AF3214

Week 7. Introduction to Risk Measures and Measuring Algorithms Performance

Part 1: Risk Measures

Obtain data from APIs

e.g., Alpha Vantage, https://github.com/RomelTorres/alpha_vantage

In [54]:
```
!pip install alpha_vantage
```

```
zsh:1: command not found: pip
```

Let's start by loading data that we will use. We need two firms. Here we pick Amazon and Apple.

In [55]:
```
# Import pandas and numpy
import pandas as pd
import numpy as np
```

In [56]:
```
# Create an empty dictionary. We name it as "stock_data"
stock_data = {}
```

In [57]:
```
# API: alpha_vantage (an API by Alpha Vantage)
from alpha_vantage.timeseries import TimeSeries
import time

# If you want to download the data please use your own Alpha Vantage key
ts = TimeSeries(key='J4TEYW0NMM3KQH5Y', output_format='pandas')

tickers = ['AAPL','AMZN']

for ticker in tickers:
    filename = ticker + '.csv'
    data, meta_data = ts.get_daily(symbol=ticker, outputsize='full')
    stock_data[ticker] = data
    data.to_csv(filename)
    time.sleep(5)

# meta data: a set of data that describes and gives information about other
```

```python
# There are two parts in the API response: "Meta Data" and "Time Series"
# The library is mapping meta_data to "Meta Data" and "Time Series" to data
```

In [58]:
```python
meta_data
```

Out[58]:
```
{'1. Information': 'Daily Prices (open, high, low, close) and Volumes',
 '2. Symbol': 'AMZN',
 '3. Last Refreshed': '2025-03-11',
 '4. Output Size': 'Full size',
 '5. Time Zone': 'US/Eastern'}
```

In [59]:
```python
stock_data
```

Out[59]:
```
{'AAPL':             1. open   2. high    3. low  4. close   5. volume
 date
 2025-03-11  223.805  225.8399  217.4500    220.84  73971209.0
 2025-03-10  235.540  236.1600  224.2200    227.48  71451281.0
 2025-03-07  235.105  241.3700  234.7600    239.07  46273565.0
 2025-03-06  234.435  237.8600  233.1581    235.33  45170419.0
 2025-03-05  235.420  236.5500  229.2300    235.74  47227643.0
 ...             ...       ...       ...       ...         ...
 1999-11-05   84.620   88.3700   84.0000     88.31   3721500.0
 1999-11-04   82.060   85.3700   80.6200     83.62   3384700.0
 1999-11-03   81.620   83.2500   81.0000     81.50   2932700.0
 1999-11-02   78.000   81.6900   77.3100     80.25   3564600.0
 1999-11-01   80.000   80.6900   77.3700     77.62   2487300.0

 [6378 rows x 5 columns],
 'AMZN':             1. open   2. high    3. low  4. close   5. volume
 date
 2025-03-11   193.90  200.1800  193.4000    196.59  52302854.0
 2025-03-10   195.60  196.7300  190.8500    194.54  61829231.0
 2025-03-07   199.49  202.2653  192.5300    199.25  59802821.0
 2025-03-06   204.40  205.7700  198.3015    200.70  49863755.0
 2025-03-05   204.80  209.9800  203.2600    208.36  38610085.0
 ...             ...       ...       ...       ...         ...
 1999-11-05    64.75   65.5000   62.2500     64.94  11091400.0
 1999-11-04    67.19   67.1900   61.0000     63.06  16759200.0
 1999-11-03    68.19   68.5000   65.0000     65.81  10772100.0
 1999-11-02    69.75   70.0000   65.0600     66.44  13243200.0
 1999-11-01    68.06   71.8800   66.3100     69.13  12824100.0

 [6378 rows x 5 columns]}
```

