Annualized returns

INTRODUCTION TO PORTFOLIO ANALYSIS IN PYTHON

Charlotte Werger
Data Scientist

Comparing returns

1. *Annual Return*: Total return earned over a period of one calendar year
2. *Annualized return*: Yearly **rate of return** inferred from any time period
3. *Average Return*: Total return realized over a longer period, spread out evenly over the (shorter) periods.
4. *Cumulative (compounding) return*: A return that includes the compounded results of re-investing interest, dividends, and capital gains.

Why annualize returns?

Average return versus annualized return		
	Value of the portfolio	Return
Year 1	$100	
Year 2	$200	100%
Year 3	$100	-50%

- Average return = (100 - 50) / 2 = 25%
- Actual return = 0% so average return is not a good measure for performance!
- How to compare portfolios with different time lengths?
- How to account for compounding effects over time?

Calculating annualized returns

- **N in years:** $rate = (1 + Return)^{1/N} - 1$
- **N in months:** $rate = (1 + Return)^{12/N} - 1$
- Convert any time length to an annual rate:
- **Return** is the total return you want to annualize.
- **N** is number of periods so far.

Annualized returns in python

```
# Check the start and end of timeseries
apple_price.head(1)
```

```
date
2015-01-06    105.05
Name: AAPL, dtype: float64
```

```
apple_price.tail(1)
```

```
date
2018-03-29    99.75
Name: AAPL, dtype: float64
```

```
# Assign the number of months
months = 38
```

Annualized returns in python

```python
# Calculate the total return
total_return = (apple_price[-1] - apple_price[0]) /
                apple_price[0]

print (total_return)
```

```
0.5397420653068692
```

Annualized returns in python

```
# Calculate the annualized returns over months
annualized_return=((1 + total_return)**(12/months))-1
print (annualized_return)
```

```
0.1460250148270876З
```

```
# Select three year period
apple_price = apple_price.loc['2015-01-01':'2017-12-31']
apple_price.tail(3)
```

```
date
2017-12-27    170.60
2017-12-28    171.08
2017-12-29    169.23
Name: AAPL, dtype: float64
```

Annualized return in Python

```python
# Calculate annualized return over 3 years
annualized_return = ((1 + total_return)**(1/3))-1

print (annualized_return)
```

```
0.1567672968419047
```

Let's practice!

Risk adjusted returns

INTRODUCTION TO PORTFOLIO ANALYSIS IN PYTHON

Charlotte Werger
Data Scientist

Choose a portfolio

Portfolio 1

- Annual return of 14%
- Volatility (standard deviation) is 8%

Portfolio 2

- Annual return of 6%
- Volatility is 3%

Risk adjusted return

- It defines an investment's return by measuring how much risk is involved in producing that return
- It's usually a ratio
- Allows you to objectively compare across different investment options
- Tells you whether the return justifies the underlying risk

Sharpe ratio

- Sharpe ratio is the most commonly used risk adjusted return ratio
- It's calculated as follows:
- $Sharpe\ ratio = \frac{R_p - R_f}{\sigma_p}$
- Where: R_p is the portfolio return, R_f is the risk free rate and σ_p is the portfolio standard deviation
- Remember the formula for the portfolio σ_p?
- $\sigma_p = \sqrt{(}Weights\ transposed(Covariance\ matrix\ *\ Weights))$

Annualizing volatility

- Annualized standard deviation is calculated as follows: $\sigma_a = \sigma_m * \sqrt{T}$
- σ_m is the measured standard deviation
- σ_a is the annualized standard deviation
- T is the number of data points per year
- Alternatively, when using variance instead of standard deviation; $\sigma_a^2 = \sigma_m^2 * T$

Calculating the Sharpe Ratio

```
# Calculate the annualized standard deviation
annualized_vol = apple_returns.std()*np.sqrt(250)
print (annualized_vol)
```

```
0.2286248397870068
```

```
# Define the risk free rate
risk_free = 0.01
```

```
# Calcuate the sharpe ratio
sharpe_ratio = (annualized_return - risk_free) / annualized_vol
print (sharpe_ratio)
```

```
0.6419569149994251
```

Which portfolio did you choose?

