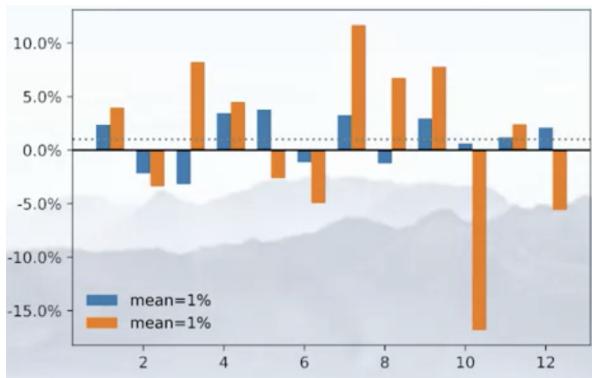


Section 1

Date	
Property	
Weeks	1주차

1. Fundamentals of Returns



Blue Asset : less volatile, less variation

Orange Asset : more volatile, more variation



평균 월간 수익률이 같다하여 최종 수익이 같은 것을 의미하지는 않음

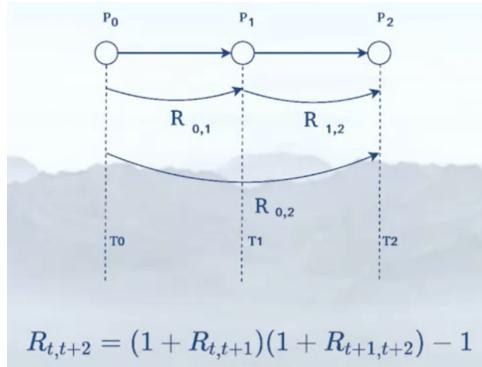
→ 그렇다면 수익에 대한 고민을 할때 어떤 방식을 사용해야 할까?

$$R_{t,t+1} = \frac{P_{t+1} - P_t}{P_t}$$

$$R_{t,t+1} = \frac{P_{t+1} + D_{t,t+1}}{P_t} - 1 = \frac{P_{t+1} + D_{t,t+1} - P_t}{P_t}$$

수익률은 +20% 혹은 1.2로 두가지 방식으로 표현할 수 있음, 1.2의 경우 P_t+1에서 원래의 자산과 수익을 함께 표현할 수 있다는 장점이 있음

1.2 Multi Period Returns



$1 + r$ 의 형태는 이와 같은 복합적인 수익률을 보기에도 유리함

10% 상승 → 10%먹락 → 손해! (단순 덧셈 뺄셈이 아님)

1.3 Annualizing Returns

매달 1%씩 수익을 내는 경우 → 12% 가 아닌 $(1+0.01)^{12} - 1$

2. Lab Session-Basics of returns

여러 기간 내 가격으로부터 수익 계산

how to go from a multi-period return to a compounded return

위에서 봤던 수익률 계산식

$$R_{t,t+1} = \frac{P_{t+1} - P_t}{P_t}$$

or

$$R_{t,t+1} = \frac{P_{t+1}}{P_t} - 1$$

3일동안 연속적으로 일어난 3가지 가격이 있다 가정.

list로 저장한 값끼리는 list가 벡터가 아니기 때문에 계산이 안됨

이제 계산할 수 있는 방법 알아볼 것.

```
▶ prices_a = [8.70, 8.91, 8.71]
[2] prices_a[1:]
↳ [8.91, 8.71]
[3] prices_a[:-1]
↳ [8.7, 8.91]
[4] # python은 이거 예리남
# list가 벡터가 아니어서
# numpy 쓰자 !
prices_a[1:]/prices_a[:-1] -1
↳ -----
TypeError                                         Traceback (most recent call last)
<ipython-input-4-27d68d3aba12> in <module>()
      2 # list가 벡터가 아니어서
      3 # numpy 쓰자 !
----> 4 prices_a[1:]/prices_a[:-1] -1
TypeError: unsupported operand type(s) for /: 'list' and 'list'

SEARCH STACK OVERFLOW
```

