In [1]:

```
# import library
import pickle, os
import pandas as pd
import numpy as np
import tqdm.notebook as tqdm
import copy
from scipy import stats
from scipy.stats.mstats import winsorize
from statsmodels.formula.api import ols
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
import seaborn as sns


os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩팩터프로젝트')
# # Data Download
with open('FS_손익계산서보정본', 'rb') as f:
    data = pickle.load(f)

with open('Final_FS_ind', 'rb') as f:
    ind = pickle.load(f)

# 특정 자산만 뽑아서 csv로 정리하기
def get_item(concept_id):
    item = []
    item_ind = []
    # check id's address
    id_address = None
    for i in range(len(data)):
        for j in range(3):
            try:
                if concept_id in data[i][j].iloc[:,0].values:
                    id_address = j
            except AttributeError:
                pass
        if id_address != None:
            break

    for i in range(len(data)):
        if type(data[i][id_address]) == type(data[0][id_address]):
            item.append(data[i][id_address][data[i][id_address][data[i][id_address].columns[0]] == concep
            if len(data[i][id_address][data[i][id_address][data[i][id_address].columns[0]] == concept_id]) >
                item_ind.append(ind[i])
    df_item = pd.concat(item)
    date = list(filter(lambda x: x[1] == ('연결재무제표',), df_item.columns))
    if id_address ==0:
        date_clean = list(map(lambda x: x[0], df_item[date].columns))
        df_item[date].columns = date_clean
        df_item2 = df_item[date]
        df_item2.columns = date_clean
        date_shifted = list(map(lambda x: shift_q(x), date_clean))
        df_item3 = df_item2
        df_item3.columns = date_shifted
        df_item4 = df_item3.groupby(level=0, axis=1).last()
        df_item3 = df_item4
        date_shifted2 = list(map(lambda x: shift_d(x), df_item4.columns))
        df_item3.columns = date_shifted2
        df_item5 = df_item3.groupby(level=0, axis=1).last()
        corp_code = list(map(lambda x:x[2], item_ind))
        df_item5['corp_code'] = corp_code
        df_item5 = df_item5.set_index('corp_code')
    else:
```

```
        date_clean = list(map(lambda x: x[0], df_item[date].columns))
        df_item[date].columns = date_clean
        df_item2 = df_item[date]
        df_item2.columns = date_clean
        date_shifted = list(map(lambda x: shift_flow_date(x), date_clean))
        df_item2.columns = date_shifted
        df_item3 = df_item2.groupby(level=0, axis=1).last()
        corp_code = list(map(lambda x:x[2], item_ind))
        df_item3['corp_code'] = corp_code
        df_item3 = df_item3.set_index('corp_code')
        df_item5 = df_item3
    return df_item5

def shift_q(date):
    if date[4:6]=='01' or date[4:6]=='02' or date[4:6]=='03':
        date = date[:4] + '0331'
    elif date[4:6]=='04' or date[4:6]=='05' or date[4:6]=='06':
        date = date[:4] + '0630'
    elif date[4:6]=='07' or date[4:6]=='08' or date[4:6]=='09':
        date = date[:4] + '0930'
    elif date[4:6]=='10' or date[4:6]=='11' or date[4:6]=='12':
        date = date[:4] + '1231'
    return date

def shift_d(date):
    # 케이스 두개 밖에 안되니까 그냥 수동으로 따져주자.
    if date[6:]=='01':
        date = date[:6] + '31'
    elif date[6:]=='29':
        date = date[:6] + '30'
    return date

def shift_flow_date(date):
    # 우선은 연간 데이터만 활용하도록 만들어놓아도 될 듯? 나중에 분기별로 뽑을 수 있는 옵션을 추가
    if date[4:6]=='01' and date[-4:-2]=='12':
        date = date[-8:-2] + '31'
    else:
        date = None
    return date

def get_item_on_date(item, date):
    return get_item(item)[date]

def get_item_for_period(item, date_from, date_to, term = 'Y'):
    # term은 Y, H, Q의 세 가지 옵션으로, 연도별, 반기별, 분기별 데이터를 얻어올 수 있도록 하는 옵션(
    data = []
    item_data = get_item(item)
    if term == 'Y':
        num = int(date_to[:4]) - int(date_from[:4]) +1
        date_working = date_from
        for i in range(num):
            data.append(item_data[date_working])
            date_working = date_working[:2] + str(int(date_working[2:4]) + 1) + date_working[4:]

    elif term == 'H':
        num = int((int(date_to[:4]) - int(date_from[:4])) * 2 + (int(date_to[4:6]) - int(date_from[4:6]))/6 +
        date_working = date_from
        for i in range(num):
            try:
                data.append(item_data[date_working])
            except:
                try:
                    date_working = date_working[:6] + str(61 - int(date_working[6:]))
                    data.append(item_data[date_working])
```

```
                except KeyError:
                    print('해당 날짜의 데이터가 없습니다:', date_working)
            if int(date_working[4:6])+6 > 12:
                date_working = date_working[:2] + str(int(date_working[2:4]) + 1) +'0'+ str(int(date_work
            else:
                date_working = date_working[:4] + str(int(date_working[4:6]) + 6).zfill(2) + '31'

    else:
        num = int((int(date_to[:4]) - int(date_from[:4])) * 4 + (int(date_to[4:6]) - int(date_from[4:6]))/3 +
        date_working = date_from
        for i in range(num):
            try:
                data.append(item_data[date_working])
            except:
                try:
                    date_working = date_working[:6] + str(61 - int(date_working[6:]))
                    data.append(item_data[date_working])
                except KeyError:
                    print('해당 날짜의 데이터가 없습니다:', date_working)
            if int(date_working[4:6])+3 > 12:
                date_working = date_working[:2] + str(int(date_working[2:4]) + 1) +'0331'
            else:
                date_working = date_working[:4] + str(int(date_working[4:6]) + 3).zfill(2) + '31'

    data = pd.concat(data, axis = 1)
    return data


def get_market_data_on_date(date, print_error = True):
    # 에러 메시지를 출력하고 싶지 않으면 print_error를 False로 설정
    os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩팩터프로젝트₩₩KRX_price')
    try:
        data = pd.read_csv(date[2:4] + '_' + str(int(date[4:6])) + '_' + str(int(date[6:8])) + '.csv', encoding
        data['종목코드'] = data['종목코드'].map(lambda x:str(x).zfill(6) if x != None else None)
        if data.isna()['종가'].sum() == len(data):
            raise FileNotFoundError
    except FileNotFoundError:
        if print_error:
            print('해당 날짜의 데이터가 존재하지 않습니다:', date)
        data = None
    return data

def get_market_cap_on_date(date, get_code = False):
    if get_code:
        data = get_market_data_on_date(date)[['종목코드','시장구분','시가총액']].set_index('종목코드')
        data.columns = [['시장구분','Market_Cap']]
    else:
        data = get_market_data_on_date(date)[['종목코드','시가총액']].set_index('종목코드')
        data.columns = [['Market_Cap']]
    data.index.rename('corp_code', inplace = True)
    return data

def last_market_date(date):
    # date의 기본형식은 'YYYYMM' : ex)'201212'. -> 'YYYYMMDD'도 가능하도록 코드 변경했음
    get_data = False
    day = 31
    while not get_data:
        if type(get_market_data_on_date(date[:6] + str(day), print_error = False)) != type(None):
            market_date = date[:6] + str(day)
            get_data = True
        else:
            day -=1
    return market_date
```

```
def find_next_market_date(date, today_opt = False):
    get_data = False
    if today_opt:
        market_date = pd.Timestamp(date)
    else:
        market_date = pd.Timestamp(date) + pd.Timedelta(days=1)
    while not get_data:
        if type(get_market_data_on_date(market_date.strftime('%Y%m%d'), print_error = False)) != type
            get_data = True
        else:
            market_date += pd.Timedelta(days=1)

    return market_date.strftime('%Y%m%d')

def get_market_cap_for_period(date_from, date_to, term = 'Y', ind_match = True):
    # ind_match는 fundamental과 합치기 편하도록 실제 날짜가 아니라 월말 날짜를 표시하도록 하는 옵
    data = []
    ind = []
    if term == 'Y':
        num = int(date_to[:4]) - int(date_from[:4]) +1
        date_working = date_from
        for i in range(num):
            data.append(get_market_cap_on_date(last_market_date(date_working), get_code = True))
            if ind_match:
                ind.append(date_working)
            else:
                ind.append(last_market_date(date_working))
            date_working = date_working[:2] + str(int(date_working[2:4]) + 1) + date_working[4:]

    elif term == 'H':
        num = int((int(date_to[:4]) - int(date_from[:4])) * 2 + (int(date_to[4:6]) - int(date_from[4:6]))/6 +
        date_working = date_from
        for i in range(num):
            data.append(get_market_cap_on_date(last_market_date(date_working), get_code = True))
            if ind_match:
                ind.append(date_working)
            else:
                ind.append(last_market_date(date_working))
            if int(date_working[4:6])+6 > 12:
                date_working = date_working[:2] + str(int(date_working[2:4]) + 1) +'0'+ str(int(date_work
            else:
                date_working = date_working[:4] + str(int(date_working[4:6]) + 6).zfill(2) + '31'

    else:
        num = int((int(date_to[:4]) - int(date_from[:4])) * 4 + (int(date_to[4:6]) - int(date_from[4:6]))/3 +
        date_working = date_from
        for i in range(num):
            data.append(get_market_cap_on_date(last_market_date(date_working), get_code = True))
            if ind_match:
                ind.append(date_working)
            else:
                ind.append(last_market_date(date_working))
            if int(date_working[4:6])+3 > 12:
                date_working = date_working[:2] + str(int(date_working[2:4]) + 1) +'0331'
            else:
                date_working = date_working[:4] + str(int(date_working[4:6]) + 3).zfill(2) + '31'
    cap_data = [x.iloc[:,1] for x in data]
    cat_data = [x.iloc[:,0] for x in data]
    cap_data = pd.concat(cap_data, axis =1)
    cap_data.columns = ind
    cat_data = pd.concat(cat_data, axis =1)
    cat_data.columns = ind
    return cap_data, cat_data
```

```python
def get_market_category(data):
    market_cat = []
    for i in range(len(data.columns)):
        market_cat.append(get_market_cap_on_date(last_market_date(data.columns[i]), get_code = True
    market_cat = pd.concat(market_cat, axis=1)
    market_cat.columns = data.columns
    return market_cat

def get_breakpoint(data, breakpoint):
    absolute_point = []
    rank = data.rank(pct=True)
    for i in range(len(breakpoint)):
        absolute_point.append((data[rank < breakpoint[i]].max() + data[rank > breakpoint[i]].min())/2)
    return absolute_point

def get_mask(data, market_cat = None, market_bp = 'All', breakpoint: list = [0.5]):
    # data는 mask의 기준이 되는 baseline data
    # market_bp는 'ALL', 'KOSPI', 'KOSDAQ'으로, 기준이 되는 breakpoint의 market을 의미
    # breakpoint는 masking의 구분점으로, list안에 float이 담긴 형태를 받음
    breakpoint = np.sort(breakpoint)
    mask_list = []

    data_ind = data.index
    if type(market_cat) != type(None):
        market_cat = market_cat[market_cat.index.map(lambda x: x in data.index)]
        for i in range(len(data)):
            if data_ind[i] not in market_cat.index:
                market_cat.loc[data_ind[i]] = None
        market_cat = market_cat.reindex(index = data.index)
    for i in range(len(data.columns)):
        mask = data.iloc[:,i] * 0 +1
        mask_np = -np.ones(len(mask))
        #rank = data.iloc[:,i].rank(pct = True)
        if market_bp == 'All':
            absolute_point = get_breakpoint(data.iloc[:,i], breakpoint)
        if market_bp == 'KOSPI':
            absolute_point = get_breakpoint(data[market_cat.iloc[:,i]=='KOSPI'].iloc[:,i], breakpoint)
        if market_bp == 'KOSDAQ':
            absolute_point = get_breakpoint(data[market_cat.iloc[:,i]=='KOSDAQ'].iloc[:,i], breakpoint)
        for j in range(len(breakpoint)+1):
            if j < len(breakpoint):
                for k in range(len(mask)):
                    mask_np[k] = j if data.iloc[k,i] < absolute_point[j] and mask_np[k] == -1 else mask_n
            else:
                for k in range(len(mask)):
                    mask_np[k] = j if data.iloc[k,i] > absolute_point[j-1] and mask_np[k] == -1 else mask_
        mask = mask * mask_np
        mask_list.append(mask)
    mask_df = pd.concat(mask_list, axis =1)
    return mask_df

def shift_date_quarter(data, num_of_quarters):
    # data의 date를 원하는 만큼 밀어줌.
    dates = data.columns
    years = [int(x[:4]) for x in dates]
    months = [int(x[4:6]) for x in dates]
    shifted_months = [x + num_of_quarters * 3 for x in months]
    shifted_years = [x + (shifted_months[i]-1) // 12 for i, x in enumerate(years)]
    shifted_months = [(x -1) % 12 +1 for x in shifted_months]
    dates = [str(shifted_years[i]) + str(shifted_months[i]).zfill(2)+ '30' if shifted_months[i] in [6, 9] else
    data2 = copy.deepcopy(data)
    data2.columns = dates
    return data2
```

In [2]:

```
def get_factor_on_date_by_mask(mask, term = 'd', winsorize_limits = 0.01, weight = 'EW'):
    # @@@@@@@@@@@@ option 추가해야할듯! : daily / weekly / monthly / quarterly 정도까지는
    # monthly 계산할 때 첫 달은 어떻게 계산하는거지? 전달 말일 기준으로 계산하나? 아니면 해당 달
    # -> 우선 첫 달은 시가 기준, 이후로는 전달 말일에 리밸런싱 하는 걸 기준으로 계산했음.
    term_conservative = pd.Timestamp(mask.columns[1]) - pd.Timestamp(mask.columns[0]) - pd.Time
    if pd.Timestamp(mask.columns[-1]) + term_conservative >  pd.Timestamp('20210630'):
        date_to = pd.Timestamp('20210630')
    else:
        date_to = pd.Timestamp(last_market_date((pd.Timestamp(mask.columns[-1]) + term_conservati
    date_from = pd.Timestamp(mask.columns[0])
    factor_days = (date_to - date_from).days
    factor_months = (date_to.year - date_from.year)*12 + (date_to.month - date_from.month)

    date_point = date_from.strftime('%Y%m%d')
    # this is for storing the last date
    date_point_marked = None

    total_factor = []
    date_list = []

    if term == 'd':
        flag = -1
#         for _ in range(factor_days):
        for _ in tqdm.tqdm(range(10)):
            if pd.Timestamp(find_next_market_date(date_point)) > pd.Timestamp(mask.columns[flag+1])
                flag+=1
                c_mask = mask.iloc[:,flag]
            date_point_marked = date_point
            first = get_market_data_on_date(date_point_marked, print_error = False).loc[:,['종목코드', '종
            date_point = find_next_market_date(date_point_marked)
            second = get_market_data_on_date(date_point, print_error = False).loc[:,['종목코드', '종가', '시
            date_list.append(date_point)
            return_list = [[] for x in np.arange(mask.max()[0]+1)]
            value_list = [[] for x in np.arange(mask.max()[0]+1)]
            for i in range(len(first)):
                if first.iloc[i,0] in c_mask.index and c_mask.notna().loc[first.iloc[i,0]]:
                    if second.notna()[second['종목코드'] == first.iloc[i,0]].iloc[0,0]:
                        return_list[int(c_mask.loc[first.iloc[i,0]])].append(second[second['종목코드'] == first.i
                    else:
                        return_list[int(c_mask.loc[first.iloc[i,0]])].append(-0.99)
                    value_list[int(c_mask.loc[first.iloc[i,0]])].append(first.iloc[i,2])

            if weight == 'VW':
                for i in range(len(return_list)):
                    # len을 곱해준 건 나중에 mean에서 EW와 같은 형태로 들어가게 하도록 하기 위함.
                    return_list[i] *= value_list[i]/ np.sum(value_list[i]) * len(value_list[i])

            # winsorize
            if winsorize_limits > 0:
                for i in range(len(return_list)):
                    return_list[i] = winsorize(return_list[i], limits = winsorize_limits)

            factor = [np.mean(x) for x in return_list]
            total_factor.append(factor)
        return pd.DataFrame(total_factor, index = date_list)
    elif term == 'm':
        flag = -1
        for _ in tqdm.tqdm(range(factor_months)):
            if flag +1 == mask.shape[1]:
                c_mask = mask.iloc[:,flag]
                date_point_marked = date_point
                first = (get_market_data_on_date(date_point_marked, print_error = False).loc[:,['종목코드',
                c_mask = c_mask.loc[list(filter(lambda x: x in first.index, c_mask[c_mask.notna()].index))]
```

```
        elif pd.Timestamp(find_next_market_date(date_point)) > pd.Timestamp(mask.columns[flag+1
            flag+=1
            c_mask = mask.iloc[:,flag]
            date_point_marked = last_market_date(date_point)
            first = (get_market_data_on_date(date_point_marked, print_error = False).loc[:,['종목코드',
            c_mask = c_mask.loc[list(filter(lambda x: x in first.index, c_mask[c_mask.notna()].index))]
        else:
            date_point_marked = date_point
            first = (get_market_data_on_date(date_point_marked, print_error = False).loc[:,['종목코드',

        # 통일된 수식으로 flag가 바뀐 경우의 말일을 계산할 수 있음
        date_point = last_market_date(find_next_market_date(date_point_marked))
        second = (get_market_data_on_date(date_point, print_error = False).loc[:,['종목코드', '종가', '
        date_list.append(date_point[:6])
        return_value = get_market_return_for_period(date_point_marked, date_point)
        return_list = []
        value_list = []
        for i in range(int(mask.max()[0]+1)):
            return_list.append(return_value.loc[c_mask[c_mask == i].index].loc[:,['등락률']].values)
            value_list.append(return_value.loc[c_mask[c_mask == i].index].loc[:,['시가총액']].values)
        c_mask.drop(list(set(c_mask.index) - set(second.index)), inplace=True)
        if weight == 'VW':
            for i in range(len(return_list)):
                # len을 곱해준 건 나중에 mean에서 EW와 같은 형태로 들어가게 하도록 하기 위함.
                return_list[i] *= value_list[i]/ np.sum(value_list[i]) * len(value_list[i])
        # winsorize
        if winsorize_limits > 0:
            for i in range(len(return_list)):
                return_list[i] = winsorize(return_list[i], limits = winsorize_limits)
        factor = [np.mean(x) for x in return_list]
        total_factor.append(factor)
    return pd.DataFrame(total_factor, index = date_list)
```

