A picture containing text, outdoor, alcohol

Description automatically generated

Gangmin Li

  Cambridge, UK 2023，02，10

**风险分析方法**

重庆理工大学

操作手册 2

内容提要：

本手册是风险分析方法三个实验操作的第二部分： 使用Pandas的高级风险分析。

所有材料都可以从github上获得。

[GangminLi/Financial-Risk-Management-Return-Analysis: A full course with Python on financial risk and return. Includes video content in 8 lessons with JuPyter Notebooks. (github.com)](https://github.com/GangminLi/Financial-Risk-Management-Return-Analysis)

（在code 按钮下，选择Download Zip）

本手册基于Rune的2.5小时商务分析课程。原课程手册可以从这里获得。[Python for Finance: Risk and Return - Learn Python With Rune](https://www.learnpythonwithrune.org/python-for-finance-risk-and-return/)

如果你遇到问题：

1. 使用Google 或者ChatGPT
2. 使用stackoverflow （<https://stackoverflow.com/>）
3. 和同学讨论
4. 询问老师

**前 言**

想象一下，您正坐在餐桌旁，人们正在谈论投资，突然他们问起您的投资。

沉默片刻，

然后，您告诉他们您已经使用 Efficient Frontier 的蒙特卡罗模拟优化了您的投资组合，以平衡回报与您选择的最佳风险。

哇。 想象一下他们会有多深刻的印象。

你怎么能那样做？ 谁帮助了你？ 它的价格是多少？

在本课程结束时，您将了解如何在自己的笔记本电脑上免费完成所有这些操作。 您现在将了解如何计算股票的波动率、夏普比率、知道什么以及如何进行蒙特卡洛模拟、有效边界、计算相关性、使用线性回归、计算 Beta 和 CAPM。

听起来很多吧？

但是您会惊讶于使用正确的工具完成工作是多么容易。 这就是为什么我将在这个 2.5 小时的实践课中向您介绍 Python 以及 pandas 和 NumPy（Python 库）。并通过使用它们进行风险和回报的技术分析。

这个实践课程包括8个部分：

* [**Lesson 1**](https://www.learnpythonwithrune.org/python-for-finance-risk-and-return/#lesson-1): Use **pandas** and **NumPy** to calculate the **Return** of a **Portfolio**
* [**Lesson 2**](https://www.learnpythonwithrune.org/python-for-finance-risk-and-return/#lesson-2): **Risk** and **Volatility** of a stock calculated as **Average True Range (ATR)**
* [**Lesson 3**](https://www.learnpythonwithrune.org/python-for-finance-risk-and-return/#lesson-3): Combine **Risk** and **Return** into **Sharpe Ratio**
* [**Lesson 4**](https://www.learnpythonwithrune.org/python-for-finance-risk-and-return/#lesson-4): Use **Monte Carlo Simulation** to optimize portfolio with **Risk** and **Return**
* [**Lesson 5**](https://www.learnpythonwithrune.org/python-for-finance-risk-and-return/#lesson-5): How to balance your portfolio with **Correlation**
* [**Lesson 6**](https://www.learnpythonwithrune.org/python-for-finance-risk-and-return/#lesson-6): Use **Linear Regression** to find how X causes Y
* [**Lesson 7**](https://www.learnpythonwithrune.org/python-for-finance-risk-and-return/#lesson-7): Measure a stock’s volatility in relation to the overall market as**Beta**.
* [**Lesson 8**](https://www.learnpythonwithrune.org/python-for-finance-risk-and-return/#lesson-8): **CAPM** – Relationship between systematic **risk** and **expected** **return**

# How to get the most out of this course?

To get the most out of this course you should do the following.

1. Download Example Notebook （\*.ipynb）from Github (download Zip file) or download from QQ
2. Open in your Jupyter Notebook (see how to get it below).
3. Read this Manual.
4. Watch companion video.
5. Try run it line-by-line by yourself in Jupyter Notebook
6. Add your interpretation or understanding of each cell of code. Some of them have been done for you.

## Tutorial 1: Understand ROI and Portfolio with your Data with Python Pandas

This tutorial provides hand on experience on **ROI** and **Portfolio**.

### **Learning objectives:** after this tutorial, you should have a good idea what is:

* **RoI (Return of Invest) : np.log(data/data.iloc[0]).tail(1)**
* **Portfolio: portfolios = [.25, .15, .40, .20]**
* **Total return: np.sum((data/data.iloc[0])\*portfolios\*100000, axis=1)**

### **Resources:**

* Pandas Datareader <https://pandas-datareader.readthedocs.io/> (<https://youtu.be/sgndYho8RyI>)
* Pandas [https://pandas.pydata.org](https://pandas.pydata.org/) (<https://youtu.be/m8ahf_c9hEc>)
* NumPy [http://numpy.org](http://numpy.org/) (Focus here)

**Code File: “01 - Intro to NumPy.ipynb”**

**Video:** [**https://youtu.be/zoyzoNClXE**](https://youtu.be/zoyzoNClXE)**4**

First, we need some historic **time series stock prices**. This can be easily done with [**Pandas Datareader**](https://pandas-datareader.readthedocs.io/).

import numpy as np

import pandas\_datareader as pdr

import datetime as dt

import pandas as pd

start = dt.datetime(2020, 1, 1)

data = pdr.get\_data\_yahoo("AAPL", start)

This will read historic stock prices from Apple (ticker AAPL) starting from 2020 and up until today. The data is in a DataFrame (Pandas main data structure).

It is a good habit to verify that the data is as expected to avoid surprises later in the process. That can be done by calling head() on the DataFrame data, which will show the first 5 lines.

data.head()

Resulting in.

High Low Open Close Volume Adj Close

Date

2020-01-02 75.150002 73.797501 74.059998 75.087502 135480400.0 73.894333

2020-01-03 75.144997 74.125000 74.287498 74.357498 146322800.0 73.175934

2020-01-06 74.989998 73.187500 73.447502 74.949997 118387200.0 73.759003

2020-01-07 75.224998 74.370003 74.959999 74.597504 108872000.0 73.412109

2020-01-08 76.110001 74.290001 74.290001 75.797501 132079200.0 74.593040

Recall that the index should be a DatetimeIndex. This makes it possible to take advantage of being a time series.

data.index

The above gives the index.

DatetimeIndex(['2020-01-02', '2020-01-03', '2020-01-06', '2020-01-07',

'2020-01-08', '2020-01-09', '2020-01-10', '2020-01-13',

'2020-01-14', '2020-01-15',

...

'2022-02-11', '2022-02-14', '2022-02-15', '2022-02-16',

'2022-02-17', '2022-02-18', '2022-02-22', '2022-02-23',

'2022-02-24', '2022-02-25'],

dtype='datetime64[ns]', name='Date', length=543, freq=None)

To remind ourselves further, we recall that each column in a DataFrame has a datatype.

data.dtypes

Shown below here.