Portfolio 1

- Annual return of 14%
- Volatility (standard deviation) is 8%
- Sharpe ratio of 1.75

Portfolio 2

- Annual return of 6%
- Volatility is 3%
- Sharpe ratio of 2

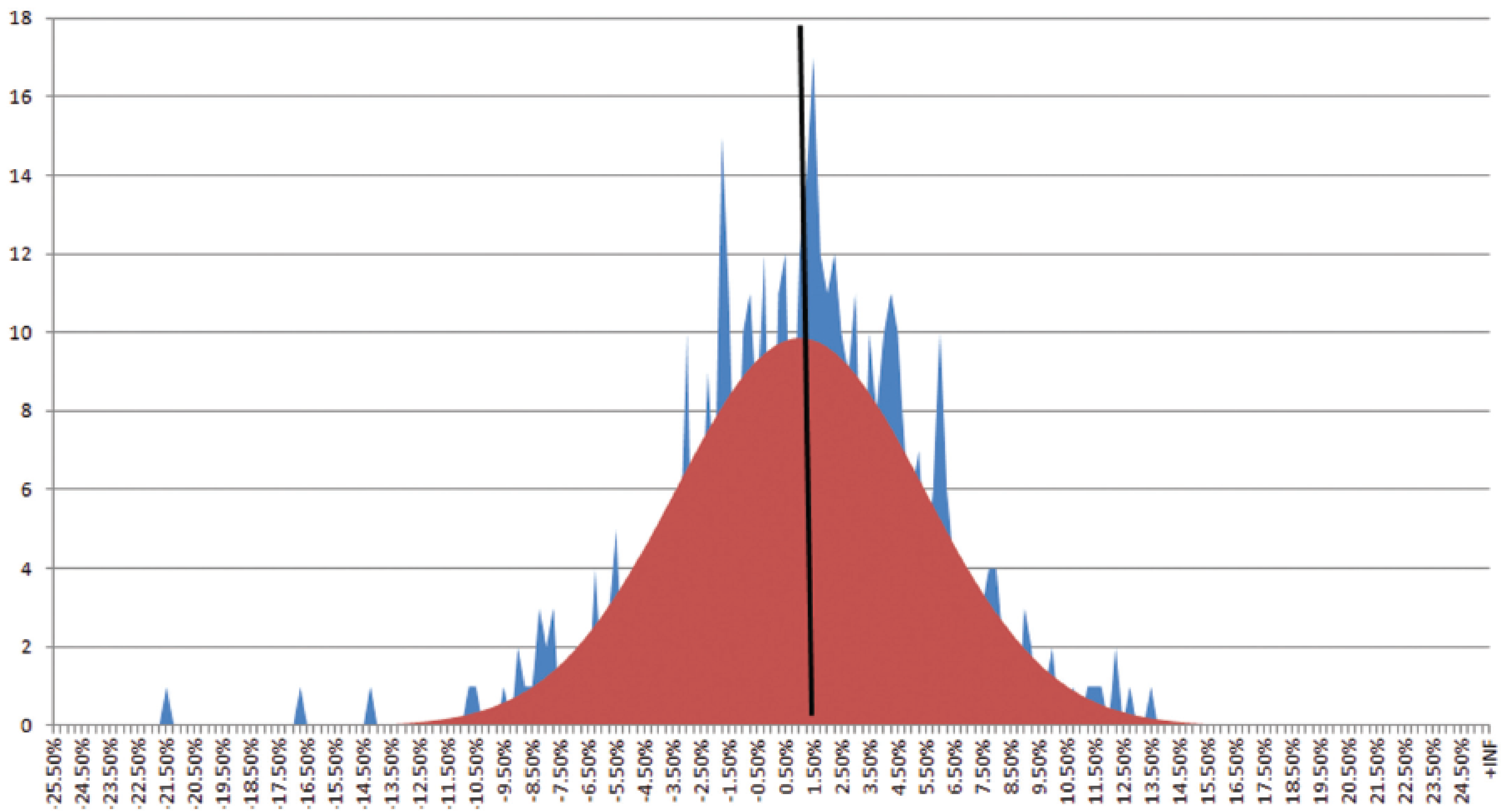
Let's practice!