# Pfo Construction

In [3]:

```
def make_portfolio(pfo, factor, market_bp= 'All', breakpoint=[0.5]):
    # 여기서 앞에 들어가는 거는 마스크, 뒤에 들어가는 거는 실제값.
    max_num = int(pfo.max()[0]+1)
    new_df = pd.DataFrame(index = pfo.index, columns = pfo.columns)
    for i in range(len(pfo.columns)):
        for j in range(max_num):
            get_index = list(filter(lambda x: x in factor.index, pfo.iloc[:,i][pfo.iloc[:,i]==j].index))
            pfo_mask = get_mask(pd.DataFrame(factor.loc[get_index].iloc[:,i]), get_market_category(pd.D
            new_df.iloc[:,i].loc[pfo_mask.index] = pfo_mask.values[:,0]
    return new_df

def serialize_pfo(mask_list: list):
    # 사전식 배열. 앞에서부터 사전식으로 serialize하여 뱉음
    serial_num = [x.max().max()+1 for x in mask_list]
    for i in reversed(range(len(serial_num)-1)):
        serial_num[i] *= serial_num[i+1]
    serial_num += [1]
    serial_num = serial_num[1:]
    serial_elem = [x * mask_list[i] for i, x in enumerate(serial_num)]
    for i in range(len(serial_elem)-1):
        serial_elem[0] += serial_elem[i+1]
    return serial_elem[0]
```

# Regression 해볼 수 있도록 만들기

In [4]:

```
def get_market_return_for_period(date_from, date_to):
    first = get_market_data_on_date(find_next_market_date(date_from, today_opt = True), print_error =
    df = pd.DataFrame()
    flag = find_next_market_date(date_from, today_opt = True)
    while flag != date_to:
        flag = find_next_market_date(flag)
        df = pd.concat([df, get_market_data_on_date(flag, print_error = False).loc[:,['종목코드', '등락률']]
        if flag == date_to:
            second = get_market_data_on_date(flag, print_error = False).loc[:, ['종목코드', '종가']].set_ind
    df = df.applymap(lambda x: (100 + x)/100)
    for i in range(len(df.columns)):
        if i == 0:
            geo_sum = df.iloc[:,i].copy()
        else:
            geo_sum *= df.values[:,i]
    compute_by_price = second['종가'] / first['종가']
    # 가정 : 3% 이상 차이가 날 수 없다. 계산상 최대 1프로 정도 차이지만, 좀 갭을 두었음.
    final_return = (compute_by_price - geo_sum < 0.03) * compute_by_price + (compute_by_price - g
    final_return = pd.concat([final_return.loc[first.index].fillna(0.01), first['시가총액']], axis=1).rename({0:'
    final_return['등락률'] = final_return['등락률'] -1
    return final_return
```

In [5]:

```
def get_market_return(mask, term = 'd'):
    term_conservative = pd.Timestamp(mask.columns[1]) - pd.Timestamp(mask.columns[0]) - pd.Time
    if pd.Timestamp(mask.columns[-1]) + term_conservative >  pd.Timestamp('20210630'):
        date_to = pd.Timestamp('20210630')
    else:
        date_to = pd.Timestamp(last_market_date((pd.Timestamp(mask.columns[-1]) + term_conservati
    date_from = pd.Timestamp(mask.columns[0])
    factor_days = (date_to - date_from).days
    factor_months = (date_to.year - date_from.year)*12 + (date_to.month - date_from.month)

    date_point = date_from.strftime('%Y%m%d')
    # this is for storing the last date
    date_point_marked = None

    total_return = []
    date_list = []

    if term == 'd':
        flag = -1
        date_point = find_next_market_date(date_point)
        for _ in tqdm.tqdm(range(factor_days)):
            if pd.Timestamp(find_next_market_date(date_point)) > pd.Timestamp(mask.columns[flag+1])
                flag+=1
                c_mask = mask.iloc[:,flag]
            first = get_market_data_on_date(date_point, print_error = False).loc[:,['종목코드', '종가', '시가
            date_list.append(date_point)
            date_point = find_next_market_date(date_point)

            market_cap = first['시가총액'].sum()
            R = first['등락률'] * (first['시가총액'] / market_cap)
            total_return.append(R.fillna(0).sum())

        return pd.DataFrame(total_return, index = date_list)
    elif term == 'm':
        flag = -1
        for _ in tqdm.tqdm(range(factor_months)):
            if flag +1 == mask.shape[1]:
                c_mask = mask.iloc[:,flag]
                date_point = find_next_market_date(date_point)
```

```
            date_point_marked = date_point
        elif pd.Timestamp(find_next_market_date(date_point)) > pd.Timestamp(mask.columns[flag+1
            flag+=1
            c_mask = mask.iloc[:,flag]
            date_point_marked = last_market_date(date_point)
        else:
            date_point_marked = date_point

        date_point = last_market_date(find_next_market_date(date_point_marked))
        return_value = get_market_return_for_period(date_point_marked, date_point)
        date_list.append(date_point[:6])
        market_cap = return_value['시가총액'].sum()
        R = return_value['등락률'] * (return_value['시가총액'] / market_cap)

        total_return.append(R.replace([np.inf, -np.inf], np.nan).fillna(0).sum())
    return pd.DataFrame(total_return, index = date_list)
```

# 모듈화하기

In [245...

```
def get_risk_free_rate(pfo, risk_free):
    return risk_free.loc[pfo.index]

def get_ols_model(mask, R_m, pfo_column, risk_free, *args):
    R_f = (get_risk_free_rate(pfo_column, risk_free).astype(float)/1200).squeeze() #monthly로 바꾸고 %
    df = pd.DataFrame([(pfo_column - R_f.T).values.squeeze(), (R_m - R_f).values.squeeze()] + [x.values
    df.columns = ['pfo', 'market'] + ['factor' + str(i) for i, x in enumerate(args)]
    var_str = ''
    for i in range(len(df.columns)-2):
        var_str += ' + ' + ['factor' + str(i) for i, x in enumerate(args)][i]
    model = ols(formula = 'pfo ~ market' + var_str,data = df).fit()
    return model
```

# 이제 FF5 구현해보자.

## FF5 model with 크롤링 데이터

- 2009년(CMA), 2010년(HML, RMW), 2011년(SMB) 데이터부터 써서 2011년 6월30일 종가 기준 리밸런싱이 처음.
- 2018년(CMA), 2019년(HML, RMW), 2020년(SMB) 데이터까지 써서 2020년 6월30일 종가 기준 리밸런싱이 마지막.
- 따라서 regression의 기간은 2011년 7월 ~ 2021년 6월
- 상하위 1% winsorization
- (size/BEME), (size/OP), (size/INV)로 25포트폴리오에 대해 OLS분석 #### 차례

1. (size/BEME) 25포트폴리오에 대해 OLS분석
   (1) Rm, SMB, HML, RMW, CMA로 regression
   (2) HML를 다른 4개 팩터에 projection시켜서 HMLO로 대체

2. (size/OP) 25포트폴리오에 대해 OLS분석
   (1) Rm, SMB, HML, RMW, CMA로 regression
   (2) HML를 다른 4개 팩터에 projection시켜서 HMLO로 대체

3. (size/INV)로 25포트폴리오에 대해 OLS분석
   (1) Rm, SMB, HML, RMW, CMA로 regression
   (2) HML를 다른 4개 팩터에 projection시켜서 HMLO로 대체

4. 블룸버그 데이터와의 비교 (우리 데이터가 괜찮음을 증명)

5. Size/BEME 25 portfolio에 대해 Window Regression

In [103…

```
# risk free rate
os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩팩터프로젝트')
cd_91 = pd.read_csv('cd금리.csv', encoding='cp949', header=2, index_col=0).T.iloc[:, [4]]
cd_91.index = cd_91.index.map(lambda x: x[:6])
cd_91.columns = ['cd_91']
os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩팩터프로젝트₩₩KRX_price')
```

In [8]:

```
Market_cap = get_market_cap_for_period('20101231', '20191231')

equity = get_item_for_period('ifrs_Equity', '20101231', '20191231')

BEME = equity/Market_cap[0]
```

```
C:₩Users₩jky93₩Anaconda3₩lib₩site-packages₩pandas₩core₩common.py:241: VisibleDeprecationW
arning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-o
r ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  result = np.asarray(values, dtype=dtype)
```

In [9]:

```
Market_cap = get_market_cap_for_period('20110630', '20200630')
```

In [10]:

```
# SMB
SMB_mask = get_mask(Market_cap[0], get_market_category(Market_cap[0]), market_bp = 'KOSPI')

SMB = get_factor_on_date_by_mask(SMB_mask, term = 'm', winsorize_limits = 0.01, weight = 'VW')
```

In [11]:

```
# HML
BEME_shifted = shift_date_quarter(BEME, 2)

HML_mask = get_mask(BEME_shifted, get_market_category(BEME_shifted), market_bp = 'KOSPI', breal

HML = get_factor_on_date_by_mask(HML_mask, term = 'm', winsorize_limits = 0.01, weight = 'VW')
```

```
C:₩Users₩jky93₩Anaconda3₩lib₩site-packages₩pandas₩core₩indexing.py:723: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value, self.name)
```

In [12]:

```
# RMW 계산하기
# (영업이익 - 이자비용) / 총자산
Total_asset = get_item_for_period('ifrs_Assets', '20101231', '20191231')
fcost1 = get_item_for_period('ifrs_FinanceCosts', '20101231', '20191231')
fcost2 = get_item_for_period('dart_InterestExpenseFinanceExpense', '20101231', '20191231')
```

```
fcost = pd.concat([fcost1, fcost2], axis = 1).groupby(level=0, axis=1).last()
OI = get_item_for_period('dart_OperatingIncomeLoss', '20101231', '20191231')
```

In [13]:
```
RMW_base = (OI - fcost) / Total_asset

RMW_base_shifted = shift_date_quarter(RMW_base, 2)

RMW_mask = get_mask(RMW_base_shifted, get_market_category(RMW_base_shifted), market_bp = 'K

RMW = get_factor_on_date_by_mask(RMW_mask, term = 'm', winsorize_limits = 0.01, weight = 'VW')
```

In [14]:
```
# CMA 계산하기

Total_asset = get_item_for_period('ifrs_Assets', '20091231', '20181231')
pnl = get_item_for_period('ifrs_ProfitLoss', '20101231', '20191231')
Total_asset_shifted = shift_date_quarter(Total_asset, 4)
CMA_base = pnl / Total_asset_shifted
CMA_base_shifted = shift_date_quarter(CMA_base, 2)
CMA_mask = get_mask(CMA_base_shifted, get_market_category(CMA_base_shifted), market_bp = 'KO
CMA = get_factor_on_date_by_mask(CMA_mask, term = 'm', winsorize_limits = 0.01, weight = 'VW')
```

In [111…
```
Rf = (get_risk_free_rate(SMB, cd_91).astype(float)/1200).squeeze()
```

In [112…
```
#팩터 정리
factors = pd.concat([(get_market_return(SMB_mask, term='m').squeeze()), Rf, SMB.iloc[:,0] - SMB.iloc[:
factors.columns = ['Rm', 'risk-free', 'SMB', 'HML', 'RMW', 'CMA']
factors
```

Out[112…

| | Rm | risk-free | SMB | HML | RMW | CMA |
|---|---|---|---|---|---|---|
| **201107** | 0.022227 | 0.002992 | 0.067616 | 0.027316 | -0.036970 | 0.042315 |
| **201108** | -0.114518 | 0.002992 | 0.014303 | 0.054783 | 0.020513 | -0.001481 |
| **201109** | -0.060949 | 0.002983 | -0.041692 | -0.039419 | 0.018520 | -0.063712 |
| **201110** | 0.079782 | 0.002983 | 0.034043 | -0.038057 | -0.044172 | 0.038830 |
| **201111** | -0.029902 | 0.002967 | 0.032046 | 0.017156 | 0.021897 | -0.001703 |
| **…** | … | … | … | … | … | … |
| **202102** | -0.019686 | 0.000608 | 0.008442 | 0.027952 | -0.020603 | 0.004320 |
| **202103** | 0.008959 | 0.000625 | 0.049235 | 0.043832 | 0.001327 | 0.030240 |
| **202104** | -0.000869 | 0.000617 | 0.049014 | 0.031919 | -0.036813 | 0.043665 |
| **202105** | 0.020393 | 0.000567 | 0.004343 | 0.034795 | 0.022790 | 0.015164 |
| **202106** | 0.026090 | 0.000550 | 0.032685 | -0.000403 | -0.000863 | 0.014150 |

120 rows × 6 columns

In [16]:
```
HMLO_ols = ols(formula = 'HML ~ Rm+SMB+RMW+CMA', data = factors).fit()
HMLO_ols.summary()
```

Out[16]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | HML | **R-squared:** | 0.218 |
| **Model:** | OLS | **Adj. R-squared:** | 0.191 |
| **Method:** | Least Squares | **F-statistic:** | 8.003 |
| **Date:** | Wed, 23 Mar 2022 | **Prob (F-statistic):** | 9.96e-06 |
| **Time:** | 10:39:36 | **Log-Likelihood:** | 260.94 |
| **No. Observations:** | 120 | **AIC:** | -511.9 |
| **Df Residuals:** | 115 | **BIC:** | -497.9 |
| **Df Model:** | 4 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | 0.0047 | 0.003 | 1.776 | 0.078 | -0.001 | 0.010 |
| **Rm** | 0.0287 | 0.067 | 0.427 | 0.671 | -0.105 | 0.162 |
| **SMB** | 0.0723 | 0.097 | 0.742 | 0.459 | -0.121 | 0.265 |
| **RMW** | -0.1546 | 0.165 | -0.939 | 0.350 | -0.481 | 0.172 |
| **CMA** | 0.3090 | 0.198 | 1.560 | 0.121 | -0.083 | 0.701 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 9.530 | **Durbin-Watson:** | 2.022 |
| **Prob(Omnibus):** | 0.009 | **Jarque-Bera (JB):** | 15.969 |
| **Skew:** | -0.316 | **Prob(JB):** | 0.000341 |
| **Kurtosis:** | 4.672 | **Cond. No.** | 96.6 |

