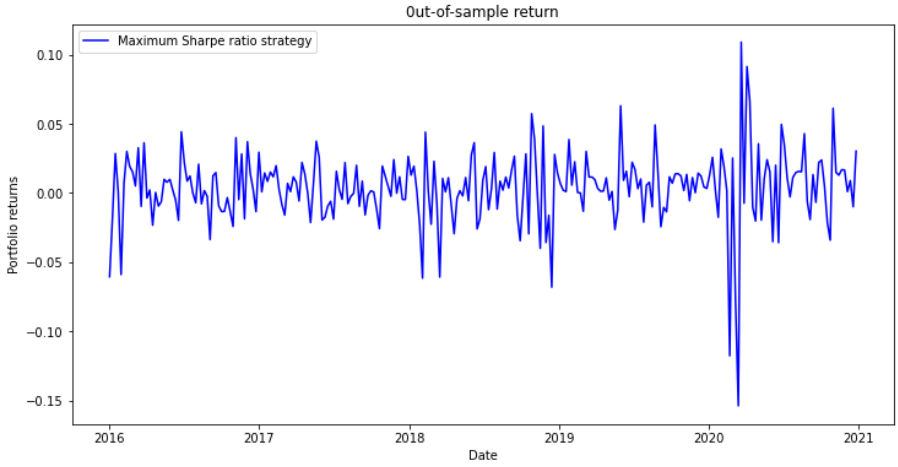
Source: own calculation

In the minimum variance strategy, the portfolio has an annualized return of 13.88% and an annualized standard deviation of 16.32%. Based on the U.S. 10-year Treasury rate, we obtain an annual risk-free rate of 2.82% and a Sharpe ratio of 0.6778 under the minimum variance strategy. For Roy's safety-first ratio, we need to determine the minimum acceptable rate of return (MAR). Based on IMF data, we know the inflation rate for the U.S. in 2020, which we set to a MAR of 1.23%. Under minimum variance strategy, Roy's safe first ratio is 0.7752.

4.3.3 Maximum Sharpe ratio strategy

In this subchapter we analyze the out-of-sample performance of the maximum Sharpe ratio portfolio given in Table 4.6. The out-of-sample period of return for maximum Sharpe ratio strategy is shown in Figure 4.12.

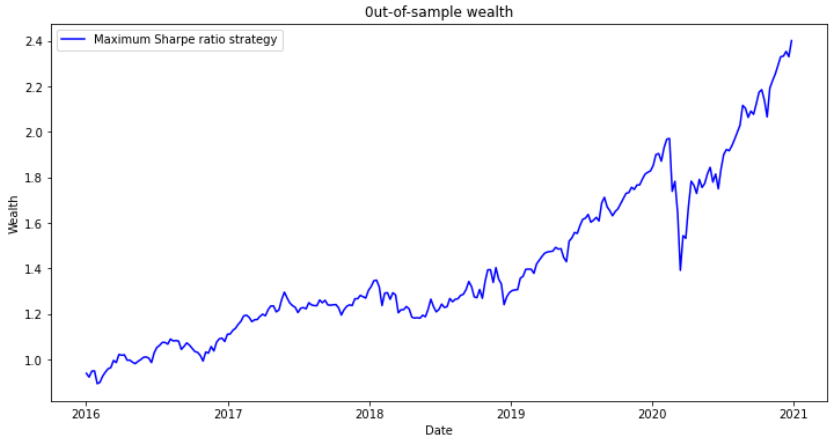
Figure 4.12 The out-of-sample period of return for maximum Sharpe ratio strategy



Source: own calculation

The out-of-sample period of wealth for maximum Sharpe ratio strategy is shown in Figure 4.13.

Figure 4.13 The out-of-sample period of wealth for maximum Sharpe ratio strategy



Source: own calculation

According to Figure 4.13, the evolution of the wealth during the out-of-sample period under the maximum Sharpe ratio strategy, the wealth of the portfolio generally increases, but from 24/10/2020-7/11/2020, there is a significant drop in wealth. The increase from an initial wealth of $1 to $2.4017 at the end of the period. The wealth reaches the highest value in the sample of $2.4017on 26/12/2020.

The results of the maximum Sharpe ratio strategy in the out-of-sample period are shown in Table 4.10.

Table 4.10 Results of maximum Sharpe ratio strategy

Mean annualized return	19.29%
Annualized standard deviation	18.89%
Final wealth	2.4017
Sharpe ratio	0.8720
RoySF ratio	0.9526

Source: own calculation

In the maximum Sharpe ratio strategy, the portfolio has an annualized return of 19.29% and an annualized standard deviation of 18.89%. Based on the U.S. 10-year government bond rate, we obtain an annualized risk-free rate of 2.82% and a Sharpe ratio of 0.8720% under a naive strategy. For Roy's safety-first ratio, we need to determine the minimum acceptable rate of return (MAR). According to the IMF, we know the inflation rate for 2020 in the U.S., which we set at 1.23% MAR. Under the maximum Sharpe ratio strategy, Roy's safe first ratio is 0.9526%.