Non-normal distribution of returns

INTRODUCTION TO PORTFOLIO ANALYSIS IN PYTHON

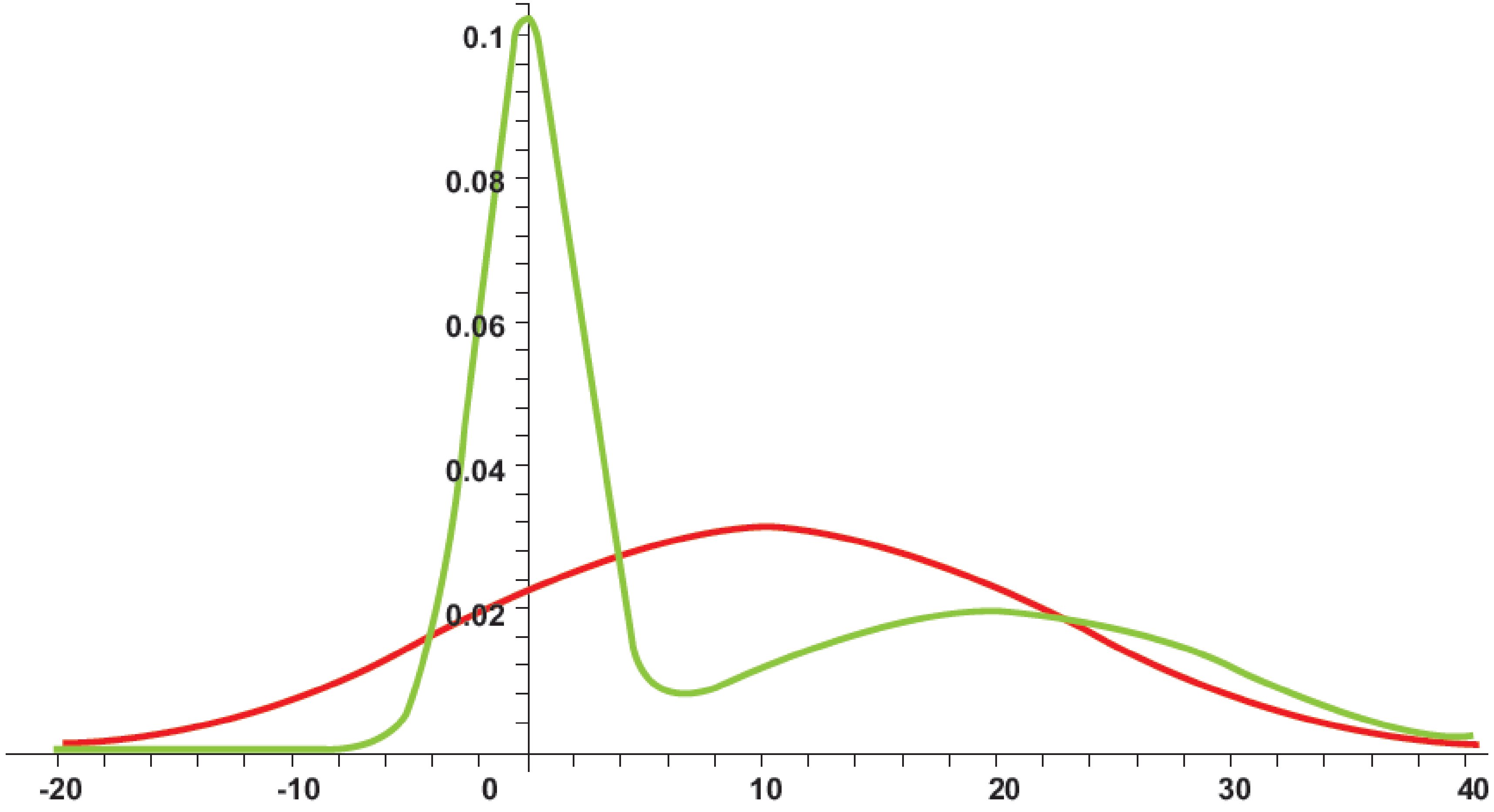
Charlotte Werger
Data Scientist

In a perfect world returns are distributed normally



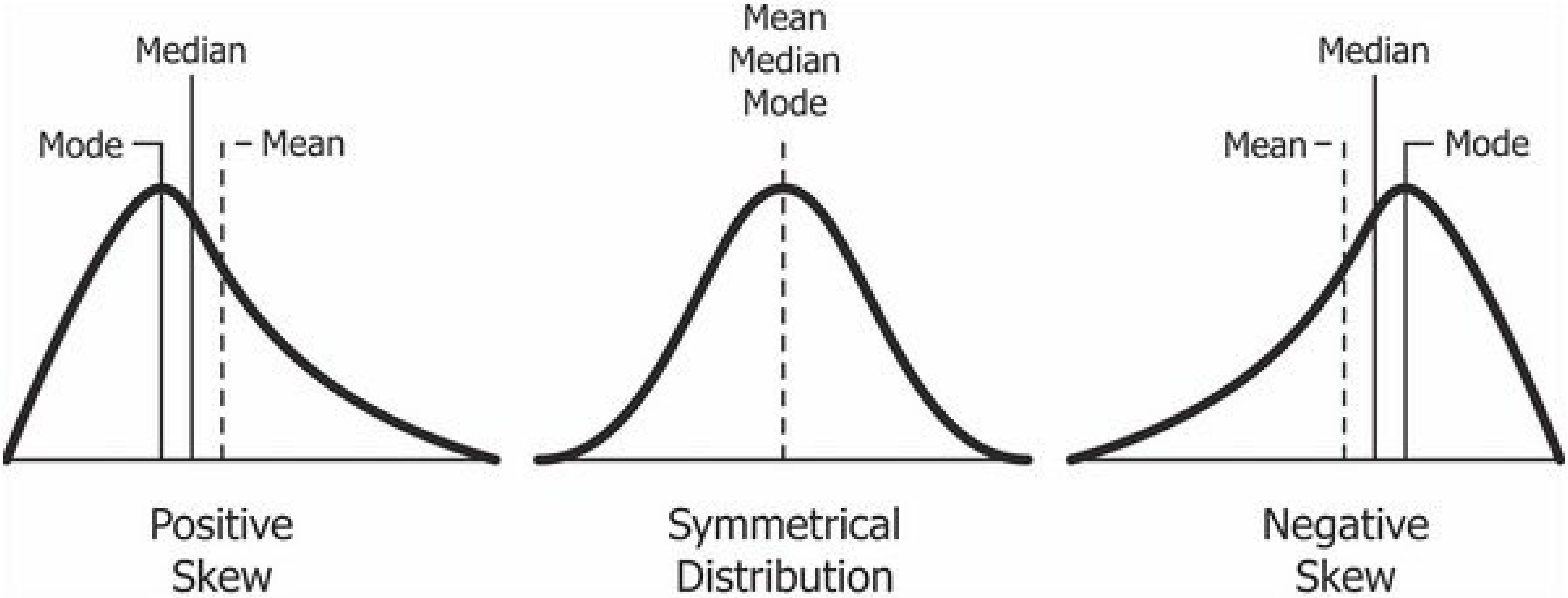
[1] Source: Distribution of monthly returns from the S&P500 from evestment.com

But using mean and standard deviations can be deceiving



[1] Source: “An Introduction to Omega, Con Keating and William Shadwick, The Finance Development Center, 2002

