

Optimization of Dividend Aristocrat Portfolio Utilizing Fundamental Data

34기 김서연 34기 조성윤

35기 박상현 35기 석정원 35기 황지희



Y-FoRM 24-1 장기 프로젝트

Table of Contents

-  연구 배경 01
-  선례 연구 02
-  이론적 배경 03
-  모델링 04
-  최적화 05
-  한계 06

01. 연구 배경

연구 배경

- 가속화 되는 인구 고령화와 낮은 연금 소득 수준
- 리스크 국면에서도 안정적 수익을 창출하는 포트폴리오에 대한 수요
- 귀족 배당주를 중심으로 자산군 포지셔닝 및 향후 배당 예측
- 펀더멘털 데이터 (PER & P/FCF)를 기반으로 한 제약조건 추가
=> 귀족 배당주를 중심으로 포트폴리오 구성!

02. 선례 연구

〈배당 귀족주 포트폴리오 최적화 연구〉 착안

- 귀족 배당주: 25년 이상 지속적인 배당 증가 및 지급
- 주가차익보다 배당 수익에 더 초점을 둔 포트폴리오
- 매달 수령받는 배당금을 통해 연금 효과를 창출
- 수리적 모델을 통한 포트폴리오 최적화

개선된 귀족 배당주 포트폴리오

- 데이터 전처리 → 특별배당, 배당 이상치 및 지속 지급
- 종목 수 30개 → 종목 수 63개로 투자 유니버스 확대
- 백테스팅 기간 : 2016년~2023년
- 3년치 데이터로 1년 주기 리밸런싱 → 1년 주기 리밸런싱 결과로 예측
- 고든성장모형 가정 타당성 확보
- 펀더멘탈 데이터 제약조건으로 추가

포트폴리오 종목 및 선정 과정

- 최근 10년의 배당 데이터
: 25년간 배당을 꾸준하게
증가시킨 귀족배당주
- 주식 분할 데이터
- 주가 데이터
- 펀더멘탈 데이터
: PER, P/FCF

NYSE - Nasdaq Real Time Price • USD

Leggett & Platt, Incorporated (LEG) [Follow](#)

11.60 +0.58 (+5.26%) 11.78 +0.18 (+1.55%)

At close: May 31 at 4:00 PM EDT Pre-Market: 8:23 AM EDT

Mar 17, 1980 - Dec 31, 2023 Dividends Only Daily

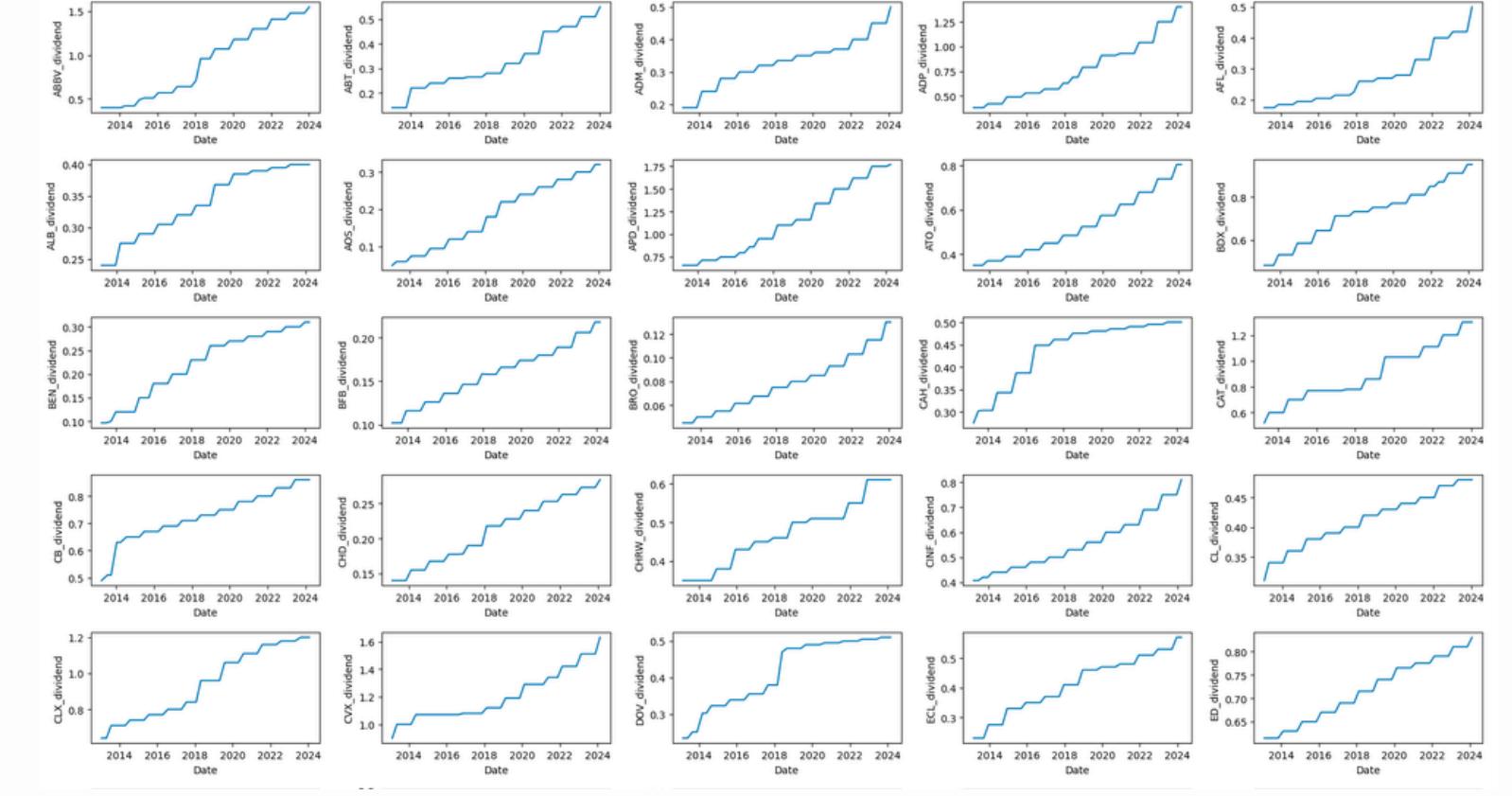
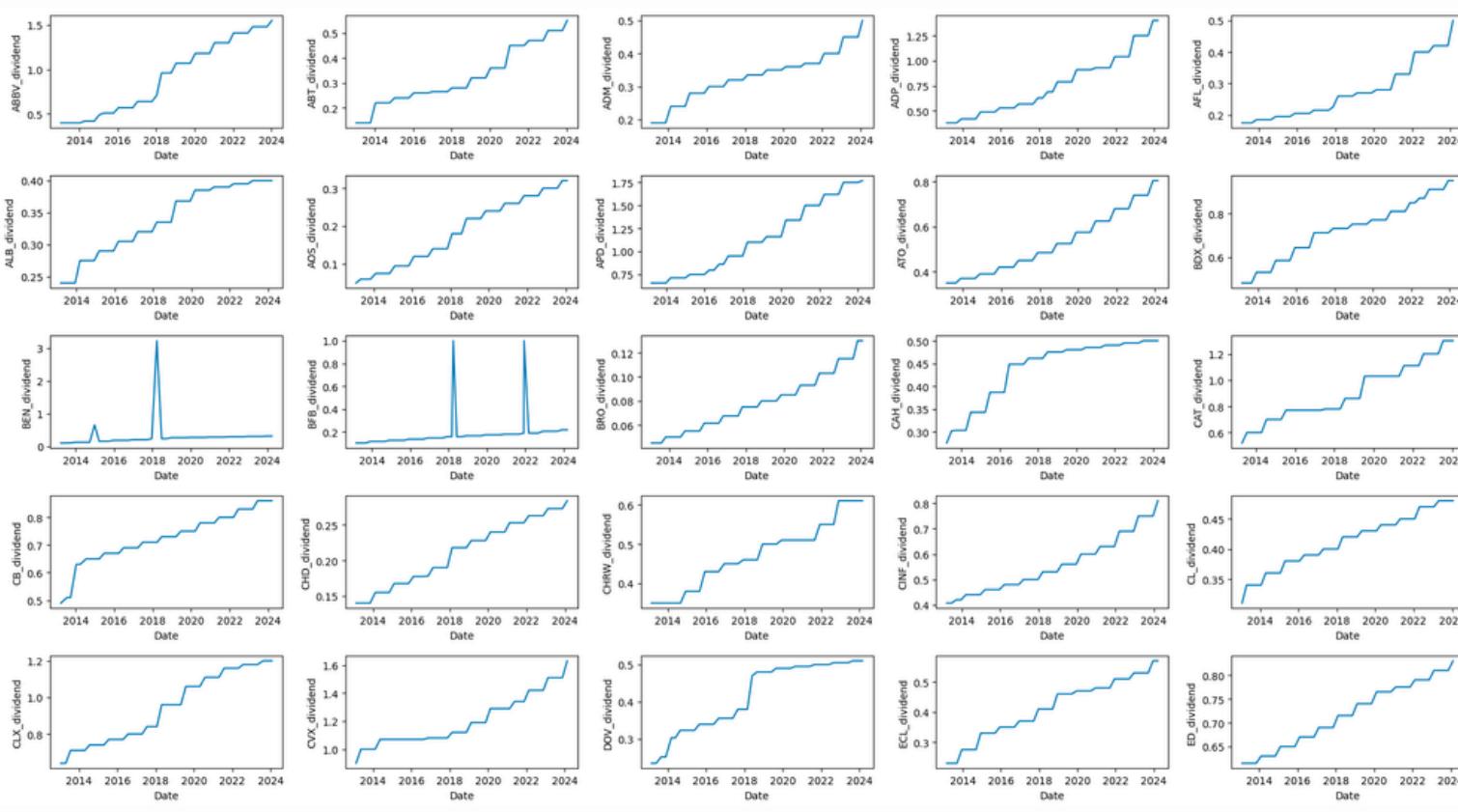
Currency in USD | [Download](#)