In [60]:
```python
type(stock_data['AAPL'])
```

Out[60]:
```
pandas.core.frame.DataFrame
```

In [61]:
```python
stock_data['AAPL']
```

Out[61]:

date	1. open	2. high	3. low	4. close	5. volume
2025-03-11	223.805	225.8399	217.4500	220.84	73971209.0
2025-03-10	235.540	236.1600	224.2200	227.48	71451281.0
2025-03-07	235.105	241.3700	234.7600	239.07	46273565.0
2025-03-06	234.435	237.8600	233.1581	235.33	45170419.0
2025-03-05	235.420	236.5500	229.2300	235.74	47227643.0
...	...	...	...	...	...
1999-11-05	84.620	88.3700	84.0000	88.31	3721500.0
1999-11-04	82.060	85.3700	80.6200	83.62	3384700.0
1999-11-03	81.620	83.2500	81.0000	81.50	2932700.0
1999-11-02	78.000	81.6900	77.3100	80.25	3564600.0
1999-11-01	80.000	80.6900	77.3700	77.62	2487300.0

6378 rows × 5 columns

In [62]:

```
stock_data['AMZN']
```

Out[62]:

date	1. open	2. high	3. low	4. close	5. volume
2025-03-11	193.90	200.1800	193.4000	196.59	52302854.0
2025-03-10	195.60	196.7300	190.8500	194.54	61829231.0
2025-03-07	199.49	202.2653	192.5300	199.25	59802821.0
2025-03-06	204.40	205.7700	198.3015	200.70	49863755.0
2025-03-05	204.80	209.9800	203.2600	208.36	38610085.0
...	...	...	...	...	...
1999-11-05	64.75	65.5000	62.2500	64.94	11091400.0
1999-11-04	67.19	67.1900	61.0000	63.06	16759200.0
1999-11-03	68.19	68.5000	65.0000	65.81	10772100.0
1999-11-02	69.75	70.0000	65.0600	66.44	13243200.0
1999-11-01	68.06	71.8800	66.3100	69.13	12824100.0

6378 rows × 5 columns

In [63]:

```
stock_final_data = pd.DataFrame()
for ticker in tickers:
    stock_final_data[ticker] = stock_data[ticker]['4. close']
```

```python
idx_sort = stock_final_data.sort_values(by="date")
print(idx_sort)
```

```
              AAPL    AMZN
date
1999-11-01   77.62   69.13
1999-11-02   80.25   66.44
1999-11-03   81.50   65.81
1999-11-04   83.62   63.06
1999-11-05   88.31   64.94
...            ...     ...
2025-03-05  235.74  208.36
2025-03-06  235.33  200.70
2025-03-07  239.07  199.25
2025-03-10  227.48  194.54
2025-03-11  220.84  196.59

[6378 rows x 2 columns]
```

In [64]:
```python
type(stock_final_data)
```

Out[64]:
```
pandas.core.frame.DataFrame
```

In [65]:
```python
stock_final_data['AAPL']
```

Out[65]:
```
date
2025-03-11    220.84
2025-03-10    227.48
2025-03-07    239.07
2025-03-06    235.33
2025-03-05    235.74
               ...
1999-11-05     88.31
1999-11-04     83.62
1999-11-03     81.50
1999-11-02     80.25
1999-11-01     77.62
Name: AAPL, Length: 6378, dtype: float64
```

In [66]:
```python
stock_final_data['AMZN']
```

Out[66]:
```
date
2025-03-11    196.59
2025-03-10    194.54
2025-03-07    199.25
2025-03-06    200.70
2025-03-05    208.36
               ...
1999-11-05     64.94
1999-11-04     63.06
1999-11-03     65.81
1999-11-02     66.44
1999-11-01     69.13
Name: AMZN, Length: 6378, dtype: float64
```

In [67]:
```python
stock_final_data.head()
```

Out[67]:

date	AAPL	AMZN
2025-03-11	220.84	196.59
2025-03-10	227.48	194.54
2025-03-07	239.07	199.25
2025-03-06	235.33	200.70
2025-03-05	235.74	208.36

In [68]:

```
stock_final_data.tail()
# how to print rows in between?
```

Out[68]:

date	AAPL	AMZN
1999-11-05	88.31	64.94
1999-11-04	83.62	63.06
1999-11-03	81.50	65.81
1999-11-02	80.25	66.44
1999-11-01	77.62	69.13

In [69]:

```
# Sort the data by 'date'
stock_final_data = stock_final_data.sort_values(by='date')
```

In [70]:

```
stock_final_data.head()
```

Out[70]:

date	AAPL	AMZN
1999-11-01	77.62	69.13
1999-11-02	80.25	66.44
1999-11-03	81.50	65.81
1999-11-04	83.62	63.06
1999-11-05	88.31	64.94