Skewness: leaning towards the negative



Pearson's Coefficient of Skewness

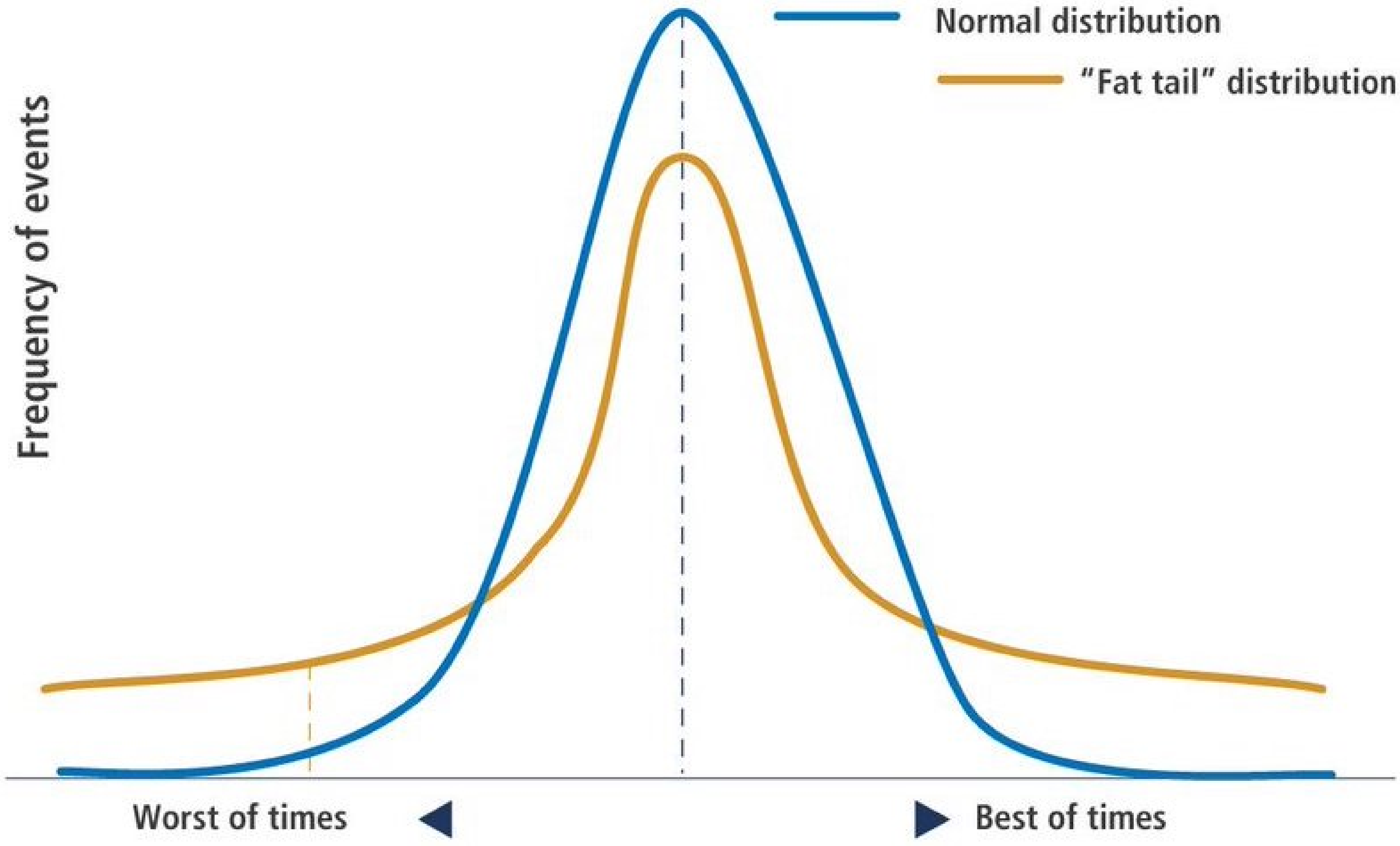
$$Skewness = \frac{3(mean - median)}{\sigma}$$

Rule of thumb:

- $Skewness < -1$ or $Skewness > 1 \Rightarrow$ Highly skewed distribution
- $-1 < Skewness < -0.5$ or $0.5 < Skewness < 1 \Rightarrow$ Moderately skewed distribution
- $-0.5 < Skewness < 0.5 \Rightarrow$ Approximately symmetric distribution

[1] Source: https://brownmath.com/stat/shape.htm

Kurtosis: Fat tailed distribution



[1] Source: Pimco

Interpreting kurtosis

"*Higher kurtosis means more of the variance is the result of infrequent extreme deviations, as opposed to frequent modestly sized deviations.*"

- A normal distribution has kurtosis of exactly 3 and is called (mesokurtic)
- A distribution with kurtosis <3 is called platykurtic. Tails are shorter and thinner, and central peak is lower and broader.
- A distribution with kurtosis >3 is called leptokurtic: Tails are longer and fatter, and central peak is higher and sharper (fat tailed)