1. numpy 행렬로 전환 시 가능

```
[6] import numpy as np
[8] prices_a = np.array([8.70, 8.91, 8.71])
prices_a[1:]/prices_a[:-1] -1
↳ array([ 0.02413793, -0.02244669])
```

2. pandas로 dataframe로 전환 시 가능

```
[9] import pandas as pd
[10] prices = pd.DataFrame({'Blue':[8.70,8.91,8.71,8.43,8.73],
                           'Orange':[10.66,11.08,10.71,11.59,12.11]})

[11] prices
↳      Blue  Orange
0    8.70   10.66
1    8.91   11.08
2    8.71   10.71
3    8.43   11.59
4    8.73   12.11
```

pandas의 iloc 은 dataframe 인덱싱 함수.

하지만 갖고있는 dataframe 형태가 다르면 겹치는 값에 대해서만 계산, 나머지는 NaN값

```
[ 0] prices.iloc[1:][]
    Blue   Orange
1  8.91   11.08
2  8.71   10.71
3  8.43   11.59
4  8.73   12.11

[13] prices.iloc[:-1]
    Blue   Orange
0  8.70   10.66
1  8.91   11.08
2  8.71   10.71
3  8.43   11.59

[21] # index로 정렬된 상태에서 계산 불가
     prices.iloc[1:]/prices.iloc[:-1]

    Blue   Orange
0    NaN    NaN
1    1.0    1.0
2    1.0    1.0
3    1.0    1.0
4    NaN    NaN
```

#그럴때는 .values를 사용해서 dataframe의 값만 추출해서 계산하면 됨.

```
[19] # pulls the values out and gives you back that NumPy array
# 값만 넘파이 행렬로 추출
     prices.iloc[1:]/prices.iloc[:-1].values-1

    Blue   Orange
1  0.024138  0.039400
2 -0.022447 -0.033394
3 -0.032147  0.082166
4  0.035587  0.044866

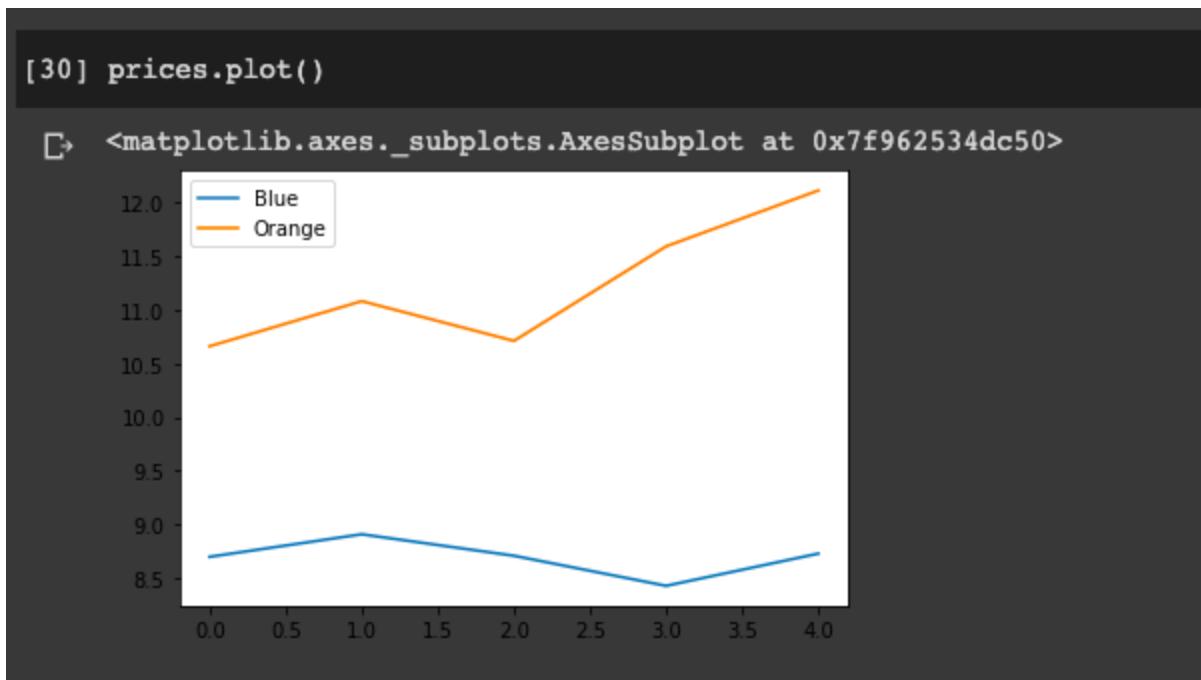
[23] # 이것도 가능
     prices.iloc[1:].values/prices.iloc[:-1]-1

    Blue   Orange
0  0.024138  0.039400
1 -0.022447 -0.033394
2 -0.032147  0.082166
3  0.035587  0.044866
```