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [113…

```
beta_Rm = HMLO_ols.params['Rm']
beta_SMB = HMLO_ols.params['SMB']
beta_RMW = HMLO_ols.params['RMW']
beta_CMA = HMLO_ols.params['CMA']
HMLO = pd.DataFrame(factors["HML"] - beta_Rm * factors['Rm'].squeeze() - beta_SMB*factors['SMB']
HMLO.columns = ['HMLO']
factors = pd.concat([factors, HMLO], axis=1)
factors
```

Out[113…

| | Rm | risk-free | SMB | HML | RMW | CMA | HMLO |
|---|---|---|---|---|---|---|---|
| **201107** | 0.022227 | 0.002992 | 0.067616 | 0.027316 | -0.036970 | 0.042315 | 0.002997 |
| **201108** | -0.114518 | 0.002992 | 0.014303 | 0.054783 | 0.020513 | -0.001481 | 0.060668 |
| **201109** | -0.060949 | 0.002983 | -0.041692 | -0.039419 | 0.018520 | -0.063712 | -0.012103 |
| **201110** | 0.079782 | 0.002983 | 0.034043 | -0.038057 | -0.044172 | 0.038830 | -0.061638 |

| | Rm | risk-free | SMB | HML | RMW | CMA | HMLO |
|---|---|---|---|---|---|---|---|
| **201111** | -0.029902 | 0.002967 | 0.032046 | 0.017156 | 0.021897 | -0.001703 | 0.019611 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **202102** | -0.019686 | 0.000608 | 0.008442 | 0.027952 | -0.020603 | 0.004320 | 0.023386 |
| **202103** | 0.008959 | 0.000625 | 0.049235 | 0.043832 | 0.001327 | 0.030240 | 0.030876 |
| **202104** | -0.000869 | 0.000617 | 0.049014 | 0.031919 | -0.036813 | 0.043665 | 0.009215 |
| **202105** | 0.020393 | 0.000567 | 0.004343 | 0.034795 | 0.022790 | 0.015164 | 0.032734 |
| **202106** | 0.026090 | 0.000550 | 0.032685 | -0.000403 | -0.000863 | 0.014150 | -0.008021 |

120 rows × 7 columns

## Confirm monotonitcity and t-test

In [86]:
```
SMB_mon_mask = get_mask(Market_cap[0], get_market_category(Market_cap[0]), market_bp = 'KOSPI

SMB_mon = get_factor_on_date_by_mask(SMB_mon_mask, term = 'm', winsorize_limits = 0.01, weight
```

In [27]:
```
HML_mon_mask = get_mask(BEME_shifted, get_market_category(BEME_shifted), market_bp = 'KOSPI',

HML_mon = get_factor_on_date_by_mask(HML_mon_mask, term = 'm', winsorize_limits = 0.01, weigh
```

```
C:₩Users₩jky93₩Anaconda3₩lib₩site-packages₩pandas₩core₩indexing.py:723: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value, self.name)
```

In [28]:
```
RMW_mon_mask = get_mask(RMW_base_shifted, get_market_category(RMW_base_shifted), market_b

RMW_mon = get_factor_on_date_by_mask(RMW_mon_mask, term = 'm', winsorize_limits = 0.01, weig
```

In [29]:
```
CMA_mon_mask = get_mask(CMA_base_shifted, get_market_category(RMW_base_shifted), market_bp

CMA_mon = get_factor_on_date_by_mask(CMA_mon_mask, term = 'm', winsorize_limits = 0.01, weigh
```

In [291...
```
def article(excess_name, mon, Rf, cols=['P1 (low)', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7', 'P8', 'P9', 'P10 (high)']):
    from scipy import stats
    excess = mon.apply(lambda x: x- Rf, axis=0)
    t_stat, p_val = stats.ttest_1samp(excess, 0)
    plt.title('Excess return of {} sorted portfolios'.format(excess_name))
    graph = plt.bar(x=cols, height=excess.mean())
    excess_mean = SMB_excess_return.mean().values
    table = pd.DataFrame([excess_mean, t_stat, p_val])
    table.columns = cols
```

```
    table.index=['excess return', 't-stat', 'p-value']
    return graph, table
```

In [347...

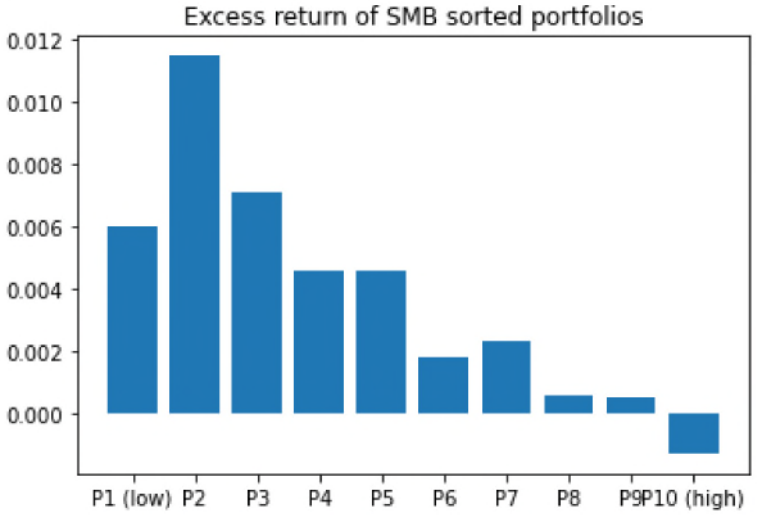
```
def reshape(row, width, w_num, length, l_num):
    df = pd.concat([row.iloc[(i)*(w_num):(i+1)*(l_num)].reset_index(drop=True) for i in range(0, w_num
    df.index = df.index.map(lambda x: x+1)
    df.index.name = width
    df.columns = range(1, l_num+1)
    df.columns.name = length
    return df.T
```

In [292...

```
article('SMB', SMB_mon, factors['risk-free'])
```

Out[292...

```
(<BarContainer object of 10 artists>,
                P1 (low)        P2        P3        P4        P5        P6  ₩
excess return  0.006001  0.011483  0.007090  0.004611  0.004585  0.001797
t-stat         1.352477  2.230558  1.365923  0.883952  0.902121  0.353797
p-value        0.178787  0.027585  0.174539  0.378505  0.368815  0.724117

                    P7        P8        P9  P10 (high)
excess return  0.002340  0.000572  0.000537   -0.001272
t-stat         0.504866  0.122562  0.122757   -0.277891
p-value        0.614586  0.902661  0.902507    0.781578  )
```

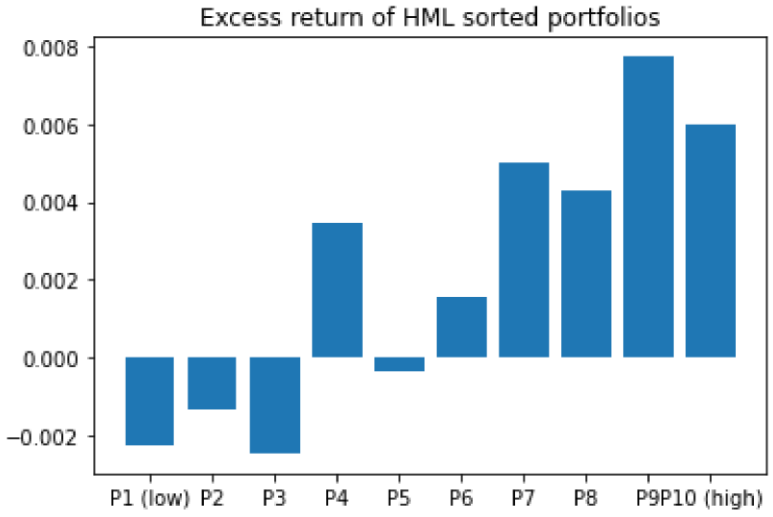



In [293...

```
article('HML', HML_mon, factors['risk-free'])
```

Out[293...

```
(<BarContainer object of 10 artists>,
                P1 (low)        P2        P3        P4        P5        P6  ₩
excess return  0.006001  0.011483  0.007090  0.004611  0.004585  0.001797
t-stat        -0.551246 -0.340867 -0.842341  1.064894 -0.086682  0.319444
p-value        0.582498  0.733805  0.401287  0.289080  0.931070  0.749950

                    P7        P8        P9  P10 (high)
excess return  0.002340  0.000572  0.000537   -0.001272
t-stat         1.191384  0.818441  1.411412    1.116153
p-value        0.235873  0.414741  0.160732    0.266605  )
```




In [294...

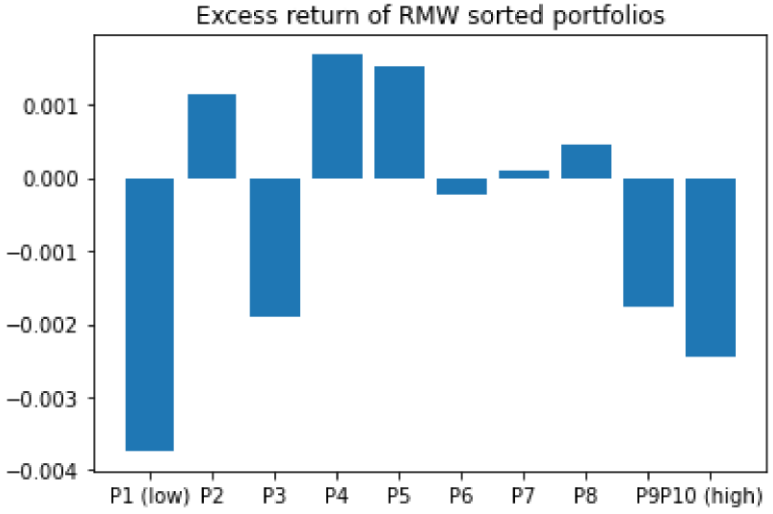
```
article('RMW', RMW_mon, factors['risk-free'])
```

Out[294...

(<BarContainer object of 10 artists>,

| | P1 (low) | P2 | P3 | P4 | P5 | P6 ₩ |
|---|---|---|---|---|---|---|
| excess return | 0.006001 | 0.011483 | 0.007090 | 0.004611 | 0.004585 | 0.001797 |
| t-stat | -0.764551 | 0.223228 | -0.414917 | 0.378814 | 0.353226 | -0.061508 |
| p-value | 0.446052 | 0.823741 | 0.678950 | 0.705501 | 0.724544 | 0.951058 |

| | P7 | P8 | P9 | P10 (high) |
|---|---|---|---|---|
| excess return | 0.002340 | 0.000572 | 0.000537 | -0.001272 |
| t-stat | 0.018414 | 0.111369 | -0.454886 | -0.708655 |
| p-value | 0.985339 | 0.911511 | 0.650020 | 0.479925 ) |




In [295...

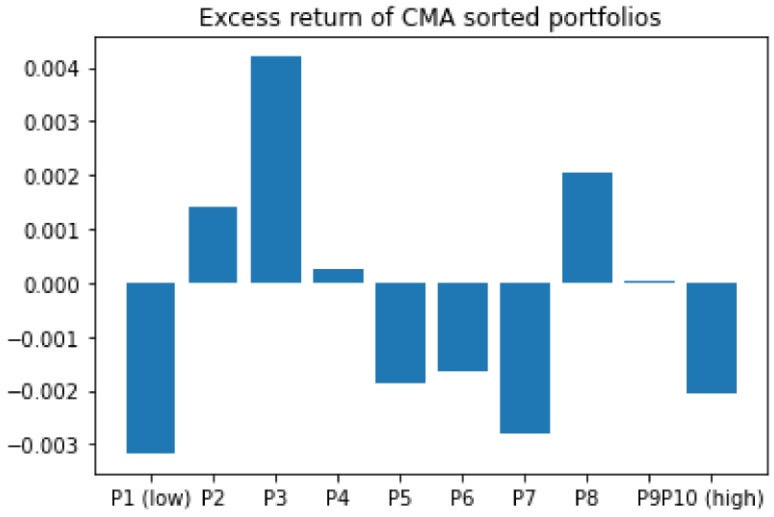
```
article('CMA', CMA_mon, factors['risk-free'])
```

Out[295...

(<BarContainer object of 10 artists>,

| | P1 (low) | P2 | P3 | P4 | P5 | P6 ₩ |
|---|---|---|---|---|---|---|
| excess return | 0.006001 | 0.011483 | 0.007090 | 0.004611 | 0.004585 | 0.001797 |
| t-stat | -0.649305 | 0.273548 | 0.917555 | 0.056450 | -0.392137 | -0.324329 |
| p-value | 0.517392 | 0.784906 | 0.360708 | 0.955078 | 0.695659 | 0.746258 |

| | P7 | P8 | P9 | P10 (high) |
|---|---|---|---|---|
| excess return | 0.002340 | 0.000572 | 0.000537 | -0.001272 |
| t-stat | -0.652680 | 0.488139 | 0.006490 | -0.806856 |
| p-value | 0.515221 | 0.626349 | 0.994832 | 0.421359 ) |




## CAPM alpha

In [296...

```
SMB_OLS_data = pd.concat([SMB_mon, factors['Rm']], axis=1)

coef = []
tvalues = []
r = []

for i in range(10):
    model = get_ols_model(SMB, factors['Rm'], SMB_OLS_data.iloc[:,i], cd_91)
    coef.append(model.params)
    tvalues.append(model.tvalues)
    r.append(model.rsquared_adj)
```

In [297...

```
pd.concat(coef, axis=1)
```

Out[297...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.006471 | 0.012178 | 0.007846 | 0.005377 | 0.005350 | 0.002554 | 0.003029 | 0.001294 | 0.00 |
| **market** | 0.555788 | 0.821931 | 0.893609 | 0.904471 | 0.904771 | 0.895820 | 0.814204 | 0.854366 | 0.83 |

In [298...

```
pd.concat(tvalues, axis=1)
```

Out[298...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 1.705583 | 3.167543 | 2.171983 | 1.492363 | 1.572938 | 0.743149 | 0.960354 | 0.43048( |
| **market** | 6.693773 | 9.768865 | 11.304453 | 11.472179 | 12.155875 | 11.909373 | 11.795623 | 12.98303 |

In [299...

```
r
```

Out[299...

```
[0.269071415904483,
 0.4424420396366695,
 0.5158481757864919,
 0.5232579770258186,
 0.5522364935536769,
 0.5420138073667053,
```

```
0.5372111904025378,
0.5847281756882841,
0.6303646558777123,
0.9207943802881485]
```

In [300…]
```
HML_OLS_data = pd.concat([HML_mon, factors['Rm']], axis=1)

coef = []
tvalues = []
r = []

for i in range(10):
    model = get_ols_model(SMB, factors['Rm'], HML_OLS_data.iloc[:,i],cd_91)
    coef.append(model.params)
    tvalues.append(model.tvalues)
    r.append(model.rsquared_adj)
```

In [301…]
```
pd.concat(coef, axis=1)
```

Out[301…]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | -0.001668 | -0.000705 | -0.001995 | 0.003923 | 0.000261 | 0.002319 | 0.005583 | 0.00501 | 0.0 |
| **market** | 0.676728 | 0.725246 | 0.554692 | 0.574052 | 0.745470 | 0.875474 | 0.705949 | 0.87718 | 0.8 |

In [302…]
```
pd.concat(tvalues, axis=1)
```

Out[302…]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | -0.570052 | -0.293068 | -1.116893 | 1.813364 | 0.089667 | 0.697315 | 1.876932 | 1.347 |
| **market** | 10.565915 | 13.771456 | 14.187198 | 12.125353 | 11.694905 | 12.031769 | 10.845705 | 10.783 |

In [303…]
```
r
```

Out[303…]
```
[0.4817943452568685,
 0.6132005964155368,
 0.6272823990941467,
 0.5509843752251391,
 0.5329135188725229,
 0.5471212000007231,
 0.4949694522694814,
 0.49208942976585224,
 0.4263618227324695,
 0.5145374517256782]
```