4.4 Comparison of the results

In this part, we compare the results, including the performances in in-sample and out-of-sample and a comparison of the strategies based on out-of-sample performances.

4.4.1 Comparison of the performances in in-sample and out-of-sample

The whole sample period is divided into the in-sample period (1/1/2011-26/12/2015) and the out-of-sample period (2/1/2016-26/12/2020). According to naive strategy, minimum variance strategy and maximum Sharpe ratio strategy, we first calculate the portfolio's weekly return, standard deviation and annualize it. Next, we compute two performance indicators. They are the Sharpe ratio and Roy's Safety-First ratio.

Table 4.11 Performance of naive strategy in in-sample and out-of-sample

	in-sample	out-of-sample
Mean annualized return	15.31%	24.55%
Annualized standard deviation	16.77%	19.86%
Final wealth	2.0073	3.0945
Sharpe ratio	0.7448	1.0942
RoySF ratio	0.8396	1.1743

Source: own calculation

The performance of naive strategy in in-sample and out-of-sample is shown in Table 4.11. From the table, we can see that in the in-sample period, the portfolio's mean annualized return is 15.31%, and the annualized standard deviation is 16.76%. In the out-of-sample period, the portfolio's mean annualized return is 24.55%, and the annualized standard deviation is 18.86%. The performance of out-of-sample mean annualized return is better than in-sample, but the performance of annualized standard deviation is worse than in-sample, the Sharpe and Roy's safety-first ratio increases significantly in the out-of-sample period compared to the in-sample period. A High Sharpe Ratio means that the asset has a high ability to diversify and reduce unsystematic risk and has room for increasing returns. Roy's safety-first ratio is very similar to the Sharpe ratio, and the best decision for investors is to select a portfolio with the highest Roy's safety-first ratio.

Table 4.12 Performance of minimum variance strategy in in-sample and out-of-sample

	in-sample	out-of-sample
Mean annualized return	13.16%	13.88%
Annualized standard deviation	11.04%	16.32%
Final wealth	1.8762	1.8738
Sharpe ratio	0.9363	0.6778
RoySF ratio	1.0803	0.7752

Source: own calculation

The performance of the minimum variance strategy in in-sample and out-of-sample is shown in Table 4.12. From the table, it can be seen that the minimum variance strategy in the in-sample period, the portfolio's mean annualized return is 13.16%, and the annualized standard deviation is 11.04 %. In the out-of-sample period, the portfolio's mean annualized return is 13.88%, and the annualized standard deviation is 16.32%. We can find that the performance of out-of-sample mean annualized return is better than in-sample, but the annualized standard deviation is worse than in-sample. In the minimum variance strategy, the Sharpe ratio and Roy's safety-first ratio of the in-sample portfolios perform better than the out-of-sample ones, which means that the performance of the minimum variance strategy worsen in the out-of-sample period.

Table 4.13 Performance of max Sharpe ratio strategy in in-sample and out-of-sample

	in-sample	out-of-sample
Mean annualized return	23.88%	19.29%
Annualized standard deviation	14.31%	18.89%
Final wealth	3.1419	2.4017
Sharpe ratio	1.4718	0.8720
RoySF ratio	1.5829	0.9562

Source：own calculation

The performance of the maximum Sharpe ratio strategy in in-sample and out-of-sample is shown in Table 4.13. From the table, it can be seen that in the maximum Sharpe ratio strategy in the in-sample period, the portfolio's mean annualized return is 23.88%, and the annualized standard deviation is 14.31 %. In the out-of-sample period, the portfolio's mean annualized return is 19.29%, and the annualized standard deviation is 18.89%. We can find that the performance of out-of-sample annualized standard deviation and mean annualized return are worse than in-sample. In the maximum Sharpe ratio strategy, the Sharpe ratio and Roy's safety-first ratio of the in-sample period perform better than the out-of-sample ones, which means that the performance of the maximum Sharpe ratio strategy worsen in the out-of-sample period.

4.4.2 Comparison of the strategies based on out-of-sample performances

The performance of three strategies in out-of-sample period is shown in Table 4.14.

Table 4.14 Performance of three strategies in out-of-sample

	Naive strategy	Minimum variance strategy	Maximum Sharpe ratio strategy
Mean annualized return	24.55%	13.88%	19.29%
Annualized standard deviation	19.86%	16.32%	18.89%
Final wealth	3.0945	1.8738	2.4017
Sharpe ratio	1.0942	0.6778	0.8720
RoySF ratio	1.1743	0.7752	0.9562

Source：own calculation

From Table 4.14, we can see that the naive strategy has the highest Sharpe ratio and Roy's safety-first ratio, and the second highest ranking is the maximum Sharpe ratio strategy. Moreover, the naive strategy has the highest mean annualized return and final wealth. Although the annualized standard deviation of the naive strategy is the highest, it is not much higher compared to the other strategies. So, if we should choose from these three strategies, the naive strategy is the best.