똑같이 두 열(Blue,Orange) 인덱스 안에 값이 있을 때만 계산 가능

```
[25] # 첫번째 row 가 NaN인 데이터프레임  
prices.shift(1)  
  
[26] Blue Orange  
0     NaN     NaN  
1    8.70   10.66  
2    8.91   11.08  
3    8.71   10.71  
4    8.43   11.59  
  
[28] # 두 값 모두 index안에 같이 있을 때만 계산 가능  
prices/prices.shift(1) -1  
  
[29] Blue Orange  
0      NaN      NaN  
1  0.024138  0.039400  
2 -0.022447 -0.033394  
3 -0.032147  0.082166  
4  0.035587  0.044866
```

plot으로 price 나타내기



pct_change() 함수 사용하면 퍼센트값 알려줌

```
# percent change
prices.pct_change()
```

	Blue	Orange
0	NaN	NaN
1	0.024138	0.039400
2	-0.022447	-0.033394
3	-0.032147	0.082166
4	0.035587	0.044866

```
[32] %matplotlib inline
```

```
[36] returns = prices.pct_change()
      returns.plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f962393a908>
```

Period	Blue (%)	Orange (%)
0	NaN	NaN
1	0.024138	0.039400
2	-0.022447	-0.033394
3	-0.032147	0.082166
4	0.035587	0.044866

데이터프레임은 벡터니까 연산 가능

```
[ 38] returns.std()
    ↗ Blue      0.033565
    ↗ Orange    0.048328
    ↗ dtype: float64

[ 39] returns.mean()
    ↗ Blue      0.001283
    ↗ Orange    0.033260
    ↗ dtype: float64

[ 40] # 데이터프레임은 벡터여서 연산 가능
      returns+1
```

	Blue	Orange
0	NaN	NaN
1	1.024138	1.039400
2	0.977553	0.966606
3	0.967853	1.082166
4	1.035587	1.044866

np.prod() : 배열 내 원소 간 곱

```
# 모든 값에 1더하고 각 열의 평균  
# np.prod()  
np.prod(returns+1)-1
```

```
[41] ⌞ Blue      0.003448  
     Orange    0.136023  
     dtype: float64
```

```
[42] # 이방법도 됨  
     (returns+1).prod()-1
```

```
[43] ⌞ Blue      0.003448  
     Orange    0.136023  
     dtype: float64
```

annualization

직접 리턴을 구해보자!

수익률 0.01% , 월별 리턴 1년후 수익률?

12%가 아닌 12.68% !

