FINAL PROJECT

Risky Asset 1

정재현

강수연

김채연

이우주

이상윤

전승원

CONTENTS

Project 개요 보고

펀드 종류 소개

프로젝트 개요 보고

Passive fund 코드 공유

프로젝트 가정 소개

구현 코드 공유

Active fund 코드 공유

코드 구현 과정 개관

구현 코드 공유

PROJECT 개요 보고

펀드
종류 소개

Passive Fund

벤치마크 지수를 기준으로 운용 방식을 사전에
정의하고 정해진 수익률을 따라가도록 설계해
펀드매니저의 개입 없이 자동으로 운용하는 펀드

Active Fund

벤치마크 지수 이상의 수익률을 목표로 삼고,
상황에 적합하게 펀드매니저의 판단에 따라
다양한 운용 전략을 선택하는 펀드

Final Project 개요

Python을 활용한 펀드(집합투자증권) 기획

Goal

간단한 passive fund 모델 및 active fund 투자 전략 구성 시스템 코드 구현

Procedure

Passive fund 모델 구현 /// Active fund 관련 Paper 탐구 /// Active fund 코드 구현

Passive Fund

ETF 추종 Passive fund

추종 지수 구성

펀드 함수 작성

수익률 비교 및 추적오차 계산

Active Fund

경제 국면 분석

팩터 선정

자산 유니버스 구성

자산 별 가중치 계산

…

PASSIVE FUND 코드 공유

프로젝트 가정 소개

운용 기간 중 펀드 구성종목은 변하지 않는다.

운용 기간 중 각 구성종목이 유동주식의 수는 변하지 않는다.

운용 보수, 주식 매매 수수료, 증권 거래세 등 제반 비용은 고려하지 않는다.

주식에서 받는 배당은 고려하지 않는다.

운용 기간 중 이자율은 변하지 않는다.

최초의 설정은 100만좌로 시작하고, 운용 기간 중 50만좌 이하로 내려가지 않는다.

Passive fund

코드 공유

```
[2]  # 라이브러리 호출

     import datetime as dt
     from urllib.request import urlopen
     import bs4
     import re
     import pandas as pd
     import requests
```

```
# 지수 종목 구성(23.06.30 기준 한국 거래소 시총 상위 10개 종목)

'''
005930 삼성전자
373220 LG에너지솔루션
000660 SK하이닉스
207940 삼성바이오로직스
005935 삼성전자우
051910 LG화학
006400 삼성SDI
005380 현대차
000270 기아
005490 POSCO홀딩스
035420 NAVER
'''

k11_component = ['005930', '373220', '000660', '207940', '005935', '051910', '006400','005380', '000270', '005490', '035420']
```

지수 구성

```python
# 일자별 주가 수집 함수(데이터 추출 함수)
# find.all 함수 >>> 찾은 데이터를 모두 리스트 형태로 저장

def historical_stock_naver(stock_cd, start_date='', end_date='', page_n=1, last_page=0):

    if start_date:``
        start_date = date_format(start_date)
    else:
        start_date = dt.date.today()
    if end_date:
        end_date = date_format(end_date)
    else:
        end_date = dt.date.today()

    naver_stock = 'https://finance.naver.com/item/sise_day.naver?code=' + stock_cd + '&page=' + str(page_n)

    headers = {'user-agent' : 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36'}
    #payload = {'as_epq': 'James Clark', 'tbs':'cdr:1,cd_min:01/01/2015,cd_max:01/01/2015', 'tbm':'nws'}
    req = requests.get(naver_stock, headers = headers)
    source = bs4.BeautifulSoup(req.text, 'html.parser')

    dates = source.find_all('span', class_='tah p10 gray03') # 날짜 수집 #XPATH 열어서 확인
    prices = source.find_all('td', class_='num') # 종가 수집

    for n in range(len(dates)):

        if len(dates) > 0:

           # 날짜 처리
           this_date = dates[n].text
           this_date = date_format(this_date)

           if this_date <= end_date and this_date >= start_date:
           # start_datedhk end_date 사이에서 데이터 저장
              # 종가 처리
              this_close = prices[n*6].text # num 태그를 쓰는 칼럼이 모두 6개 >> 6배수에 해당하는 값을 추출
              this_close =this_close.replace(',', '')
              this_close = float(this_close)

              # 딕셔너리에 저장
              historical_prices[this_date] = this_close

           elif this_date < start_date:
           # start_date 이전이면 함수 종료
              return historical_prices


    # 페이지 네비게이션
    if last_page == 0:
        last_page = source.find_all('table')[1].find('td', class_='pgRR').find('a')['href'] #XPath 수에서 -1 해주기
        last_page = last_page.split('&')[1]
        last_page = last_page.split('=')[1]
        last_page = float(last_page)

    # 다음 페이지 호출
    if page_n < last_page:
        page_n = page_n + 1
        historical_stock_naver(stock_cd, start_date, end_date, page_n, last_page)


    return historical_prices
```