5 Conclusion

Portfolio optimization determines the optimal combination of weights associated with the financial assets held in the portfolio. This thesis uses Python software to analyze portfolio optimization based on the naive strategy, the minimum variance strategy, and the maximum Sharpe ratio strategy. We selected thirty stocks in the NASDAQ Composite Index in the last ten years. The entire sample period is divided into an in-sample period (1/1/2011-26/12/2015) and an out-of-sample period (2/1/2016-26/12/2020), and the best portfolio is selected by evaluating the performance of each portfolio over the different periods.

For the naive strategy, in the in-sample period, the portfolio's mean annualized return is 15.31%, and the annualized standard deviation is 16.76%. In the out-of-sample period, the portfolio's mean annualized return is 24.55%, and the annualized standard deviation is 19.86%. We can find that the performance of out-of-sample mean annualized return is better than in-sample, but the performance of annualized standard deviation is worse than in-sample. The Sharpe ratio and Roy's safety-first ratio also increases significantly in the out-of-sample period compared to the in-sample period.

For the minimum variance strategy and maximum Sharpe ratio strategy, the Sharpe ratio and Roy’s safety-first ratio of the in-sample period perform better than the out-of-sample ones, which means that the performance in the out-of-sample period is worse than in the in-sample period. This is caused mainly by higher standard deviation of returns in the out-of-sample period and in case of maximum Sharpe ratio strategy also by the lower mean annualized returns in the out-of-sample period compared to the in-sample period.

Among the three strategies, the naive strategy has the highest Sharpe ratio and Roy's safety-first ratio in the out-of-sample period, and the second-ranked strategy is the maximum Sharpe ratio strategy. Furthermore, the naive strategy has the best performance in mean annualized return and final wealth. Although the annualized standard deviation of the naive strategy is the highest, it is not much higher compared to the other strategies. The higher Sharpe ratio indicates a higher reward per unit of risk and the more desirable the investment vehicle is. The optimal decision for investors is to choose the investment portfolio with the highest Sharpe ratio and Roy's safety-first ratio. Considering all performance metrics, as in our 30 stocks, the naive strategy is sound and arguably the best, and asset allocation through this model is a good choice.

Bibliography

[1] BRUGIÈRE, Pierre. *Quantitative portfolio management: with applications in Python*. Cham, Switzerland: Springer, 2020. ISBN 978-3-030-37739-7.

[2] DEMIGUEL, Victor, GARLAPPI, Lorenzo and Raman UPPAL. Optimal Versus Naive Diversification: How Inefficient is the 1/N portfolio strategy? *Review of Financial Studies*. 2009, 22(5), pp. 1915–1953.

[3] HILPISCH, Yves J. *Python for finance: mastering data-driven finance*. Second edition. Sebastopol, CA: O'Reilly, 2018. ISBN 978-1-492-02433-0.

[4] KRESTA, Aleš. *Financial Engineering in Matlab: Selected Approaches and Algorithms*. Ostrava: VŠB-TU Ostrava, 2015. ISBN 978-80-248-3702-4.

[5] REILLY, Frank K., Keith C. BROWN and Sanford J. LEEDS. *Investment analysis and portfolio management*. 11th ed. Boston: Cengage, 2019. ISBN 978-1-305-26299-7. Studies. 2009, 22(5), pp. 1915–1953.

[6] ZMEŠKAL, Z., D. DLUHOŠOVÁ and T. TICHÝ. *Financial Models*. Ostrava: VŠB-TU Ostrava, 2004. ISBN 80-248-0754-8

Electronic documents and others

[7] ANALYSPREP. *CFA Level 1, 2 & 3 Question Bank | FRM part 1 & 2 QBank* [online]. 2014-2022 [21.6.2022]. Available on: https://analystprep.com/

[8] DISFOLD. *Top companies in the NASDAQ Composite idex ranked by market capitalization* [online]. 2022 [20.6.2022]. Available on: https://disfold.com/stock-index/nasdaq-composite/companies/

[9] GEEKSFORGEEKS. *History of Python – GreeksforGreeks* [online]. 2022 [11.7.2022]. Available on: https://www.geeksforgeeks.org/history-of-python/

[10] INTERNATIONAL MONETARY FUND. *United States and the IMF* [online]. 2022 [21.6.2022]. Available on: https://www.imf.org/en/Countries/USA#countrydata/

[11] PROJECT JUPYTER. *JupyterLab Documentation – JupyterLab 3.4.3 documentation* [online]. 2018 [12.4.2022]. Available on: https://jupyterlab.readthedocs.io/en/stable/

[12] WANG, Anlan. *Verification on the Performance of Classical and Modern Portfolio Optimization Models*. Ostrava, 2022. VŠB-TUO, Faculty of Economics. Doctoral thesis.