High float64

Low float64

Open float64

Close float64

Volume float64

Adj Close float64

dtype: object

To confirm our data type is a DataFrame, we can do

type(data)

We will see,

pandas.core.frame.DataFrame

The next step in our journey is to see how **NumPy** is different from **Pandas** DataFrames.

We can get the DataFrame from Pandas as a NumPy array as follows.

arr = data.to\_numpy()

The shape of a NumPy array gives the dimensions.

(543, 6)

Please notice, that you might get more rows than 543, as you run this later than we do here in the tutorial. There will be a row for each day open on the stock exchange market since beginning of 2020.

But you should get 6 columns, as there are 6 columns in our DataFrame, where the NumPy array comes from.

The first row of data can be accessed as follows.

arr[0]

Which gives the the data of the first row, as we know it from the DataFrame.

[7.51500015e+01 7.37975006e+01 7.40599976e+01 7.50875015e+01

1.35480400e+08 7.43335114e+01]

Notice the scientific notation. Other than that, you can see the figures are the same (except the last adjusted close value) .

Now to an interesting difference from DataFrames. The NumPy array only has one datatype. That means, that all columns have the same datatype. The full array has the same datatype.

arr.dtype

Resulting in the following output.

dtype('float64')

To access the top 10 entries of the first column in our NumPy array (the one representing the **High** column), we can use the following notation.

small = arr[:10, 0].copy()

small

Which will output a one-dimensional array of size 10, containing the 10 first values of column 0.

array([75.15000153, 75.14499664, 74.98999786, 75.22499847, 76.11000061,

77.60749817, 78.16750336, 79.26750183, 79.39250183, 78.875 ])

Some nice functionality to master.

np.max(small)

79.39250183105469

small.max()

79.39250183105469

small.argmax()

8

Where the first two return the maximum value of the array, small. The argmax() returns the index of the maximum value.

The NumPy functionality works well on DataFrames, which comes in handy when working with financial data.

We can get the **logarithm** of values in a NumPy array as follows.

The logarithm base 10 (that is b = 10) is called the decimal or [common logarithm](https://www.wikiwand.com/en/Common_logarithm) and is commonly used in science and engineering. The [natural logarithm](https://www.wikiwand.com/en/Natural_logarithm) has the [number *e*](https://www.wikiwand.com/en/E_(mathematical_constant)) (that is *b* ≈ 2.718) as its base; its use is widespread in mathematics and [physics](https://www.wikiwand.com/en/Physics), because of its simpler [integral](https://www.wikiwand.com/en/Integral) and [derivative](https://www.wikiwand.com/en/Derivative). The [binary logarithm](https://www.wikiwand.com/en/Binary_logarithm) uses base 2 (that is *b* = 2) and is frequently used in [computer science](https://www.wikiwand.com/en/Computer_science).

np.log(small)

array([4.31948614, 4.31941954, 4.31735474, 4.3204836 , 4.33217967,

4.35166405, 4.358854 , 4.37282823, 4.37440393, 4.36786432])

* Notice that we have used natural logarithm (ln)

Similarly, we can apply the logarithm on all entries in a DataFrame as follows.

np.log(data)

We will see,

Table

Description automatically generated

While the logarithm of all the columns here does not make sense. Later we will use this and it will all make sense.

We can calculate the daily return as follows.

data/data.shift()

Resulting in the following output (or first few lines).

Table

Description automatically generated

Let’s investigate that a bit. Recall the data (you can get the first 5 lines: data.head())

Table

Description automatically generated

Notice the calculation.

75.144997/75.150002 (3nd of Jan / the 2nd of Jan)

Gives.

0.9999333998687053

The daily return as data/data.shift(), is effectively the second row (e.g., High) divided by the first row (High). That is the line takes each entry in data and divides it with the corresponding entry in data.shift(), and it happens that data.shift() is shifted one forward by date. Hence, it will divide by the previous row.

Now we understand that, let’s get back to the logarithm. Because, we love log returns. Why? Let’s see this example.

np.sum(np.log(data/data.shift()))

Giving.

High 0.787186

Low 0.779272

Open 0.794015

Close 0.786382

Volume -0.388325

Adj Close 0.802400

dtype: float64

And the following.

np.log(data/data.iloc[0]).tail(1)

Giving the following.

Try to explain that is this mean?

Table

Description automatically generated with medium confidence

Now why are we so excited about that?

Well, because we can **sum the log daily returns and get the full return**. This is really handy when we want to calculate returns of changing portfolios or similar.

**We do not care where the log returns comes from. If our money was invested one day in one portfolio, we get the log return from that. The next day our money is invested in another portfolio. Then we get the log return from that. The sum of those two log returns give the full return.**

That’s the magic.

We also cover how to reshape data in the video lecture.

**Then we consider how to calculate with portfolio and get the return.**

This requires us to read data from multiple tickers to create a portfolio.

Again, the following code may not work. Here is ChatGPT saying.

Graphical user interface, text, application

Description automatically generated

tickers = ['AAPL', 'MSFT', 'TWTR', 'IBM']

start = dt.datetime(2020, 1, 1)

data = pdr.get\_data\_yahoo(tickers, start)

This gives data in the following format.

Attributes Adj Close ... Volume

Symbols AAPL MSFT ... TWTR IBM

Date ...

2020-01-02 74.333511 158.571075 ... 10721100.0 3148600.0

2020-01-03 73.610840 156.596588 ... 14429500.0 2373700.0

2020-01-06 74.197395 157.001373 ... 12582500.0 2425500.0

2020-01-07 73.848442 155.569855 ... 13712900.0 3090800.0

2020-01-08 75.036385 158.047836 ... 14632400.0 4346000.0

Where the column has **two layers** of names. First, the attributes then the second layer of the tickers.

If we only want work with the **Adj Close** values, which is often the case, we can access them as follows.

data = data['Adj Close']

Giving data in the following format.

Text

Description automatically generated

(The actual data can be different (this is run at 27th of Feb, 2022)

Now that is convenient.

Now consider a portfolio as follows.

portfolios = [.25, .15, .40, .20]

That is, 25%, 15%, 40%, and 20% to AAPL, MSFT, TWTR, and IBM, respectively.

Assume we have 10000 USD to invest as above.

**(data/data.iloc[0])\*portfolios\*100000**

What this formula mean?

Well, first we calculate the data (data/data.iloc[0]), this is called “normalization”.

Then we multiply with the portfolio and the amount we invest.

This result in the following.

Table

Description automatically generated

As we can see the first row, this distributes the money as the portfolio is allocated. Then it shows how it evolves.

We can get the sum of the full return as follows.

np.sum((data/data.iloc[0])\*portfolios\*100000, axis=1)

Where we show the summary here.

Text

Description automatically generated

As you see, we start with 100000 USD and end with 149288 USD (at the date to 25th Feb,2022) in this case. You might get a bit different result, as you run yours on a later day.

This is handy to explore a portfolio composition.