In [304…]
```
RMW_OLS_data = pd.concat([RMW_mon, factors['Rm']], axis=1)

coef = []
tvalues = []
r = []

for i in range(10):
    model = get_ols_model(SMB, factors['Rm'], RMW_OLS_data.iloc[:,i], cd_91)
    coef.append(model.params)
```

```
    tvalues.append(model.tvalues)
    r.append(model.rsquared_adj)
```

In [305...

```
pd.concat(coef, axis=1)
```

Out[305...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | -0.003070 | 0.001885 | -0.001248 | 0.002339 | 0.002182 | 0.000307 | 0.000813 | 0.001128 |
| **market** | 0.792596 | 0.890825 | 0.770808 | 0.774008 | 0.766429 | 0.630329 | 0.856314 | 0.789553 |

In [306...

```
pd.concat(tvalues, axis=1)
```

Out[306...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **Intercept** | -0.850566 | 0.547243 | -0.382627 | 0.764960 | 0.742946 | 0.119321 | 0.251936 |
| **market** | 10.035760 | 11.820020 | 10.801871 | 11.564972 | 11.924458 | 11.197387 | 12.126096 |

In [307...

```
r
```

Out[307...

```
[0.455916639507427,
 0.5382457936476319,
 0.49292744255234444,
 0.5273061616927621,
 0.5426466402092507,
 0.5110556291525401,
 0.5510148826523888,
 0.6385127706480623,
 0.6032339243724412,
 0.6142705518520581]
```

In [308...

```
CMA_OLS_data = pd.concat([CMA_mon, factors['Rm']], axis=1)

coef = []
tvalues = []
r = []

for i in range(10):
    model = get_ols_model(SMB, factors['Rm'], CMA_OLS_data.iloc[:,i], cd_91)
    coef.append(model.params)
    tvalues.append(model.tvalues)
    r.append(model.rsquared_adj)
```

In [309...

```
pd.concat(coef, axis=1)
```

Out[309...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | -0.002522 | 0.002191 | 0.004879 | 0.001007 | -0.001120 | -0.000829 | -0.002111 | 0.002679 |
| **market** | 0.764520 | 0.899623 | 0.804794 | 0.871402 | 0.865247 | 0.947828 | 0.795404 | 0.747222 |

In [310…
```
pd.concat(tvalues, axis=1)
```

Out[310…

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | -0.681803 | 0.601851 | 1.569997 | 0.325779 | -0.368164 | -0.267513 | -0.789063 | 0.955 |
| **market** | 9.444163 | 11.291640 | 11.834250 | 12.888375 | 12.991399 | 13.982208 | 13.585097 | 12.178 |

In [311…
```
r
```

Out[311…
```
[0.4256540899005392,
 0.5152771668602183,
 0.5388481285078217,
 0.58114846368766,
 0.5850426394151667,
 0.6204172694723316,
 0.6066828981345,
 0.5531685403995223,
 0.4389455522294041,
 0.627500334764548]
```

In [ ]:

In [ ]:

## 1. (size/BEME) 25포트폴리오에 대해 OLS분석

# 기간에 따라서 정반대의 결과! 11년 ~ 했을 때는 RMW, CMA, HML가 좋았다 근데, 13년부터 돌리니까, RMW, CMA는 전혀 유의하지 않고, market이랑 SMB가 유효!

In [266…
```
# HML pfo
hml_pfo_mask = make_portfolio(get_mask(Market_cap[0], Market_cap[1], breakpoint = [0.2, 0.4, 0.6, 0

pfo25_hml_pfo = serialize_pfo([get_mask(Market_cap[0], Market_cap[1], breakpoint = [0.2, 0.4, 0.6, 0.8

pfo_return_hml_pfo = get_factor_on_date_by_mask(pfo25_hml_pfo, term = 'm', winsorize_limits= 0.01,
```

In [312…
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

## 1. (size/BEME) 25포트폴리오에 대해 OLS분석

(1) Rm, SMB, HML, RMW, CMA로 regression

In [313…
```
coef = []
tvalues = []
```

```
pvalues = []
se = []
r = []
for i in range(25):
    model = get_ols_model(pfo25_hml_pfo, factors['Rm'], pfo_return_hml_pfo.iloc[:,i], cd_91, factors['SM
    coef.append(model.params)
    tvalues.append(model.tvalues)
    pvalues.append(model.pvalues)
    se.append(model.bse)
    r.append(model.rsquared_adj)
```

In [314...

```
pd.concat(coef, axis=1)
```

Out[314...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.015071 | 0.008765 | 0.014264 | 0.014938 | 0.009253 | -0.001482 | -0.000547 | 0.000123 |
| **market** | 0.396935 | 0.650940 | 0.717838 | 0.645721 | 0.450448 | 0.579658 | 0.581812 | 0.541126 |
| **factor0** | 0.870551 | 1.418320 | 1.300617 | 1.450804 | 1.192347 | 1.245038 | 1.184390 | 1.129570 |
| **factor1** | -0.673763 | 0.021836 | 0.049120 | 0.216895 | 0.255501 | -0.440631 | -0.145501 | 0.066873 |
| **factor2** | 0.255837 | -0.056645 | 0.296010 | 0.134296 | 0.583250 | -0.059018 | 0.216297 | 0.124514 |
| **factor3** | 1.709128 | -0.187832 | -0.092038 | -0.162957 | 0.664805 | 0.225102 | 0.431374 | 0.348947 |

In [315...

```
pd.concat(tvalues, axis=1)
```

Out[315...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 1.714681 | 2.165766 | 3.410969 | 3.598947 | 2.458947 | -0.428728 | -0.243074 | 0.05435 |
| **market** | 1.803121 | 6.421749 | 6.853474 | 6.211342 | 4.779402 | 6.695739 | 10.331328 | 9.54015 |
| **factor0** | 2.726624 | 9.647460 | 8.561698 | 9.622221 | 8.722845 | 9.915969 | 14.500887 | 13.73080 |
| **factor1** | -2.208230 | 0.155426 | 0.338355 | 1.505295 | 1.955933 | -3.672260 | -1.864113 | 0.85062 |
| **factor2** | 0.472834 | -0.227361 | 1.149822 | 0.525585 | 2.517818 | -0.277364 | 1.562659 | 0.89313 |
| **factor3** | 2.608068 | -0.622475 | -0.295181 | -0.526565 | 2.369532 | 0.873465 | 2.573160 | 2.06659 |

In [316...

```
pv = pd.concat(pvalues, axis=1)
pv
```

Out[316...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.089121 | 3.241216e-02 | 8.962776e-04 | 4.741236e-04 | 1.543737e-02 | 6.689302e-01 | 8.083846e-01 | 9.5 |
| **market** | 0.074011 | 3.214611e-09 | 3.894342e-10 | 8.805306e-09 | 5.286733e-06 | 8.474602e-10 | 4.668435e-18 | 3.2 |
| **factor0** | 0.007410 | 1.845829e-16 | 6.014146e-14 | 2.113514e-16 | 2.564883e-14 | 4.363216e-17 | 1.219112e-27 | 6.4 |

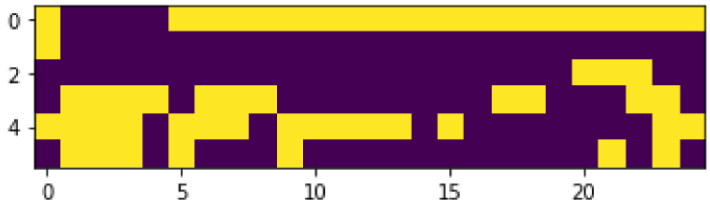
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **factor1** | 0.029229 | 8.767602e-01 | 7.357179e-01 | 1.350141e-01 | 5.291945e-02 | 3.675351e-04 | 6.487849e-02 | 3.9 |
| **factor2** | 0.637236 | 8.205501e-01 | 2.526235e-01 | 6.001976e-01 | 1.319675e-02 | 7.820031e-01 | 1.209045e-01 | 3.7 |
| **factor3** | 0.010324 | 5.348727e-01 | 7.683922e-01 | 5.995185e-01 | 1.949112e-02 | 3.842458e-01 | 1.136089e-02 | 4.1 |

In [326...

```
TF =pv>0.05 #False이면 통계적 유의성있음 - 보라색이면 설명력 좋은 것!!
plt.imshow(TF.values)

## 기간에 따라서 정반대가 아니라 계속 설명력 좋다 RMW랑 CMA팩터는 한국시장에서 잘 작동하지 않
```

Out[326...

```
<matplotlib.image.AxesImage at 0x1ac7bdbe7c8>
```




In [ ]:

```
pd.concat(se, axis=1)
```

In [319...

```
r
```

Out[319...

```
[0.33584718020157833,
 0.6674693632044355,
 0.6319604039172082,
 0.6668026244186944,
 0.6914104994847745,
 0.7113566331967238,
 0.8593428899135657,
 0.8524207294843529,
 0.8436021831581562,
 0.7956283735398665,
 0.6809499621206649,
 0.7950888535358689,
 0.8404541647963482,
 0.7764830113783531,
 0.7671752738401504,
 0.6421015251215294,
 0.6888733977871755,
 0.6939558542469342,
 0.7061748207799925,
 0.7028759165280598,
 0.5861596503513213,
 0.5073167341504279,
 0.7450083307061995,
 0.5937466009964874,
 0.7591183388189037]
```

## 1. (size/BEME) 25포트폴리오에 대해 OLS분석

(2) HML를 다른 4개 팩터에 projection시켜서 HML0로 대체

In [327...]

```
coef_1 = []
tvalues_1 = []
pvalues_1 = []
se_1 = []
r_1 = []
for i in range(25):
    model = get_ols_model(pfo25_hml_pfo, factors['Rm'], pfo_return_hml_pfo.iloc[:,i], cd_91, factors['SM
    coef_1.append(model.params)
    tvalues_1.append(model.tvalues)
    pvalues_1.append(model.pvalues)
    se_1.append(model.bse)
    r_1.append(model.rsquared_adj)
```

In [328...]

```
coef_1_table = pd.concat(coef_1, axis=1)
coef_1_table
```

Out[328...]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.015040 | 0.008767 | 0.014267 | 0.014948 | 0.009265 | -0.001502 | -0.000553 | 0.000126 |
| **market** | 0.377619 | 0.651566 | 0.719246 | 0.651939 | 0.457773 | 0.567025 | 0.577640 | 0.543044 |
| **factor0** | 0.821787 | 1.419900 | 1.304172 | 1.466502 | 1.210839 | 1.213147 | 1.173859 | 1.134410 |
| **factor1** | -0.673877 | 0.021681 | 0.048952 | 0.216920 | 0.255516 | -0.440666 | -0.145467 | 0.066880 |
| **factor2** | 0.360056 | -0.060023 | 0.288412 | 0.100746 | 0.543729 | 0.009139 | 0.238803 | 0.114170 |
| **factor3** | 1.501004 | -0.181087 | -0.076865 | -0.095958 | 0.743729 | 0.088992 | 0.386429 | 0.369604 |

In [329...]

```
tvalues_1_table = pd.concat(tvalues_1, axis=1)
tvalues_1_table
```

Out[329...]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 1.711648 | 2.166764 | 3.412591 | 3.602300 | 2.462732 | -0.434712 | -0.246211 | 0.05572 |
| **market** | 1.716721 | 6.432923 | 6.872245 | 6.276056 | 4.860916 | 6.554970 | 10.265169 | 9.58142 |
| **factor0** | 2.580088 | 9.681385 | 8.605684 | 9.749709 | 8.879409 | 9.685252 | 14.406353 | 13.82275 |
| **factor1** | -2.208593 | 0.154319 | 0.337191 | 1.505458 | 1.956024 | -3.672540 | -1.863639 | 0.85070 |
| **factor2** | 0.667994 | -0.241838 | 1.124583 | 0.395790 | 2.356175 | 0.043117 | 1.731832 | 0.82206 |
| **factor3** | 2.314601 | -0.606437 | -0.249112 | -0.313336 | 2.678741 | 0.348954 | 2.329304 | 2.21197 |

In [344...]

```
pv_1 = pd.concat(pvalues_1, axis=1)
pv_1<0.1 #True이면 통계적 유의성 충분. (설명력 ㅇ)
```

Out[344...]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | True | True | True | True | True | False | False | False | True | True | False | False | False | Fals |
| **market** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **factor0** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor1** | True | False | False | False | True | True | True | False | False | True | True | True | True | Tru |
| **factor2** | False | False | False | False | True | False | True | False | True | False | False | False | False | Fals |
| **factor3** | True | False | False | False | True | False | True | True | True | True | True | True | True | Tru |

In [345...

```
TF_1 =pv_1<0.05 #노란색이 좋은 것
TF_1
```

Out[345...

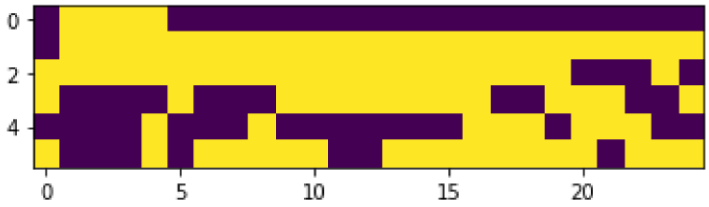
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | False | True | True | True | True | False | False | False | False | False | False | False | False | Fal |
| **market** | False | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor0** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor1** | True | False | False | False | False | True | False | False | False | True | True | True | True | Tru |
| **factor2** | False | False | False | False | True | False | False | False | True | False | False | False | False | Fal |
| **factor3** | True | False | False | False | True | False | True | True | True | True | True | False | False | Tru |

In [342...

```
plt.imshow(TF_1.values)
```

Out[342...

```
<matplotlib.image.AxesImage at 0x1ac7356dd48>
```




In [346...

```
shaped_r1= reshape(pd.Series(r_1), 'B/M', 5,'Size',5)
shaped_r1
```

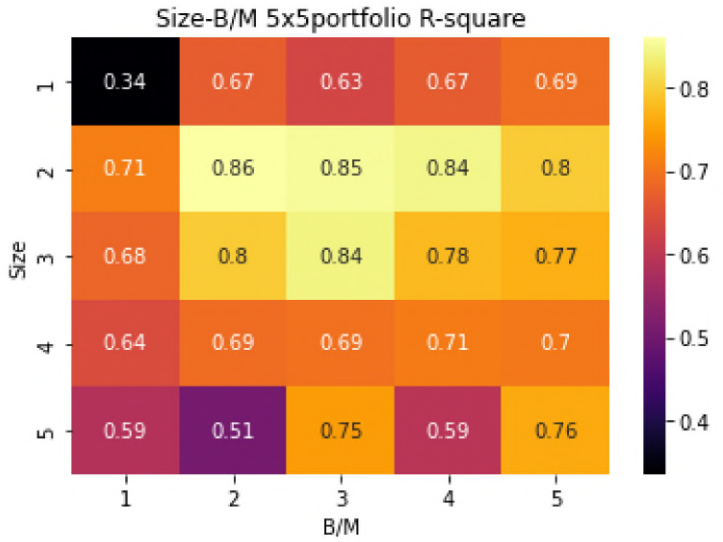
Out[346...

| B/M | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **Size** | | | | | |
| **1** | 0.335856 | 0.667468 | 0.631958 | 0.666804 | 0.691411 |
| **2** | 0.711361 | 0.859341 | 0.852421 | 0.843604 | 0.795632 |
| **3** | 0.680976 | 0.795095 | 0.840456 | 0.776482 | 0.767187 |
| **4** | 0.642093 | 0.688874 | 0.693948 | 0.706173 | 0.702884 |
| **5** | 0.586144 | 0.507310 | 0.745006 | 0.593755 | 0.759149 |

In [348...

```
os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩아티클')
sns.heatmap(shaped_r1, cmap = 'inferno', annot=True)
```

```
plt.title('Size-B/M 5x5portfolio R-square')
plt.savefig('r1.png', dpi=400)
```




In [422...