쿼터(분기)별, 일별

```
[45] ## Annualization
```

더블클릭 또는 Enter 키를 눌러 수정

```
[47] # monthly return  
rm = 0.01  
(1+rm)**12 -1
```

```
↳ 0.12682503013196977
```

```
[48] #quarterly returns  
rq = 0.04  
(1+rq)**4 -1
```

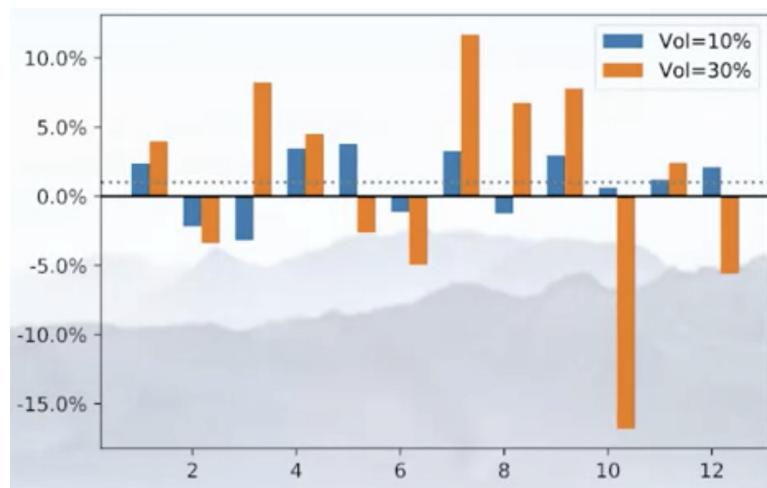
```
↳ 0.16985856000000002
```

```
[50] #daily return  
rd = 0.0001  
(1+rd)**252 - 1
```

```
↳ 0.025518911987694626
```

3. Measures of Risk and Reward

3.1 Volatility : Standard Deviation And Variance



$$\sigma_R^2 = \frac{1}{N} \sum_{i=1}^N (R_i - \bar{R})^2$$

평균에서 얼마나 벗어나 있는가? → Variance

$$\sigma_R = \sqrt{\frac{1}{N} \sum_{i=1}^N (R_i - \bar{R})^2}$$

3.2 Annualizing Volatility

우리는 일간 데이터와 월간 데이터의 변동성(Volatility)을 비교할 수 없다.

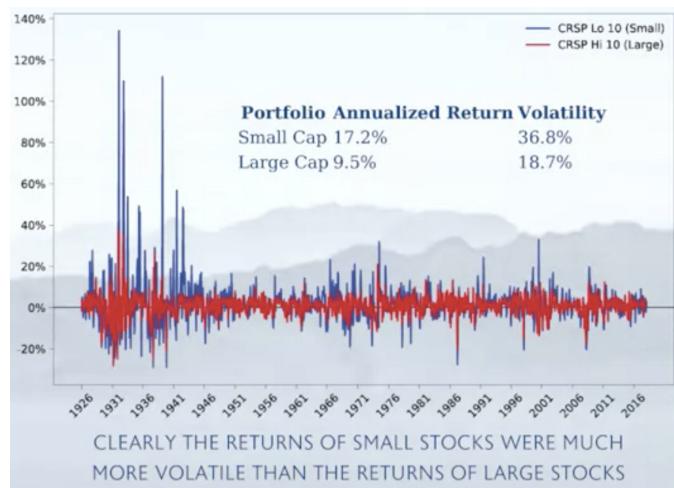
$$\sigma_{ann} = \sigma_p \sqrt{p}$$

p : 영업일(연간 : 252)

▼ The annualized volatility is always greater than the monthly volatility. True or False?

True / Since you are multiplying the monthly volatility by the square root of 12, the result will always be greater than the monthly volatility

3.3 Risk Adjusted Measures



- blue line : US small caps / red line : US large caps
→ small caps are far more volatile

- 그렇다면 위의 두 그래프를 어떻게 비교할 것인가?

- 수치적 단순 비율 비교

Small Caps → $17.2\% / 36.8\% = 0.47$

Large Caps → $9.5\% / 18.7\% = 0.50$

→ Large Caps가 낮은 수익률을 갖고 있지만 변동성에 비한다면 더 나은 수익률임을 알 수 있음

▼ The risk free rate is the return of an investment that carries:

No Risk : Although there is no perfectly riskless investment, the very short term US Treasury Bill (30 days or less) is typically used as a proxy for the risk free rate.