Actually, when we get **to Monte Carlo Simulation**, this will be handy. There, we will generate **multiple random portfolios and calculate the return and risk for each of them, to optimize the portfolio composition.**

A random portfolio can be generated as follows with NumPy.

weight = np.random.random(4)

weight /= weight.sum()

Notice, that we generate 4 random numbers (one for each ticker) and then we divide by the sum of them. This ensures the sum of the weights will be 1, hence, representing a portfolio.

This is the end of the first Tutorial.

## Tutorial 2: Volatility (Average True Range) and Risk

In this Tutorial we will learn about the **Volatility** of a stock and how it is one measure of investment risk.

### **Learning objectives:**

* How volatility represent the risk.
* Calculation of Average True Range (ATR) – a volatility and risk measure.
* How to visualize the volatility.

### **Resources:**

* Volatility <https://www.investopedia.com/terms/v/volatility.asp>
* Average True Range <https://www.investopedia.com/terms/a/atr.asp>
* Matplotlib [http://matplotlib.org](http://matplotlib.org/) (<https://youtu.be/2ywUfs0rgtU>)

**File: 02 – Volatitlity.ipynb**

**Video:** [**https://youtu.be/ZpI-JDfuCs4**](https://youtu.be/ZpI-JDfuCs4)

To get started, we need some historic stock prices. This can be done as follows and is covered in Tutorial 1.

import numpy as np

import pandas\_datareader as pdr

import datetime as dt

import pandas as pd

start = dt.datetime(2020, 1, 1)

data = pdr.get\_data\_yahoo("NFLX", start)

This reads the time series data of the historic stock prices of Netflix (ticker NFLX).

To calculate the Average True Range (ATR) we need a formula, which is given on [investoperia.org](https://www.investopedia.com/terms/a/atr.asp).

Text, letter

Description automatically generated

The formula of Average True Range (ATR)

The Average True Range (ATR) is a moving average of the True Range (TR). And the TR is given by **the maximum of the current high (H) minus current low (L),** **the absolute value of current high (H) minus previous close (Cp),** **and the absolute value of current low (L) and previous close (Cp).**

This can look intimidating at first, but don’t worry, this is where the power of Pandas DataFrames and Series come into the picture.

It is always a good idea to make your calculations simple.

high\_low = data['High'] - data['Low']

high\_cp = np.abs(data['High'] - data['Close'].shift())

low\_cp = np.abs(data['Low'] - data['Close'].shift())

Here we make a Series for each of the values needed.

Notice, that we get the previous close by using shift() **(data[‘Close’].shift()).**

Then a great way to get the maximum value of these is to create a DataFrame with all the values.

df = pd.concat([high\_low, high\_cp, low\_cp], axis=1)

true\_range = np.max(df, axis=1)

Now that is nice.

Then we get the ATR as the moving average of 14 days (14 days is the default).

average\_true\_range = true\_range.rolling(14).mean()

Finally, let’s try to visualize it. Often visualization helps us understand it better.

import matplotlib.pyplot as plt

%matplotlib notebook

fig, ax = plt.subplots()

average\_true\_range.plot(ax=ax)

ax2 = data['Close'].plot(ax=ax, secondary\_y=True, alpha=.3)

ax.set\_ylabel("ATR")

ax2.set\_ylabel("Price")

Resulting in the following chart.

Graphical user interface, chart, histogram

Description automatically generated

The resulting output

This shows how the Average True Range (ATR) moves in relation to the stock price. The blue line is ATR and the orange (semi-transparent is the stock price).

In periods with big changes in price, the ATR moves up. When the price is more stable, the ATR moves down.

Now Let us see how to combine **Risk** **and Return** in one measure, **Sharpe Ratio**.

## Tutorial 3: Sharpe Ratio – Combining Risk and Return

In this Tutorial we will combine Risk and Return into one number, the [**Sharpe Ratio**](https://www.investopedia.com/terms/s/sharperatio.asp).

### **Learning objectives**

* Risk and return combined in one number.
* Using standard deviation as a measure for risk.
* Sharpe Ratio calculation combining risk and return.

### **Resources:**

* **Risk free return** 10 Year Treasury Note <https://www.treasury.gov/resource-center/data-chart-center/interest-rates/pages/textview.aspx?data=yield>
* Sharpe Ratio <https://www.investopedia.com/terms/s/sharperatio.asp>
* Standard deviation <https://www.investopedia.com/terms/s/standarddeviation.asp>

**File: 03 - Sharpe Ratio.ipynb**

**Video:** [**https://youtu.be/k5qVtR57MTE**](https://youtu.be/k5qVtR57MTE)

The **Sharpe Ratio** is the average return earned in excess of the risk-free rate per unit of volatility or total risk.

The idea with Sharpe Ratio, is to have one number to represent both return and risk. This makes it easy to compare different weights of portfolios. We will use that in the next Tutorial about Monte Carlo Simulations for Portfolio Optimization.

Now that is a lot of words. How does the Sharpe Ratio look like?

Text

Description automatically generated

We need the return of the portfolio and the **risk free return**, as well as the standard deviation of the portfolio.

* The return of the portfolio we covered in Tutorial 1, but we will calculate it with log returns here.
* It is custom for the risk free return to use the [**10 Year Treasury Note**](https://www.treasury.gov/resource-center/data-chart-center/interest-rates/pages/textview.aspx?data=yield), but as it has been low for long time, often 0 is used.
* The standard deviation is a measure of the volatility and is used here to represent the risk. This is similar to Average True Range.

To get started, we need to read time series data of historic stock prices for a portfolio. This can be done as follows.

import numpy as np

import pandas\_datareader as pdr

import datetime as dt

import pandas as pd

tickers = ['AAPL', 'MSFT', 'TWTR', 'IBM']

start = dt.datetime(2020, 1, 1)

data = pdr.get\_data\_yahoo(tickers, start)

data = data['Adj Close']

Where our portfolio will consist of the tickers for Apple, Microsoft, Twitter and IBM (AAPL, MSFT, TWTR, IBM). We read the data from start 2020 from the Yahoo! Finance API using Pandas Datareader.

Finally, we only keep the **Adjusted Close price**.

Let’s assume our portfolio is balanced as follows, 25%, 15%, 40%, and 20% to AAPL, MSFT, TWTR, IBM, respectively.

Then we can calculate the daily log return of the portfolio as follows.

portfolio = [.25, .15, .40, .20]

log\_return = np.sum(np.log(data/data.shift())\*portfolio, axis=1)

Where we use the np.log to take the logarithm of the daily change, we apply the portfolio. Finally, we sum (np.sum) along the rows (axis=1).

For the fun, we can visualize the daily log returns as follows.

import matplotlib.pyplot as plt

%matplotlib notebook

fig, ax = plt.subplots()

log\_return.hist(bins=50, ax=ax)

Resulting in this.

Chart, histogram

Description automatically generated

This gives an impression of how volatile the portfolio is. The more data is centered around 0.0, the less volatile and risky.