[13] YAHOO. *Nasdaq (^IXIC) Charts, Data & News – Yahoo Finance* [online]. 2022 [20.6.2022]. Available on: https://finance.yahoo.com/

List of Abbreviations

Abbreviations

CAGR	Compound Annual Growth Rate
CML	Capital market line
MAR	Minimum acceptable rate of return
MDD	Maximum downturn
MPT	Modern portfolio theory
SFRatio	Roy's safety-first ratio
Std	Standard deviation

Variables

P_t	Asset’s price
$R_{P,t}$	Rate of return
$R_{P,t}$	Return of the portfolio
R_f	Risk-free rate
R_t	Return
$w_{i,t}$	Weights of assets in the portfolio
w_i	Weights of each stock
x_F	Share of the riskless asset in the portfolio
x_i	The weight to invest in asset i in a portfolio
σ_p	Standard deviation
σ_p^2	Variance of the portfolio return
$E(R)$	Vector of expected returns
Q	Covariance matrix
r	Ex-post observed returns
w	Vector of weights

List of Annexes

Annex A:
Python programs

0. Import packages

```
# Import packages
import warnings
warnings.filterwarnings('ignore')

import datetime as dt
import numpy as np
import pandas as pd

import scipy.optimize as sco

import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Read original price data
df_price = pd.read_csv('stock.csv', parse_dates=True)

print('Asset Prices (Whole Period):')
print(df_price.head())
```

```
# Data Preprocessing
# Set Date column as index
df_price.set_index('Date', inplace=True)

# Change index type
df_price.index = pd.to_datetime(df_price.index)
```

```
# Split data into two samples: in-sample and out-of-sample
split_date = '2016-01-02'

# In-sample data
df_rtn_in_sample = df_rtn.loc[:split_date]

# Out-of-sample data
df_rtn_out_of_sample = df_rtn.loc[split_date:]
```

```
# Explore in-sample data
# Mean return
print('Mean return of assets(in-sample):')
df_rtn_in_sample.mean()
```

```
# Standard deviation
print('Standard deviation of assets(in-sample):')
df_rtn_in_sample.std()
```

```
# Explore out-of-sample data
# Mean return
print('Mean return of assets(out-of-sample):')
df_rtn_out_of_sample.mean()
```

```
print('Standard deviation of assets(out-of-sample):')
df_rtn_out_of_sample.std()
```

```
# Correlation matrix between assets
df_rtn_in_sample.corr()
```

```
# Correlation heatmap between assets
plt.figure(num=1, figsize=(10,8))
sns.heatmap(df_rtn_in_sample.corr())
```

3. Optimiza weights

Define some functions for optimization

```
def portfolio(weights):
    """
        Compute portfolio return, standard deviation, and sharpe ratio
    """
    # Use global variables
    global stk_returns, stk_cov

    weights = np.array(weights)
    # Compute portfolio return
    port_return = np.sum(stk_returns * weights)
    # Compute portfolio standard deviation
    port_standard_deviation = np.sqrt(np.dot(weights, np.dot(stk_cov, weights)))
    # Compute portfolio SR, assuming that Rf = 0.0282
    port_sharp_ratio = port_return / port_standard_deviation

    return np.array([port_return, port_standard_deviation, port_sharp_ratio])


def max_func_sharp(weights):
    """
        Use SR as objective function
    """
    # Use global variables
    global stk_returns, stk_cov
    # Return sharpe ratio
    return -portfolio(weights)[2]


def min_func_variance(weights):
    """
        Use variance as objective function
    """
    # Return volatility
    return portfolio(weights)[1]
```

The main optimization process goes here

```
# Number of assets
N = len(df_rtn_in_sample.columns)

# Asset weights for different portfolios
port_names = ['naive_strategy', 'minimum_variance_strategy', 'max_Sharpe_ratio_strategy']

# Initialize portfolio weights
df_weights = pd.DataFrame(data=None, index=port_names, columns=df_rtn.columns)

# Asset mean returns: annualized
stk_returns = 52 * df_rtn_in_sample.mean()

# Covariance between assets
stk_cov = 52 * df_rtn_in_sample.cov()

# Weights allocation
# Initialize weights
for idx in df_weights.index:
    df_weights.loc[idx, :] = np.array(N * [1.0 / N, ])

# Bounds: no short-sale allowed
bnds = tuple((0, 1) for _ in range(N))

# Constraints: sum of weights must be equal to 1
cons = ({"type": "eq", "fun": lambda x: np.sum(x) - 1})

# Optimize for minimum variance strategy
opt_res_min_var = sco.minimize(min_func_variance,
                               df_weights.loc['minimum_variance_strategy',:],
                               method="SLSQP",
                               bounds=bnds,
                               constraints=cons)

df_weights.loc['minimum_variance_strategy', :] = opt_res_min_var["x"].round(4)

# Optimize for max Sharpe ratio strategy
opt_res_max_sr = sco.minimize(max_func_sharp,
                               df_weights.loc['max_Sharpe_ratio_strategy',:],
                               method="SLSQP",
                               bounds=bnds,
                               constraints=cons)

df_weights.loc['max_Sharpe_ratio_strategy', :] = opt_res_max_sr["x"].round(4)

# Final weights
print('Weights for 12 portfolios:')
print(df_weights.T)
```