Date	Dividend
Dec 14, 2023	0.46 Dividend
Sep 14, 2023	0.46 Dividend
Jun 14, 2023	0.46 Dividend
Mar 14, 2023	0.44 Dividend
Dec 14, 2022	0.44 Dividend
Sep 14, 2022	0.44 Dividend
Jun 14, 2022	0.44 Dividend
Mar 14, 2022	0.42 Dividend
Dec 14, 2021	0.42 Dividend

Year	Current	2023	2022	2021	2020	2019
Market Capitalization	284,729	273,605	285,804	239,371	189,171	130,935
Market Cap Growth	-	-4.27%	19.40%	26.54%	44.48%	-5.58%
Enterprise Value	340,661	320,174	339,846	306,225	266,748	157,739
PE Ratio	47.85	56.26	24.15	20.74	40.98	16.61
PS Ratio	5.23	5.04	4.92	4.26	4.13	3.94
PB Ratio	35.56	26.41	16.57	15.54	14.47	-16.02
P/FCF Ratio	13.01	12.40	11.79	10.89	11.27	10.25
P/OCF Ratio	12.55	11.98	11.46	10.51	10.76	9.83

데이터 확인 및 전처리

배당금 이상치, 특별배당 지급, 배당 지급 지속성 등의 조건에 맞추어 데이터 조정



03. 이론적 배경

샤프지수

1. 샤프지수: 위험대비 투자수익에 대한 지표

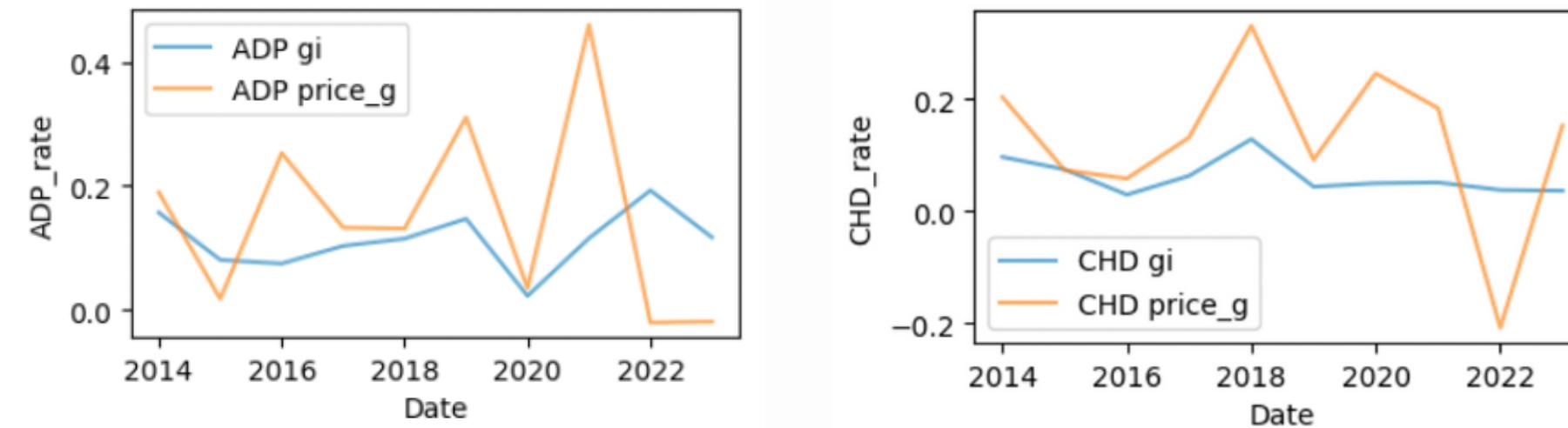
$$\frac{R_p - R_f}{\sigma_p} \left\{ \begin{array}{l} R_p : \text{포트폴리오 초과수익} \\ R_f : \text{무위험 초과수익 (미국 국채 수익률 사용)} \\ \sigma_p : \text{포트폴리오 초과수익의 표준편차} \end{array} \right.$$

2. 샤프지수의 의미

- 포트폴리오의 초과수익을 포트폴리오의 변동성으로 나눔
→ 투자자가 위험에 대한 대가로 얼마나 많은 초과수익을 얻었는지를 평가
- 샤프지수가 높을수록 리스크가 조정된 포트폴리오를 의미한다.
→ 샤프지수 최대화하는 수리 모델 구현

고든 성장 모형

- 기업의 이익과 배당이 매년 $g\%$ 만큼 동일하고 일정하게 성장한다고 가정
- 하지만, 기업의 이익과 배당이 $g\%$ 라는 일정하고 동일한 값으로 증가하지는 않았음



- 데이터의 주가 연간 변화율과 배당성장률의 연간 변화율을 비교했을 때 두가지 변화율이 정확히 일치하지는 않았음
- 그럼에도 배당성장률의 상승/하락 경향성은 같아지는 것을 확인.
- 이를 바탕으로, 기업의 주가 성장률을 배당성장률로 볼 수 있다는 근거가 마련되면 고든성장모형에서의 기대수익률 구현에 대한 현실적인 타당성을 확보