The Sharpe Ratio can be calculate directly as follows.

sharpe\_ratio = log\_return.mean()/log\_return.std()

This gives a daily Sharpe Ratio, where we have the return to be the mean value. That is, the average return of the investment. And divided by the standard deviation.

The greater is the standard deviation the greater the magnitude of the deviation from the mean value can be expected.

To get an annualized Sharpe Ratio.

## ASR : Annualized Sharpe Ratio

**it equals: The monthly Sharpe Ratio times the square root of 12. or The daily Sharpe Ratio times the square root of 252.**

asr = sharpe\_ratio\*252\*\*.5

This is the measure we will use in the next Tutorial, where we will optimize the portfolio using Monte Carlo Simulation.

## Tutorial 4: Monte Carlo Simulation for Portfolio Optimization

In Tutorial we will learn about **Monte Carlo Simulation**. First an introduction to the concept and then how to use **Sharpe Ratio**to find the optimal portfolio with Monte Carlo Simulation.

### **Learning objectives**

* The principles behind Monte Carlo Simulation
* Applying Monte Carlo Simulation using Sharpe Ratio to get the optimal portfolio
* Create a visual **Efficient Frontier** based on Sharpe Ratio

### Resources:

* Monte Carlo Simulation <https://en.wikipedia.org/wiki/Monte_Carlo_method>
* Efficient Frontier <https://www.investopedia.com/terms/e/efficientfrontier.asp>

**File: 04 - Monte Carlo Simulation and Efficient Frontier.ipynb**

**Video:** [**https://youtu.be/E2zkJnzr1D4**](https://youtu.be/E2zkJnzr1D4)

[**Monte Carlo Simulation**](https://en.wikipedia.org/wiki/Monte_Carlo_method) is a great tool to master. It can be used to simulate risk and uncertainty that can affect the outcome of different decision options.

Simply said, if there are too many variables affecting the outcome, then it can simulate them and find the optimal based on the values.

Here we will first use it for simple example, which we can precisely calculate. This is only to get an idea of what Monte Carlo Simulations can do for us.

The game we play.

* You roll two dice.
* When you roll 7, then you gain 5 dollars.
* If you roll anything else than 7, you lose 1 dollar.

How can we simulate this game?

Well, the roll of two dice can be simulated with NumPy as follows.

import numpy as np

def roll\_dice():

return np.sum(np.random.randint(1, 7, 2))

Where are roll is simulated with a call to the roll\_dice(). It simply uses the np.random.randint(1, 7, 2), which returns an array of length 2 with 2 integers in the range 1 to 7 (where 7 is not included, but 1 is). The np.sum(…) sums the two integers into the sum of the two simulated dice.

Now to the Monte Carlo Simulation.

This is simply to make a trial run and then see if it is a good game or not.

def monte\_carlo\_simulation(runs=1000):

results = np.zeros(2)

for \_ in range(runs):

if roll\_dice() == 7:

results[0] += 1

else:

results[1] += 1

return results

This is done by keeping track of the how many games We win and lose.

A run could look like this.

monte\_carlo\_simulation()

**It could return array([165., 835.]), which would result in my win of 165\*5 = 825 USD and lose of 835 USD. A total loss of 10 USD.**

Each run will most likely give different conclusions.

A way to get a more precise picture is to make more runs. Here, we will try to record a series of runs and visualize them.

results = np.zeros(1000)

for i in range(1000):

results[i] = monte\_carlo\_simulation()[0]

import matplotlib.pyplot as plt

%matplotlib notebook

fig, ax = plt.subplots()

ax.hist(results, bins=15)

Resulting in this figure.

Chart, histogram

Description automatically generated

This gives an idea of how a game of 1000 runs (each run roll two dice 1000 timers) returns the times we have two dice adding together equals to 7 (results[i]= monte\_carlo\_simulation()[0]) .

From the plot we can see how volatile it is. See, if the game was less volatile, it would centre around one place.

For these particular runs we have that results.mean()\*5 gives the average return of 833.34 USD (notice, you will not get the exact same number due to the randomness involved).

The average loss will be 1000 – results.mean() = 833.332 USD.

This looks like a pretty even game.

**Can we calculate this exactly?**

Yes. The reason is, that this is a simple situation are simulating. When we have more variable (as we will have in a moment with portfolio simulation) it will not be the case.

A nice way to visualize it is as follows.

d1 = np.arange(1, 7)

d2 = np.arange(1, 7)

mat = np.add.outer(d1, d2)

Where the matrix mat looks as follows.

array([[ 2, 3, 4, 5, 6, 7],

[ 3, 4, 5, 6, 7, 8],

[ 4, 5, 6, 7, 8, 9],

[ 5, 6, 7, 8, 9, 10],

[ 6, 7, 8, 9, 10, 11],

[ 7, 8, 9, 10, 11, 12]])

The exact probability for rolling 7 is.

mat[mat == 7].size/mat.size

Where we count how many occurrences of 7 divided by the number of possibilities. This gives 0.16666666666666667 or 1/6.

Think about it, is it a fair game?

This is the same we concluded with the Monte Carlo Simulation.

Now we have some understanding of Monte Carlo Simulation, we are ready to use it for portfolio optimization.

To do that, we need to read some time series of historic stock prices.

import pandas\_datareader as pdr

import datetime as dt

import pandas as pd

tickers = ['AAPL', 'MSFT', 'TWTR', 'IBM']

start = dt.datetime(2020, 1, 1)

data = pdr.get\_data\_yahoo(tickers, start)

data = data['Adj Close']

To use it with Sharpe Ratio, we will calculate the log returns.

log\_returns = np.log(data/data.shift())

The Monte Carlo Simulations can be done as follows.

# Monte Carlo Simulation

n = 5000

weights = np.zeros((n, 4))

exp\_rtns = np.zeros(n)

exp\_vols = np.zeros(n)

sharpe\_ratios = np.zeros(n)

for i in range(n):

weight = np.random.random(4)

weight /= weight.sum()

weights[i] = weight

exp\_rtns[i] = np.sum(log\_returns.mean()\*weight)\*252

exp\_vols[i] = np.sqrt(np.dot(weight.T, np.dot(log\_returns.cov()\*252, weight)))

sharpe\_ratios[i] = exp\_rtns[i] / exp\_vols[i]

The code will run 5000 experiments. We will keep all the data from each run. The weights of the portfolios (weights), the expected return (exp\_rtns), the expected volatility (exp\_vols) and the Sharpe Ratio (sharpe\_ratios).

Then we iterate over the range.

First, we create a random portfolio in weight (notice it will have the sum 1). Then we calculate the expected annual return. **The expected volatility is calculated a bit different than we learned in the Tutorial about Sharpe Ratio. This is only to make it perform faster.**

Finally, the Sharpe Ratio is calculated.

In this specific run (you might get different values) we get that the maximum Sharpe Ratio is, given by sharpe\_ratios.max(), 1.1398396630767385.