지수 구성

일자 별 주가 수집

find.all 사용 및 함수 작성

지수 구성

시가총액 비중 계산

일자 별 인덱스 계산

[14]
```
# k11 지수 산출
# 일자별로 각 종목별 시가총액 비중을 구함.

k11_historical_mc = k11_historical_price * k11_info['Outstanding'] * k11_info['Floating'] * 0.01
k11_historical_mc.tail(3)
```

	000270	000660	005380	005490	005930	005935	006400	035420	051910	207940	373220
2023-04-05	1.976534e+13	4.546500e+13	2.577309e+13	2.291347e+13	5.787649e+12	2.436428e+14	3.913973e+13	2.722779e+13	3.443012e+13	1.388667e+13	2.034986e+13
2023-04-04	1.947360e+13	4.541126e+13	2.537276e+13	2.251471e+13	5.760477e+12	2.431900e+14	3.652370e+13	2.747383e+13	3.395976e+13	1.388667e+13	1.989994e+13
2023-04-03	1.974103e+13	4.686227e+13	2.534515e+13	2.404840e+13	5.715190e+12	2.436428e+14	3.747956e+13	2.709111e+13	3.348940e+13	1.371045e+13	2.028064e+13

[15]
```
# 시가총액 비중 계산

k11 = pd.DataFrame()
k11['Market Cap'] = k11_historical_mc.sum(axis=1)
k11.head(3)
```

	Market Cap
2023-06-30	5.339956e+14
2023-06-29	5.376834e+14
2023-06-28	5.367899e+14

[16]
```
# 지수 산출 기준일 설정

base_date = dt.date(2023, 4, 3)
```

```
# 일자별 k11 인덱스 계산

k11['Index'] = k11['Market Cap'] / k11['Market Cap'][base_date] * 100
k11.head(3)
```

	Market Cap	Index
2023-06-30	5.339956e+14	107.356074
2023-06-29	5.376834e+14	108.097489
2023-06-28	5.367899e+14	107.917858

```
[28] # K11과 KOSPI200 비교 그래프

     plt.plot(k11['Adj Index'], color='orange', label='K11')
     plt.plot(k200['Index'], color='blue', label='KOSPI200')
     plt.legend(loc=0)
     plt.grid(True)
```

K11의 수익률이 더 높다.....스고이,,0ㅁ0

지수 구성

인덱스 비교 그래프

펀드 작성

[29]
```
import numpy as np
```

[30]
```
# 펀드 기본정보 세팅
# 설정과 환매 단위

CU = 50000
base_date = dt.date(2023, 4, 3) # 설정 기준일
volume = 1000000 # 최초 설정 수량
interest_rate = 0.02 # 이자율
```

[79]
```
#설정과 환매 random data

def creation_redemption(v):
    creation = np.random.randint(0, 5) * CU # 0~5 사이의 임의의 정수 * CU
    if v>500000:      # 50만 좌 이상인 경우에만 환매 발생
       redemption = np.random.randint(0, 5) * CU

    else:
       redemption = 0

    volume = v + creation - redemption  # 총 좌수 = 기존 좌수 + 설정 - 환매
    return(creation, redemption, volume)
```

```
# 일자별 보유비중 계산 및 저장

## 보유비중 산정 ##

k11_stock_ratio = pd.DataFrame()
for s in k11_info.index:
    k11_stock_ratio[s] = k11_historical_mc[s] / k11_historical_mc.sum(axis=1)
```