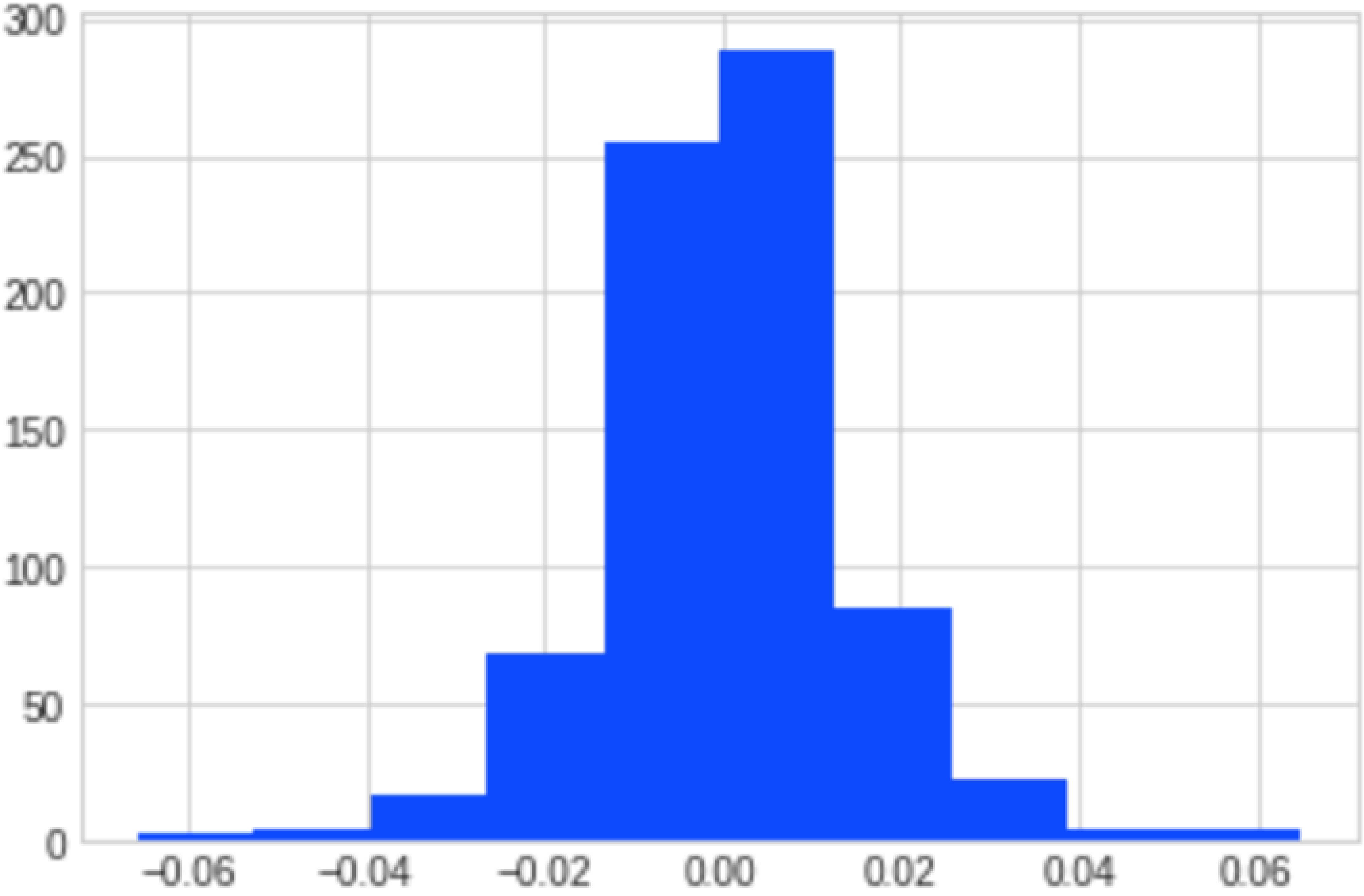
[1] Source: https://brownmath.com/stat/shape.htm

Calculating skewness and kurtosis

```
apple_returns=apple_price.pct_change()
apple_returns.head(3)
```

```
date
2015-01-02          NaN
2015-01-05    -0.028172
2015-01-06     0.000094
Name: AAPL, dtype: float64
```

```
apple_returns.hist()
```


300
250
200
150
100
50
0
-0.06
-0.04
-0.02
0.00
0.02
0.04
0.06

Calculating skewness and kurtosis

```
print("mean : ", apple_returns.mean())
print("vol  : ", apple_returns.std())
print("skew : ", apple_returns.skew())
print("kurt : ", apple_returns.kurtosis())
```

```
mean :  0.0006855391415724799
vol  :  0.014459504468360529
skew :  -0.012440851735057878
kurt :  3.197244607586669
```

Let's practice!