To get the optimal weight (portfolio), call weights[sharpe\_ratios.argmax()]. In this specific run, array([4.57478167e-01, 6.75247425e-02, 4.74612301e-01, 3.84789577e-04]). This concludes to hold 45.7% to AAPL, 6.7% to MSFT, 47.5% to TWTR, and 0,03% to IBM is optimal.

This can be visualized as follows in an **Efficient Frontier**.

import matplotlib.pyplot as plt

%matplotlib notebook

fig, ax = plt.subplots()

ax.scatter(exp\_vols, exp\_rtns, c=sharpe\_ratios)

ax.scatter(exp\_vols[sharpe\_ratios.argmax()], exp\_rtns[sharpe\_ratios.argmax()], c='r')

ax.set\_xlabel('Expected Volatility')

ax.set\_ylabel('Expected Return')

Resulting in this chart.

Chart, scatter chart

Description automatically generated

## Tutorial 5: Correlation Calculations

In this Tutorial we will learn about correlation of assets, calculations of correlation, and risk and coherence.

### **Learning objectives**

* What is correlation and how to use it
* Calculate correlation
* Find negatively correlated assets

### **Resources**

* Correlation <https://www.investopedia.com/terms/c/correlation.asp>
* SP500 by Market Cap <https://www.slickcharts.com/sp500>

### **File： 05 – Correlation.ipynb**

**Video:** [**https://youtu.be/gIZEiQsNqBo**](https://youtu.be/gIZEiQsNqBo)

Correlation is a **statistic** measurement that measures **the degree to which two variables move in relation to each other**. Correlation measures association but doesn’t show if **x** causes **y** or vice versa.

The correlation between two stocks is a number from -1 to 1 (both inclusive).

* A positive correlation means, when stock x goes up, we expect stock y to go up, and opposite.
* A negative correlation means, when stock x goes up, we expect stock y to go down, and opposite.
* A zero correlation, we cannot say anything in relation to each other.

The formula for calculating the correlation is quite a mouthful.

Text, letter

Description automatically generated

Luckily, the **DataFrames**object in Pandas has a function that can calculate it for us. Hence, we do not need to master how to do it.

Let’s get started. First, we need to load some time series of historic stock prices.

import pandas as pd

import pandas\_datareader as pdr

import datetime as dt

import numpy as np

tickers = ['AAPL', 'TWTR', 'IBM', 'MSFT']

start = dt.datetime(2020, 1, 1)

data = pdr.get\_data\_yahoo(tickers, start)

data = data['Adj Close']

log\_returns = np.log(data/data.shift())

Where we also calculate the log returns.

The correlation can be calculated as follows.

log\_returns.corr()

That was easy, right? Remember we do it on the log returns to keep it on the same range.

Symbols AAPL TWTR IBM MSFT

Symbols

AAPL 1.000000 0.531973 0.518204 0.829547

TWTR 0.531973 1.000000 0.386493 0.563909

IBM 0.518204 0.386493 1.000000 0.583205

MSFT 0.829547 0.563909 0.583205 1.000000

This result is normally called correlation Matrix. In this matrix we can see that the correlation on the **diagonal is 1.0**. This is obvious, since the diagonal shows the correlation between itself (AAPL and AAPL, and so forth).

Other than that, we can conclude that AAPL and MSFT are correlated the most.

Let’s add the S&P 500 to our DataFrame.

sp500 = pdr.get\_data\_yahoo("^GSPC", start)

log\_returns['SP500'] = np.log(sp500['Adj Close']/sp500['Adj Close'].shift())

log\_returns.corr()

Resulting in this.

Table

Description automatically generated

Where we see that AAPL and MSFT are mostly correlated to S&P 500 index. This is not surprising, as they are a big part of the weight of the market cap in the index.

We can define a function to calculate any “Ticker’s” correlations (with one we have. Notice, this is not calculate any one ticker but the ticker’s with any other tickers already been calculated)

def test\_correlation(ticker):

df = pdr.get\_data\_yahoo(ticker, start)

lr = log\_returns.copy()

lr[ticker] = np.log(df['Adj Close']/df['Adj Close'].shift())

return lr.corr()

Use this function can help us find assets with a **negative** correlation. Think about it, Why do we want that? Well, to minimize the risk.

Now, let’s test.

test\_correlation("TLT")

Resulting in this following.

Table

Description automatically generated

The negative correlation we are looking for.

This can be visualized to get a better understanding as follows.

import matplotlib.pyplot as plt

%matplotlib notebook

def visualize\_correlation(ticker1, ticker2):

df = pdr.get\_data\_yahoo([ticker1, ticker2], start)

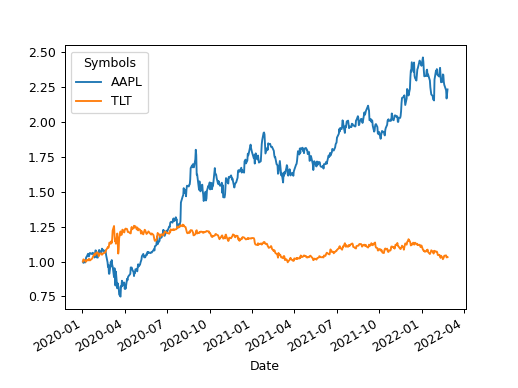
df = df['Adj Close']

df = df/df.iloc[0]

fig, ax = plt.subplots()

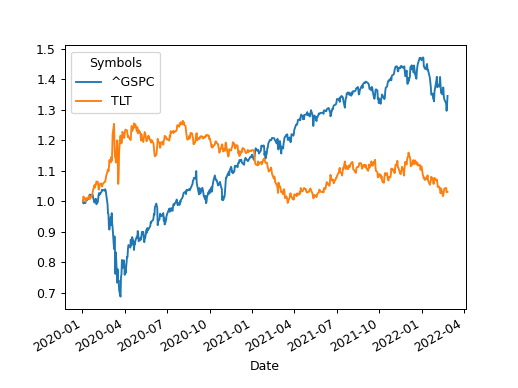
df.plot(ax=ax)

With visualize\_correlation(“AAPL”, “TLT”) we get.



Where we see, when AAPL goes down, the TLT goes up.

And if we look at visualize\_correlation(“^GSPC”, “TLT”) (the S&P 500 index and TLT).



We can conclude that the TLT moves opposite with SP500.

## Tutorial 6: Linear Regression Calculations

In this Tutorial we will learn about **Linear Regression**, difference from Correlation and how to **visualize Linear Regression**.

### **Learning objectives**

* Understand the difference between Linear Regression and Correlation.
* Understand the difference between **true random** and **correlated variables**
* Visualize linear regression.