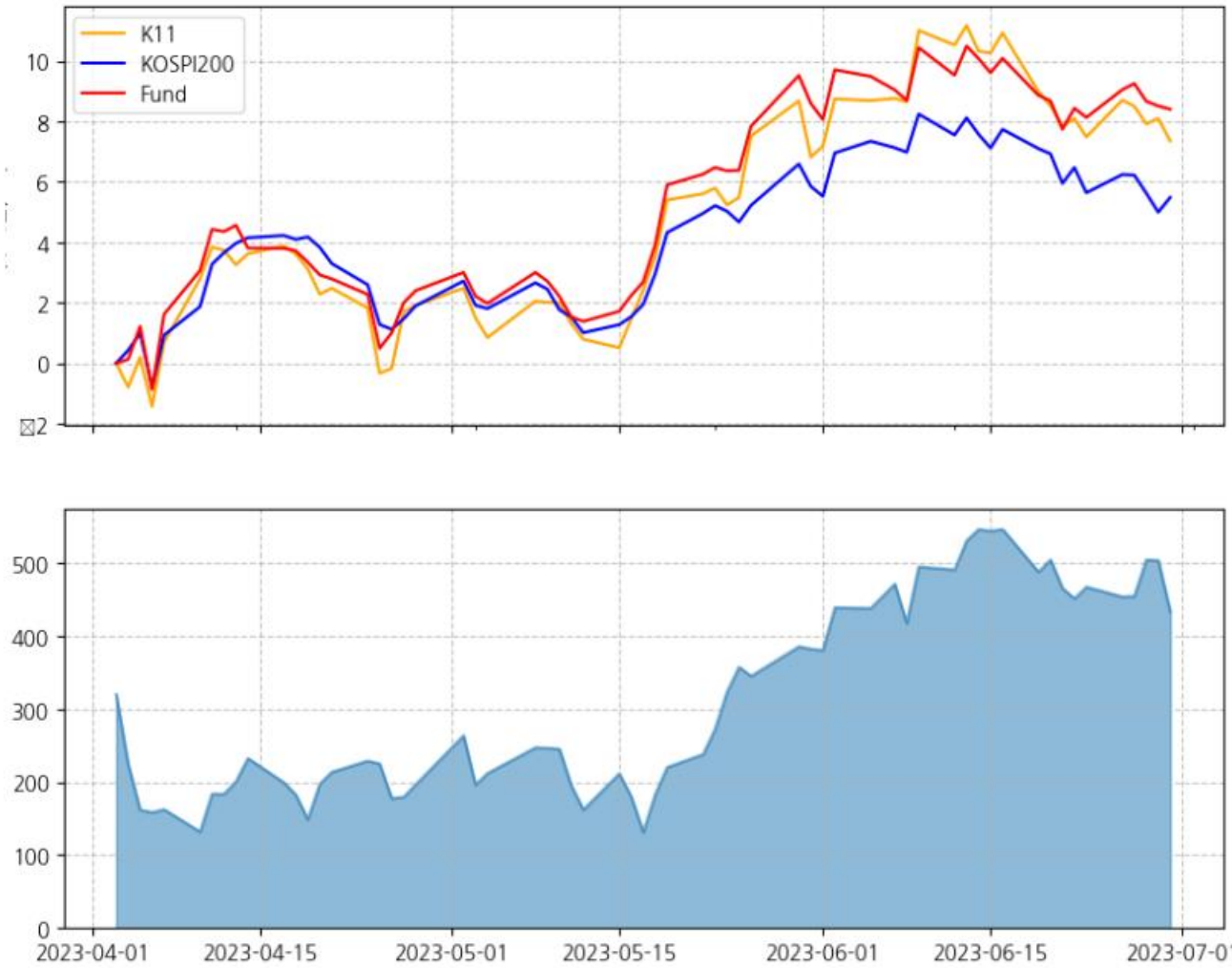
시가총액 비중 계산

일자 별 인덱스 계산

펀드 작성

```
[88]  # DataFrame 선언

      Fund_NAV = pd.DataFrame()
      Fund_Chg = pd.DataFrame()

      # 일자별 순환 계산 선언

      for d in k11_historical_price.index:
          # 포트폴리오 구성용 정보 (당일 주가, 자산 비중)
          stock_price = np.array(k11_historical_price.loc[d])
          stock_weight = np.array(k11_stock_ratio.loc[d])

          # 기존 주식 포트폴리오 NAV 계산
          if (d <= base_date):    # 기준일 이전
             # 최초 주식 포트폴리오 (보유량 0)
             stock_holdings = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
             NAV_cash = 0  # 최초 현금 보유량

          else: # 기준일 이후
             NAV_stock = sum(stock_holdings * stock_price) # 주식 잔고
             NAV = NAV_stock + NAV_cash # 전체 잔고

              # 기준 가격 산정
          if (d <= base_date):
             # 최초 기준가를 기준일자의 KOSPI 200 지수와 맞춤
              price = k200['Index'][base_date] * 100
          else:
              price = NAV / volume

          # 신규 펀드 설정 및 환매 좌수 계산

          if (d == base_date):
             volume = 0  # 펀드 좌수
             volume_chg = 1000000  # 첫날 설정액

          else:
              vol = creation_redemption(volume)  # 설정 및 환매 함수 호출
              volume_chg = vol[0] - vol[1]  # 좌수 변동
          # 총 펀드 좌수에 반영
          volume = volume + volume_chg
          # 펀드 입출금액
          aum_chg = price * volume_chg
          # 신규 주식 거래량 계산
          stock_trade = np.floor( price * volume_chg * stock_weight / stock_price )
          # 주식 매매 금액
          trade_amt = sum(stock_trade * stock_price)
          # 현금 잔고 변동
          cash_chg = aum_chg - trade_amt

          # 총 주식 보유량 = 기 보유량 + 신규 거래량
          stock_holdings = stock_holdings + stock_trade
          # 현금 보유량 증가 (이자율 반영)
          cash_holdings = np.floor(NAV_cash * np.exp(interest_rate/365))

          # NAV 업데이트
          NAV_stock = sum(stock_holdings * stock_price)    # 주식 잔고
          NAV_cash = cash_holdings + cash_chg    # 현금 잔고
          NAV = NAV_stock + NAV_cash   # 전체 잔고

          date = pd.Series(d)

          # Fund NAV 정보를 데이터 프레임에 저장
          NAV_tmp = {'Stock' : NAV_stock, 'Cash' : NAV_cash, 'Total' : NAV, 'Price' : price}
          tmp = pd.DataFrame(NAV_tmp, index=date)
          Fund_NAV = Fund_NAV.append(tmp)

          # 일자별 설정 및 환매 좌수 정보를 데이터 프레임에 저장
          Chg_tmp = {'Amount Change' : aum_chg, 'Trade Amount' : trade_amt, 'Cash Change' : cash_chg}
          tmp = pd.DataFrame(Chg_tmp, index=date)
          Fund_Chg = Fund_Chg.append(tmp)
```