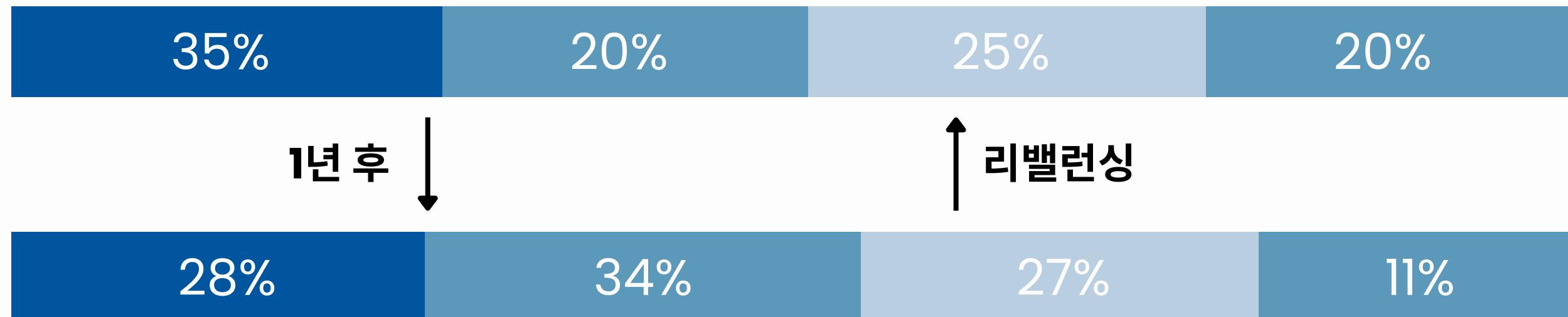
고든 성장 모형

- 이를 바탕으로, 기업의 주가 성장률을 배당성장률로 볼 수 있다는 근거가 마련되면 고든성장모형에서의 기대수익률 구현에 대한 현실적인 타당성을 확보할 수 있었음
- 샤프지수의 포트폴리오 기대수익률에 고든성장모형의 가정을 차용하여 구현하면,

$$\frac{R_p - R_f}{\sigma_p} = g_i(\%) \text{(배당성장률)} + d_i(\%) \text{(배당수익률)}$$

리밸런싱

- 리밸런싱(rebalancing) : 투자 포트폴리오 내의 자산 비율을 목표 비율에 맞추기 위해 주기적으로 조정하는 과정을 의미함
- 이는 시간이 지남에 따라 각 자산의 가치 변화로 인해 초기 목표 비율에서 벗어난 포트폴리오를 다시 원래의 비율로 되돌리는 것을 목표
- 기업마다 배당금을 지급하는 시기가 각각 다르기 때문에 리밸런싱 주기를 1년으로 설정

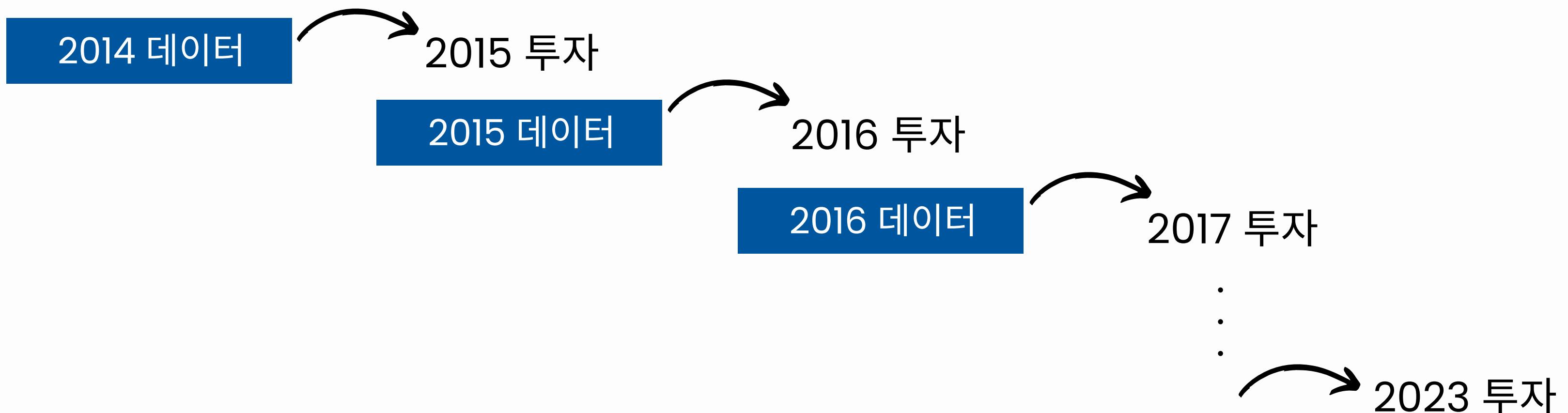


지난 1년 간의 데이터를 통해
향후 1년간 투자할 투자비율
설정

주가의 차익으로 인한
자산별 비중의 변화

슬라이딩 윈도우

- 슬라이딩 윈도우 : 포트폴리오 리밸런싱을 위한 시점을 결정하는 한 방법으로, 일정한 기간 동안 데이터를 분석하고 그 기간이 지나면 윈도우를 앞으로 이동시키는 방식
- 윈도우 이동 간격을 1년으로 지정해놓고, 과거 1년의 데이터를 분석해 그 다음 해의 새로운 데이터를 리밸런싱을 결정하는 방식으로 진행



04. 모델링

목적식 Parameter

i, j : 종목

w_i : 종목별 투자비중 = 종목 수 * 종목 가격 / 총 투자비용

d_i : 배당수익률 = 직전년도 배당금 합 / 마지막 배당 지급한 날의 주가

g_i : 배당성장률 = 배당금의 차 / 평균 배당금



내년도의 배당금 상승분은 작년도의 배당금 상승분과 같다고 가정하고 계산했는데
시각화 결과 다소 맞지 않는 부분이 있었음. 따라서 해당 변수를 활용하는 경우, 즉
고든성장모형을 활용하는 최적화와, 고든성장모형을 활용하지 않는 최적화를 따로 진행

목적식 정의

주가성장률(=배당성장률)