### **Resources:**

1. The difference between correlation and linear regression. <https://www.graphpad.com/support/faq/what-is-the-difference-between-correlation-and-linear-regression/>
2. Linear regression model in Python. <https://www.kdnuggets.com/2019/03/beginners-guide-linear-regression-python-scikit-learn.html>

### **File:** [**06 - Linear Regression.ipynb**](https://github.com/LearnPythonWithRune/PythonForFinanceRiskAndReturn/blob/main/06%20-%20Linear%20Regression.ipynb)

**Video:** [**https://youtu.be/H65yckhLyBE**](https://youtu.be/H65yckhLyBE)

Let’s first see what the similarities and difference between Linear Regression and Correlation is.

Similarities.

* Quantify the direction and strength of the relationship between two variables, here we look at stock prices.

Differences.

* Correlation is a single statistic. It is just a number between -1 and 1 (both inclusive).
* Linear regression produces an equation Y = a + bX.

A great way to learn about relationships between variables is to compare it to random variables.

Let’s start by doing that.

import pandas as pd

import numpy as np

from sklearn.linear\_model import LinearRegression

import pandas\_datareader as pdr

import datetime as dt

import matplotlib.pyplot as plt

%matplotlib notebook

X = np.random.randn(5000)

Y = np.random.randn(5000)

fig, ax = plt.subplots()

ax.scatter(X, Y, alpha=.2)

Giving the following scatter chart.

Chart, scatter chart

Description automatically generated

Which shows the how two non-correlated variables look like.

To compare that to two correlated, we need some data.

tickers = ['AAPL', 'TWTR', 'IBM', 'MSFT', '^GSPC']

start = dt.datetime(2020, 1, 1)

data = pdr.get\_data\_yahoo(tickers, start)

data = data['Adj Close']

log\_returns = np.log(data/data.shift())

Let’s make a function to calculate the Liner Regression and visualize it.

def linear\_regression(ticker\_a, ticker\_b):

X = log\_returns[ticker\_a].iloc[1:].to\_numpy().reshape(-1, 1)

Y = log\_returns[ticker\_b].iloc[1:].to\_numpy().reshape(-1, 1)

lin\_regr = LinearRegression()

lin\_regr.fit(X, Y)

Y\_pred = lin\_regr.predict(X)

alpha = lin\_regr.intercept\_[0]

beta = lin\_regr.coef\_[0, 0]

fig, ax = plt.subplots()

ax.set\_title("Alpha: " + str(round(alpha, 5)) + ", Beta: " + str(round(beta, 3)))

ax.scatter(X, Y)

ax.plot(X, Y\_pred, c='r')

The function takes the two tickers and get’s the log returns in NumPy arrays. They are reshaped to fit the required format.

We then use these two (pair) values to **train** the Linear Regression model. Remember the Linear Regress model is in a form of: Y = a + bX. Or, (, where,

Alpha () and Beta () are the parameters (fixed quantities) of the model.

* Alpha is called constant or intercept and measures the value where the regression line crosses the y-axis,
* Beta is called coefficient/slope. And it measures the steepness of the regression line.

The actual regression analysis (**Training**) is to estimates the parameters alpha and beta by using the given observations for X and Y. (as we hypothesize a linear relationship).

Once we can find the model, we can use it to predict new values.

The Linear Regression model (LinearRegression) is used and applied to predict values. The alpha and beta are the liner variables. Finally, we scatter plot all the points and a prediction line.

Let’s try linear\_regression(“AAPL”, “^GSPC”).

Chart, scatter chart

Description automatically generated

Where we see the red line as the prediction line.

Other examples linear\_regression(“AAPL”, “MSFT”)

Chart, scatter chart

Description automatically generated

And linear\_regression(“AAPL”, “TWTR”).

Chart, scatter chart

Description automatically generated

Where it visually shows that AAPL and TWTR are not as closely correlated as the other examples.

## Tutorial 7: Stock’s Volatility Measure: Beta and S&P 500

In this Tutorial we will learn about Stock market’s volatility measure **Beta**with **S&P 500** index, how to calculate it, and comparison of calculations from last Tutorial.

### **Learning Objectives**

* Understand what market Beta tells you.
* How to calculate the market (S&P 500) Beta.
* See how Beta is related with Linear Regression.

### Resources

* Beta <https://www.investopedia.com/investing/beta-know-risk/>

### **File: 07- Beta.ipynb**

### **Video:** [**https://youtu.be/u\_rXWGfHKeM**](https://youtu.be/u_rXWGfHKeM)

**Beta** is a measure of a stock’s volatility in relation to the overall market (represented by S&P 500 index for example). **The S&P 500 index has Beta 1**. If a stock **moves less** than the market, the stock's beta is less than 1.0. On the contrary, if a stock **moves** **more** than the market, the stock's beta is more than 1.0.

High-beta (>1.0) stocks are supposed to be riskier but provide higher potential return. While low-beta (<1.0) stocks pose less risk but also lower returns.

Interpretation

* Beta above 1: stock is more volatile than the market but expects higher return.
* Beta below 1: stock with lower volatility and expects less return.

The formula for Beta is **Covariance divided by Variance. (**Beta=Variance/Covariance**)**

This sound more scary than it is.

The Beta on financial pages, like Yahoo! Finance, are calculated on the monthly price.

Let’s make an example here.

import numpy as np

import pandas\_datareader as pdr

import datetime as dt

import pandas as pd

from sklearn.linear\_model import LinearRegression

tickers = ['AAPL', 'MSFT', 'TWTR', 'IBM', '^GSPC']

start = dt.datetime(2015, 12, 1)

end = dt.datetime(2021, 1, 1)

data = pdr.get\_data\_yahoo(tickers, start, end, interval="m")

data = data['Adj Close']

log\_returns = np.log(data/data.shift())

Where we notice that we read data on interval=”m”, which gives the monthly data.

Then the Beta is calculated as follows.

cov = log\_returns.cov()

var = log\_returns['^GSPC'].var()

cov.loc['AAPL', '^GSPC']/var

For Apple, it was 1.25.

If you wonder if it is related to the Beta value from Linear Regression. Let’s check it out.

X = log\_returns['^GSPC'].iloc[1:].to\_numpy().reshape(-1, 1)

Y = log\_returns['AAPL'].iloc[1:].to\_numpy().reshape(-1, 1)

lin\_regr = LinearRegression()

lin\_regr.fit(X, Y)

lin\_regr.coef\_[0, 0]

Also giving 1.25. Hence, it is the same calculation behind it.

**Question:** What the calculation tells us about Apple share in the market?

## Tutorial 8: Capital Asset Pricing Model (CAPM)

The well-worn definition of risk is the possibility of suffering a loss. Of course, when investors consider risk, they are thinking about the chance that the stock they buy will decrease in value. Beta is a measure of a stock's volatility in relation to the overall market. Beta is calculated using [regression](https://www.investopedia.com/terms/r/regression.asp) analysis. Numerically, it represents the tendency for a security's returns to respond to swings in the market. The formula for calculating beta is the covariance of the return of an asset with the return of the [benchmark](https://www.investopedia.com/terms/b/benchmark.asp) divided by the variance of the return of the benchmark over a certain period.