Calculating the Log Returns:

Log Return:

$$Log\_Return_t = Log(Price_t) - Log(Price_{t-1})$$

Python code: log_ret = np.log(df) - np.log(df.shift(1))

For more details, please refer to: https://www.allquant.co/post/magic-of-log-returns-concept-part-1 and https://www.allquant.co/post/magic-of-log-returns-practical-part-2

In [71]:
```
# Method 1:
stock_log_ret = np.log(stock_final_data) - np.log(stock_final_data.shift(1))
stock_log_ret
```

Out[71]:

date	AAPL	AMZN
1999-11-01	NaN	NaN
1999-11-02	0.033322	-0.039690
1999-11-03	0.015456	-0.009527
1999-11-04	0.025680	-0.042685
1999-11-05	0.054571	0.029377
...	...	...
2025-03-05	-0.000806	0.022128
2025-03-06	-0.001741	-0.037456
2025-03-07	0.015768	-0.007251
2025-03-10	-0.049694	-0.023923
2025-03-11	-0.029624	0.010483

6378 rows × 2 columns

$$Log\_Return_t = Log(Price_t) - Log(Price_{t-1}) = Log(Price_t/Price_{t-1})$$

In [72]:
```
# Method 2:
stock_log_ret = np.log(stock_final_data/stock_final_data.shift(1))
```

In [73]:
```
stock_log_ret
```

Out[73]:

date	AAPL	AMZN
1999-11-01	NaN	NaN
1999-11-02	0.033322	-0.039690
1999-11-03	0.015456	-0.009527
1999-11-04	0.025680	-0.042685
1999-11-05	0.054571	0.029377
...	...	...
2025-03-05	-0.000806	0.022128
2025-03-06	-0.001741	-0.037456
2025-03-07	0.015768	-0.007251
2025-03-10	-0.049694	-0.023923
2025-03-11	-0.029624	0.010483

6378 rows × 2 columns

Calculating Expected Return

Realized returns are often used as a proxy for expected returns. The use of average realized returns as a prxoy for expected returnes relies on a belief that information surprises tend to cancel out over the period of the study and realized returns are therefore an unbiased estimate of expected returns.

In [74]:

```
# Daily Expected Return (i.e., mean of returns)
aapl_er = stock_log_ret['AAPL'].mean()
print("The daily Expected Return is "+ str(aapl_er*100) + '%')
```

```
The daily Expected Return is 0.016396633706681492%
```

In [75]:

```
# Daily Expected Return (i.e., mean of returns)
amzn_er = stock_log_ret['AMZN'].mean()
print (str(amzn_er * 100) +'%')
```

```
0.016389078748037383%
```

Annualized Expected Return:

What is annualized return?

Annualized return: Yearly rate of return inferred from any time period.

(1) The annualized return is the return that an investment earns each year for a given period.

(2) It is useful when comparing investments with different lengths of time.

Since we are using log returns, we do not need to compound it as log returns are already continuously compounded. We just need to multiply by the # of tradings days (assuming 252 trading days per year).

https://www.nyse.com/publicdocs/Trading_Days.pdf

In [76]:
```
# Annualized return for Apple
aapl_ann_ret = aapl_er * 252
print ('Annualized return is ' + str(aapl_ann_ret*100)+' %')
```

```
Annualized return is 4.131951694083736 %
```

In [77]:
```
# Annualized return for Amazon
amzn_ann_ret = amzn_er * 252
print('Annualized return is ' + str(amzn_ann_ret*100)+' %')
```

```
Annualized return is 4.1300478445054205 %
```

Now let's work on portfolio

Learn a new Python function that we will use: "np.array"

Know more about the "array" function in Numpy:

np.array: Create an array

What is an array? https://2.bp.blogspot.com/-TUYyIovFJXc/VhU8CxS68tI/AAAAAAAAD6o/EbIM_W5YdPs/w1200-h630-p-k-no-nu/What%2Bis%2Bin%2Barray.jpg