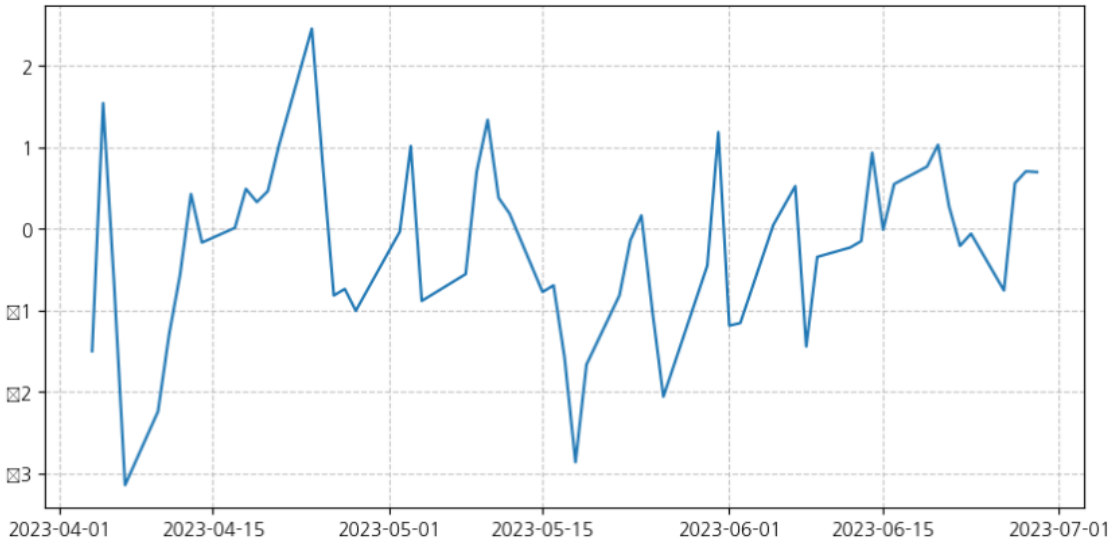
수익률 및 거래량



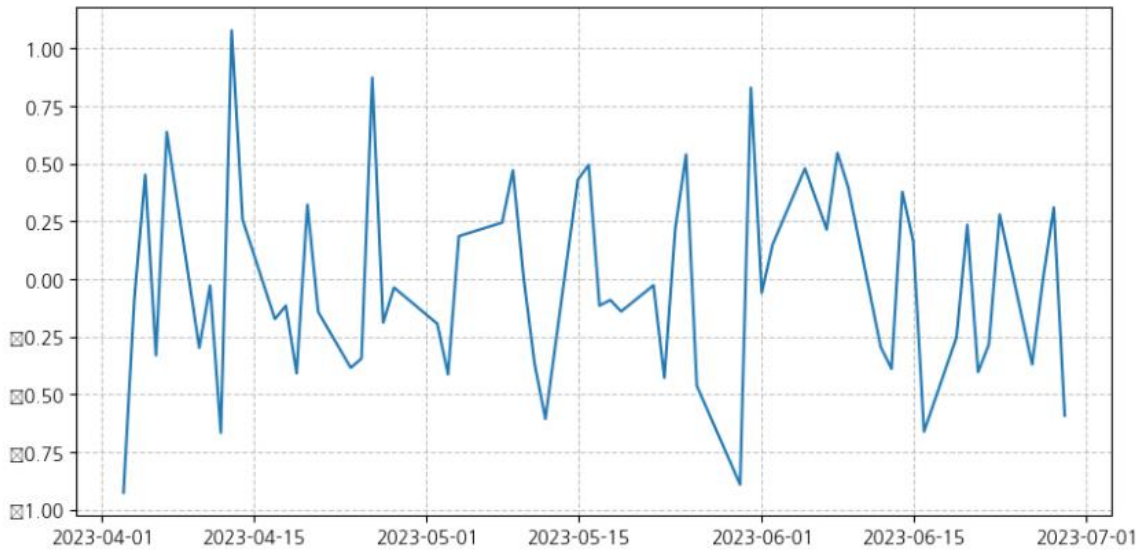
지수와 수익률 비교

일별 거래량 그래프

추적 오차



vs KOSPI200



vs K11

추적 오차 = 펀드 수익률 - 지수 수익률

ACTIVE FUND 코드 공유

코드 구현 과정 개관

경제 국면 분석

팩터 선정

자산 유니버스 구성

자산 별 가중치 계산

Active fund

Active fund
코드 공유

경제 국면 분석

Static Model

Factor와 Weight 고정

모델 구축 용이

Look - Ahead Bias 가능성

Dynamic Model

Factor와 Weight 가변

모델 구축 까다로움

Look - Ahead Bias 가능성 감소

Macro Dynamic Model

국면 인식

Efficient Factors 검색

최종 Factor & Weight 선정

경제 국면 분석

Macro Dynamic Model

국면 인식	현재 시점과 유사했던 과거의 시점을 검색하는 과정

Feature 선택

국면 인식 모델 상의 측정 경제 변수 리스트 및 변수별 유사도 측정 방식

경제 변수	Correl-adj Distance 적용법
KOSPI	Indexed Distance 사용
원/달러	Indexed Distance 사용
원/엔	Indexed Distance 사용
WTI	Indexed Distance 사용
ISM 제조업	월통계치를 일별 데이터로 전환 및 Lagging 적용. Distance 사용
중국 PMI	월통계치를 일별 데이터로 전환 및 Lagging 적용. Distance 사용
KOSPI P/E	Distance 사용
국고3년	Distance 사용
Yield Curve	(통안1년, 국고3년, 국고5년, 국고10년)의 4차원 벡터 적용. Distance 사용
G10 ESI	Distance 사용

자료: 삼성증권

경제 국면 분석

Macro Dynamic Model

국면 인식

Feature 선택

```
path0 = '/content/drive/MyDrive/macro_data/'
path = '/content/drive/MyDrive/macro_data/*'

file_list = glob.glob(path)
csv_list = [file for file in file_list if file.endswith('.csv')]

#폴더안에 있는 csv들의 2003년부터 6월 리밸런싱 시점까지의 데이터 가져오기
k = 0
for x in csv_list:
  longSentence = csv_list[k]
  shortSentence = path0
  newSentence = longSentence.replace(shortSentence, "")
  globals()['{}{}'.format('df_',newSentence[:-4])]=pd.read_csv(x,index_col=0).loc['2003-01-01':'2023-06-05'].fillna(method='ffill')