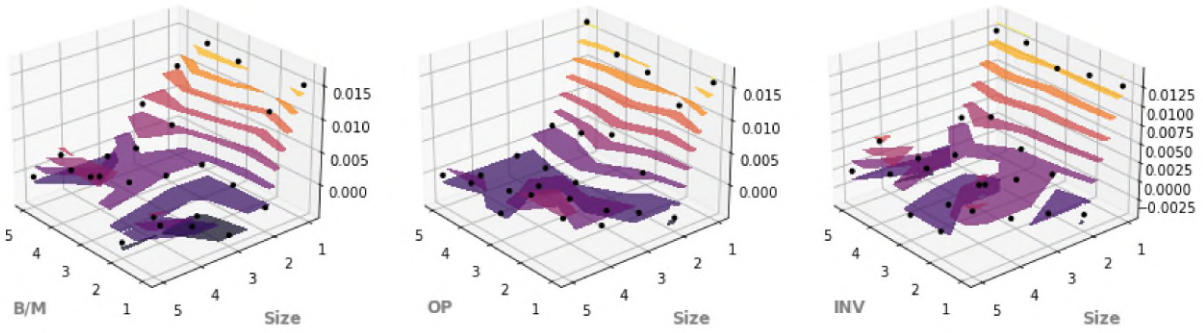
```
shaped_int_coef1 = reshape(coef_1_table.loc['Intercept',:], 'B/M', 5,'Size',5)
shaped_SMB_coef1 = reshape(coef_1_table.loc['factor0',:], 'B/M', 5,'Size',5)
shaped_HMLO_coef1 = reshape(coef_1_table.loc['factor1',:], 'B/M', 5,'Size',5)
shaped_RMW_coef1 = reshape(coef_1_table.loc['factor2',:], 'B/M', 5,'Size',5)
shaped_CMA_coef1 = reshape(coef_1_table.loc['factor3',:], 'B/M', 5,'Size',5)
```

In [419...

```
def graph(data:list, figsize:tuple, name:str):
    os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩아티클')
    fig = plt.figure(figsize=(15,15))
    ax0 = fig.add_subplot(131, projection='3d')
    ax1 = fig.add_subplot(132, projection='3d')
    ax2 = fig.add_subplot(133, projection='3d')
    axs = [ax0, ax1, ax2]

    fontlabel = {"fontsize":"large", "color":"gray", "fontweight":"bold"}
    for ax, datum in zip(axs, data):
        data_pt = datum
        X_ = data_pt.columns.tolist() #B/M
        Y_ = data_pt.index.tolist()     #Size
        X = [X_ for _ in range(len(Y_))]
        Y = [[y_]*len(X_) for y_ in Y_]
        Z = data_pt.values

        ax.set_xlabel(datum.columns.name, fontdict=fontlabel, labelpad=16)
        ax.set_ylabel("Size", fontdict=fontlabel, labelpad=16)
    #     ax.set_title("Z", fontdict=fontlabel)
        ax.view_init(elev=25, azim=140)

        ax.contourf(X, Y, Z, cmap="inferno", alpha=0.7)
        ax.scatter(X, Y, Z, color='black', s=10, alpha=1)
    plt.savefig(name + ' coefficients.png', dpi=400)
```

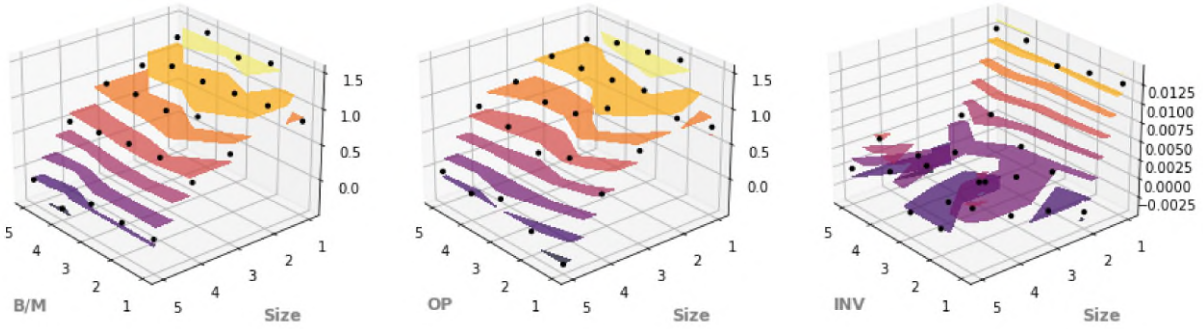
In [435...

```
graph([shaped_int_coef1, shaped_int_coef2, shaped_int_coef3], (15,15), 'Intercept')
```



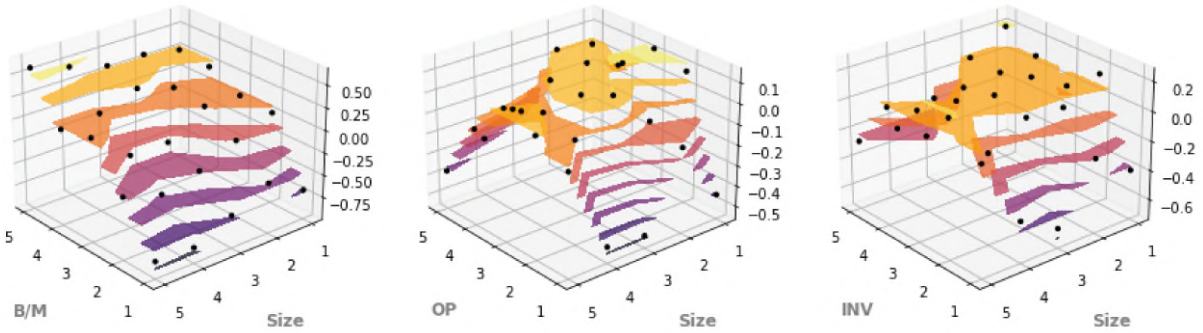

In [436...

```
graph([shaped_SMB_coef1, shaped_SMB_coef2, shaped_int_coef3], (15,15), 'SMB')
```



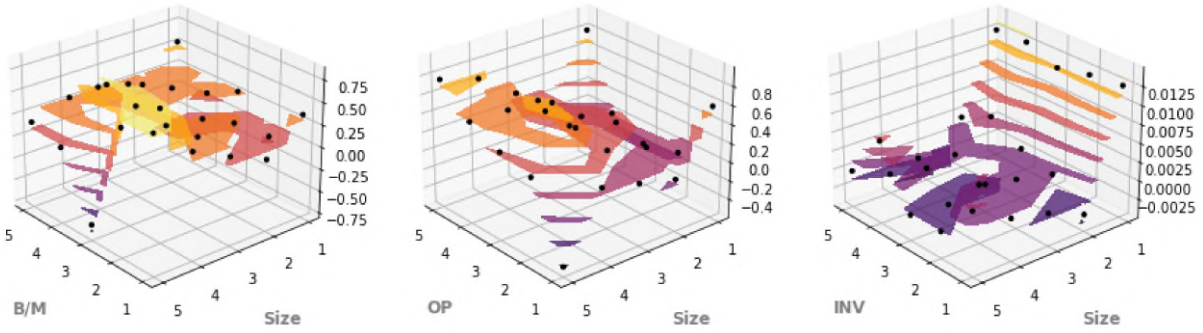

In [437...

```
graph([shaped_HMLO_coef1, shaped_HMLO_coef2, shaped_HMLO_coef3], (15,15), 'HMLO')
```



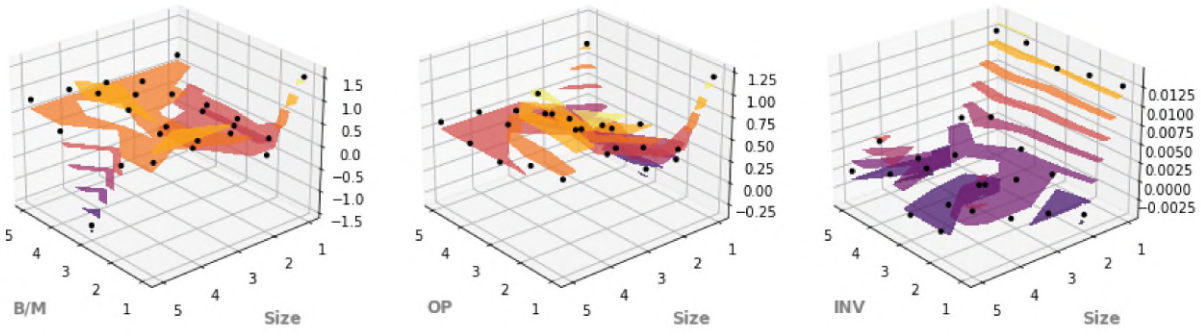

In [438...

```
graph([shaped_RMW_coef1, shaped_RMW_coef2, shaped_int_coef3], (15,15), 'RMW')
```




In [439...

```
graph([shaped_CMA_coef1, shaped_CMA_coef2, shaped_int_coef3], (15,15), 'CMA')
```




## 2. (size/OP) 25포트폴리오에 대해 OLS분석

In [352...

```
# OP pfo
rmw_pfo_mask = make_portfolio(get_mask(Market_cap[0], Market_cap[1], breakpoint = [0.2, 0.4, 0.6, 0

pfo25_rmw_pfo = serialize_pfo([get_mask(Market_cap[0], Market_cap[1], breakpoint = [0.2, 0.4, 0.6, 0.8

pfo_return_rmw_pfo = get_factor_on_date_by_mask(pfo25_rmw_pfo, term = 'm', winsorize_limits= 0.01
```

## 2. (size/OP) 25포트폴리오에 대해 OLS분석

(1) Rm, SMB, HML, RMW(OP), CMA(Inv)로 regression

In [450...

```
coef = []
tvalues = []
pvalues = []
se = []
r = []
for i in range(25):
    model = get_ols_model(pfo25_rmw_pfo, factors['Rm'], pfo_return_rmw_pfo.iloc[:,i], cd_91, factors['S
    coef.append(model.params)
    tvalues.append(model.tvalues)
    pvalues.append(model.pvalues)
    se.append(model.bse)
    r.append(model.rsquared_adj)
```

In [451...

```
pd.concat(coef, axis=1)
```

Out[451...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.015578 | 0.009854 | 0.012737 | 0.013365 | 0.016442 | -0.003061 | 0.001408 | 0.004912 |
| **market** | 0.506428 | 0.472383 | 0.719053 | 0.610477 | 0.497071 | 0.442427 | 0.674397 | 0.615922 |
| **factor0** | 0.766488 | 1.461584 | 1.373842 | 1.320386 | 1.181070 | 1.049338 | 1.220215 | 1.145014 |
| **factor1** | -0.445184 | 0.081097 | 0.123022 | -0.016253 | 0.011551 | -0.148447 | -0.099071 | 0.099279 |
| **factor2** | 0.519279 | -0.059031 | -0.165702 | -0.072831 | 0.803872 | -0.064458 | 0.149180 | 0.343832 |
| **factor3** | 1.328682 | 0.171708 | -0.255212 | -0.142689 | 0.932835 | 0.462309 | 0.402670 | 0.380794 |

In [452...

```
pd.concat(tvalues, axis=1)
```

Out[452...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 1.773579 | 2.135484 | 2.894438 | 3.426550 | 3.217473 | -0.939499 | 0.542286 | 2.07238 |
| **market** | 2.302051 | 4.087223 | 6.523896 | 6.248939 | 3.883631 | 5.421940 | 10.370859 | 10.37418 |
| **factor0** | 2.402307 | 8.719335 | 8.594274 | 9.318890 | 6.362401 | 8.866549 | 12.937844 | 13.29734 |
| **factor1** | -1.460051 | 0.506255 | 0.805308 | -0.120034 | 0.065113 | -1.312548 | -1.099204 | 1.20647 |
| **factor2** | 0.960368 | -0.207806 | -0.611668 | -0.303314 | 2.555326 | -0.321390 | 0.933361 | 2.35621 |
| **factor3** | 2.028883 | 0.499073 | -0.777832 | -0.490643 | 2.448291 | 1.903200 | 2.080118 | 2.15455 |

In [453...

```
pv = pd.concat(pvalues, axis=1)
pv
```

Out[453...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.078803 | 3.486164e-02 | 4.552732e-03 | 8.509586e-04 | 1.683885e-03 | 3.494612e-01 | 5.886797e-01 | 4.0 |
| **market** | 0.023148 | 8.156095e-05 | 1.960801e-09 | 7.362390e-09 | 1.730117e-04 | 3.341052e-07 | 3.773443e-18 | 3.7 |
| **factor0** | 0.017907 | 2.613012e-14 | 5.063621e-14 | 1.073286e-15 | 4.277647e-09 | 1.196757e-14 | 4.130361e-24 | 6.2 |
| **factor1** | 0.147026 | 6.136554e-01 | 4.223183e-01 | 9.046678e-01 | 9.481983e-01 | 1.919716e-01 | 2.739951e-01 | 2.3 |
| **factor2** | 0.338902 | 8.357515e-01 | 5.419755e-01 | 7.622032e-01 | 1.192574e-02 | 7.485033e-01 | 3.526064e-01 | 2.0 |
| **factor3** | 0.044800 | 6.186896e-01 | 4.382791e-01 | 6.246221e-01 | 1.587739e-02 | 5.953643e-02 | 3.975592e-02 | 3.3 |

In [454...

```
pv<0.1
```

Out[454...

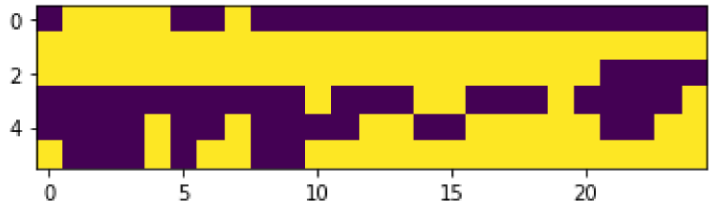
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | True | True | True | True | True | False | False | True | False | False | False | False | False | Fal |
| **market** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor0** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor1** | False | False | False | False | False | False | False | False | False | False | True | False | False | Fal |
| **factor2** | False | False | False | False | True | False | False | True | False | False | False | False | True | Tru |
| **factor3** | True | False | False | False | True | True | True | True | False | False | True | True | True | Tru |

In [455...

```
TF =pv<0.05
plt.imshow(TF.values)
```

<matplotlib.image.AxesImage at 0x1ac7ddc9b88>

Out[455...