배당수익률

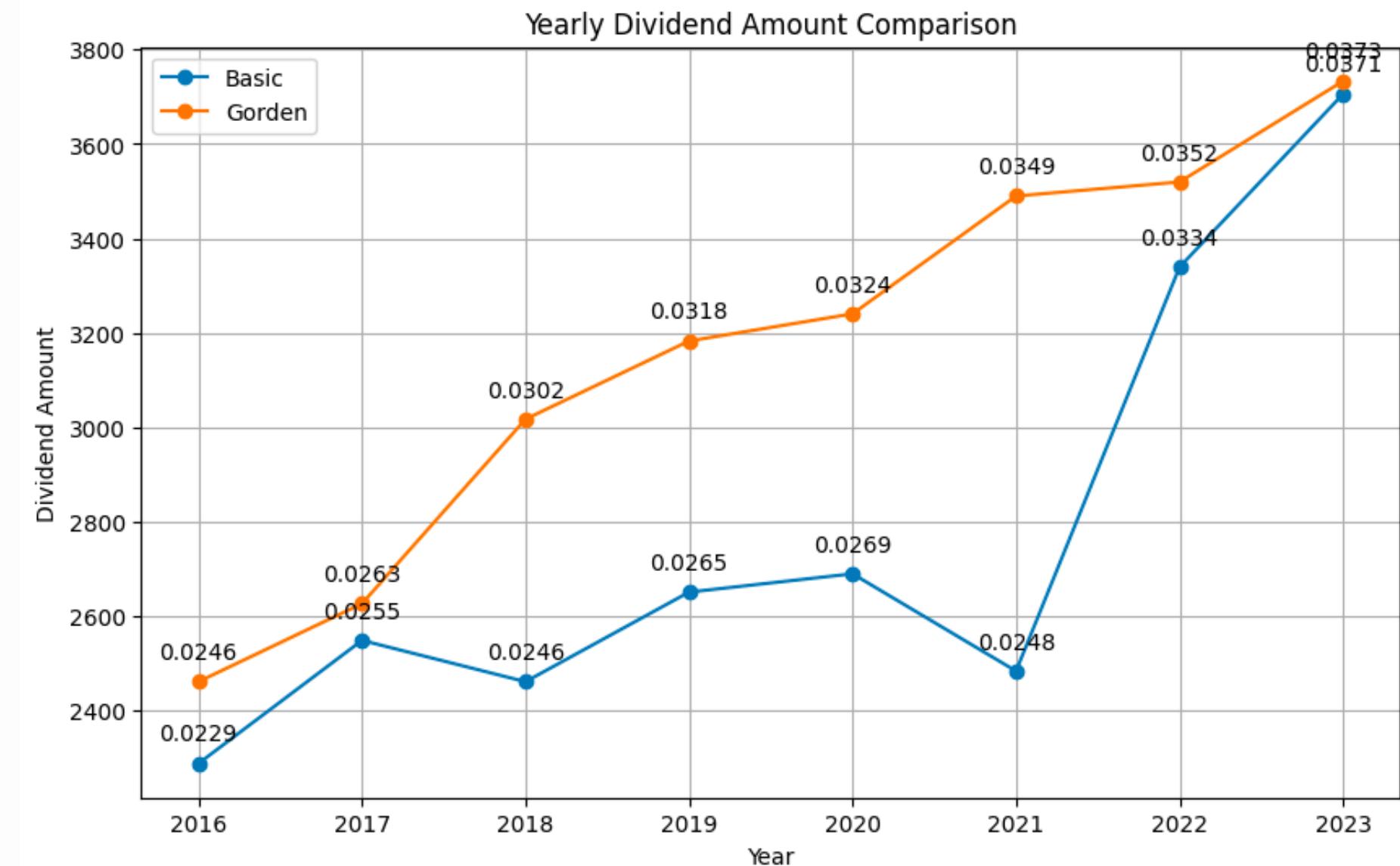
$$Max \frac{\sum_{i=1}^{63} g_i w_i + \sum_{i=1}^{63} w_i d_i (1 + g_i) - R_f}{\sqrt{\sum_{i=1}^{63} \sum_{j=1}^{63} w_i w_j Cov(i, j)}}$$

$$Max : Sharpe = \frac{R_p - R_f}{\sigma_p}, \quad E(p) = \sum_{i=1}^{63} g_i w_i + \sum_{i=1}^{63} w_i d_i (1 + g_i), \sigma_p = \sqrt{\sum_{i=1}^{63} \sum_{j=1}^{63} w_i w_j Cov(i, j)}$$

w_i : 종목별 투자비중, d_i : 배당수익률, g_i : 배당성장률, i, j : 종목

목적식 정의

$$Max \frac{\sum_{i=1}^{63} g_i w_i + \sum_{i=1}^{63} w_i d_i (1 + g_i) - R_f}{\sqrt{\sum_{i=1}^{63} \sum_{j=1}^{63} w_i w_j Cov(i, j)}}$$



제약식 정의

1. 주식분배금액 = 초기투자금액

2. 매달 배당금 기댓값 $\geq \$100$

$$\sum_i x_i v_{it} > \text{당해 연도 초기 투자금액의 } 1\% / 12, \forall i$$

3. 개별 종목 비중 $\leq 20\%$

$$w_i \leq 0.2, \forall i$$

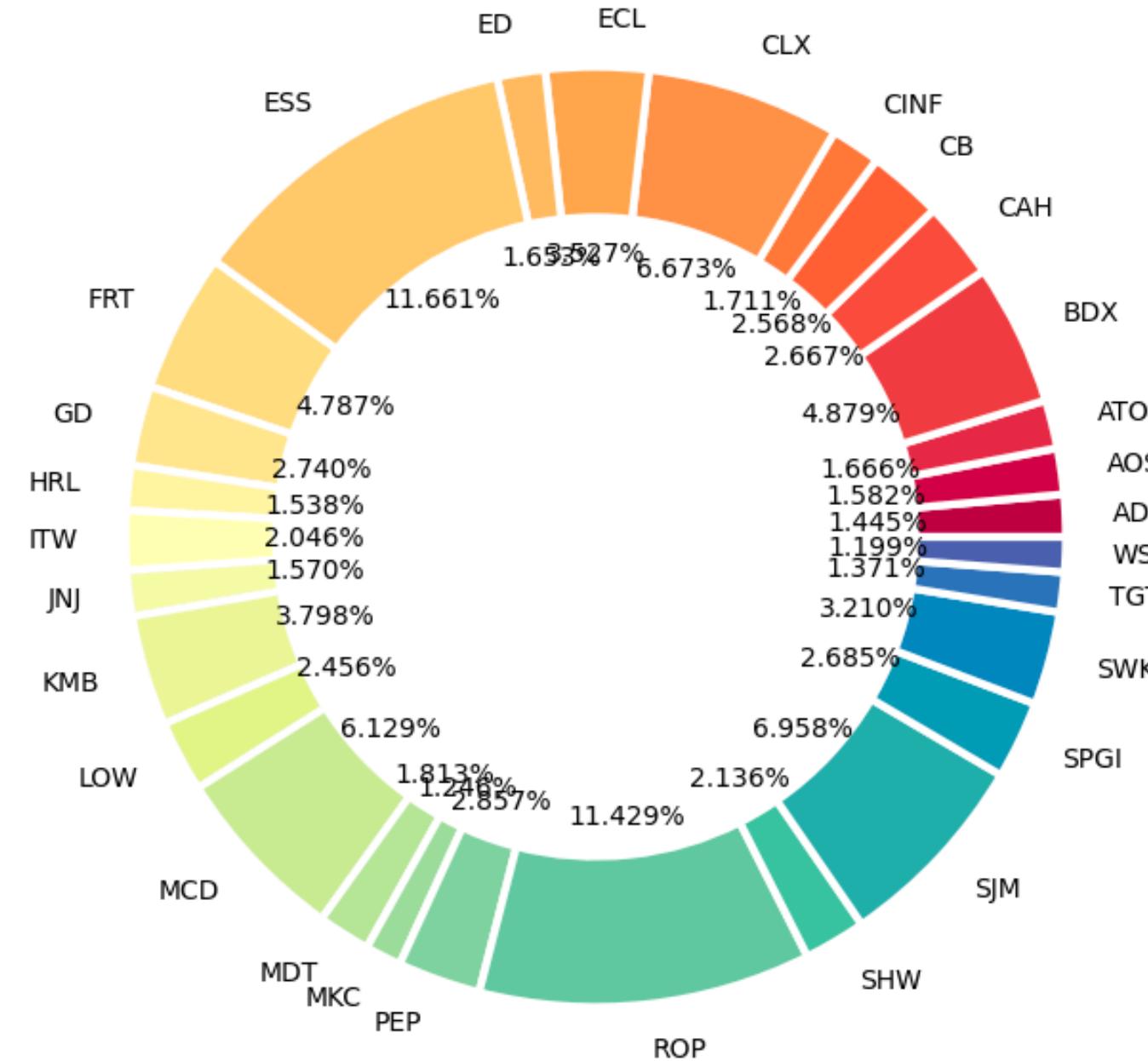
4. 펀더멘털 데이터(PER, P/FCF) 고려 \rightarrow if $\text{PER} < 0$ or $\text{PER} > 50$, $\max(w_i) = 1\%$