In [78]:
```
# Example of np.array
import numpy as np
np.array([0, 1, 2])
```

Out[78]:
```
array([0, 1, 2])
```

Now let's use "np.array" to calculate expected return of a portfolio

In [79]:
```
# Cacluated Expected Return of a Portfolio
# Assuming an equally weighted portfolio
portfolio_weights = np.array([0.5, 0.5])
```

Expected return of a portfolio

$$Expected\ Return\ of\ Portfolio = \sum Expected\ Return\ of\ Stock_i * Weight_i$$

In [80]:
```
expected_return = np.sum( (stock_log_ret.mean() * portfolio_weights))
expected_return
```

Out[80]:
```
np.float64(0.00016392856227359436)
```

In [81]:
```
# Annualized return
ann_return = expected_return*252
ann_return
```

Out[81]:
```
np.float64(0.04130999769294578)
```

Calculate daily return of a portfolio

In [82]:
```
stock_port_ret = (stock_log_ret*portfolio_weights)
```

In [83]:
```
stock_port_ret
```

Out[83]:

	AAPL	AMZN
date		
1999-11-01	NaN	NaN
1999-11-02	0.016661	-0.019845
1999-11-03	0.007728	-0.004764
1999-11-04	0.012840	-0.021343
1999-11-05	0.027285	0.014689
...	...	...
2025-03-05	-0.000403	0.011064
2025-03-06	-0.000870	-0.018728
2025-03-07	0.007884	-0.003625
2025-03-10	-0.024847	-0.011961
2025-03-11	-0.014812	0.005241

6378 rows × 2 columns

In [84]:
```
# df.loc[:,'New_Column'] = 'value' - You can use '.loc' with ':' to add a sp
# https://www.re-thought.com/blog/how-to-add-new-columns-in-a-dataframe-in-p

stock_port_ret.loc[:,'Portfolio']= stock_port_ret.sum(axis=1)

"""
pandas.DataFrame.sum(axis=1):
to find the sum of all rows in DataFrame;
```

```
axis=1 specifies that the sum will be done on the rows.
"""
```

Out[84]:

```
'\npandas.DataFrame.sum(axis=1):\nto find the sum of all rows in DataFrame;
\naxis=1 specifies that the sum will be done on the rows.\n'
```

In [85]:

```
stock_port_ret
```

Out[85]:

date	AAPL	AMZN	Portfolio
1999-11-01	NaN	NaN	0.000000
1999-11-02	0.016661	-0.019845	-0.003184
1999-11-03	0.007728	-0.004764	0.002964
1999-11-04	0.012840	-0.021343	-0.008503
1999-11-05	0.027285	0.014689	0.041974
...	...	...	...
2025-03-05	-0.000403	0.011064	0.010661
2025-03-06	-0.000870	-0.018728	-0.019598
2025-03-07	0.007884	-0.003625	0.004258
2025-03-10	-0.024847	-0.011961	-0.036808
2025-03-11	-0.014812	0.005241	-0.009571

6378 rows × 3 columns

In [86]:

```
ticker ='DIA'
filename = ticker + '.csv'
data, meta_data = ts.get_intraday(symbol=ticker, outputsize='full')
data.to_csv(filename)
data
```

Out[86]:

date	1. open	2. high	3. low	4. close	5. volume
2025-03-11 20:00:00	414.7400	414.7400	414.7400	414.7400	47945.0
2025-03-11 19:45:00	415.3000	415.6500	415.3000	415.6500	299.0
2025-03-11 19:30:00	415.2100	415.2500	415.1010	415.2500	11.0
2025-03-11 19:15:00	415.4600	415.4800	415.3150	415.4050	50.0
2025-03-11 19:00:00	415.5600	415.5888	415.4910	415.4910	156.0
...	...	...	...	...	...
2025-02-10 05:00:00	443.8265	443.8765	443.7466	443.7466	76.0
2025-02-10 04:45:00	443.6867	443.6867	443.6867	443.6867	5.0
2025-02-10 04:30:00	443.7766	443.7866	443.7267	443.7367	159.0
2025-02-10 04:15:00	443.6967	443.7766	443.6967	443.7766	33.0
2025-02-10 04:00:00	443.0277	443.6468	443.0277	443.6468	446.0