4. Performance evaluation

```
# Portfolio returns over in-sample period
df_port_rtn = pd.DataFrame(index=df_rtn_in_sample.index, columns=df_weights.index, data=None)

for d in df_port_rtn.index:
    for p in df_port_rtn.columns:
        df_port_rtn.loc[d, p] = np.dot(df_rtn_in_sample.loc[d, :], df_weights.loc[p,:])

# Cumulative return
df_port_cum = (1+df_port_rtn).cumprod()
```

```
# Performance evaluation over in-sample period
df_perf = pd.DataFrame(index=['Mean annualized return', 'Annualized standard deviation', 'Final wealth', 'Sharpe Ratio', 'RoySF ratio'], columns=df_weights

# Mean Annualized Return
df_perf.loc['Mean annualized return', :] = 52 * df_port_rtn.mean()

# Annualized Volatility
df_perf.loc['Annualized standard deviation', :] = np.sqrt(52) * df_port_rtn.std()

# Cumulative Return
df_perf.loc['Final wealth', :] = df_port_cum.iloc[-1, :]

# Sharpe ratio
Rf = 0.0282
df_perf.loc['Sharpe Ratio', :] = (df_perf.loc['Mean annualized return', :] - Rf) / df_perf.loc['Annualized standard deviation', :]

# SF Ratio
RI = 0.0123
df_perf.loc['RoySF ratio', :] = (df_perf.loc['Mean annualized return', :] - RI) / df_perf.loc['Annualized standard deviation', :]

df_perf
```

```
# Print Weekly returns
print(df_port_rtn.loc[:, 'naive_strategy'], 'r')
print(df_port_rtn.loc[:, 'minimum_variance_strategy'], 'g')
print(df_port_rtn.loc[:, 'max_Sharpe_ratio_strategy'], 'b')
df_port_rtn.to_excel('port_rtn(1).xlsx')
```

```
# Print cumulative returns in-sample
print(df_port_cum.loc[:, 'naive_strategy'], 'r')
print(df_port_cum.loc[:, 'minimum_variance_strategy'], 'g')
print(df_port_cum.loc[:, 'max_Sharpe_ratio_strategy'], 'b')
df_port_cum.to_excel('port_cum(1).xlsx')
```

```
# Plot Weekly returns-Naive strategy
plt.figure(num=3, figsize=(12,6))
plt.plot(df_port_rtn.loc[:, 'naive_strategy'], 'r')
plt.xlabel('Date')
plt.ylabel('Portfolio returns')
plt.title('In-sample return')
plt.legend(['Naive strategy'])
plt.show()
# Plot Weekly returns-Mimium variance strategy
plt.figure(num=3, figsize=(12,6))
plt.plot(df_port_rtn.loc[:, 'minimum_variance_strategy'], 'g')
plt.xlabel('Date')
plt.ylabel('Portfolio returns')
plt.title('In-sample return')
plt.legend(['Mimium variance strategy'])
plt.show()
# Plot Weekly returns-Maximum Sharpe ratio strategy
plt.figure(num=3, figsize=(12,6))
plt.plot(df_port_rtn.loc[:, 'max_Sharpe_ratio_strategy'], 'b')
plt.xlabel('Date')
plt.ylabel('Portfolio returns')
plt.title('In-sample return')
plt.legend(['Maximum Sharpe ratio strategy'])
plt.show()
# Plot Weekly returns
plt.figure(num=3, figsize=(12,6))
plt.plot(df_port_rtn.loc[:, 'naive_strategy'], 'r')
plt.plot(df_port_rtn.loc[:, 'minimum_variance_strategy'], 'g')
plt.plot(df_port_rtn.loc[:, 'max_Sharpe_ratio_strategy'], 'b')
plt.xlabel('Date')
plt.ylabel('Portfolio returns')
plt.title('Portfolio returns over in-sample period')
plt.legend(['Naive strategy', 'Mimium variance strategy', 'Maximum Sharpe ratio strategy'])
plt.show()
```

```
# Plot comulative return
plt.figure(num=4, figsize=(12,6))
plt.plot(df_port_cum.loc[:, 'naive_strategy'], 'r')
plt.xlabel('Date')
plt.ylabel('Wealth')
plt.title('In-sample wealth')
plt.legend(['Naive Strategy'])
plt.show()
plt.figure(num=4, figsize=(12,6))
plt.plot(df_port_cum.loc[:, 'minimum_variance_strategy'], 'g')
plt.xlabel('Date')
plt.ylabel('Wealth')
plt.title('In sample wealth')
plt.legend(['Mimium variance strategy'])
plt.show()
plt.figure(num=4, figsize=(12,6))
plt.plot(df_port_cum.loc[:, 'max_Sharpe_ratio_strategy'], 'b')
plt.xlabel('Date')
plt.ylabel('Wealth')
plt.title('In sample wealth')
plt.legend(['Maximum Sharpe ratio strategy'])
plt.show()
plt.figure(num=4, figsize=(12,6))
plt.plot(df_port_cum.loc[:, 'naive_strategy'], 'r')
plt.plot(df_port_cum.loc[:, 'minimum_variance_strategy'], 'g')