In [456...

```
r
```

Out[456...

```
[0.2578147716160414,
 0.6321274171383615,
 0.6448999888925913,
 0.6562265189155313,
 0.5546114866192593,
 0.7140913509404672,
 0.8430330465364406,
 0.8467858540273662,
 0.8356749612578432,
 0.7754439860679773,
 0.6835393653416698,
 0.8323522826030997,
 0.7856548954795439,
 0.8173084972728966,
 0.7644254549883429,
 0.6996751651438846,
 0.7080388490907327,
 0.6717161907363975,
 0.6759988818156604,
 0.6402714492976003,
 0.7412688932914497,
 0.6575691562629538,
 0.5883969391023971,
 0.5194558105688671,
 0.6444397166406628]
```

In [ ]:

## 2. (size/OP) 25포트폴리오에 대해 OLS분석

(2) HML를 다른 4개 팩터에 projection시켜서 HML0로 대체

In [361...

```
coef_2 = []
tvalues_2 = []
pvalues_2 = []
se_2 = []
r_2 = []
for i in range(25):
    model = get_ols_model(pfo25_rmw_pfo, factors['Rm'], pfo_return_rmw_pfo.iloc[:,i], cd_91, factors['S
    coef_2.append(model.params)
    tvalues_2.append(model.tvalues)
    pvalues_2.append(model.pvalues)
    se_2.append(model.bse)
    r_2.append(model.rsquared_adj)
```

In [362...

```
coef_2_table = pd.concat(coef_2, axis=1)
coef_2_table
```

Out[362...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.015559 | 0.009858 | 0.012743 | 0.013364 | 0.016443 | -0.003068 | 0.001403 | 0.004917 |
| **market** | 0.493665 | 0.474708 | 0.722580 | 0.610011 | 0.497403 | 0.438171 | 0.671556 | 0.618768 |
| **factor0** | 0.734268 | 1.467453 | 1.382746 | 1.319210 | 1.181906 | 1.038594 | 1.213045 | 1.152199 |
| **factor1** | -0.445485 | 0.081084 | 0.122961 | -0.016227 | 0.011439 | -0.148479 | -0.099086 | 0.099306 |
| **factor2** | 0.588140 | -0.071576 | -0.184732 | -0.070317 | 0.802085 | -0.041496 | 0.164504 | 0.328476 |
| **factor3** | 1.191166 | 0.196759 | -0.217210 | -0.147709 | 0.936403 | 0.416454 | 0.372067 | 0.411461 |

In [363...

```
tvalues_2_table = pd.concat(tvalues_2, axis=1)
tvalues_2_table
```

Out[363...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 1.771878 | 2.136875 | 2.896555 | 3.427229 | 3.218528 | -0.941817 | 0.540686 | 2.07482 |
| **market** | 2.245809 | 4.110538 | 6.560985 | 6.249033 | 3.889245 | 5.373984 | 10.335241 | 10.43027 |
| **factor0** | 2.306876 | 8.775366 | 8.670716 | 9.332941 | 6.382189 | 8.796864 | 12.892716 | 13.41296 |
| **factor1** | -1.461039 | 0.506168 | 0.804896 | -0.119841 | 0.064481 | -1.312824 | -1.099356 | 1.20679 |
| **factor2** | 1.091889 | -0.252926 | -0.684513 | -0.293962 | 2.559380 | -0.207692 | 1.033171 | 2.25958 |
| **factor3** | 1.838064 | 0.577902 | -0.668978 | -0.513252 | 2.483522 | 1.732478 | 1.942262 | 2.35257 |

In [364...

```
pv_2 = pd.concat(pvalues_2, axis=1)
pv_2
```

Out[364...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.079087 | 3.474570e-02 | 4.524240e-03 | 8.490346e-04 | 1.678220e-03 | 3.482780e-01 | 5.897788e-01 | 4.0 |
| **market** | 0.026644 | 7.471114e-05 | 1.637307e-09 | 7.359097e-09 | 1.695191e-04 | 4.133032e-07 | 4.571122e-18 | 2.7 |
| **factor0** | 0.022868 | 1.941675e-14 | 3.380103e-14 | 9.955863e-16 | 3.889467e-09 | 1.732440e-14 | 5.240206e-24 | 3.4 |
| **factor1** | 0.146755 | 6.137159e-01 | 4.225550e-01 | 9.048199e-01 | 9.487004e-01 | 1.918789e-01 | 2.739292e-01 | 2.3 |
| **factor2** | 0.277184 | 8.007808e-01 | 4.950405e-01 | 7.693213e-01 | 1.179517e-02 | 8.358397e-01 | 3.037107e-01 | 2.5 |
| **factor3** | 0.068657 | 5.644700e-01 | 5.048621e-01 | 6.087683e-01 | 1.446376e-02 | 8.589304e-02 | 5.457214e-02 | 2.0 |

In [369...

```
pv_2<0.1
```

Out[369...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | True | True | True | True | True | False | False | True | False | False | False | False | False | Fal |
| **market** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor0** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor1** | False | False | False | False | False | False | False | False | False | False | True | False | False | Fal |
| **factor2** | False | False | False | False | True | False | False | True | False | False | False | False | True | Tru |
| **factor3** | True | False | False | False | True | True | True | True | False | False | True | True | True | Tru |

In [370...

```
TF_2 =pv_2<0.05
TF_2
```

Out[370...

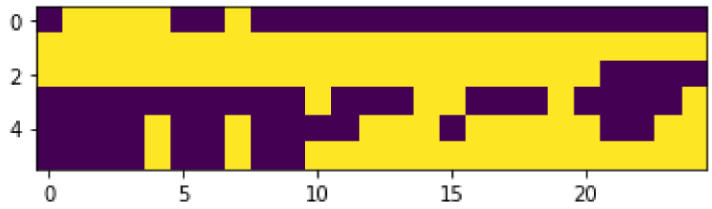
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | False | True | True | True | True | False | False | True | False | False | False | False | False | Fal |
| **market** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor0** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor1** | False | False | False | False | False | False | False | False | False | False | True | False | False | Fal |
| **factor2** | False | False | False | False | True | False | False | True | False | False | False | False | True | Tru |
| **factor3** | False | False | False | False | True | False | False | True | False | False | True | True | True | Tru |

In [371...

```
plt.imshow(TF_2.values)
```

Out[371...

```
<matplotlib.image.AxesImage at 0x1ac7997cd88>
```




In [372...

```
pd.concat(se_2, axis=1)
```

Out[372...

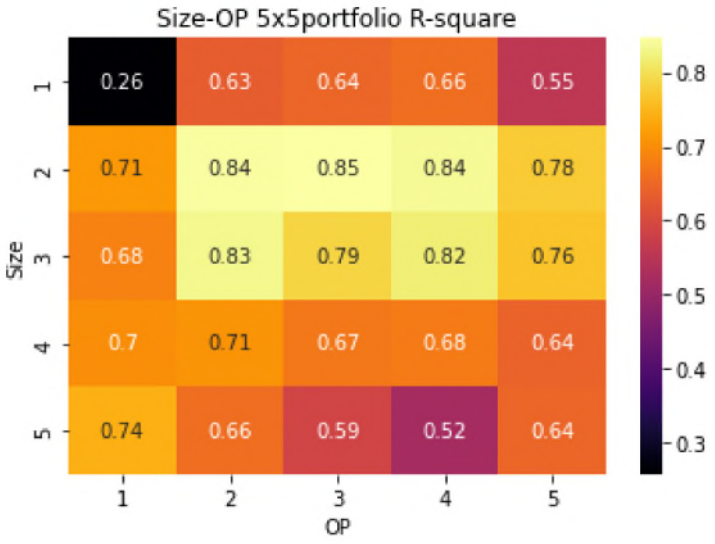
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.008781 | 0.004613 | 0.004399 | 0.003899 | 0.005109 | 0.003257 | 0.002596 | 0.002370 | 0.00 |
| **market** | 0.219816 | 0.115486 | 0.110133 | 0.097617 | 0.127892 | 0.081536 | 0.064977 | 0.059324 | 0.05 |
| **factor0** | 0.318295 | 0.167224 | 0.159473 | 0.141350 | 0.185188 | 0.118064 | 0.094088 | 0.085902 | 0.08 |
| **factor1** | 0.304910 | 0.160192 | 0.152767 | 0.135405 | 0.177400 | 0.113099 | 0.090131 | 0.082289 | 0.07 |
| **factor2** | 0.538645 | 0.282990 | 0.269873 | 0.239204 | 0.313390 | 0.199798 | 0.159223 | 0.145370 | 0.13 |
| **factor3** | 0.648055 | 0.340471 | 0.324690 | 0.287791 | 0.377046 | 0.240381 | 0.191564 | 0.174898 | 0.16 |

In [373...
```
r_2
```

Out[373...
```
[0.25783321219855904,
 0.6321271353830658,
 0.644897934399647,
 0.65622637974727,
 0.5546111666400055,
 0.7140931406705627,
 0.8430335011902266,
 0.8467868766753167,
 0.8356754565797844,
 0.7754442961140828,
 0.6835324806003273,
 0.8323503126411215,
 0.7856550835047349,
 0.8173085131351486,
 0.7644307319455415,
 0.6996532400877093,
 0.7080376820875735,
 0.6717162016109723,
 0.6759990383086816,
 0.6402774394868207,
 0.7412682032641875,
 0.6575712769477934,
 0.5884025811003627,
 0.5194439829363591,
 0.6444480441724416]
```

In [433...
```
shaped_r2= reshape(pd.Series(r_2), 'OP', 5,'Size',5)
shaped_int_coef2 = reshape(coef_2_table.loc['Intercept',:], 'OP', 5,'Size',5)
shaped_SMB_coef2 = reshape(coef_2_table.loc['factor0',:], 'OP', 5,'Size',5)
shaped_HMLO_coef2 = reshape(coef_2_table.loc['factor1',:], 'OP', 5,'Size',5)
shaped_RMW_coef2 = reshape(coef_2_table.loc['factor2',:], 'OP', 5,'Size',5)
shaped_CMA_coef2 = reshape(coef_2_table.loc['factor3',:], 'OP', 5,'Size',5)
```

In [392...
```
os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩아티클')
sns.heatmap(shaped_r2, cmap='inferno', annot=True)
plt.title('Size-OP 5x5portfolio R-square')
plt.savefig('r2.png', dpi=400)
```




## 3. (size/INV)로 25포트폴리오에 대해 OLS분석

In [353...]
```
# INV pfo
cma_pfo_mask = make_portfolio(get_mask(Market_cap[0], Market_cap[1], breakpoint = [0.2, 0.4, 0.6, 0

pfo25_cma_pfo = serialize_pfo([get_mask(Market_cap[0], Market_cap[1], breakpoint = [0.2, 0.4, 0.6, 0.8

pfo_return_cma_pfo = get_factor_on_date_by_mask(pfo25_cma_pfo, term = 'm', winsorize_limits= 0.01
```

## 3. (size/INV)로 25포트폴리오에 대해 OLS분석

(1) Rm, SMB, HML, RMW, CMA로 regression

In [457...]
```
coef = []
tvalues = []
pvalues = []
se = []
r = []
for i in range(25):
    model = get_ols_model(pfo25_cma_pfo, factors['Rm'], pfo_return_cma_pfo.iloc[:,i], cd_91, factors['SI
    coef.append(model.params)
    tvalues.append(model.tvalues)
    pvalues.append(model.pvalues)
    se.append(model.bse)
    r.append(model.rsquared_adj)
```

In [458...]
```
pd.concat(coef, axis=1)
```

Out[458...]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.012569 | 0.012037 | 0.011067 | 0.012678 | 0.012540 | -0.002715 | 0.000478 | 0.001780 |
| **market** | 0.435617 | 0.619372 | 0.648543 | 0.505274 | 0.716459 | 0.458038 | 0.599387 | 0.628608 |
| **factor0** | 0.969343 | 1.401882 | 1.200640 | 1.346006 | 1.330071 | 1.113523 | 1.236621 | 1.174919 |
| **factor1** | -0.400702 | 0.149607 | -0.031381 | 0.079710 | 0.192513 | -0.229330 | 0.008054 | 0.112851 |
| **factor2** | 0.373218 | 0.002744 | 0.056185 | 0.250618 | 0.040596 | 0.268903 | 0.232351 | 0.083129 |
| **factor3** | 1.136638 | 0.063290 | 0.151900 | 0.307862 | -0.034890 | 0.613373 | 0.528099 | 0.146628 |

In [459...]
```
pd.concat(tvalues, axis=1)
```

Out[459...]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 1.288015 | 2.759236 | 2.647392 | 3.667282 | 2.641216 | -0.863926 | 0.181343 | 0.72671 |
| **market** | 1.782312 | 5.668442 | 6.194351 | 5.835517 | 6.025038 | 5.819040 | 9.081733 | 10.24535 |
| **factor0** | 2.734530 | 8.846051 | 7.906708 | 10.718294 | 7.712042 | 9.753836 | 12.918856 | 13.20327 |
| **factor1** | -1.182856 | 0.987862 | -0.216251 | 0.664195 | 1.168048 | -2.102047 | 0.088050 | 1.32704 |
| **factor2** | 0.621273 | 0.010217 | 0.218330 | 1.177621 | 0.138896 | 1.389906 | 1.432339 | 0.55124 |
| **factor3** | 1.562213 | 0.194574 | 0.487366 | 1.194394 | -0.098562 | 2.617665 | 2.687915 | 0.80279 |

In [460...]
```
pv = pd.concat(pvalues, axis=1)
pv
```

Out[460...]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.200349 | 6.751720e-03 | 9.259578e-03 | 3.739863e-04 | 9.419958e-03 | 3.894424e-01 | 8.564201e-01 | 4.6 |
| **market** | 0.077360 | 1.102191e-07 | 9.545539e-09 | 5.124693e-08 | 2.121922e-08 | 5.529427e-08 | 3.807945e-15 | 7.4 |
| **factor0** | 0.007245 | 1.334394e-14 | 1.854514e-12 | 5.811739e-19 | 5.070688e-12 | 1.042740e-16 | 4.565299e-24 | 1.0 |
| **factor1** | 0.239327 | 3.253125e-01 | 8.291786e-01 | 5.079068e-01 | 2.452243e-01 | 3.775092e-02 | 9.299911e-01 | 1.8 |
| **factor2** | 0.535660 | 9.918657e-01 | 8.275622e-01 | 2.414003e-01 | 8.897770e-01 | 1.672655e-01 | 1.547833e-01 | 5.8 |
| **factor3** | 0.121010 | 8.460728e-01 | 6.269351e-01 | 2.348032e-01 | 9.216593e-01 | 1.005483e-02 | 8.267104e-03 | 4.2 |

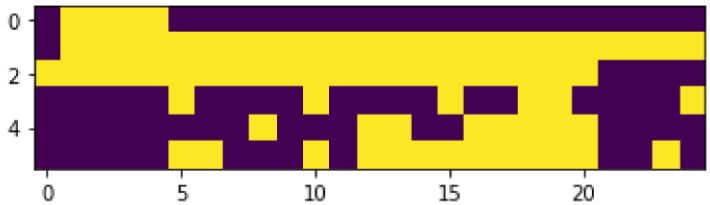
In [461...]
```
pv<0.1
```

Out[461...]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | False | True | True | True | True | False | False | False | True | False | False | False | False | Fal |
| **market** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor0** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor1** | False | False | False | False | False | True | False | False | False | False | True | False | False | Fal |
| **factor2** | False | False | False | False | False | False | False | False | True | False | False | False | True | Tru |
| **factor3** | False | False | False | False | False | True | True | False | False | False | True | True | True | Tru |

In [462...]
```
TF =pv<0.05
plt.imshow(TF.values)
```

Out[462...]
```
<matplotlib.image.AxesImage at 0x1ac7d8b46c8>
```




In [463...]
```
r
```

Out[463...]
```
[0.22159223431771835,
 0.66224717126601,
 0.6369668254185257,
 0.7317701003880448,
```

0.617420237599102,
0.7306014992314364,
0.841599099258226,
0.8375218184309722,
0.8370507902944053,
0.8123974817731476,
0.698218865358058,
0.7928121483195706,
0.8146672562558507,
0.8067212873905927,
0.7156780325650386,
0.7203775114919091,
0.6710398992687355,
0.7216448916529264,
0.5915049938777415,
0.6581116370244418,
0.7871235048688733,
0.6750701313662153,
0.6517254572510081,
0.6790779626328753,
0.5509686812686918]

## 3. (size/INV)로 25포트폴리오에 대해 OLS분석

(2) HML를 다른 4개 팩터에 projection시켜서 HML0로 대체

In [376...

```
coef_3 = []
tvalues_3 = []
pvalues_3 = []
se_3 = []
r_3 = []
for i in range(25):
    model = get_ols_model(pfo25_cma_pfo, factors['Rm'], pfo_return_cma_pfo.iloc[:,i], cd_91, factors['SI
    coef_3.append(model.params)
    tvalues_3.append(model.tvalues)
    pvalues_3.append(model.pvalues)
    se_3.append(model.bse)
    r_3.append(model.rsquared_adj)
```

In [377...

```
coef_3_table = pd.concat(coef_3, axis=1)
coef_3_table
```

Out[377...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.012552 | 0.012044 | 0.011065 | 0.012681 | 0.012549 | -0.002725 | 0.000478 | 0.001785 |
| **market** | 0.424129 | 0.623661 | 0.647643 | 0.507559 | 0.721978 | 0.451463 | 0.599618 | 0.631843 |
| **factor0** | 0.940342 | 1.412710 | 1.198368 | 1.351775 | 1.344004 | 1.096925 | 1.237204 | 1.183087 |
| **factor1** | -0.401042 | 0.149567 | -0.031395 | 0.079787 | 0.192431 | -0.229376 | 0.008097 | 0.112861 |
| **factor2** | 0.435199 | -0.020397 | 0.061039 | 0.238289 | 0.010818 | 0.304376 | 0.231105 | 0.065673 |
| **factor3** | 1.012862 | 0.109503 | 0.142207 | 0.332484 | 0.024577 | 0.542534 | 0.530587 | 0.181488 |

In [378...

```
tvalues_3_table = pd.concat(tvalues_3, axis=1)
tvalues_3_table
```