\rightarrow if $\text{P/FCF} < -100$ or $\text{P/FCF} > 100$, $\max(w_i) = 1\%$

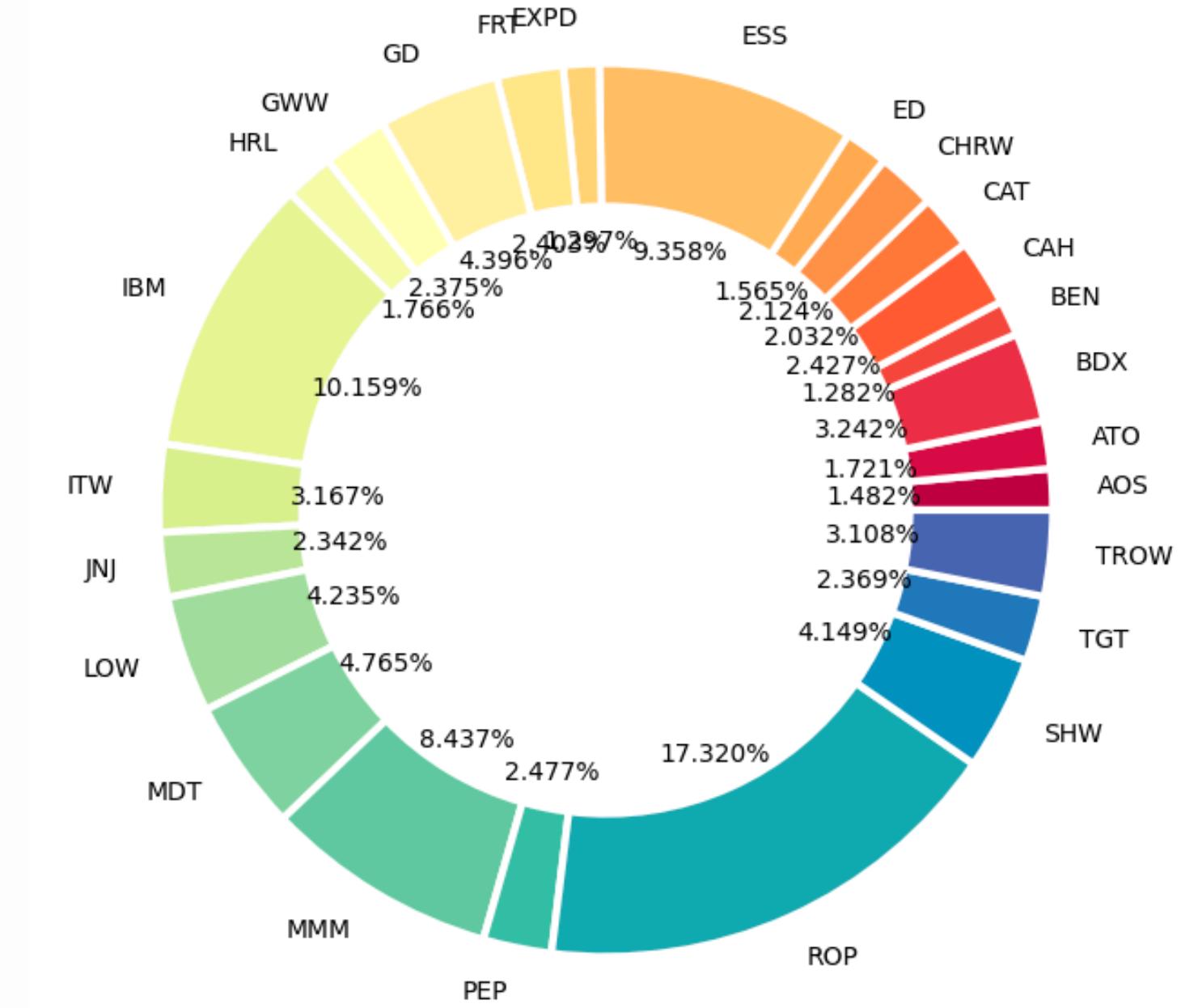
w_i : 종목별 투자비중, x_i : 종목별 구매한 주식 수, v_{it} : i종목, t월 배당금