plt.plot(df_port_cum.loc[:, 'max_Sharpe_ratio_strategy'], 'b')
plt.xlabel('Date')
plt.ylabel('Wealth')
plt.title('Wealth over in-sample period')
plt.legend(['Naive strategy', 'Mimium variance strategy', 'Maximum Sharpe ratio strategy'])
plt.show()
```

```
# Portfolio returns over out-of-sample period
df_port_rtn = pd.DataFrame(index=df_rtn_out_of_sample.index, columns=df_weights.index, data=None)

for d in df_port_rtn.index:
    for p in df_port_rtn.columns:
        df_port_rtn.loc[d, p] = np.dot(df_rtn_out_of_sample.loc[d, :], df_weights.loc[p,:])

# Cumulative return
df_port_cum = (1+df_port_rtn).cumprod()
```

```
# Plot Weekly returns-Naive strategy
plt.figure(num=3, figsize=(12,6))
plt.plot(df_port_rtn.loc[:, 'naive_strategy'], 'r')
plt.xlabel('Date')
plt.ylabel('Portfolio returns')
plt.title('Out-of-sample return')
plt.legend(['Naive strategy'])
plt.show()
# Plot Weekly returns-Mimium variance strategy
plt.figure(num=3, figsize=(12,6))
plt.plot(df_port_rtn.loc[:, 'minimum_variance_strategy'], 'g')
plt.xlabel('Date')
plt.ylabel('Portfolio returns')
plt.title('Out-of-sample return')
plt.legend(['Mimium variance strategy'])
plt.show()
# Plot Weekly returns-Maximum Sharpe ratio strategy
plt.figure(num=3, figsize=(12,6))
plt.plot(df_port_rtn.loc[:, 'max_Sharpe_ratio_strategy'], 'b')
plt.xlabel('Date')
plt.ylabel('Portfolio returns')
plt.title('Out-of-sample return')
plt.legend(['Maximum Sharpe ratio strategy'])
plt.show()
# Plot Weekly returns
plt.figure(num=3, figsize=(12,6))
plt.plot(df_port_rtn.loc[:, 'naive_strategy'], 'r')
plt.plot(df_port_rtn.loc[:, 'minimum_variance_strategy'], 'g')
plt.plot(df_port_rtn.loc[:, 'max_Sharpe_ratio_strategy'], 'b')
plt.xlabel('Date')
plt.ylabel('Portfolio returns')
plt.title('Portfolio returns over out-of-sample period')
plt.legend(['Naive strategy', 'Mimium variance strategy', 'Maximum Sharpe ratio strategy'])
plt.show()
```

```
# Plot comulative return
plt.figure(num=4, figsize=(12,6))
plt.plot(df_port_cum.loc[:, 'naive_strategy'], 'r')
plt.xlabel('Date')
plt.ylabel('Wealth')
plt.title('Out-of-sample wealth')
plt.legend(['Naive Strategy'])
plt.show()
plt.figure(num=4, figsize=(12,6))
plt.plot(df_port_cum.loc[:, 'minimum_variance_strategy'], 'g')
plt.xlabel('Date')
plt.ylabel('Wealth')
plt.title('Out-of-sample wealth')
plt.legend(['Mimium variance strategy'])
plt.show()
plt.figure(num=4, figsize=(12,6))
plt.plot(df_port_cum.loc[:, 'max_Sharpe_ratio_strategy'], 'b')
plt.xlabel('Date')
plt.ylabel('Wealth')
plt.title('Out-of-sample wealth')
plt.legend(['Maximum Sharpe ratio strategy'])
plt.show()
plt.figure(num=4, figsize=(12,6))
plt.plot(df_port_cum.loc[:, 'naive_strategy'], 'r')
plt.plot(df_port_cum.loc[:, 'minimum_variance_strategy'], 'g')
plt.plot(df_port_cum.loc[:, 'max_Sharpe_ratio_strategy'], 'b')
plt.xlabel('Date')
plt.ylabel('Wealth')
plt.title('Wealth over out-of-sample period')
plt.legend(['Naive strategy', 'Mimium variance strategy', 'Maximum Sharpe ratio strategy'])
plt.show()
```

```
# Performance evaluation over out-of-sample period
df_perf = pd.DataFrame(index=['Mean annualized return', 'Annualized standard deviation', 'Final wealth', 'Sharpe Ratio', 'RoySF ratio'], columns=df_weights

# Mean Annualized Return
df_perf.loc['Mean annualized return', :] = 52 * df_port_rtn.mean()

# Annualized Volatility
df_perf.loc['Annualized standard deviation', :] = np.sqrt(52) * df_port_rtn.std()

# Cumulative Return
df_perf.loc['Final wealth', :] = df_port_cum.iloc[-1, :]

# Sharpe ratio
Rf = 0.0282
df_perf.loc['Sharpe Ratio', :] = (df_perf.loc['Mean annualized return', :] - Rf) / df_perf.loc['Annualized standard deviation', :]

# SF Ratio
RI = 0.0123
df_perf.loc['RoySF ratio', :] = (df_perf.loc['Mean annualized return', :] - RI) / df_perf.loc['Annualized standard deviation', :]

df_perf
```