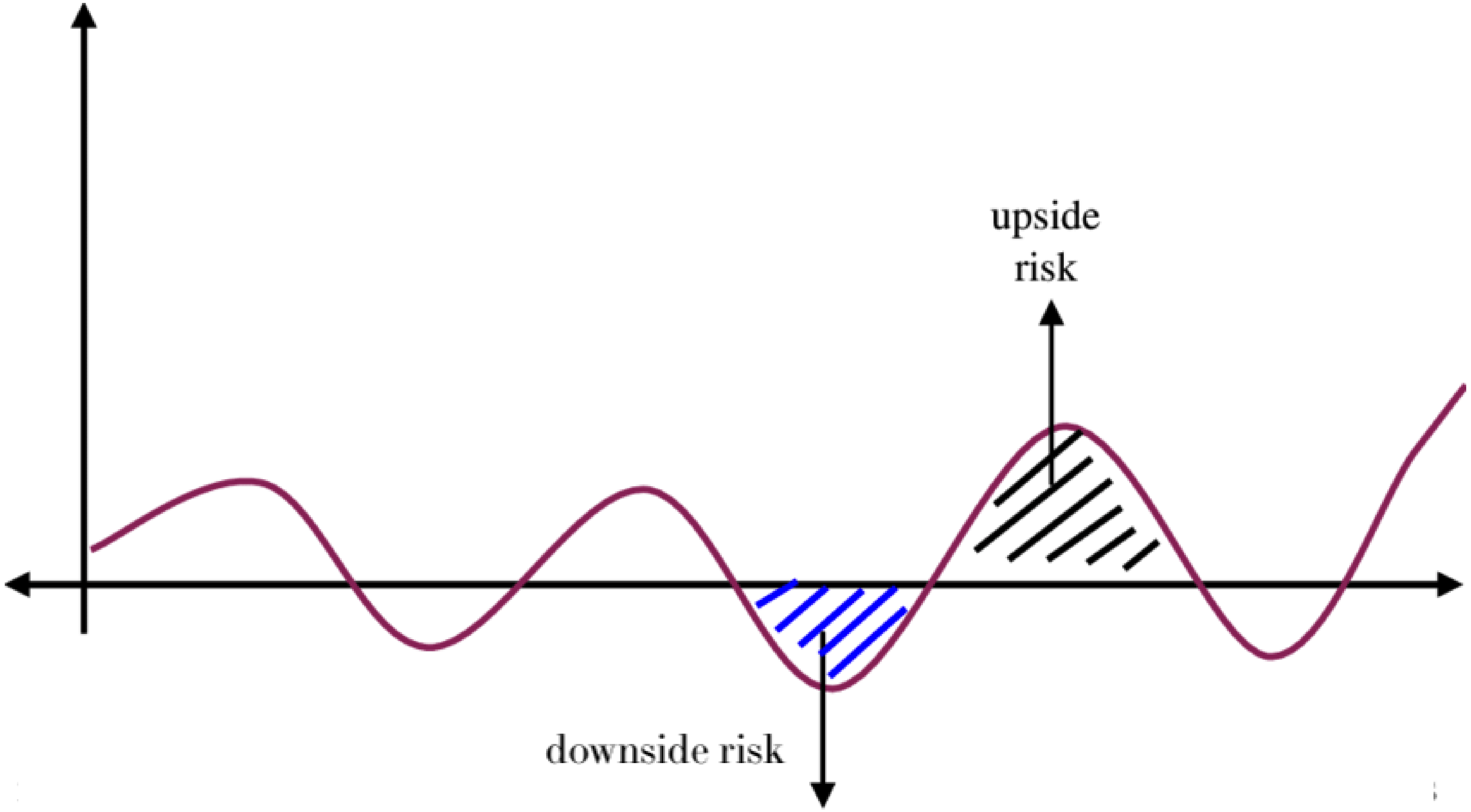
INTRODUCTION TO PORTFOLIO ANALYSIS IN PYTHON

Alternative measures of risk

INTRODUCTION TO PORTFOLIO ANALYSIS IN PYTHON

Charlotte Werger
Data Scientist

Looking at downside risk



- A good risk measure should focus on potential losses i.e. downside risk

Sortino ratio

- Similar to the Sharpe ratio, just with a different standard deviation
- $Sortino\ Ratio = \frac{R_p - R_f}{\sigma_d}$
- σ_d is the standard deviation of the downside.

$$\text{Downside risk} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{return} - \text{target return})^2 f(t)}$$

$$f(t) = 1 \text{ if return} < \text{target return}$$

$$f(t) = 0 \text{ if return} \geq \text{target return}$$

Sortino ratio in python

```
# Define risk free rate and target return of 0
rfr = 0
target_return = 0
```

```
# Calcualte the daily returns from price data
apple_returns=pd.DataFrame(apple_price.pct_change())
```

```
# Select the negative returns only
negative_returns = apple_returns.loc[apple_returns['AAPL'] < target]
```