- 초과 이익률을 이용한 비교 (Sharpe Ratio)

$$\text{Return On Risk Ratio} = \frac{\text{Return}}{\text{Volatility}} = \text{Sharpe Ratio}(P) = \frac{R_p - R_f}{\sigma_p}$$

Small Cap → $(17.2\% - 3\%) / 36.8\% = 0.39$

Large Cap → $(9.5\% - 3\%) / 18.7\% = 0.35$

→ Small Cap이 더 나은 수익률을 갖고 있음

▼ If the risk free rate goes up and the return and volatility of a portfolio are unchanged, the Sharpe Ratio ...

Goes down : Since the risk free rate is subtracted from the numerator, and the denominator is unchanged, the Sharpe Ratio goes down

4. Lab Session-Risk Adjusted returns

.std()(Standard Deviations)를 이용하여 Return의 Risk 를 계산할수 있다.

```
[3]: returns.std()
```

```
[3]: BLUE      0.023977
      ORANGE    0.079601
      dtype: float64
```

```
deviations = returns - returns.mean()
squared_deviations = deviations**2
variance = squared_deviations.mean()
import numpy as np
volatility = np.sqrt(variance)
volatility
```

```
BLUE      0.022957
ORANGE    0.076212
dtype: float64
```

직접한 계산과 .std()를 사용한 계산이 다르다.

.std() 는 $n-1$ 을 사용 , 직접하는건 n 을 사용함.

```
number_of_obs = returns.shape[0]
variance = squared_deviations.sum()/(number_of_obs - 1)
volatility = variance**0.5
```

```
volatility
```

```
BLUE      0.023977
ORANGE    0.079601
dtype: float64
```

연간 volatility 는

```
returns.std()*np.sqrt(12)
```

```
BLUE      0.083060  
ORANGE    0.275747  
dtype: float64
```

```
returns.std()*(12**0.5)
```

```
BLUE      0.083060  
ORANGE    0.275747  
dtype: float64
```

```
annualized_vol = returns.std()*np.sqrt(12)  
annualized_vol
```

```
SmallCap   0.368193  
LargeCap   0.186716  
dtype: float64
```

시가 총액이 클수록 변동성이 작다.

```
riskfree_rate = 0.03  
excess_return = annualized_return - riskfree_rate  
sharpe_ratio = excess_return/annualized_vol  
sharpe_ratio
```

```
SmallCap   0.373346  
LargeCap   0.336392  
dtype: float64
```

Sharpe Ratio 를 계산할수도 있다.

5. Measuring Max Drawdown

risk 평가 다른 요소

possibility of losing money → money를 잃는다는것 risk이므로 이 관점에서 살펴볼것.

MAX DRAWDOWN 이란?

정의: 이전 고점 대비 하락 비율

새로운 최고점이 달성되기 전, 포트폴리오의 최고점에서 최저점에 이르는 최대 관측 손실

ex. 포트폴리오가 하나 있다. 초기값은 500,000원.

얼마 후, 포트폴리오의 가치가 750,000원으로 상승

그러고 나서 얼마 후, 400,000원으로 하락장에 떨어짐.

그후, 다시 600,000원으로 상승했다가 350,000원으로 떨어졌다.

마지막으로 지금 가치는 800,000원이다. MDD는 얼마인가?

$$MDD = \frac{\text{최고값} - \text{최저값}}{\text{최고값}}$$

$$= \frac{350,000 - 750,000}{750,000}$$

$$= -53.33\%$$

특성

MDD는 최악의 경우를 상상하는 것이라 할 수 있으므로 risk measure가 됨.

최악의 경우를 상상하는 것이라고 한 이유는

최고점에서 사고, 최저점에서 팔았을때를 가정하였기 때문이다.

THE WORST POSSIBLE RETURN you could have seen

if you "Buy at its highest value, Sold at the bottom."