Annex B:

Correlation matrix between assets

	AAPL	MSFT	GOOGL	GOOG	NVDA	AMZN	TSLA	ASML	AVGO	COST	PEP	CSCO	CMCSA	VZ	AZN	INTC	ADBE	QCOM	TXN	TMUS	NFLX	AMGN	SNY	AMD	INTU	AMAT	SBUX	CHTR	ADP	BKNG
AAPL	1,00	0,29	0,39	0,40	0,36	0,37	0,23	0,29	0,45	0,35	0,07	0,28	0,35	0,18	0,28	0,38	0,30	0,42	0,45	0,13	0,14	0,15	0,27	0,32	0,31	0,41	0,39	0,22	0,39	0,41
MSFT	0,29	1,00	0,42	0,43	0,45	0,36	0,27	0,35	0,39	0,34	0,34	0,47	0,45	0,39	0,26	0,51	0,38	0,42	0,53	0,14	0,27	0,31	0,33	0,29	0,42	0,48	0,44	0,19	0,58	0,38
GOOGL	0,39	0,42	1,00	1,00	0,40	0,52	0,28	0,31	0,36	0,36	0,26	0,35	0,40	0,39	0,25	0,40	0,40	0,42	0,36	0,22	0,30	0,33	0,33	0,25	0,41	0,32	0,38	0,25	0,48	0,47
GOOG	0,40	0,43	1,00	1,00	0,40	0,52	0,27	0,31	0,37	0,36	0,26	0,35	0,40	0,39	0,25	0,41	0,39	0,42	0,38	0,22	0,30	0,33	0,32	0,26	0,42	0,32	0,38	0,25	0,49	0,47
NVDA	0,36	0,45	0,40	0,40	1,00	0,33	0,30	0,48	0,52	0,38	0,19	0,46	0,41	0,26	0,22	0,51	0,45	0,49	0,65	0,21	0,24	0,29	0,31	0,50	0,40	0,58	0,32	0,18	0,50	0,39
AMZN	0,37	0,36	0,52	0,52	0,33	1,00	0,32	0,38	0,42	0,38	0,23	0,32	0,42	0,27	0,26	0,36	0,44	0,40	0,40	0,19	0,24	0,31	0,33	0,27	0,45	0,41	0,43	0,25	0,48	0,51
TSLA	0,23	0,27	0,28	0,27	0,30	0,32	1,00	0,30	0,32	0,16	0,04	0,27	0,21	0,16	0,18	0,27	0,30	0,29	0,30	0,27	0,24	0,18	0,19	0,30	0,24	0,29	0,31	0,23	0,25	0,36
ASML	0,29	0,35	0,31	0,31	0,48	0,38	0,30	1,00	0,45	0,31	0,17	0,38	0,39	0,27	0,30	0,44	0,41	0,47	0,55	0,22	0,18	0,32	0,43	0,41	0,40	0,61	0,40	0,22	0,47	0,41
AVGO	0,45	0,39	0,36	0,37	0,52	0,42	0,32	0,45	1,00	0,32	0,14	0,39	0,40	0,27	0,30	0,52	0,40	0,54	0,64	0,34	0,16	0,27	0,27	0,40	0,45	0,55	0,39	0,25	0,50	0,43
COST	0,35	0,34	0,36	0,36	0,38	0,38	0,16	0,31	0,32	1,00	0,47	0,34	0,51	0,44	0,27	0,37	0,38	0,34	0,42	0,18	0,14	0,39	0,36	0,31	0,43	0,39	0,35	0,32	0,58	0,31
PEP	0,07	0,34	0,26	0,26	0,19	0,23	0,04	0,17	0,14	0,47	1,00	0,35	0,34	0,51	0,25	0,37	0,28	0,31	0,34	0,10	0,06	0,42	0,39	0,16	0,36	0,24	0,24	0,22	0,56	0,22
CSCO	0,28	0,47	0,35	0,35	0,46	0,32	0,27	0,38	0,39	0,34	0,35	1,00	0,40	0,28	0,23	0,46	0,39	0,41	0,55	0,19	0,16	0,27	0,37	0,36	0,39	0,50	0,39	0,08	0,56	0,34
CMCSA	0,35	0,45	0,40	0,40	0,41	0,42	0,21	0,39	0,40	0,51	0,34	0,40	1,00	0,40	0,34	0,45	0,46	0,48	0,52	0,26	0,16	0,39	0,43	0,32	0,52	0,49	0,37	0,47	0,61	0,38
VZ	0,18	0,39	0,39	0,39	0,26	0,27	0,16	0,27	0,27	0,44	0,51	0,28	0,40	1,00	0,26	0,37	0,29	0,33	0,36	0,20	0,06	0,44	0,41	0,16	0,38	0,32	0,27	0,28	0,57	0,24
AZN	0,28	0,26	0,25	0,25	0,22	0,26	0,18	0,30	0,30	0,27	0,25	0,23	0,34	0,26	1,00	0,27	0,31	0,25	0,34	0,25	0,18	0,40	0,39	0,24	0,28	0,35	0,27	0,32	0,39	0,22
INTC	0,38	0,51	0,40	0,41	0,51	0,36	0,27	0,44	0,52	0,37	0,37	0,46	0,45	0,37	0,27	1,00	0,41	0,53	0,64	0,17	0,15	0,36	0,36	0,47	0,43	0,53	0,34	0,21	0,58	0,38
ADBE	0,30	0,38	0,40	0,39	0,45	0,44	0,30	0,41	0,40	0,38	0,28	0,39	0,46	0,29	0,31	0,41	1,00	0,47	0,54	0,18	0,21	0,35	0,40	0,33	0,54	0,49	0,40	0,30	0,56	0,43
QCOM	0,42	0,42	0,42	0,42	0,49	0,40	0,29	0,47	0,54	0,34	0,31	0,41	0,48	0,33	0,25	0,53	0,47	1,00	0,63	0,25	0,18	0,32	0,35	0,36	0,48	0,52	0,36	0,36	0,56	0,53
TXN	0,45	0,53	0,36	0,38	0,65	0,40	0,30	0,55	0,64	0,42	0,34	0,55	0,52	0,36	0,34	0,64	0,54	0,63	1,00	0,23	0,25	0,39	0,38	0,51	0,53	0,70	0,46	0,27	0,66	0,42
TMUS	0,13	0,14	0,22	0,22	0,21	0,19	0,27	0,22	0,34	0,18	0,10	0,19	0,26	0,20	0,25	0,17	0,18	0,25	0,23	1,00	0,12	0,22	0,24	0,15	0,23	0,26	0,17	0,29	0,29	0,15
NFLX	0,14	0,27	0,30	0,30	0,24	0,24	0,24	0,18	0,16	0,14	0,06	0,16	0,16	0,06	0,18	0,15	0,21	0,18	0,25	0,12	1,00	0,15	0,09	0,16	0,15	0,22	0,30	0,16	0,20	0,31
AMGN	0,15	0,31	0,33	0,33	0,29	0,31	0,18	0,32	0,27	0,39	0,42	0,27	0,39	0,44	0,40	0,36	0,35	0,32	0,39	0,22	0,15	1,00	0,34	0,20	0,41	0,32	0,26	0,23	0,54	0,31
SNY	0,27	0,33	0,33	0,32	0,31	0,33	0,19	0,43	0,27	0,36	0,39	0,37	0,43	0,41	0,39	0,36	0,40	0,35	0,38	0,24	0,09	0,34	1,00	0,29	0,40	0,39	0,27	0,20	0,48	0,34
AMD	0,32	0,29	0,25	0,26	0,50	0,27	0,30	0,41	0,40	0,31	0,16	0,36	0,32	0,16	0,24	0,47	0,33	0,36	0,51	0,15	0,16	0,20	0,29	1,00	0,32	0,52	0,31	0,20	0,41	0,39
INTU	0,31	0,42	0,41	0,42	0,40	0,45	0,24	0,40	0,45	0,43	0,36	0,39	0,52	0,38	0,28	0,43	0,54	0,48	0,53	0,23	0,15	0,41	0,40	0,32	1,00	0,46	0,42	0,25	0,60	0,45