1346 rows × 5 columns

In [87]:

```
market_return = data.sort_values(by='date')
market_return = market_return['4. close']
market_return = np.log(market_return/market_return.shift(1))
market_return
```

Out[87]:

```
date
2025-02-10 04:00:00         NaN
2025-02-10 04:15:00    0.000293
2025-02-10 04:30:00   -0.000090
2025-02-10 04:45:00   -0.000113
2025-02-10 05:00:00    0.000135
                         ...
2025-03-11 19:00:00   -0.000190
2025-03-11 19:15:00   -0.000207
2025-03-11 19:30:00   -0.000373
2025-03-11 19:45:00    0.000963
2025-03-11 20:00:00   -0.002192
Name: 4. close, Length: 1346, dtype: float64
```

In [88]:

```
market_return = market_return.drop(market_return.index[0])
market_return
```

Out[88]:
```
date
2025-02-10 04:15:00    0.000293
2025-02-10 04:30:00   -0.000090
2025-02-10 04:45:00   -0.000113
2025-02-10 05:00:00    0.000135
2025-02-10 05:15:00   -0.000022
                          ...
2025-03-11 19:00:00   -0.000190
2025-03-11 19:15:00   -0.000207
2025-03-11 19:30:00   -0.000373
2025-03-11 19:45:00    0.000963
2025-03-11 20:00:00   -0.002192
Name: 4. close, Length: 1345, dtype: float64
```

In [89]:
```
# Add the market return into stock_port_ret
stock_port_ret['DIA'] = market_return
```

In [90]:
```
stock_port_ret = stock_port_ret.drop(stock_port_ret.index[0])
```

In [91]:
```
stock_port_ret
```

Out[91]:

date	AAPL	AMZN	Portfolio	DIA
1999-11-02	0.016661	-0.019845	-0.003184	NaN
1999-11-03	0.007728	-0.004764	0.002964	NaN
1999-11-04	0.012840	-0.021343	-0.008503	NaN
1999-11-05	0.027285	0.014689	0.041974	NaN
1999-11-08	0.043671	0.091623	0.135293	NaN
...	...	...	...	...
2025-03-05	-0.000403	0.011064	0.010661	NaN
2025-03-06	-0.000870	-0.018728	-0.019598	NaN
2025-03-07	0.007884	-0.003625	0.004258	NaN
2025-03-10	-0.024847	-0.011961	-0.036808	NaN
2025-03-11	-0.014812	0.005241	-0.009571	NaN

6377 rows × 4 columns

Risk Measure: Standard Deviation

Variance

In [92]:
```python
# Variance of a Single Stock
aapl_variance = stock_log_ret['AAPL'].var()
print(aapl_variance)
```

```
0.001640567273592953
```

Standard Deviation

In [93]:
```python
# Method 1:
np.sqrt(aapl_variance)
```

Out[93]:
```
np.float64(0.040503916768541696)
```

In [94]:
```python
# Method 2:
stock_log_ret['AAPL'].std()
```

Out[94]:
```
np.float64(0.040503916768541696)
```

Variance of the Portfolio of Stocks

$$Variance = (Weight_1)^2 * Var_1 + (Weight_2)^2 * Var_2 + 2 * Weight_1 * Weight$$

Method 1 :

In [95]:
```python
portfolio_weights
```

Out[95]:
```
array([0.5, 0.5])
```

In [96]:
```python
stock_log_ret.cov()
```

Out[96]:

	AAPL	AMZN
AAPL	0.001641	0.000281
AMZN	0.000281	0.002347

In [118…
```python
#df.loc['row_label', 'column_label']
stock_log_ret.cov().loc['AAPL','AMZN']
```

Out[118…
```
np.float64(0.0002807111908117361)
```

In [119…
```python
# variance of portfolio of 2 assets
# = (weight_1)^2*var_1 + (weight_2)^2*var_2 + 2*weight_1*weight_2*cov_12

port_var = portfolio_weights[0]**2 * stock_log_ret['AAPL'].var() + portfolio
# loc: Access a group of rows and columns.
```

In [120…
```python
port_var
```

Out[120… np.float64(0.0011371754695984257)

In [121…
```
# Annual variance
print (port_var*252)
```

0.2865682183388033

In [122…
```
# Standard deviation
port_std = np.sqrt(port_var*252)
print(port_std)
```

0.5353206687012966

(Optional) Method 2 :

Cacluating using Matrixes, so you could easily use this for any number of assets.

https://community.wolfram.com/c/portal/getImageAttachment?filename=var_covar_formula.gif&userId=196586

portfolio variance = weight_vector' * cov matrix * weight_vector

Learn a new Python function that we will use: "np.dot"

Explain np.dot:

What is dot product in Math: https://en.wikipedia.org/wiki/Dot_product

In Python, np.dot: product of two arrays. To compute dot product of numpy arrays, you can use numpy.dot() function.