주의해야할점

1. period 를 어떻게 설정하느냐에 따라서 값이 다르게 나온다. daily/weekly와 monthly데이터의 mdd는 다를것.

2. outlier에 취약하다.

6. Lab Session-Drawdown

```
rets.index = pd.to_datetime(rets.index, format="%Y%m")
rets.index = rets.index.to_period('M')
rets["1975"] #index the data easily when we use time-series data
rets.info()
```

return 데이터를 날짜 형식으로 바꿔서 시계열 데이터(time-series)로 처리하기 쉽게함.



Compute Drawdowns

1. Compute a wealth index - The wealth index is nothing more than the value of a portfolio as it compounds over time.
2. Compute previous peaks - keep track of the cumulative max
3. Compute drawdown - which is the wealth value as a percentage of the previous peak
(The drawdown is whatever wealth I have, right now, minus my previous peak, is the amount that I've lost)

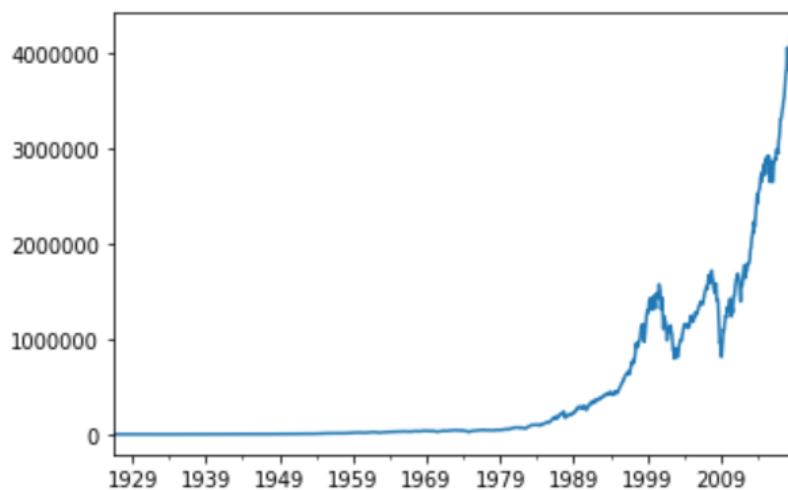
Drawdown을 계산하기 위한 절차는 위와 같다.

1. 먼저 wealth index를 계산한다. 이는 단지 포트폴리오의 값에 계속 수익률을 곱한 것이다.

```
wealth_index = 1000 * (1+rets["LargeCap"]).cumprod()
```

```
wealth_index.plot.line()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28d0edcbc88>
```

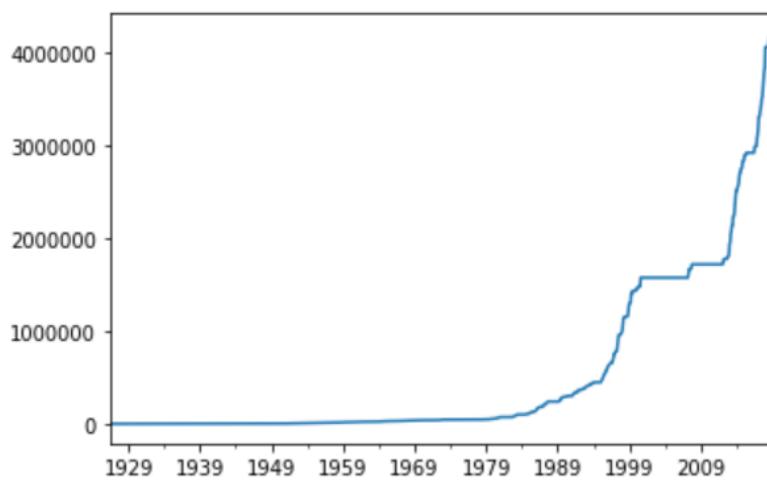


2. 과거의 peaks를 계산한다. 이는 누적 최댓값을 의미한다.

```
previous_peaks = wealth_index.cummax()
```

```
previous_peaks.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28d11187cc8>
```