AMAT	0,41	0,48	0,32	0,32	0,58	0,41	0,29	0,61	0,55	0,39	0,24	0,50	0,49	0,32	0,35	0,53	0,49	0,52	0,70	0,26	0,22	0,32	0,39	0,52	0,46	1,00	0,37	0,27	0,56	0,39
SBUX	0,39	0,44	0,38	0,38	0,32	0,43	0,31	0,40	0,39	0,35	0,24	0,39	0,37	0,27	0,27	0,34	0,40	0,36	0,46	0,17	0,30	0,26	0,27	0,31	0,42	0,37	1,00	0,19	0,46	0,43
CHTR	0,22	0,19	0,25	0,25	0,18	0,25	0,23	0,22	0,25	0,32	0,22	0,08	0,47	0,28	0,32	0,21	0,30	0,36	0,27	0,29	0,16	0,23	0,20	0,20	0,25	0,27	0,19	1,00	0,34	0,18
ADP	0,39	0,58	0,48	0,49	0,50	0,48	0,25	0,47	0,50	0,58	0,56	0,56	0,61	0,57	0,39	0,58	0,56	0,56	0,66	0,29	0,20	0,54	0,48	0,41	0,60	0,56	0,46	0,34	1,00	0,45
BKNG	0,41	0,38	0,47	0,47	0,39	0,51	0,36	0,41	0,43	0,31	0,22	0,34	0,38	0,24	0,22	0,38	0,43	0,53	0,42	0,15	0,31	0,31	0,34	0,39	0,45	0,39	0,43	0,18	0,45	1,00