05. 최적화

최적화

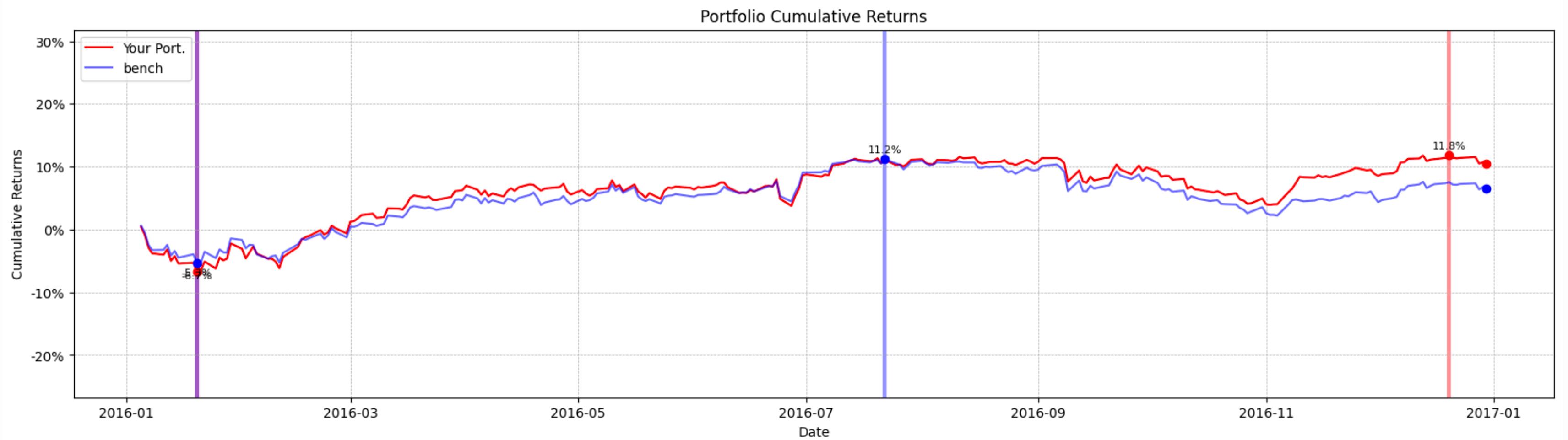


단순 샤프지수를 통한 2016 구매 비중



고든성장모형 적용한 2016 구매 비중

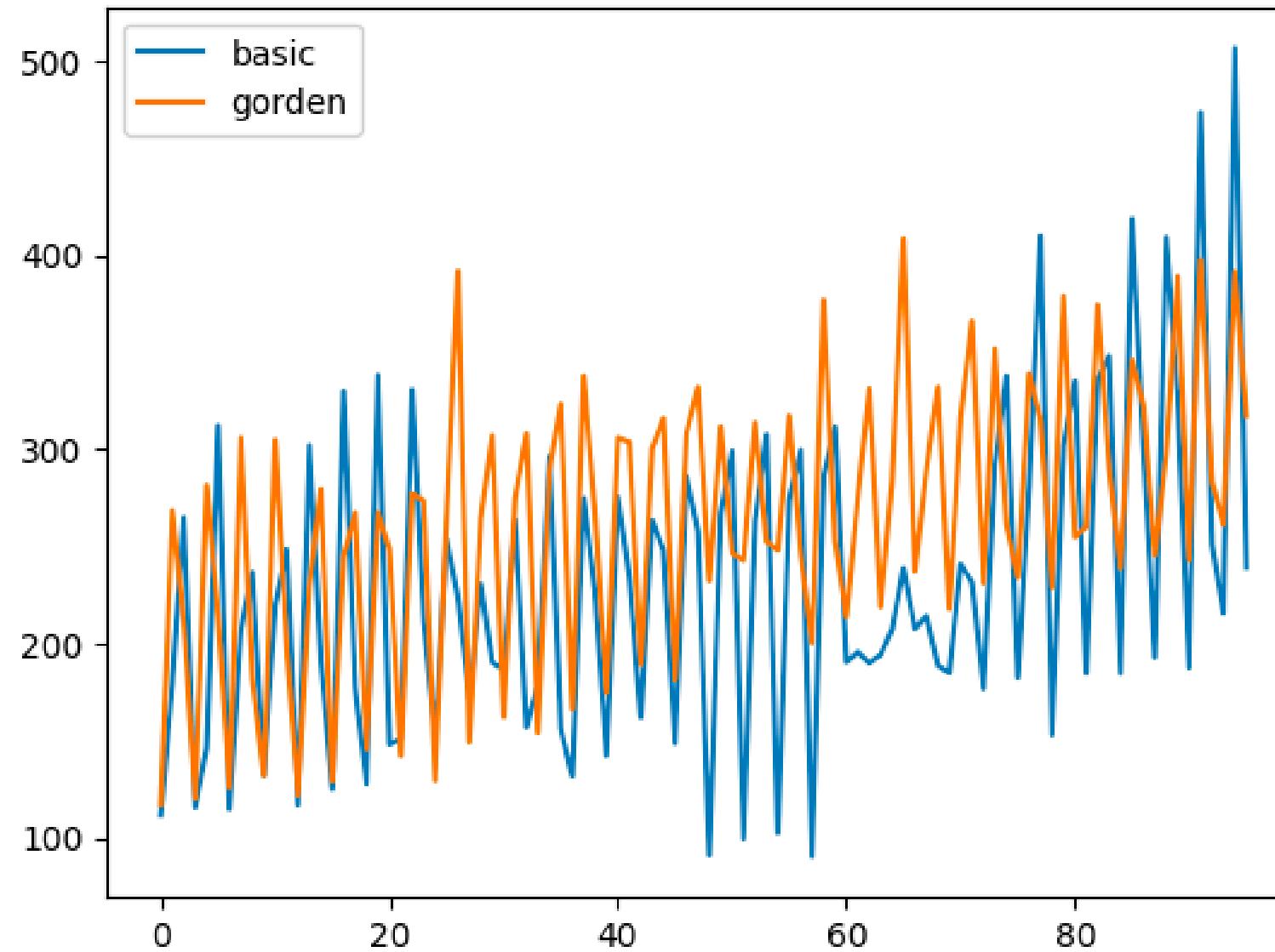
최적화



Portfolio PnL in 2016

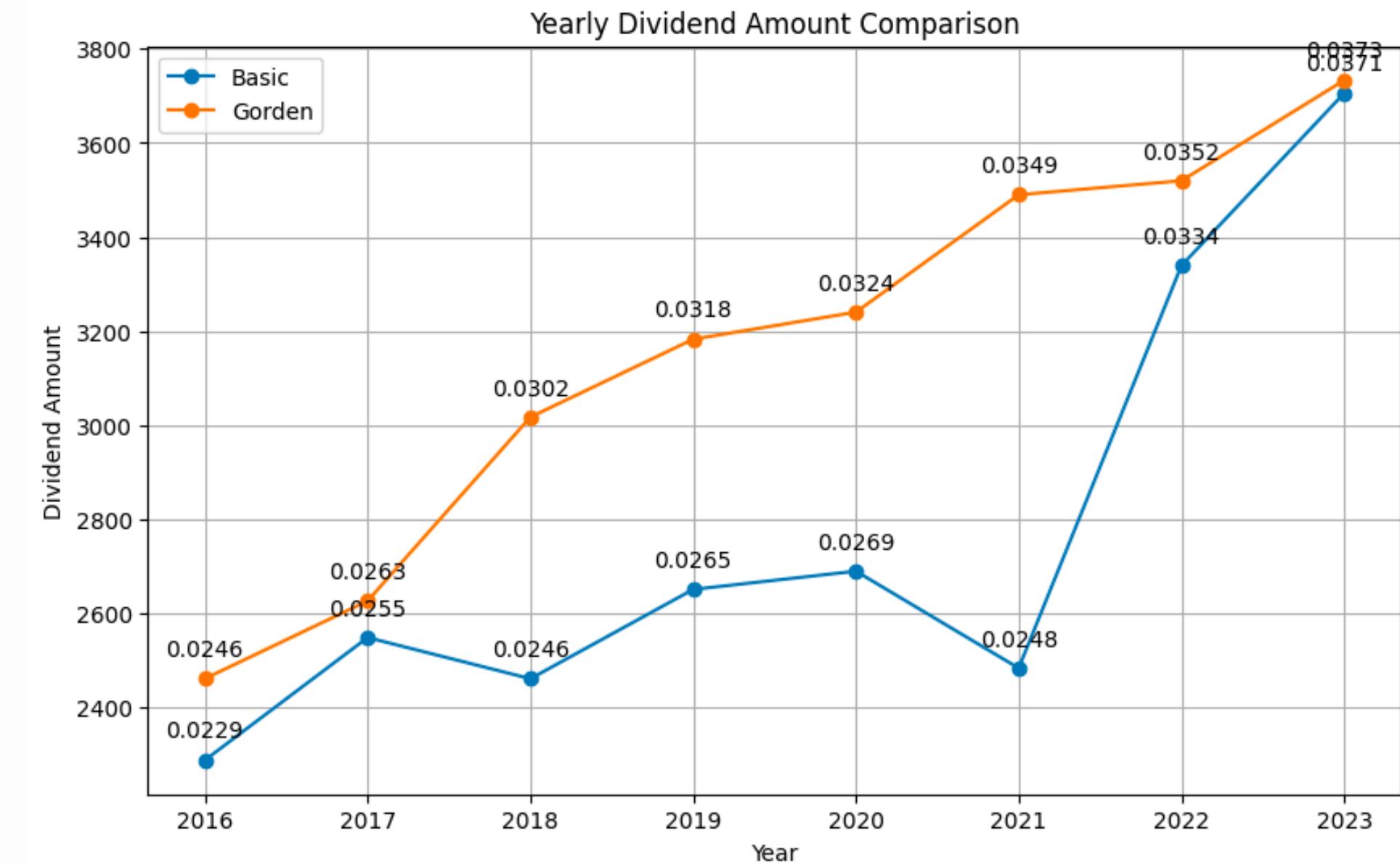
- 고든성장모형 차용한 샤프지수
- 단순 샤프지수 활용한 샤프지수

최적화



월별 실수령 배당금

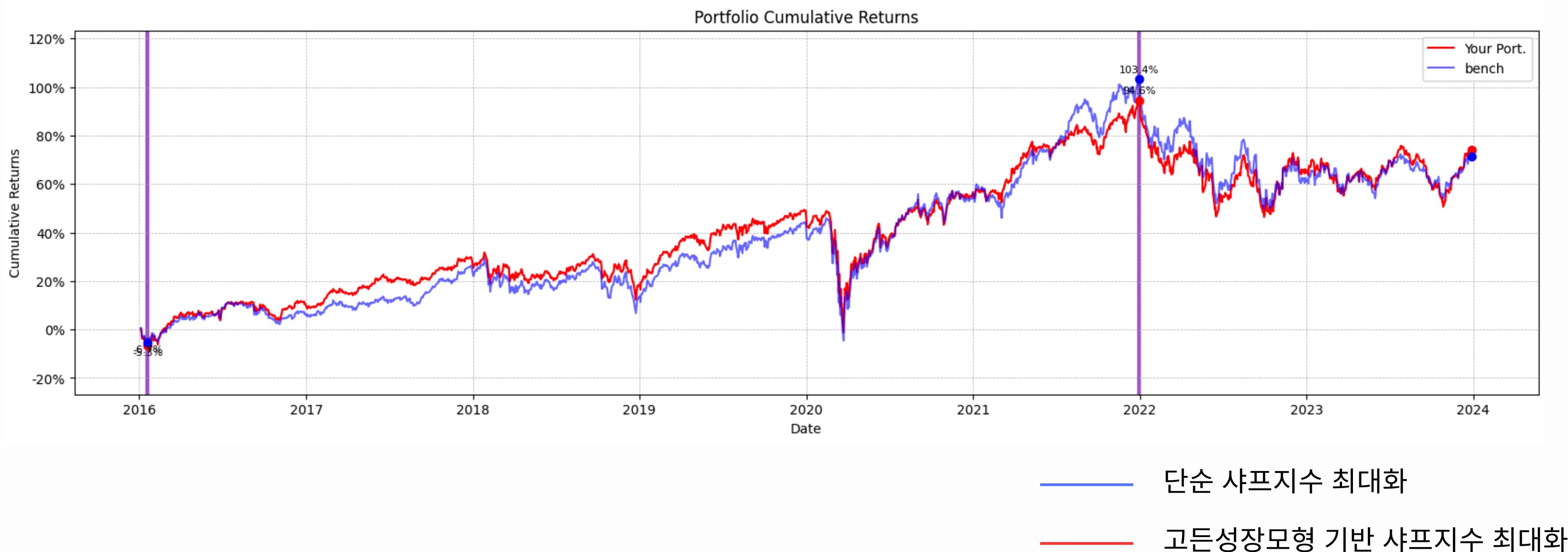
매달 실수령 배당금 > \$100 (제약조건 충족)