  k=k+1
```

“Correlation - Adjusted Distance”

경제변수 트렌드의 절댓값과 트렌드의 변화 패턴 유사도를 동시에 고려하는 Distance 계산 방식

Distance Correlation

Distance 측정 : Correlation - Adjusted Distance 사용

단순 Distance

두 시점의 경제 변수 단기 시계열 값의
일치 여부를 거리 개념으로 Distance 산출

Indexed Distance

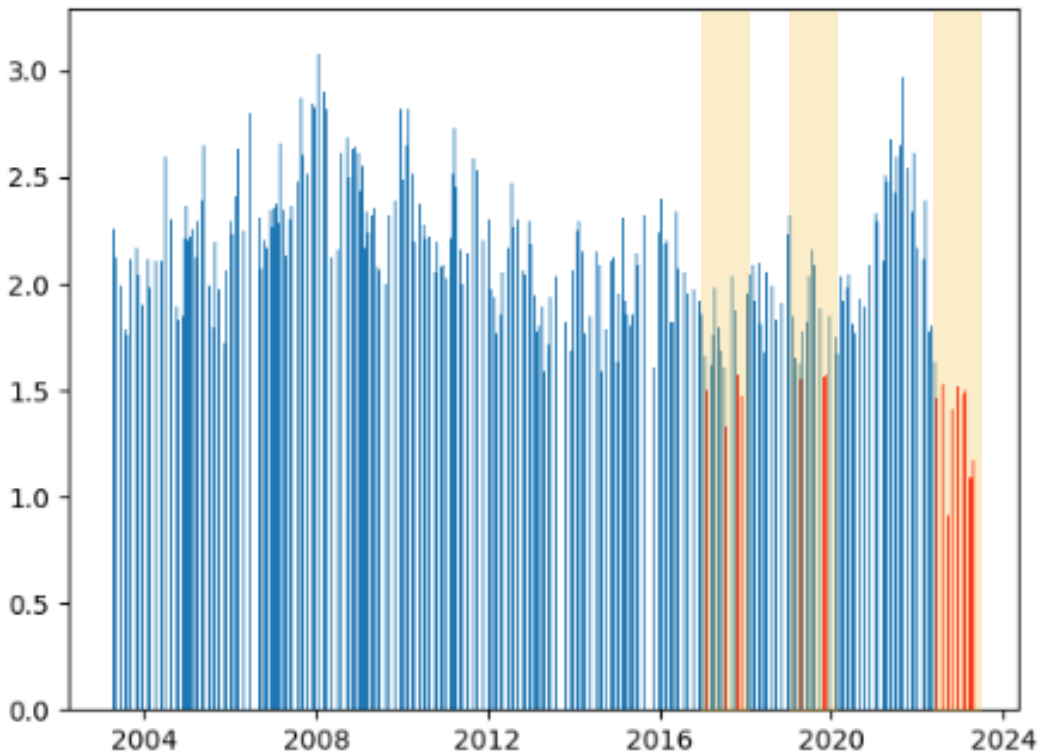
Distance가 발산할 수 있기에,
시계열을 기간 말 값 100으로 변환하는
인덱스를 적용해 Distance 산출

```
def cal_vec(df,t):
  a=(df.iloc[-t:]),iloc[:,0],to_numpy()
  df_temp = pd.DataFrame(columns=['cal'])
  for x in range(1,len(df)-t+1):
    b=(df.iloc[-t-x:-x]),iloc[:,0],to_numpy()
    all_var = df.var()[0] # 전기간 분산 사용
    sigma = 0
    for y in range(t):
      sigma = sigma +(((a[y]-b[y])**2)/all_var)
    distance = (sigma/t)**(1/2)
    cor = (np.cov(a, b)[0][1])/(a.std()*b.std())
    cor_dist=distance+(1-cor)
    df_temp.loc[df.iloc[-t-x:-x].index[-1]] = cor_dist

  global cal
  cal = df_temp.iloc[::-1]
  return cal
```

```
def cal_idxvec(df,t):
  a=(df.iloc[-t:]/(df.iloc[-t:].iloc[-1,0])*100),iloc[:,0],to_numpy()
  df_temp = pd.DataFrame(columns=['cal'])
  for x in range(1,len(df)-t+1):
    b=(df.iloc[-t-x:-x]/df.iloc[-t-x:-x].iloc[-1,0]*100),iloc[:,0],to_numpy()
    all_var = (100*df.std()[0] / df.mean()[0])**2 # 전기간 분산 사용
    sigma = 0
    for y in range(t):
      sigma = sigma +(((a[y]-b[y])**2)/((y+1)*all_var))
    distance = (sigma/t)**(1/2)
    cor = (np.cov(a, b)[0][1])/(a.std()*b.std())
    cor_dist=distance+(1-cor)
    df_temp.loc[df.iloc[-t-x:-x].index[-1]] = cor_dist
    global cal2
    cal2 = df_temp.iloc[::-1]
    return cal2
```