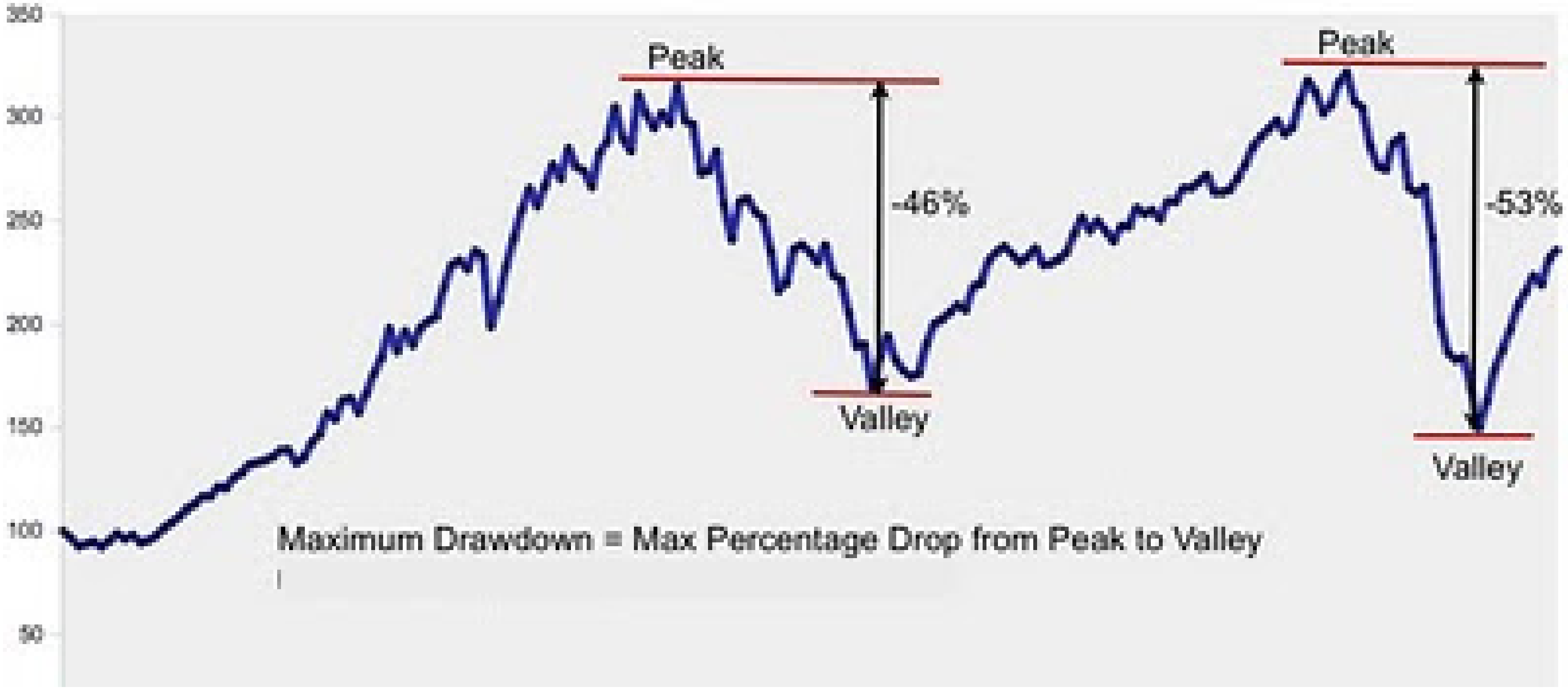
```python
# Calculate expected return and std dev of downside returns
expected_return = apple_returns['AAPL'].mean()
down_stdev = negative_returns.std()
```

```python
# Calculate the sortino ratio
sortino_ratio = (expected_return - rfr)/down_stdev
print(sortino_ratio)
```

```
0.07887683763760528
```

Maximum draw-down

- The largest percentage loss from a market peak to trough
- Dependent on the chosen time window
- The recovery time: time it takes to get back to break-even



Maximum daily draw-down in Python

```
# Calculate the maximum value of returns using rolling().max()
roll_max = apple_price.rolling(min_periods=1,window=250).max()
# Calculate daily draw-down from rolling max
daily_drawdown = apple_price/roll_max - 1.0
# Calculate maximum daily draw-down
max_daily_drawdown = daily_drawdown.rolling(min_periods=1,window=250).min()
# Plot the results
daily_drawdown.plot()
max_daily_drawdown.plot()
plt.show()
```

Maximum draw-down of Apple

Let's practice!