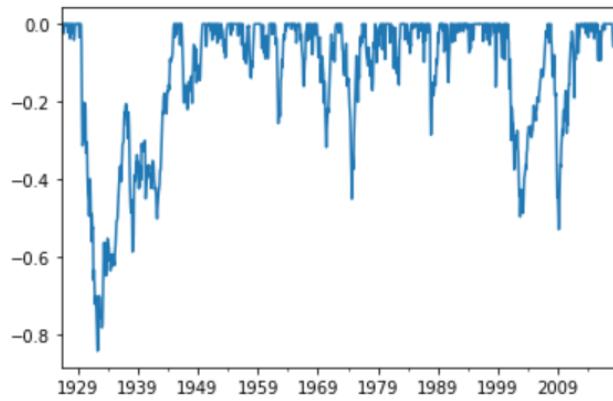


3. Drawdown을 계산한다.

```
drawdown = (wealth_index - previous_peaks) / previous_peaks
```

```
drawdown.plot() # we can see that In 1929 people lost more than 80% of wealth
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28d11d6cec8>
```



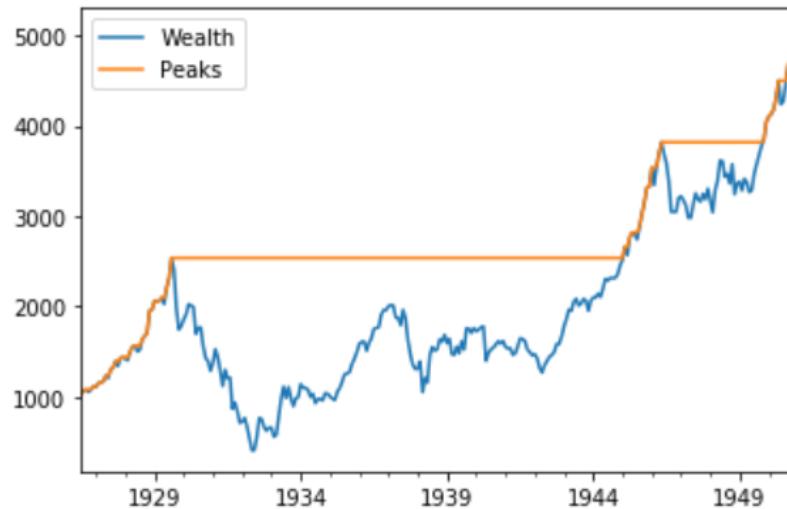
```

def drawdown(return_series: pd.Series):#expect input as pd.series
    """
    Takes a time series of asset returns
    Computes and returns a DataFrame that contains:
    the wealth index
    the previous peaks
    percent drawdowns
    """
    wealth_index = 1000 * (1+return_series).cumprod()
    previous_peaks = wealth_index.cummax()
    drawdowns = (wealth_index - previous_peaks)/previous_peaks
    return pd.DataFrame({
        "Wealth":wealth_index,
        "Peaks":previous_peaks,
        "Drawdown":drawdowns
    })

```

다음과 같이 함수를 직접 만들어서 표현할 수 있다.

```
drawdown(rets[:"1950"]["LargeCap"])[["Wealth", "Peaks"]].plot()
```



drawdown 함수를 통해 1950년대까지 데이터 중 Large Cap에 해당하는 포트폴리오를 지정하고 그 중 Wealth Index와 Previous Peaks에 대한 그래프는 위와 같다.

1929년 대공황 이후 이전 부까지 복귀하는데 약 15년이 걸린 것을 그래프를 통해 확인할 수 있으며 그 시점까지의 peaks는 일직선으로 나타남을 알 수 있다.

```
drawdown(rets["SmallCap"]["Drawdown"].idxmin())
```

```
Period('1932-05', 'M')
```

SmallCap 포트폴리오 중 Max Drawdown이 나왔던 월은 1932년 5월임을 알 수 있다.

▼ Reference