Out[378...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 1.286627 | 2.761592 | 2.647757 | 3.669228 | 2.643862 | -0.867477 | 0.181458 | 0.72903 |
| **market** | 1.736678 | 5.712130 | 6.190578 | 5.866498 | 6.076151 | 5.740022 | 9.092313 | 10.30612 |
| **factor0** | 2.659114 | 8.935758 | 7.910700 | 10.790118 | 7.811500 | 9.631588 | 12.955983 | 13.32699 |
| **factor1** | -1.183861 | 0.987585 | -0.216342 | 0.664831 | 1.167528 | -2.102464 | 0.088511 | 1.32714 |
| **factor2** | 0.727222 | -0.076239 | 0.238099 | 1.123965 | 0.037154 | 1.579278 | 1.430098 | 0.43715 |
| **factor3** | 1.406760 | 0.340192 | 0.461067 | 1.303500 | 0.070159 | 2.339733 | 2.729003 | 1.00411 |

In [379...

```
pv_3 = pd.concat(pvalues_3, axis=1)
pv_3
```

Out[379...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.200831 | 6.706293e-03 | 9.250161e-03 | 3.714514e-04 | 9.350943e-03 | 3.875029e-01 | 8.563305e-01 | 4.6 |
| **market** | 0.085145 | 9.031412e-08 | 9.718042e-09 | 4.440899e-08 | 1.669033e-08 | 7.949776e-08 | 3.599096e-15 | 5.3 |
| **factor0** | 0.008962 | 8.284203e-15 | 1.816525e-12 | 3.948169e-19 | 3.035648e-12 | 2.009922e-16 | 3.753731e-24 | 5.3 |
| **factor1** | 0.238931 | 3.254473e-01 | 8.291082e-01 | 5.075013e-01 | 2.454333e-01 | 3.771365e-02 | 9.296260e-01 | 1.8 |
| **factor2** | 0.468581 | 9.393622e-01 | 8.122321e-01 | 2.633890e-01 | 9.704276e-01 | 1.170432e-01 | 1.554239e-01 | 6.6 |
| **factor3** | 0.162219 | 7.343380e-01 | 6.456293e-01 | 1.950304e-01 | 9.441903e-01 | 2.103776e-02 | 7.360211e-03 | 3.1 |

In [380...

```
pv_3<0.1 #True이면 통계적 유의성 충분. (설명력 ㅇ)
```

Out[380...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | False | True | True | True | True | False | False | False | True | False | False | False | False | Fal |
| **market** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor0** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor1** | False | False | False | False | False | True | False | False | False | False | True | False | False | Fal |
| **factor2** | False | False | False | False | False | False | False | False | True | False | False | False | True | Tru |
| **factor3** | False | False | False | False | False | True | True | False | True | True | True | True | True | Tru |

In [389...

```
TF_3 =pv_3<0.05 #True이면 통계적 유의성 충분. (설명력 ㅇ)
TF_3
```

Out[389...

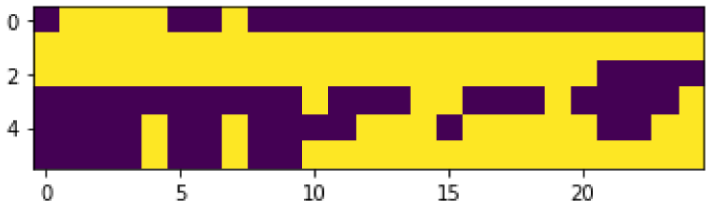
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | False | True | True | True | True | False | False | False | False | False | False | False | False | Fal |
| **market** | False | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor0** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor1** | False | False | False | False | False | True | False | False | False | False | True | False | False | Fal |
| **factor2** | False | False | False | False | False | False | False | False | True | False | False | False | True | Tru |
| **factor3** | False | False | False | False | False | True | True | False | False | False | True | False | True | Tru |

In [390...

```
plt.imshow(TF_2.values)
```

Out[390...

<matplotlib.image.AxesImage at 0x1ac78f23c48>




In [382...

```
pd.concat(se_3, axis=1)
```

Out[382...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.009756 | 0.004361 | 0.004179 | 0.003456 | 0.004746 | 0.003142 | 0.002634 | 0.002449 | 0.00 |
| **market** | 0.244218 | 0.109182 | 0.104618 | 0.086518 | 0.118822 | 0.078652 | 0.065948 | 0.061308 | 0.05 |
| **factor0** | 0.353630 | 0.158096 | 0.151487 | 0.125279 | 0.172055 | 0.113888 | 0.095493 | 0.088774 | 0.08 |
| **factor1** | 0.338758 | 0.151448 | 0.145116 | 0.120010 | 0.164819 | 0.109099 | 0.091477 | 0.085040 | 0.07 |
| **factor2** | 0.598441 | 0.267543 | 0.256358 | 0.212007 | 0.291164 | 0.192731 | 0.161601 | 0.150230 | 0.13 |
| **factor3** | 0.719996 | 0.321887 | 0.308430 | 0.255070 | 0.350306 | 0.231878 | 0.194425 | 0.180745 | 0.16 |

In [383...

```
r_3
```

Out[383...

```
[0.22160827654585669,
 0.6622455673667555,
 0.6369669501959538,
 0.731772081877837,
 0.6174162082039267,
 0.7306054883111477,
 0.8415992121759848,
 0.8375221904049485,
 0.8370515792304176,
 0.8123969675850008,
 0.698223871038518,
 0.7928117551381342,
 0.8146653816628224,
 0.8067204512188487,
 0.7156761704292233,
```

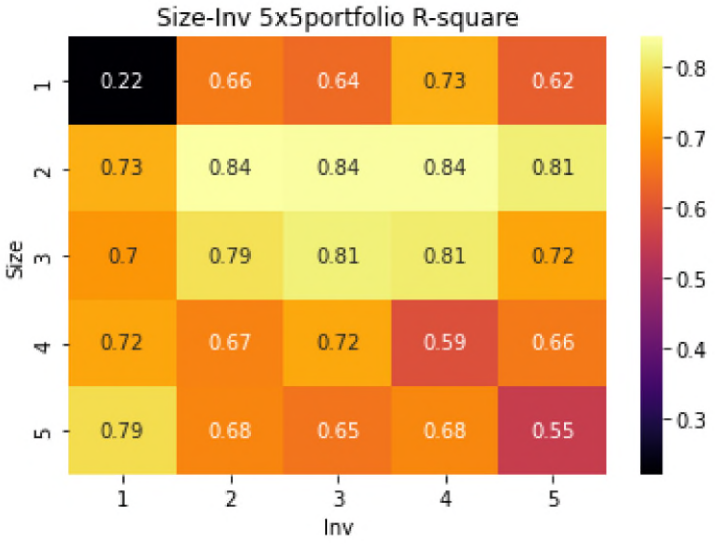
0.7203552315800883,
0.671035986279489,
0.7216424788583937,
0.591503698283174,
0.6581202241698481,
0.7871239942064954,
0.6750791715224804,
0.6517394794944419,
0.679075780967648,
0.5509497082177021]

In [434...

```
shaped_r3= reshape(pd.Series(r_3), 'Inv', 5,'Size',5)
shaped_int_coef3 = reshape(coef_3_table.loc['Intercept',:], 'INV', 5,'Size',5)
shaped_SMB_coef3 = reshape(coef_3_table.loc['factor0',:], 'INV', 5,'Size',5)
shaped_HMLO_coef3 = reshape(coef_3_table.loc['factor1',:], 'INV', 5,'Size',5)
shaped_RMW_coef3 = reshape(coef_3_table.loc['factor2',:], 'INV', 5,'Size',5)
shaped_CMA_coef3 = reshape(coef_3_table.loc['factor3',:], 'INV', 5,'Size',5)
```

In [426...

```
os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩아티클')
sns.heatmap(shaped_r3, cmap='inferno', annot=True)
plt.title('Size-Inv 5x5portfolio R-square')
plt.savefig('r3.png', dpi=400)
```




## FF3F 모델로 돌려보자

In [440...

```
coef_FF3 = []
tvalues_FF3 = []
pvalues_FF3 = []
se_FF3 = []
r_FF3 = []
for i in range(25):
    model = get_ols_model(pfo25_hml_pfo, factors['Rm'], pfo_return_hml_pfo.iloc[:,i], cd_91, factors['SM
    coef_FF3.append(model.params)
    tvalues_FF3.append(model.tvalues)
    pvalues_FF3.append(model.pvalues)
    se_FF3.append(model.bse)
    r_FF3.append(model.rsquared_adj)
```

In [441...

```
coef_FF3_table = pd.concat(coef_FF3, axis=1)
coef_FF3_table
```

Out[441...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.012515 | 0.009029 | 0.015187 | 0.015428 | 0.008955 | -0.001683 | -0.000827 | -0.000430 |
| **market** | 0.646995 | 0.619897 | 0.690093 | 0.628523 | 0.572959 | 0.583607 | 0.639677 | 0.608097 |
| **factor0** | 1.331229 | 1.364150 | 1.172432 | 1.391320 | 1.337475 | 1.247554 | 1.254707 | 1.251070 |
| **factor1** | -0.673771 | 0.021669 | 0.048940 | 0.216911 | 0.255561 | -0.440660 | -0.145442 | 0.066906 |

In [442...

```
tvalues_FF3_table = pd.concat(tvalues_FF3, axis=1)
tvalues_FF3_table
```

Out[442...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 1.392134 | 2.259049 | 3.612984 | 3.753041 | 2.341045 | -0.493692 | -0.364667 | -0.1868 |
| **market** | 3.246192 | 6.996006 | 7.404970 | 6.896341 | 6.756183 | 7.720497 | 12.722722 | 11.9259 |
| **factor0** | 4.819974 | 11.109928 | 9.078668 | 11.016483 | 11.381043 | 11.909747 | 18.008609 | 17.7060 |
| **factor1** | -2.145830 | 0.155228 | 0.333342 | 1.510737 | 1.912856 | -3.700305 | -1.836202 | 0.8329( |

In [443...

```
pv_FF3 = pd.concat(pvalues_FF3, axis=1)
pv_FF3
```

Out[443...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.166545 | 2.574850e-02 | 4.487838e-04 | 2.744756e-04 | 2.093707e-02 | 6.224569e-01 | 7.160240e-01 | 8.5 |
| **market** | 0.001529 | 1.808391e-10 | 2.277113e-11 | 2.977130e-10 | 5.973733e-10 | 4.485144e-12 | 9.538382e-24 | 6.8 |
| **factor0** | 0.000004 | 5.693960e-20 | 3.410653e-15 | 9.453813e-20 | 1.309473e-20 | 7.509879e-22 | 2.159903e-35 | 9.1 |
| **factor1** | 0.033971 | 8.769111e-01 | 7.394781e-01 | 1.335748e-01 | 5.823189e-02 | 3.307911e-04 | 6.888862e-02 | 4.0 |

In [444...

```
pv_FF3<0.1 #True이면 통계적 유의성 부족. (설명력 x)
```

Out[444...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | False | True | True | True | True | False | False | False | False | True | False | False | False | Fals |
| **market** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor0** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor1** | True | False | False | False | True | True | True | False | False | True | True | True | True | Tru |

In [445...

```
TF_FF3 =pv_FF3<0.05 #True이면 통계적 유의성 부족. (설명력 x)
```

TF_FF3

Out[445...

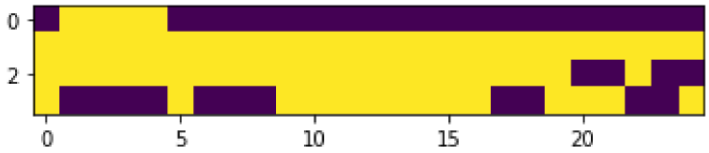
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | False | True | True | True | True | False | False | False | False | False | False | False | False | Fal |
| **market** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor0** | True | True | True | True | True | True | True | True | True | True | True | True | True | Tru |
| **factor1** | True | False | False | False | False | True | False | False | False | True | True | True | True | Tru |

In [446...

```
plt.imshow(TF_FF3.values)
```

Out[446...

<matplotlib.image.AxesImage at 0x1ac724eff48>




In [447...

```
pd.concat(se_FF3, axis=1)
```

Out[447...

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **Intercept** | 0.008990 | 0.003997 | 0.004203 | 0.004111 | 0.003825 | 0.003410 | 0.002268 | 0.002300 | 0.00 |
| **market** | 0.199309 | 0.088607 | 0.093193 | 0.091139 | 0.084805 | 0.075592 | 0.050278 | 0.050989 | 0.05 |
| **factor0** | 0.276190 | 0.122787 | 0.129141 | 0.126294 | 0.117518 | 0.104751 | 0.069673 | 0.070658 | 0.07 |
| **factor1** | 0.313991 | 0.139592 | 0.146816 | 0.143580 | 0.133602 | 0.119087 | 0.079208 | 0.080328 | 0.08 |

In [448...

```
r_FF3
```

Out[448...

```
[0.2966597155923969,
 0.6717310071649873,
 0.6235860706158249,
 0.6691569249995001,
 0.6772110471592206,
 0.7156851112122058,
 0.8551546151334887,
 0.8459258425362742,
 0.8383621194304087,
 0.7897863131287118,
 0.6422754504572432,
 0.7932444779487534,
 0.8385107930296714,
 0.7528166471595785,
 0.7477063861615936,
 0.6171026943568505,
 0.6536227959508071,
 0.6508646470750146,
 0.6537426467868499,
 0.6685598443503836,
 0.5521794719621336,
 0.4745582160294035,
```

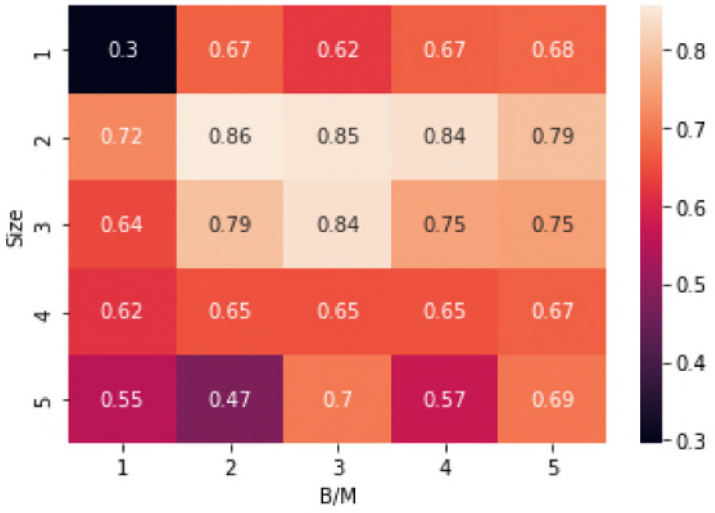
0.6999166327651067,
0.5704947067668626,
0.6942258802414307]

In [449...

```
shaped_r_FF3 = reshape(pd.Series(r_FF3), 'B/M', 5,'Size',5)
sns.heatmap(shaped_r_FF3, annot=True)
```

Out[449...