For more information, please see https://numpy.org/doc/stable/reference/generated/numpy.dot.html

In [123…
```
# Example of np.dot

import numpy as np

# Create two arrays
A = np.array([2, 1, 5, 4])
B = np.array([3, 4, 7, 8])

# dot product
output = np.dot(A, B)

print(output)
```

77

In [103…
```
"""
output = [2, 1, 5, 4].[3, 4, 7, 8]
       = 2*3 + 1*4 + 5*7 + 4*8
```

```
      = 77
"""
```

Out[103…

```
'\noutput = [2, 1, 5, 4].[3, 4, 7, 8]\n       = 2*3 + 1*4 + 5*7 + 4*8\n
= 77\n'
```

Now let's use "np.dot" to calculate variance of a portfolio

$$Variance = (Weight_1)^2 * Var_1 + (Weight_2)^2 * Var_2 + 2 * Weight_1 * Weigh$$

In [104…

```
stock_log_ret.cov()
```

Out[104…

	AAPL	AMZN
AAPL	0.001641	0.000281
AMZN	0.000281	0.002347

In [124…

```
portfolio_weights
```

Out[124…

```
array([0.5, 0.5])
```

In [128…

```
np.dot(portfolio_weights, stock_log_ret.cov())
portfolio_weights.T
```

Out[128…

```
array([0.5, 0.5])
```

In [106…

```
# port_var_mat = np.dot(np.dot(A, B), A.T)
# numpy.T, Returns an array with axes transposed, View of the transposed arr
# https://numpy.org/doc/stable/reference/generated/numpy.ndarray.T.html
port_var_mat = np.dot(np.dot(portfolio_weights, stock_log_ret.cov()), portfo
print (port_var_mat)
```

```
0.0011371754695984246
```

In [107…

```
# Alternative:
port_var_mat = np.dot(portfolio_weights.T, np.dot(stock_log_ret.cov(), portf
print (port_var_mat)
```

```
0.0011371754695984246
```

In [108…

```
# Annual variance
print (port_var*252)
```

```
0.2865682183388033
```

In [109…

```
# Standard deviation
np.sqrt(port_var_mat*252)
```

Out[109…

```
np.float64(0.5353206687012962)
```

Risk Measure: Beta (optional, for advanced students, not to cover in class)

Beta

We have a portfolio called "p", which includes Apple and Amazon. We have a market index called "m". B_p is Beta for Portfolio p. Using CAPM, we have the following

$$E(R_p) = R_f + B_p[E(R_m) - R_f]$$

That is, Expected Return of Portfolio = Risk-free Rate + Beta*(Expected Return of Market - Risk-free Rate)

Therefore we can cacluate Beta B_p using

$$B_p = [E(R_p) - R_f]/[E(R_m) - R_f]$$

Using regressions to obtain Beta

To be simple, let's assume $R_f = 0$ for now:
$E(R_p) = R_f + B_p[E(R_m) - R_f] = B_p * E(R_m)$

Now let's use a simple linear regression: $Y = \alpha + \beta X + \epsilon$, where Y is R_p, X is R_m, and β is B_p.