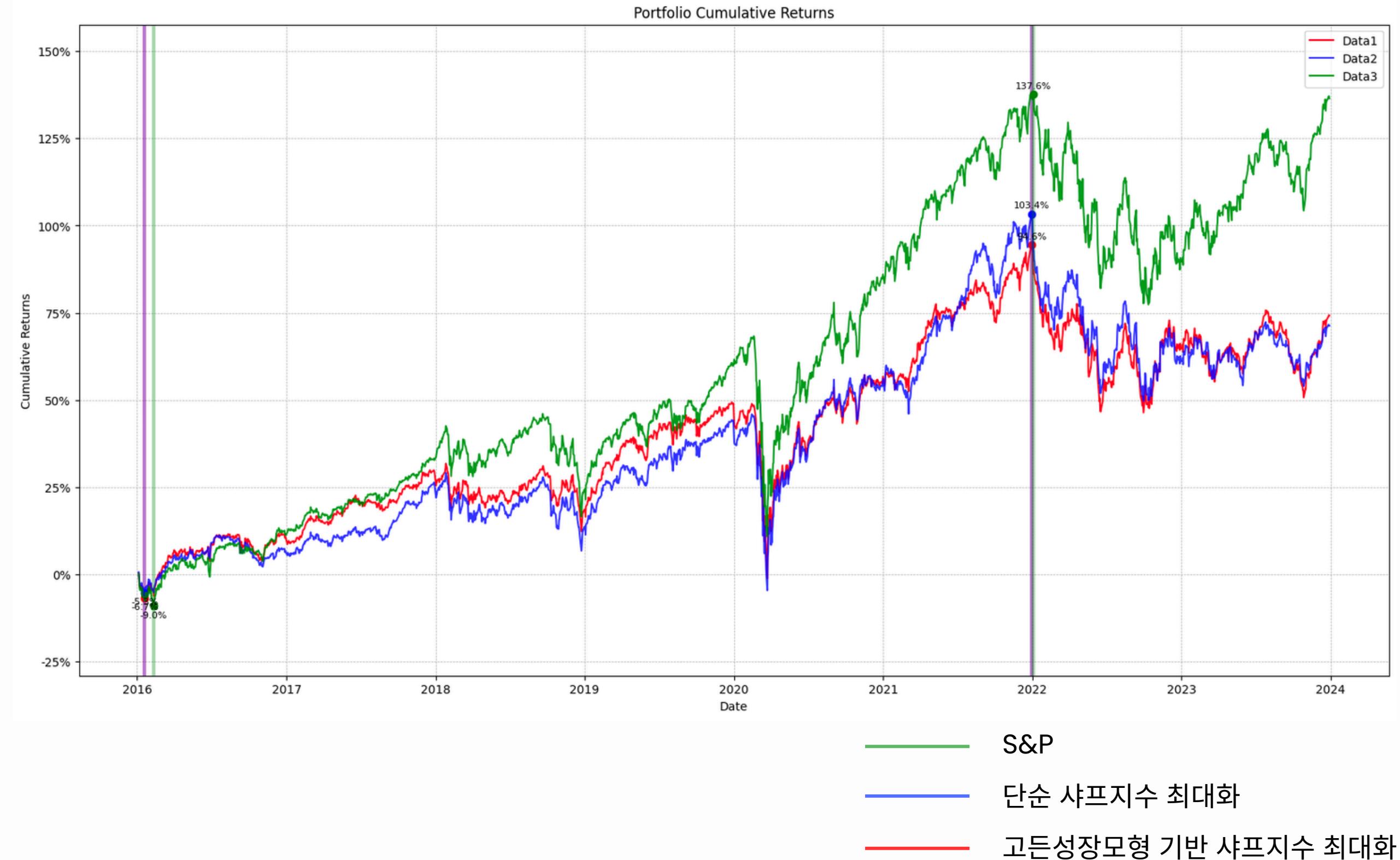
연도별 실질수령 배당금

리스크 상황(2020, 2021 코로나)에서 두드러지는 배당 수령 안정성

최적화



최적화



06. 한계

한계점

- 종가로 해당 주식을 구매할 수 있다는 가정 -> 실질적으로 불가능
- 1월1일로 리밸런싱 일자를 고정한 것 -> 마년 초만이 아닌 다른 시점에도 리밸런싱은 가능
- 이 정도의 underperform은 예상치 못함.

개선 사항

- 실제 테스팅에서 증권사 API에 연결해서 실제 주가를 가져와서 월별로 비중을 조절할 수 있음

Appendix - parameter

g_i (배당성장률)

	ABBV	ABT	ADM	ADP	AFL	ALB	AOS	APD	ATO	BDX	...	SHW	SJM	SPGI	SWK	SYY	TGT	TROW	WMT	WST	XOM	
Date	2014	0.048193	0.363636	0.208333	0.156369	0.053333	0.127273	0.200000	0.077922	0.053333	0.098435	...	0.090906	0.098361	0.066667	0.039216	0.034188	0.189474	0.136364	0.020831	0.097561	0.088889
2015	0.089109	0.083333	0.142857	0.080000	0.050633	0.051724	0.210526	0.049383	0.075472	0.097560	...	0.179105	0.045802	0.090909	0.056075	0.033333	0.074074	0.098039	0.020406	0.088889	0.055556	
2016	0.105263	0.076923	0.066667	0.074074	0.048193	0.049180	0.208333	0.066862	0.070175	0.103322	...	0.202382	0.112676	0.083333	0.053097	0.032258	0.068966	0.037037	0.020004	0.081633	0.026846	
2017	0.109375	0.018868	0.062500	0.102564	0.045977	0.046875	0.142857	0.094737	0.076294	0.027211	...	0.011764	0.039216	0.121951	0.082645	0.060606	0.032787	0.052632	0.019606	0.075472	0.026144	
2018	0.178273	0.053571	0.044776	0.114286	0.134615	0.044776	0.210526	0.136364	0.080808	0.026491	...	0.011630	0.085890	0.180000	0.046512	0.083333	0.031746	0.185714	0.019229	0.070175	0.061920	
2019	0.102804	0.125000	0.042857	0.146341	0.037037	0.089674	0.088889	0.051724	0.093023	0.025806	...	0.238938	0.034682	0.122807	0.044444	0.076923	0.030769	0.078947	0.018872	0.065574	0.058309	
2020	0.093220	0.111111	0.027778	0.021858	0.035714	0.044156	0.081633	0.134328	0.085106	0.049999	...	0.156716	0.022472	0.149254	0.014388	0.133333	0.029851	0.155556	0.018517	0.061538	0.011494	
2021	0.092308	0.200000	0.027027	0.114883	0.151515	0.012821	0.075472	0.106667	0.086106	0.047619	...	0.187878	0.095238	0.129870	0.120805	0.043478	0.278481	0.122951	0.018180	0.057971	0.011461	
2022	0.078014	0.042553	0.075000	0.192220	0.175000	0.012658	0.070175	0.074074	0.086331	0.034995	...	0.083333	0.029851	0.096386	0.012579	0.041667	0.181818	0.100000	0.017861	0.054795	0.033803	
2023	0.047297	0.078431	0.111111	0.116505	0.047619	0.012500	0.065574	0.074286	0.085950	0.043478	...	0.008264	0.038462	0.055556	0.012422	0.020202	0.018349	0.016393	0.017542	0.051948	0.043478	