Beta is a component of the [capital asset pricing model (CAPM)](https://www.investopedia.com/terms/c/capm.asp), which is used to calculate the cost of equity funding. The CAPM formula uses the total average market return and the beta value of the stock to determine the rate of return that shareholders might reasonably expect based on perceived investment risk.

In this Tutorial we will learn about the CAPM.

### **Learning objectives**

* Understand the CAPM (Capital Asset Pricing Model).
* Beta and CAPM calculations.
* Expected return of an investment.

### **Resources:**

* CAPM <https://www.investopedia.com/terms/c/capm.asp>
* **Market risk premium** <https://www.investopedia.com/terms/m/marketriskpremium.asp>
* **Risk free return** 10 Year Treasury Note <https://www.treasury.gov/resource-center/data-chart-center/interest-rates/pages/textview.aspx?data=yield>
* Between 1926 and 2014, the S&P 500 exhibited a 10.5% compounding annual rate of return

**File: 08 – CAPM.ipynb**

**Video:** [**https://youtu.be/KK6DdHyOkFs**](https://youtu.be/KK6DdHyOkFs)

The CAPM calculates the relationship between **systematic risk** and **expected return**. There are several assumptions behind the CAPM formula that have been shown not to hold in reality. But still, the CAPM formula is still widely used.

The formula is as follows.

Text, letter

Description automatically generated

Let’s get some data and calculate it.

import numpy as np

import pandas\_datareader as pdr

import datetime as dt

import pandas as pd

tickers = ['AAPL', 'MSFT', 'TWTR', 'IBM', '^GSPC']

start = dt.datetime(2015, 12, 1)

end = dt.datetime(2021, 1, 1)

data = pdr.get\_data\_yahoo(tickers, start, end, interval="m")

data = data['Adj Close']

log\_returns = np.log(data/data.shift())

Again, when we look at the formula, the risk free return is often set to 0. Otherwise, the [10 years treasury note is used](https://www.treasury.gov/resource-center/data-chart-center/interest-rates/pages/textview.aspx?data=yield). Here, we use 1.38%. You can update it for more up to date value with the link.

We can calculate the expected return on the stock market when we invest on apple share,

cov = log\_returns.cov()

var = log\_returns['^GSPC'].var()

beta = cov.loc['AAPL', '^GSPC']/var

risk\_free\_return = 0.0138

market\_return = .105

expected\_return = risk\_free\_return + beta\*(market\_return - risk\_free\_return)

It gives us this number,

0.1282156241860044

Notice, you can calculate it all simultaneously.

cov = log\_returns.cov()

var = log\_returns['^GSPC'].var()

beta = cov.loc['^GSPC']/var

risk\_free\_return = 0.0138

market\_return = .105

expected\_return = risk\_free\_return + beta\*(market\_return - risk\_free\_return)

This online course C O U R S E R E S O U R C E S

▸ Lesson Post: [https://www.learnpythonwithrune.org/g...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqa1ZSNE1MNlVuMi1pa29xSy13SlEzOUJxUHVwd3xBQ3Jtc0ttQjNMRVZKVW5PUDdFdnNLNXNEejlvNm5PcHlUa29HTVJKd2l5ZTR5TmxLenV0VjV4TWs0bXZ5QTJrc21tWVVkQWptYzFNUTU0S2hwVWtaNzZ0TWRFaGZrYS1Xc0cwTi1OdzNubFpOVXFPWEZ6Y3hIVQ&q=https%3A%2F%2Fwww.learnpythonwithrune.org%2Fget-started-with-pandas-and-numpy-for-finance-for-risk-and-return%2F)

▸ Course page: [https://www.learnpythonwithrune.org/p...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbU14WG5EbEN5eFJfdExzTXk3RHVxTzNxTXFxQXxBQ3Jtc0tsMlZ0c3dPRkVrckRjcHZVQzZCNnNkYWUwc0o0R2dnTzNsNmkwWXRQRVRwQTRRdDd3QVA4RVNQM2M5MVdNd0lmVXh1ckhkSW5oS1MyQWhaN19vY19HczZtTUxoWng1SEttWUFfazRNNy1QaHlrYjBZOA&q=https%3A%2F%2Fwww.learnpythonwithrune.org%2Fpython-for-finance-risk-and-return%2F)

▸ Jupyter Notebook download: [https://github.com/LearnPythonWithRun...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqa0dMeHlndm1tX0JnYjE1bE1oZjZLZkhpeVRnUXxBQ3Jtc0tuNW1WSzBwWk55ZFlIbEt2QnBLYnZQdWg3MmtYNUhZTFB5ajhmaEhSYjBoeWFDVVFNUU9GZVNKZjZLN3BBaHRabkdNZDhEekI2QlBfVkttQlRmMFYyRDM0OUxIUXBTQlNYRDNOUzBoOUwzZ2J4bFdwbw&q=https%3A%2F%2Fgithub.com%2FLearnPythonWithRune%2FPythonForFinanceRiskAndReturn)

▸ Anaconda (Jupyter Notebook and Python): [https://www.anaconda.com/products/ind...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqa21oai1TcU5kSXlxV2RpX1pkNnNpNnRwRTY1UXxBQ3Jtc0tuN2xpYWMwa2Zya19DTndjUXJkdjBuQnRRM2tCc2NNb2FtWmxHZ2ZMV2JNWlNOS0tCR1dkejZGa3dfU3lOd3oxaHktVjIxU2gtWmphbGhXaU9PbExIQVhWeFpzLUVnRWlJUGlXX1J1aXg0RG16blRPTQ&q=https%3A%2F%2Fwww.anaconda.com%2Fproducts%2Findividual)

▸ Get it here: [https://www.learnpythonwithrune.org/f...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbEkzR2MtQlotTk9fNHZNNDZaN1k4YU5mYUVJQXxBQ3Jtc0tsY2JiUVA1c2ZBQks3WEJOeURNLTREWE9lNnhxWFN3cnlncXM5OXdaRm0weEFaUW1ONld4d2NReGxxNEd4eFYtNVRGV1ZiUTB6bHoxaWl0RHplVGkzU1ZFUWc2QmhRcUZZS2RNbi1YWW8yMWtlY0xNMA&q=https%3A%2F%2Fwww.learnpythonwithrune.org%2Ffree-ebook-2%2F)

▸ Full code available: [https://github.com/LearnPythonWithRun...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbk9HQ1FUTHo0ZXRYa2U1RkpqUjlzbkdNMXlzZ3xBQ3Jtc0ttcUdJUENYcWo1M1ROUjlLMlZpam50eFk5X2J6NDN4SXVicFFwdHUtcWFuRkhvVEtSQVhQT2Mza3A0dExhUEpxeHhQSFZKa2N6cEQyNkw4dS15R1lXU3ZzNDZlVzB1aUk4WldTZVEtNzRjeGtONlpHaw&q=https%3A%2F%2Fgithub.com%2FLearnPythonWithRune%2FPythonForFinancialAnalysis)

* Download the 8 Notebooks ([GitHub zip-file](https://github.com/LearnPythonWithRune/FinancialDataAnalysisWithPython/archive/refs/heads/main.zip))
* Start JuPyter Notebook ([If you don’t have it, install Anaconda for FREE](https://www.anaconda.com/products/individual))
* Start the playlist on [YouTube](https://youtube.com/playlist?list=PLvMRWNpDTNwQF6t_Tq7aVX0AI6H1avSpv).