결과 : 5개 경제 변수를 활용한 과거의 유사 국면 선정

```
# df_mean : 피쳐의 평균 거리 데이터프레임
vec_dict = {'kospi_pe': df_kospi_pe, 'bond3y' : df_bond3y}
idxvec_dict = {'kospi' : df_kospi,'won_dollar' : df_won_dollar, 'won_yen' : df_won_yen}#, 'wti' : df_wti}

avg_dist(vec_dict,idxvec_dict,60)
df_mean
```



5개 경제변수의 Distance 평균을 적용한
현재 60일간의 국면과 비슷했던 과거 국면은
2017년과 2019년, 그리고 2022년말~23년 초

Factor 선정

Factor

주식 수익률에 영향을 주는 요인

Value
Size
Momentum
Growth

Multi Factor Portfolio

주가의 수익률에 영향이 있는 서로 다른

펙터를 조합하여 종목을 추출한 후,

리밸런싱 규칙에 따라 포트폴리오를

운용하는 전략

Data Import

재무제표 계정 과목 Data 및 코스피/코스닥 상장 종목의 시장 Data 불러오기

자산	63923470712	1216333773586	798252213878	270722256078	464801905765
부채	26806321140	840704737810	237435015228	85542180287	266030246530
자본	37117149572	375629035776	560817198650	185180075791	198771659235
매출채권	NaN	57094915310	NaN	2586671714	51004829563
매입채무	NaN	33647712198	NaN	NaN	66181541363
유동자산	28752831462	157283046094	34192697827	60883766068	319161183019
유동부채	24532330120	493377808213	225281358153	40438713212	165089889459
현금성자산	5351700872	34542606676	380279267	14830491504	150545870615
장기차입금	2000000000	170393902709	10172895491	34700000000	73000000000
총차입금	8500000000	105500000000	219725782621	12619000000	36725468259
매출액	7240837118	153361916261	NaN	3959016214	79215926905
매출원가	6067522153	135149374725	NaN	627061564	55393075635
매출총이익	1173314965	NaN	NaN	3331954650	23822851270
판매비와관리비	0	NaN	NaN	3055024065	8129153378
영업이익(손실)	354154101	18212541536	27500402499	276930585	11862612607
법인세비용차감전순이익(손실)	565362294	13241693625	26113267625	260565732	17552684258
당기순이익(손실)	565362294	10911765678	26113267625	319724004	13315075365
주당손익(EPS)	0.0	0.0	0.0	0.0	0.0
영업활동현금흐름	-743917467	-1816184834	-4030325520	-939353868	7171696632