<AxesSubplot:xlabel='B/M', ylabel='Size'>




In [ ]:

## 4. 블룸버그 데이터와의 비교 (우리 데이터가 괜찮음을 증명) - 이건 API Guideline에 넣자

In [231...

```
def get_item_for_period(item, date_from, date_to, term = 'Y', bloomberg=False, preprocess=[]):
    # term은 Y, H, Q의 세 가지 옵션으로, 연도별, 반기별, 분기별 데이터를 얻어올 수 있도록 하는 옵션
    if bloomberg==False:
        data = []
        item_data = get_item(item)
        if term == 'Y':
            num = int(date_to[:4]) - int(date_from[:4]) +1
            date_working = date_from
            for i in range(num):
                data.append(item_data[date_working])
                date_working = date_working[:2] + str(int(date_working[2:4]) + 1) + date_working[4:]

        elif term == 'H':
            num = int((int(date_to[:4]) - int(date_from[:4])) * 2 + (int(date_to[4:6]) - int(date_from[4:6]))/6
            date_working = date_from
            for i in range(num):
                try:
                    data.append(item_data[date_working])
                except:
                    try:
                        date_working = date_working[:6] + str(61 - int(date_working[6:]))
                        data.append(item_data[date_working])
                    except KeyError:
                        print('해당 날짜의 데이터가 없습니다:', date_working)
                if int(date_working[4:6])+6 > 12:
                    date_working = date_working[:2] + str(int(date_working[2:4]) + 1) +'0'+ str(int(date_w
                else:
                    date_working = date_working[:4] + str(int(date_working[4:6]) + 6).zfill(2) + '31'

        else:
```

```
            num = int((int(date_to[:4]) - int(date_from[:4])) * 4 + (int(date_to[4:6]) - int(date_from[4:6]))/3
            date_working = date_from
            for i in range(num):
                try:
                    data.append(item_data[date_working])
                except:
                    try:
                        date_working = date_working[:6] + str(61 - int(date_working[6:]))
                        data.append(item_data[date_working])
                    except KeyError:
                        print('해당 날짜의 데이터가 없습니다:', date_working)
                if int(date_working[4:6])+3 > 12:
                    date_working = date_working[:2] + str(int(date_working[2:4]) + 1) +'0331'
                else:
                    date_working = date_working[:4] + str(int(date_working[4:6]) + 3).zfill(2) + '31'

        data = pd.concat(data, axis = 1)
        return data
    else:
        data = []
        item_data = bloomberg_get_item(preprocess, item)
        if term == 'Y':
            num = int(date_to[:4]) - int(date_from[:4]) +1
            date_working = date_from
            for i in range(num):
                data.append(item_data[date_working])
                date_working = date_working[:2] + str(int(date_working[2:4]) + 1) + date_working[4:]

        elif term == 'H':
            num = int((int(date_to[:4]) - int(date_from[:4])) * 2 + (int(date_to[4:6]) - int(date_from[4:6]))/6
            date_working = date_from
            for i in range(num):
                try:
                    data.append(item_data[date_working])
                except:
                    try:
                        date_working = date_working[:6] + str(61 - int(date_working[6:]))
                        data.append(item_data[date_working])
                    except KeyError:
                        print('해당 날짜의 데이터가 없습니다:', date_working)
                if int(date_working[4:6])+6 > 12:
                    date_working = date_working[:2] + str(int(date_working[2:4]) + 1) +'0'+ str(int(date_w
                else:
                    date_working = date_working[:4] + str(int(date_working[4:6]) + 6).zfill(2) + '31'

        else:
            num = int((int(date_to[:4]) - int(date_from[:4])) * 4 + (int(date_to[4:6]) - int(date_from[4:6]))/3
            date_working = date_from
            for i in range(num):
                try:
                    data.append(item_data[date_working])
                except:
                    try:
                        date_working = date_working[:6] + str(61 - int(date_working[6:]))
                        data.append(item_data[date_working])
                    except KeyError:
                        print('해당 날짜의 데이터가 없습니다:', date_working)
                if int(date_working[4:6])+3 > 12:
                    date_working = date_working[:2] + str(int(date_working[2:4]) + 1) +'0331'
                else:
                    date_working = date_working[:4] + str(int(date_working[4:6]) + 3).zfill(2) + '31'

        data = pd.concat(data, axis = 1)
        return data
```

```
def preprocessing_bloomberg(data, number_of_series, name_of_series):
    temp = data.index.map(lambda x: x[:4]+x[5:7]+x[8:10] if type(x)!=type(np.NaN) else x)
    data.index = list(map(lambda x: shift_q(x) if type(x)!=type(np.NaN) else x, temp))
    preprocess = dict()
    for i in tqdm.tqdm(range(number_of_series)):
        finance = data.iloc[:,i::number_of_series]
        finance.columns = list(map(lambda x: x[:6], finance.columns))
        real = finance.T.groupby(level=0, axis=1).last()
        preprocess[str(name_of_series[i])]=real
    return preprocess

def bloomberg_get_item(preprocess, concept_id): #preprocess는 list이고 리스트의 각 element는 item
    if len(preprocess)>1:
        temp = []
        for x in preprocess:
            temp.append(x[concept_id])
        datapool = pd.concat(temp)
    else:
        datapool = preprocess[0][concept_id]
    return datapool

def bloomberg_get_item_on_date(preprocess, item, date):
    return bloomberg_get_item(preprocess, item)[date]
```

In [ ]:

```
#블룸버그 데이터 import
#블룸버그 데이터는 비지배를 배제함.
os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩팩터프로젝트₩₩data from bloomberg'
final = pd.read_csv('ff5 final yearly.csv', encoding='cp949', header=1, index_col=0).iloc[1:,:]
final_preprocess = preprocessing_bloomberg(final, 6, ['equity', 'interest expense', 'total asset', 'operati
```

In [ ]:

```
os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩팩터프로젝트₩₩data from bloomberg'
finalQ = pd.read_csv('ff5 final.csv', encoding='cp949', header=1, index_col=0).iloc[1:,:]
finalQ_preprocess = preprocessing_bloomberg(finalQ, 6, ['equity', 'interest expense', 'total asset', 'ope
```

In [234…

```
mktcap_for_SMB = get_item_for_period('mktcap', '20110630', '20200630', 'Y', True, [finalQ_preprocess]
mktcap_for_HML = shift_date_quarter(get_item_for_period('mktcap', '20101231', '20191231', 'Y', True,
equity = shift_date_quarter(get_item_for_period('equity', '20101231', '20191231', 'Y', True, [final_prepr
mktcap_for_HML = mktcap_for_HML.applymap(lambda x: float(x) if type(x)!=type(None) else x)
mktcap_for_SMB = mktcap_for_SMB.applymap(lambda x: float(x) if type(x)!=type(None) else x)
equity = equity.applymap(lambda x: float(x) if type(x)!=type(None) else x)
```

In [ ]:

```
bloom_HML_base = equity / mktcap_for_HML
bloom_SMB_base = mktcap_for_SMB

bloom_SMB_mask = get_mask(bloom_SMB_base, get_market_category(bloom_SMB_base), market_bp
bloom_SMB = get_factor_on_date_by_mask(bloom_SMB_mask, term='m', winsorize_limits=0.01, weigh
bloom_HML_mask = get_mask(bloom_HML_base, get_market_category(bloom_HML_base), market_bp
bloom_HML = get_factor_on_date_by_mask(bloom_HML_mask, term='m', winsorize_limits=0.01, weigh
```

In [ ]:

```
TA = get_item_for_period('total asset', '20101231', '20191231', 'Y', True, [final_preprocess])
icost = get_item_for_period('interest expense', '20101231', '20191231','Y', True, [final_preprocess])
operation = get_item_for_period('operation incomeloss', '20101231', '20191231','Y', True, [final_prepr

TA = TA.applymap(lambda x: float(x) if type(x)!=type(None) else x)
icost = icost.applymap(lambda x: float(x) if type(x)!=type(None) else x)
operation = operation.applymap(lambda x: float(x) if type(x)!=type(None) else x)
```

```
bloom_RMW_base = (operation - icost) / TA
bloom_RMW_base_shifted = shift_date_quarter(bloom_RMW_base, 2)

bloom_RMW_mask = get_mask(bloom_RMW_base_shifted, get_market_category(bloom_RMW_base_sh
bloom_RMW = get_factor_on_date_by_mask(bloom_RMW_mask, term = 'm', winsorize_limits = 0.01, w
```

In [ ]:
```
TA = get_item_for_period('total asset', '20091231', '20181231', 'Y', True, [final_preprocess])
TA = TA.applymap(lambda x: float(x) if type(x)!=type(None) else x)
PL = get_item_for_period('net income', '20101231', '20191231', 'Y', True, [final_preprocess])
PL = PL.applymap(lambda x: float(x) if type(x)!=type(None) else x)
TA_shifted = shift_date_quarter(TA, 4)

bloom_CMA_base = PL / TA_shifted
bloom_CMA_base_shifted = shift_date_quarter(bloom_CMA_base, 2)

bloom_CMA_mask = get_mask(bloom_CMA_base_shifted, get_market_category(bloom_CMA_base_shif
bloom_CMA = get_factor_on_date_by_mask(bloom_CMA_mask, term = 'm', winsorize_limits = 0.01, w
```

In [238...
```
ind=[]
for i in pnl.index:
    if i in PL.index:
        ind.append(i)
    else:
        pass
len(ind)
```

Out[238... 1812

In [ ]:
```
PL.loc[ind,:] #블룸버그 데이터는 비지배를 배제함.
```

In [ ]:
```
pnl.loc[ind,:]
```

In [ ]:
```
print(pnl.shape, PL.shape)
```

In [ ]:
```
Rm = get_market_return(SMB_mask, term='m').squeeze()
```

In [ ]:
```
#팩터 정리
bloom_factors = pd.concat([Rm, bloom_SMB.iloc[:,0] - bloom_SMB.iloc[:,1], bloom_HML.iloc[:,2]- bloo
bloom_factors.columns = ['Rm', 'SMB', 'HML', 'RMW', 'CMA']
bloom_factors
```

In [ ]:
```
bloom_HMLO_ols = ols(formula = 'HML ~ Rm+SMB+RMW+CMA', data = bloom_factors).fit()
bloom_HMLO_ols.summary()
```

In [ ]:
```
beta_Rm = bloom_HMLO_ols.params['Rm']
beta_SMB = bloom_HMLO_ols.params['SMB']
beta_RMW = bloom_HMLO_ols.params['RMW']
beta_CMA = bloom_HMLO_ols.params['CMA']
bloom_HMLO = pd.DataFrame(bloom_factors["HML"] - beta_Rm * bloom_factors['Rm'].squeeze() - be
bloom_HMLO.columns = ['HMLO']
```

```
bloom_factors = pd.concat([bloom_factors, HMLO], axis=1)
bloom_factors
```

In [ ]:
```
# (Yearly) 블룸버그로 뽑은 팩터랑 우리 데이터로 뽑은 팩터 상관계수
import scipy.stats as stats
for i in factors.columns:
    print(stats.pearsonr(bloom_factors[i], factors[i]))
```

In [ ]:
```
# (Quarterly) 블룸버그로 뽑은 팩터랑 우리 데이터로 뽑은 팩터 상관계수
import scipy.stats as stats
for i in factors.columns:
    print(stats.pearsonr(bloom_factors[i], factors[i]))
```

CMA 상관계수가 0.7밖에 안 되는 것 같으니 종목을 똑같이 맞추자

In [ ]:
```
print(pnl.shape, PL.shape)
ind=[]
for i in pnl.index:
    if i in PL.index:
        ind.append(i)
    else:
        pass
len(ind)
```

In [ ]:
```
adjusted_PL = pnl.loc[ind,:]
adjusted_bloom_CMA_base = adjusted_PL / TA_shifted
adjusted_bloom_CMA_base_shifted = shift_date_quarter(adjusted_bloom_CMA_base, 2)

adjusted_bloom_CMA_mask = get_mask(adjusted_bloom_CMA_base_shifted, get_market_category(adj
adjusted_bloom_CMA = get_factor_on_date_by_mask(adjusted_bloom_CMA_mask, term = 'm', winsori
```

In [ ]:
```
adjusted_CMA = pd.DataFrame(adjusted_bloom_CMA.iloc[:,0] - adjusted_bloom_CMA.iloc[:,2], column
bloom_factors = pd.concat([bloom_factors, adjusted_CMA], axis=1)
bloom_factors
```

In [ ]:
```
stats.pearsonr(bloom_factors['adjusted_CMA'], factors['CMA'])
```

와 0.97!! 종목을 통일시키니까 CMA상관계수가 97%까지 올라가네!!!

## 블룸버그 데이터 Size-B/M 포트폴리오

(1) HML로 돌리기

In [ ]:
```
# HML pfo
bloom_hml_pfo_mask = make_portfolio(get_mask(bloom_SMB_base, get_market_category(bloom_SME

bloom_pfo25 = serialize_pfo([get_mask(bloom_SMB_base, get_market_category(bloom_SMB_base), bre

bloom_pfo_return = get_factor_on_date_by_mask(bloom_pfo25, term = 'm', winsorize_limits= 0.01, we
```

In [ ]:
```
coef = []
tvalues = []
pvalues = []
```

```
se = []
r = []
for i in range(25):
    model = get_ols_model(bloom_pfo25, bloom_factors['Rm'], bloom_pfo_return.iloc[:,i], cd_91, bloom
    coef.append(model.params)
    tvalues.append(model.tvalues)
    pvalues.append(model.pvalues)
    se.append(model.bse)
    r.append(model.rsquared_adj)
```

In [ ]:
```
pd.concat(coef, axis=1)
```

In [ ]:
```
pd.concat(tvalues, axis=1)
```

In [ ]:
```
pd.concat(pvalues, axis=1)
```

In [ ]:
```
pv<0.05
```

In [ ]:
```
pv = pd.concat(pvalues, axis=1)
plt.imshow(pv<0.05, ) #노란색이 통계적 유의성 있음
```

In [ ]:
```

```

In [ ]:
```
pd.concat(se, axis=1)
```

In [ ]:
```
r
```

In [ ]:
```

```

In [ ]:
```

```

(2) HMLO로 돌리기

In [ ]:
```
coef = []
tvalues = []
pvalues = []
se = []
r = []
for i in range(25):
    model = get_ols_model(bloom_pfo25, bloom_factors['Rm'], bloom_pfo_return.iloc[:,i], cd_91, bloom
    coef.append(model.params)
    tvalues.append(model.tvalues)
    pvalues.append(model.pvalues)
    se.append(model.bse)
    r.append(model.rsquared_adj)
```

In [ ]:
```
pd.concat(coef, axis=1)
```

In [ ]:
```
pd.concat(tvalues, axis=1)
```

In [ ]:
```
pd.concat(pvalues, axis=1)
```

In [ ]:
```
pv<0.05
```

In [ ]:
```
pv = pd.concat(pvalues, axis=1)
plt.imshow(pv<0.05, ) #노란색이 통계적 유의성 있음
```

In [ ]:
```
pd.concat(se, axis=1)
```

In [ ]:
```
r
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
#get_market_category()를 블룸버그 버전으로 따로 만들려다가 그만 둠. category도 기존꺼 써도 될 듯

os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩팩터프로젝트₩₩data from bloomberg'
kospi_data = pd.read_csv('kospi mktcap.csv', encoding='cp949', index_col=0).iloc[1:,:]
kosdaq_data = pd.read_csv('kosdaq mktcap.csv', encoding='cp949', index_col=0).iloc[1:,:]
kospi_data2 = preprocessing_bloomberg(kosdaq_data, 1, ['kosdaq_data'])['kosdaq_data'].loc[:, SMB_sta
kosdaq_data2 = preprocessing_bloomberg(kospi_data,  1, ['kosdaq_data'])['kosdaq_data'].loc[:, SMB_st
```

In [ ]:
```
def bloomberg_get_market_category(division):
    os.chdir('C:₩₩Users₩₩jky93₩₩KYdrive₩₩바탕 화면₩₩창업₩₩팩터프로젝트₩₩data from bloombe
    kospi_data = pd.read_csv('kospi mktcap.csv', encoding='cp949', index_col=0).iloc[1:,:]
    kosdaq_data = pd.read_csv('kosdaq mktcap.csv', encoding='cp949', index_col=0).iloc[1:,:]
    kospi_data2 = preprocessing_bloomberg(kosdaq_data, 1, ['kosdaq_data'])['kosdaq_data'].loc[:, divisi
    kosdaq_data2 = preprocessing_bloomberg(kospi_data,  1, ['kosdaq_data'])['kosdaq_data'].loc[:, divis
```

```
    answersheet = SMB_standard * kospi_data2.iloc[:,::4]
    category = division.applymap(lambda x: whether_kospi(x, answersheet))
    return category

def whether_kospi(element, answersheet):
    if type(element)!= type(np.NaN):
        for i in tqdm.tqdm(range(answersheet.shape[0])):
            for j in range(answersheet.shape[1]):
                if type(answersheet) != type(np.NaN):
                    return 'KOSPI'
                else:
                    return 'KOSDAQ'
    else:
        print(type(element))
        return np.NaN
```

In [ ]:
```
answersheet = SMB_standard * kospi_data2.iloc[:,::4]
```

In [ ]:
```
SMB_standard
```

In [ ]:
```
kospi_data2.iloc[:,::4]
```

In [ ]:
```
kosdaq_data2.iloc[:,::4]
```

In [ ]:
```
get_market_category(RMW_base_shifted)
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```