So we will run a regression below:

$$R_p = \alpha + B_p * R_m + \epsilon$$

Run regressions in Python: use statsmodels module

statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration.

https://www.statsmodels.org/stable/install.html

In [110…

```
# Install statsmodels
%pip install statsmodels
```

```
Requirement already satisfied: statsmodels in /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages (from statsmodels) (2.2.3)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages (from statsmodels) (1.15.2)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages (from statsmodels) (2.2.3)
Requirement already satisfied: patsy>=0.5.6 in /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages (from statsmodels) (1.0.1)
Requirement already satisfied: packaging>=21.3 in /Users/isaackaiyuilee/Library/Python/3.13/lib/python/site-packages (from statsmodels) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/isaackaiyuilee/Library/Python/3.13/lib/python/site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2025.1)
Requirement already satisfied: six>=1.5 in /Users/isaackaiyuilee/Library/Python/3.13/lib/python/site-packages (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4->statsmodels) (1.17.0)

[notice] A new release of pip is available: 24.2 -> 25.0.1
[notice] To update, run: pip3 install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [129…

```python
# import statsmodels API
import statsmodels.api as sm

"""
statsmodels.api: Cross-sectional models and methods.
The API focuses on models and the most frequently used statistical test,
and tools.
Canonically imported using import statsmodels.api as sm.
"""
```

Out[129…

```
'\nstatsmodels.api: Cross-sectional models and methods.\nThe API focuses on models and the most frequently used statistical test, \nand tools.\nCanonically imported using import statsmodels.api as sm.\n'
```

Regression Code:

In [130…

```python
# X is the market index return
X = stock_port_ret['AAPL']

# y is the portfolio return
y = stock_port_ret['Portfolio']
```

```python
# Add a constant in the regression model
# An intercept is not included by default and should be added by the user
X1 = sm.add_constant(X)

# Regression model
# OLS: Ordinary Least Squares
model = sm.OLS(y,X1)

# Fit the model and print results
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:              Portfolio   R-squared:                       0.4
95
Model:                            OLS   Adj. R-squared:                  0.4
95
Method:                 Least Squares   F-statistic:                     624
0.
Date:                Wed, 12 Mar 2025   Prob (F-statistic):               0.
00
Time:                        10:05:13   Log-Likelihood:                 1474
4.
No. Observations:                6377   AIC:                       -2.948e+
04
Df Residuals:                    6375   BIC:                       -2.947e+
04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
==
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const       6.792e-05      0.000      0.226      0.821      -0.001       0.0
01
AAPL           1.1711      0.015     78.994      0.000       1.142       1.2
00
==============================================================================
==
Omnibus:                    18145.248   Durbin-Watson:                   1.9
86
Prob(Omnibus):                  0.000   Jarque-Bera (JB):      1446140689.3
22
Skew:                         -37.347   Prob(JB):                         0.
00
Kurtosis:                    2334.739   Cond. No.                         4
9.4
==============================================================================
==

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```

In [113…
```python
print(results.params)
```

```
const    0.000068
AAPL     1.171106
dtype: float64
```

In [114…
```python
beta = results.params[1]
print('Beta is ' + str(beta))
```

```
Beta is 1.1711061748763039
```

```
/var/folders/qz/xwx1r5sx2k9dgf6nxmgph3200000gn/T/ipykernel_5445/1074429198.p
y:1: FutureWarning: Series.__getitem__ treating keys as positions is depreca
ted. In a future version, integer keys will always be treated as labels (con
sistent with DataFrame behavior). To access a value by position, use `ser.il
oc[pos]`
  beta = results.params[1]
```

In [115…
```
print('R2: ', results.rsquared)
```

```
R2:  0.4946512507246904
```

In [ ]:
```

```

Sharpe Ratio

Assume the risk-free rate is 0%

Formula and Calculation for Sharpe Ratio

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p}$$

$$\textbf{where:}$$
$$R_p = \text{return of portfolio}$$
$$R_f = \text{risk-free rate}$$
$$\sigma_p = \text{standard deviation of the portfolio's excess return}$$

In [116…
```
sharpe = (ann_return - 0)/port_std
print(sharpe)
```

```
0.07716869552816824
```

Treynor Ratio

The Formula for the Treynor Ratio is:

$$\text{Treynor Ratio} = \frac{R_p - R_f}{\beta_p}$$

$$\textbf{where:}$$
$$R_p = \text{Portfolio return}$$
$$R_f = \text{Risk-free rate}$$
$$\beta_p = \text{Beta of the portfolio}$$

In [117…
```
treynor = (ann_return - 0)/beta
print(treynor)
```

```
0.03527434025980529
```

```
In [ ]:
```