감가상각비	455505966	NaN	NaN	NaN	NaN
이자수익	22489239	NaN	NaN	NaN	NaN
이자비용	62616222	NaN	NaN	NaN	NaN
법인세비용(수익)	0	NaN	NaN	NaN	NaN
투자활동현금흐름	-243529291	23924179579	-4416940	-679384049	-1450337855
재무활동현금흐름	-36,683,280	-3,132,093,697	4,415,021,727	-2,414,443,520	9,742,081,756

by **Open Data API**

2022년도 1분기 ~ 2023년도 1분기까지, 코스피 및 코스닥 시장에 상장된 전종목의 분기별 주요 재무제표 데이터 로드

Data Import

재무제표 계정 과목 Data 및 코스피/코스닥 상장 종목의 시장 Data 불러오기

종목명	3S	AJ네트웍스	AK홀딩스	APS홀딩스	AP시스템
20220331	162413010630.0	289830006050.0	290783963950.0	NaN	362933748750.0
20220630	143673047865.0	302003802750.0	212623354050.0	NaN	253671588600.0
20220929	103879546685.0	315114045350.0	185465854000.0	NaN	251379375450.0
20221229	101684264990.0	267823527400.0	227858049200.0	NaN	280414075350.0
20230331	110906226970.0	220533009450.0	236866390680.0	NaN	340775688300.0
종목코드	060310	095570	006840	NaN	265520
BPS	829.0	8075.0	45961.0	NaN	12713.0
PER	63.48	3.35	0.0	NaN	4.67
PBR	2.53	0.71	0.37	NaN	1.44
EPS	33.0	1707.0	0.0	NaN	3932.0
DIV	0.0	4.72	1.16	NaN	1.31
DPS	0.0	270.0	200.0	NaN	240.0
TICKER	60310.0	95570.0	6840.0	NaN	265520.0
SEC_NM_KOR	산업재	산업재	소재	NaN	IT

by **financeDataReader 모듈**

상장 종목의 종가, 거래량과 분기별 마감 영업일의 업종명, 시가총액, BPS, PER, EPS, DIV, DPS Data 로드 후, 시점 및 종목코드 기준으로 OUTER_JOIN 방식의 병합 수행

* 기간
: 2022.01.02 ~ 2023.07.14

* 분기별 마감 영업일
: 22.03.31, 22.06.30, 22.09.29, 23.03.31)

팩터 선정

분석 Data 생성

최종 팩터 선정을 위한 분석 Data 생성

회계 숫자(,) 처리

해결을 위해 “,”를 공백 처리하는 문자열 대체 함수 str.replace()를 활용하였으나 대체되지 않는 일부 계정과목 컬럼이 존재하였고 연속형으로 타입 변환되지 않은 문자열은 팩터 필터조건 적용이 불가함

업종별 분류 후 표준화 수행

각 섹터 내에서 “(수익률 - 평균) / 표준편차 “의 산식을 통해

표준화 수행 후 섹터 별 결과를 수익률 기준 내림차순 정렬

팩터 선정

팩터 종류 탐구 for 멀티팩터 포트폴리오 구성

밸류에이션 팩터(Valuation Factor)

개념

특정 Valuation level에서 투자하였을 때, 좋은 투자 성과가 나타나는지, 확인하는 팩터

의의

“ 근본적으로 기대되는 현금흐름(가치)보다 낮은 가격에 거래되고 있는 종목에 투자한다는 관점 ”

입력 Data

자산, 장기차입금, 유동자산, 유동부채 매출액, 당기손익, 매출총이익 영업활동현금흐름

구현 절차

- 가치주(Value)인지 고평가주/성장주(glamour)인지 여부를 EV/EBITDA 멀티플을 활용하여 분류한다. (시장의 기대치 = 평가)
- 분류한 주식종목 그룹 내에서 → 펀더멘털 F-Score(피오트로스키)를 활용하여 추가적으로 분류한다. (실제 펀더멘털 = 내재가치)
- 저평가인 동시에 낮은 펀더멘탈 스코어를 가진 그룹에 long 포지션 & 고평가인 동시에 낮은 펀더멘털 스코어를 가진 그룹에 short 포지션
- 즉, 시장의 기대치와 실제 펀더멘털의 괴리가 큰 그룹에 초과수익이 집중된다는 가설

구현 산식

- F - Score(조셉 피오트로스키)
  각 조건에 부합할 경우, 점수를 1점 씩 부여하는 방식 (수익성, 재무적 성과, 운영 효율성 고려)
- [EV / EBITDA] = [기업의 시장가치 / 세전영업이익]
  해당 기업의 실제 현금성 이익을 확인하기 위해 활용

팩터 선정

팩터 종류 탐구 for 멀티팩터 포트폴리오 구성

사이즈 팩터(SMB)

개념

기업의 규모를 고려하여 투자 성과를 측정하는 팩터

의의

사이즈 프리미엄을 “시가총액 소형주의 주가수익률”을 “기업 규모에 대한 리스크 프리미엄” 으로 보는 관점

* 효과적으로 사이즈 팩터 투자 전략을 실행하는 방법으로, 밸류, 퀄리티 백터 종목에 투자하는 팩터 포트폴리오 내 소형주 비중을 높이는 방식이 있다.