10 rows x 63 columns

d_i (배당수익률)

	ABBV	ABT	ADM	ADP	AFL	ALB	AOS	APD	ATO	BDX	...	SHW	SJM	SPGI	SWK	SYY	TGT	TROW	WMT	WST	XOM	
Date	2014-02-01	0.000000	0.000000	0.007916	0.000000	0.007557	0.000000	0.003669	0.000000	0.010346	0.000000	...	0.003213	0.007998	0.004202	0.000000	0.000000	0.010047	0.000000	0.000000	0.000000	0.010455
2014-03-01	0.000000	0.000000	0.000000	0.007715	0.000000	0.004867	0.000000	0.008164	0.000000	0.005472	...	0.000000	0.000000	0.000000	0.007768	0.000000	0.000000	0.007599	0.007872	0.000000	0.000000	
2014-04-01	0.012734	0.006981	0.000000	0.000000	0.000000	0.000000	0.003799	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.010468	0.000000	0.000000	0.002408	0.000000	0.000000	
2014-05-01	0.000000	0.000000	0.007156	0.000000	0.007554	0.000000	0.000000	0.000000	0.009360	0.000000	...	0.003014	0.007682	0.004259	0.000000	0.000000	0.009806	0.000000	0.007614	0.000000	0.010405	
2014-06-01	0.000000	0.000000	0.000000	0.007533	0.000000	0.004433	0.000000	0.007819	0.000000	0.005297	...	0.000000	0.000000	0.000000	0.007093	0.000000	0.000000	0.007445	0.000000	0.000000	0.000000	
...	
2023-10-01	0.010330	0.005461	0.000000	0.000000	0.000000	0.000000	0.004749	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.007843	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
2023-11-01	0.000000	0.000000	0.006222	0.000000	0.005194	0.000000	0.000000	0.000000	0.007280	0.000000	...	0.002307	0.009667	0.002266	0.009315	0.000000	0.000000	0.000000	0.000000	0.000584	0.009218	
2023-12-01	0.000000	0.000000	0.000000	0.006053	0.000000	0.002925	0.000000	0.006582	0.000000	0.004001	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.011870	0.003706	0.000000	0.000000	
2024-01-01	0.009612	0.004910	0.000000	0.000000	0.000000	0.000000	0.004010	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.006645	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
2024-02-01	0.000000	0.000000	0.009348	0.000000	0.006379	0.000000	0.000000	0.000000	0.007154	0.000000	...	0.002282	0.008355	0.002090	0.000000	0.000000	0.007478	0.000000	0.000000	0.000000	0.009256	

121 rows x 64 columns

w_i (종목별 비중)

	ABBV	ABT	ADM	ADP	AFL	ALB	AOS	APD	ATO	BDX	...	SHW	SJM	SPGI	SWK	SYY	TGT	TROW	WMT	WST	XOM
Weight	0.0																				

Appendix - 목적식 구현

```
def sharpe_obj(self,x): #목적함수 구현

    x = np.array(x)

    seed = np.dot(x, self.current_price) # 전체 투자 금액, 내적 개념, 종목수*종목가격 = 투자금액

    weights = (x*self.current_price)/ seed # 투자 비중 / 총투자비용 : [1번종목에 총투자비용대비 얼마큼투자함, ~~~~]

    er = (self.past_period_price.iloc[-252:,:].pct_change().dropna().mean()) * 252 # 252를 start-end 해서 실제 영업일 기준 투자 기간으로 바꿔도 좋을듯

    rf = self.rf.resample('M').mean().loc[:self.start_date].iloc[-1][0]

    cov = self.past_period_price.iloc[-252:,:].pct_change().dropna().cov() # 공분산

    variance = np.dot(weights.T, np.dot(cov,weights)) # variance = sigma(wiwjcovij)

    std = np.sqrt(variance) #시그마, 표준편차

    standard_ER = np.dot(weights,er)

    standard_sharpe = -(standard_ER-rf)/std

    return standard_sharpe
```

단순 샤프지수 최대화

Appendix - 목적식 구현

```
def gorden_obj(self,x): #목적함수 구현
    x = np.array(x)

    seed = np.dot(x, self.current_price) # 전체 투자 금액, 내적 개념, 종목수*종목가격 = 투자금액

    weights = (x*self.current_price)/ seed # 투자 비중 / 총투자비용 : [1번종목에 총투자비용대비 몇만큼투자함, ~~~~]

    rf = self.rf.resample('M').mean().loc[:self.start_date].iloc[-1][0]

    cov = self.past_period_price.iloc[-252:,:].pct_change().dropna().cov() # 공분산

    variance = np.dot(weights.T, np.dot(cov,weights)) # variance = sigma(wiwjcovij)

    std = np.sqrt(variance) #시그마, 표준편차

    growth_rate = self.loc[self.start_date -1]

    div_rate = self.df_dy.loc[self.start_date -1]

    ep = np.dot(weights,growth_rate) + np.dot(weights,div_rate*(1+growth_rate)) #기대 자본성장률 + 기대배당수익률

    custom_sharpe = -(ep-rf)/std

    return custom_sharpe
```

고든성장모형에 기반한 샤프지수 최대화

Appendix - 최적화

```
class Optimizing():
    """
    Base class for optimizing
    """

    def __init__(self,
                 price: pd.DataFrame,
                 investment_period: list,
                 initial_investment: int,
                 rf: pd.DataFrame,
                 div: pd.DataFrame,
                 div_gi: pd.DataFrame,
                 div_di: pd.DataFrame,
                 fund: dict):
        self.price = price
        self.investment_period = investment_period
        self.start_date = price.loc[investment_period[0]:].index[0]
        self.end_date = price.loc[:investment_period[1]].index[-1]
        self.initial_investment = initial_investment
        self.rf = rf
        self.div = div
        self.div_gi = div_gi
        self.div_di = div_di
        self.fundamental = fund

        self.ticker_lst = price.columns.to_list()

        #need to OOP
        self.current_price = price.loc[self.start_date]
        self.past_period_price = price.loc[:self.start_date]
```

```
def optimizing_weight(self):
    """
    Optimize given obj,params,cons to return weight
    """

    # self.period? (투자 기간이 필요할듯) # version 1에서는 과거1년으로 고정

    #예상 주당 배당금 ( 추후 Gi 곱을 통해 미래 추정값으로 바꿔야함)
    expect_div = self.div.loc[:str(self.start_date.year-1)]
    expect_growth = self.div_gi.loc[:str(self.start_date.year-1)]
    expect_div = expect_div*(1+expect_growth)

    min_month_div = self.initial_investment*(0.01)*(1/12)

    num_stocks = len(self.ticker_lst)

    #boundary
    bnds = ((0,None), ) * num_stocks # Long only port
    #initial decision variable
    x0 = np.array([100] * num_stocks)
```

최적화 함수