Tutorial D E S C R I P T I O N

Part 1: [https://youtu.be/zoyzoNClXE4](https://www.youtube.com/watch?v=zoyzoNClXE4&t=0s) - Update Pandas and Pandas DataReader - Learn differences between DataFrames and NumPy - NumPy basics - Portfolios and returns

Part 2: [https://youtu.be/ZpI-JDfuCs4](https://www.youtube.com/watch?v=ZpI-JDfuCs4&t=0s) - Volatility and Risk - Average True Range (ATR) - Visualization of the result

Part 3: [https://youtu.be/k5qVtR57MTE](https://www.youtube.com/watch?v=k5qVtR57MTE&t=0s) - Risk and return combined - Risk as standard deviation - Sharpe Ratio calculation

Part 4: [https://youtu.be/E2zkJnzr1D4](https://www.youtube.com/watch?v=E2zkJnzr1D4&t=0s) - Introduction to Monte Carlo Simulation - Portfolio optimization - Visualization of the result

Part 5: [https://youtu.be/gIZEiQsNqBo](https://www.youtube.com/watch?v=gIZEiQsNqBo&t=0s) - Correlation of assets - Calculations of correlation - Risk and coherence

Part 6: [https://youtu.be/H65yckhLyBE](https://www.youtube.com/watch?v=H65yckhLyBE&t=0s) - Linear regression vs correlation - True random distribution vs correlated - Visualization of linear regression

Part 7: [https://youtu.be/u\_rXWGfHKeM](https://www.youtube.com/watch?v=u_rXWGfHKeM&t=0s) - Market beta (S&P 500) - Beta calculation with S&P 500 - Comparison of calculations

Part 8: [https://youtu.be/KK6DdHyOkFs](https://www.youtube.com/watch?v=KK6DdHyOkFs&t=0s) - Capital wealth pricing model (CAPM) - Beta and CAPM calculations - Expected return on investment

——————————————

Financial analysis with Python

▸ Course page: [https://www.learnpythonwithrune.org/s...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbWJ1Ui1LLWVZWjh0Rm5KTXh3a2s5NTJkSTdHZ3xBQ3Jtc0ttVjRuQ0ctbUJ3bjUybTdfY0paRUJ4NWktaG5JajBWU3l2b0IzSTE3UWZPRUpvYTdsQ1lQaXNWcGRRbjU1WEVEeDdxY3VqMmhyZGJMRXhkU1YzalRyWEVEelpqcURuWFk4M241UmJKUVZlUjVHMS1SRQ&q=https%3A%2F%2Fwww.learnpythonwithrune.org%2Fstart-python-with-pandas-for-financial-analysis%2F)

▸ GitHub: [https://github.com/LearnPythonWithRun...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbTdieVVtOVZyU0IyNU81RW9WRllJZTdjUTNGZ3xBQ3Jtc0tsVUp5Mk1aWllBbXFVR21KaXpYQWdmdDVQMUF5R2ZNZHRKbkRCTU1tYUVJWXNuNFh6SlJab1ZLVFEzY09YUGNSdjNRZGMxd2ZVTWZzY1RsZmFMZzI4UHVUczNFTzBpLW9LenZfdlJoR1VlTWpqVXpaaw&q=https%3A%2F%2Fgithub.com%2FLearnPythonWithRune%2FFinancialDataAnalysisWithPython) Learn Python for Beginners.

▸ A 8 hour full Python Course.

▸ Including FREE eBook and Jupyter Notebooks.

▸ Course page: [https://www.learnpythonwithrune.org/l...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbV9xa0tKT2ZVQ3oyLWI0NlB2QWpNT05ETm91QXxBQ3Jtc0tuSy1JVDV6bGtDMGdvVEpqWnpHRmhnQkxtYXRFTWlJR084RlYwZVZFenNEbFNFcm5sMEkwRXBfSHpLWDRHZlEyMDVJZWxMVG5vZDZxU0FtT0tKaVBVOEhpUGJJdjhUVTJJcmxUOG1GWUstMXRUZlYtbw&q=https%3A%2F%2Fwww.learnpythonwithrune.org%2Flearn-python%2F)

——————————————

U D E M Y C O U R S E S Python for Finance 2021: Financial Analysis for Investing

▸ Link to Udemy: [https://bit.ly/3sLbMAf](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbnk1QmlCQUV1U0tTV21OQUZHRlBXQ1ZvMlVyZ3xBQ3Jtc0ttdUZ5bmJvNzBNVlN5NlkyN1NnVG9UeXhScjJCNkJPVmRiaThtczVRSXc0MEhxdVFPaHpSdTl6MnFrVGZfYVRUU3ZycV81cmRNVWM4WVJKZUQ0MVgxaDZYZ0tzbDA2U0Rqd3RWOVpuN2xqUzBDQ2RvSQ&q=https%3A%2F%2Fbit.ly%2F3sLbMAf)

▸ 21.5h course

▸ Use Python to Find Good Investments.

▸ Learn Pandas, NumPy, Matplotlib for Financial Analysis & Automate Value Investing.

——————————————

S T A Y C O N N E C T E D

▸ Blog: [http://learnpythonwithrune.org](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbmt4c284S0RqOWVaNWFnQU9Ka3U1WEYyUHhuQXxBQ3Jtc0tsVGw1NUhrTS13bTRfS2pIbk82QUNOcGlMMFQzQzZvNExXQjhpSzQxT0lMX3BYWXJIelZUeE5DSnNDSlpmVzdKOUNiSlBxWkhEbVJxTHVHWXM4cXowZjQwNTNjYW5VWi10VWpNbHI0bFdxOWJWcWp6TQ&q=http%3A%2F%2Flearnpythonwithrune.org)

▸ Online courses: [https://www.learnpythonwithrune.org/m...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbXJBc1NJenRsMmtOczlsVWNuUkIwSldnU19Ed3xBQ3Jtc0ttN1U1UFhzeFlWTHE4NmUzSGR0UnF0NGp0NHBPNmdhRVN4dTBnVjF1ZkM2NjFzRFRyWkFyZ3c5a1N0bk9GaF95WGh3eDV6Y1FhUUNLU09PS3E1VDZwdDNFQ0tkMTlVTF9QR0w1TV9XSjNBZ3JzaF9fQQ&q=https%3A%2F%2Fwww.learnpythonwithrune.org%2Fmy-online-courses%2F)