입력 Data

시가총액, 자본 총계, 우선주 자본금

구현 절차

- 사이즈 변수를 가지고 KOSPI 종목을 기준으로 20%씩 5개 그룹을 나눈 다음
- KOSPI 및 KOSDAQ 종목을 해당 구간에 할당
- 각 사이즈 내에서 → BE/ME 별로 5개 그룹으로 세분화하여 총 25개 그룹으로 분류
- 장부가치 대비 시가총액이 낮아 저평가되어 있을 수 있는 주식종목에 투자하는 관점

구현 산식

- Market Cap(시가총액)
  : 보통주 종가 * 발생 주식의 수
- BM(Book value : Market Value, 장부 가치 대비 시장 가치)
  : 시가총액 / (자본총계 - 우선주 자본금)

팩터 선정

팩터 종류 탐구 for 멀티팩터 포트폴리오 구성

성장성 팩터(Growth)

개념: 회사의 재무 데이터를 분석하여 성장 가능성이 높은 기업의 투자 성과를 측정하는 팩터

입력 Data: 매출액, 매출총이익, 영업이익, 당기손익

구현 절차:

- KOSPI, KOSDAQ 상장종목의 전분기 대비 당기 매출액 변화율, 매출총이익 변화율, 영업이익 변화율, 당기손익 변화율을 산출(QoQ)
- 대표적인 수익성 지표인 영업이익, 매출액, 당기순이익의 전환지표
  (전분기 대비하여 각각의 상태가 계속 지속 중인지 아니면 상태가 전환되었는지 확인)
- - → + : 흑자 전환 / + → - : 적자 전환 / + → + : 흑자 지속 / - → - : 적자 지속
- 다만, 이익 증가율 같은 지표는 "시가총액이나 자산총계와 비교해서 의미 없는 중복들을 걸러낸다."

구현 산식:

- (2023년도 1분기 영업이익 - 2022년 4분기 영업이익) / 2022년 4분기 마감 영업일 시가총액 * 100
- (2023년도 1분기 영업이익 - 2022년 4분기 영업이익) / 2022년 4분기 마감 영업일 자산총계 * 100

모멘텀 팩터(MoM) = 시간민감도

개념: 회사의 실적 추이나 과거의 주가 상승 트렌드가 주가에 반영 되는지 확인하는 팩터 (최근 상승세를 기록한 종목)

구현 산식: t-2월부터 t-12월까지 11개월 간 누적 수익률을 연간 수익률로 환산한 값

팩터 선정

주성분 분석 및 팩터 별 비중 결정 : 선형 결합하여 고윳값으로 비중 결정

수익률에 대한 각 팩터의 기여도(고윳값, 팩터별 계수)를 구하고, 이를 활용하여 가중치 배분

```
class FactorEvaluation_PrincipalComponentAnalysis():
  # 요인의 수익 기여도

  def __init__(self, file_path: str):
    self.differencing_market_return = pd.read_csv(file_path, index_col = 0)

  def linearCombinationPCA(self, k: int):

  # 4. 원하는 결합정도에 따라 상위 k의 고유벡터를 선택하는데 모든 고유값의 누적합계로  설명된 분산 기여율 계산
    sum_eigen_values = np.sum(sorted_eigen_values)
    explained_variance = sorted_eigen_values / sum_eigen_values
    cummulative_variance = np.cumsum(explained_variance)

  # 5. 변환된 데이터로 고유벡터의 전치행렬의 내적을 취하여 원본 데이터를 결합(데이터를 중앙에 배치하기 위해 평균을 뺌)
    n_components = k
    eigen_vectors_subset = sorted_eigen_vectors[:, 0:n_components]

  # 6. 고유벡터를 사용하여 데이터의 좌표방향변환(회전) -> 정사영 / 투영 / 내적
    pca_matrix = np.dot(substract_mean_matrix, eigen_vectors_subset)

    reduction_matrix = pd.Series(np.array(pca_matrix, ndmin = 1).tolist())
    for parenthesis in ["[", "]"]:
      reduction_matrix = reduction_matrix.apply(lambda data: str(data).replace(parenthesis, ""))
    reduction_matrix = reduction_matrix.astype("float")
    reduction_matrix.index = self.differencing_market_index.index
    return reduction_matrix
```

자산 유니버스 구성

자산 별 가중치 계산

Active fund!

...

~

개선 및 보완 사항

유니버스 및 사용 데이터 소개

Constraint 및 리밸런싱

성과 측정 : 백테스팅, 자산군 분석

감사합니다 :)

appendix

- 유진투자증권 안지선, 시계열 분해로 살펴본 현재 매크로, 지수, 업종, 2022.09
- 이동훈, 경기지표를 활용한 스마트베타 팩터 가중치 결정 전략, 2022.06.
- 교보증권 김지혜, 밸류와 모멘텀 팩터를 결합하는 몇가지 방법, 2016.12.
- 김은총 외, 멀티 팩터에 기반한 원자재 포트폴리오 구성 전략, 2020.05.
- 장수정, 팩터 구성 방법론 : 한국 주식시장의 규모, 가치요인을 중심으로 실증 분석, 2022.
- 김용환, 금융공학 레시피, 한빛 미디어, 2018
- 에릭 르윈슨, 금융 파이썬 쿡북, 에이콘출판주식회사, 2021
- 이브 힐피시, 파이썬을 활용한 금융 분석, 한빛 미디어, 2016
- 정호성, 파이썬을 이용한 경제 및 금융 데이터 분석, 자유아카데미, 2023