Code 1, Part 1 - START
##

Get started

Loard packages and Eikon Lisence

In[2]:

```
import copy
from datetime import date

import eikon as ek
import numpy as np
import pandas as pd
import statsmodels.api as sm

ek.set_app_key('-YOUR EIKON APP KEY-')
```

Todays date

In[3]:

```
today = str(date.today())
```

Importing Data From Eikon Refinitiv

Financial data

In[73]:

```
del_all_ind = ['.OSEAX','.OMXSPI','.OMXCPI','.OMXHPI']
del_all, e = ek.get_data(del_all_ind,
['TR.IndexJLConstituentChangeDate','TR.IndexJLConstituentRIC.change','TR.IndexJLConstituentRIC'], {'Sdate':'2015-
01-01', 'EDate':'2021-12-31', 'IC':'B'})
s_del = del_all[del_all["Change"]=="Leaver"]["Constituent RIC"].to_list()
```

Tickers

In[74]:

```
# Listed
s_nor = "SCREEN(U(IN(Equity(active,public,primary))/*UNV:Public*/), IN(TR.ExchangeCountryCode,""NO""),
NOT_IN(TR.ExchangeMarketIdCode,""NOTC""),
CURN=USD)";"TR.CommonName;TR.ExchangeCountry;TR.ExchangeName";"curn=USD RH=In CH=Fd"
```

```
s_swe = "SCREEN(U(IN(Equity(active,public,primary))/*UNV:Public*/), IN(TR.ExchangeCountryCode,""SE""),
CURN=USD)";"TR.CommonName;TR.ExchangeCountry;TR.HeadquartersCountry;TR.ExchangeName";"curn=USD
RH=In CH=Fd"
s_fin = "SCREEN(U(IN(Equity(active,public,primary))/*UNV:Public*/), IN(TR.ExchangeCountryCode,""FI""),
CURN=USD)";"TR.CommonName;TR.ExchangeCountry;TR.HeadquartersCountry;TR.ExchangeName";"curn=USD
RH=In CH=Fd"
s_den = "SCREEN(U(IN(Equity(active,public,primary))/*UNV:Public*/), IN(TR.ExchangeCountryCode,""DK""),
CURN=USD)";"TR.CommonName;TR.ExchangeCountry;TR.HeadquartersCountry;TR.ExchangeName";"curn=USD
RH=In CH=Fd"

# Delisted
del_all_ind = ['.OSEAX','.OMXSPI','.OMXCPI','.OMXHPI']
del_all, e = ek.get_data(del_all_ind,
['TR.IndexJLConstituentChangeDate','TR.IndexJLConstituentRIC.change','TR.IndexJLConstituentRIC'], {'Sdate':'2015-
01-01', 'EDate':today, 'IC':'B'})
s_del = del_all[del_all["Change"]=="Leaver"]["Constituent RIC"].to_list()


# ### Total Return

# In[ ]:


# Norway
total_ret_nor, e = ek.get_data(s_nor,['TR.TotalReturn','TR.TotalReturn.date'], {'Frq':'M','Sdate':'2015-01-01',
'EDate':'2021-12-31'})

# Sweden
total_ret_swe, e = ek.get_data(s_swe,['TR.TotalReturn','TR.TotalReturn.date'], {'Frq':'M','Sdate':'2015-01-01',
'EDate':'2021-12-31'})

# Finland
total_ret_fin, e = ek.get_data(s_fin,['TR.TotalReturn','TR.TotalReturn.date'], {'Frq':'M','Sdate':'2015-01-01',
'EDate':'2021-12-31'})

# Denmark
total_ret_den, e = ek.get_data(s_den,['TR.TotalReturn','TR.TotalReturn.date'], {'Frq':'M','Sdate':'2015-01-01',
'EDate':'2021-12-31'})

# Delisted
total_ret_del, e = ek.get_data(s_del,['TR.TotalReturn','TR.TotalReturn.date'], {'Frq':'M','Sdate':'2015-01-01',
'EDate':'2021-12-31'})

# Combine Total Return
total_ret_all = pd.concat([total_ret_nor,total_ret_swe,total_ret_fin,total_ret_den,total_ret_del], ignore_index=True)
total_ret_all["Date"] = pd.to_datetime(total_ret_all["Date"])
total_ret_all["Date"] = total_ret_all["Date"].dt.strftime("%Y-%m-%d")

total_ret_all["Instrument"] = [word.split("^")[0] for word in total_ret_all["Instrument"].to_list()]

# Save to CSV
total_ret_all.to_csv("total_ret_all.csv")
```

Risk Free Rate

In[5]:

risk_free_rate_daily, e = ek.get_data('USD1MFSR=X',['TR.FIXINGVALUE','TR.FIXINGVALUE.date'],
{"Sdate":'2014-12-31', "Edate":'2021-12-31'})

rf = risk_free_rate_daily.copy()
rf.drop("Instrument",axis=1,inplace=True)
rf["Date"] = pd.to_datetime(rf["Date"])
rf["Date"] = rf["Date"].dt.strftime("%Y-%m-%d")
rf["Date"] = pd.to_datetime(rf["Date"])
rf['month'] = rf['Date'].to_numpy().astype('datetime64[M]')
rf["test"] = rf.duplicated(subset=['month'])
rf = rf[rf["test"]==False]
rf = rf.drop(["Date","test"],axis=1)
rf.rename(columns={"month":"Date"},inplace=True)
rf.set_index("Date",inplace=True)

# Save to CSV
rf.to_csv("risk_free_rate.csv")

Reference indices

In[6]:

reff_ind, e = ek.get_data([".OBX",".OMXS30",".OMXH25",".OMXC20"],['TR.Close','TR.TotalReturn.date'],
{'Frq':'M','Sdate':'2015-01-01', 'EDate':'2021-12-31'})
reff_ind["Date"] = pd.to_datetime(reff_ind["Date"])

reff_ind["Date"] = reff_ind["Date"].dt.strftime("%Y-%m-%d")
reff_ind = pd.pivot_table(reff_ind, values="Price Close", index="Date", columns=['Instrument']) #dropna=False
reff_ind.index = pd.to_datetime(reff_ind.index)

reff_ind.to_csv("ref_ind.csv")

Accounting Data

Tickers

In[7]:

# Listed
s_all_on = "SCREEN(U(IN(Equity(active,public,primary))/*UNV:Public*/),
IN(TR.ExchangeCountryCode,""DK"",""FI"",""NO"",""SE""), NOT_IN(TR.ExchangeMarketIdCode,""NOTC""),
CURN=USD)";"TR.CommonName;TR.HeadquartersCountry";"curn=USD RH=In CH=Fd"

# Delisted
del_all_ind = ['.OSEAX','.OMXSPI','.OMXCPI','.OMXHPI']

```
del_all, e = ek.get_data(del_all_ind,
['TR.IndexJLConstituentChangeDate','TR.IndexJLConstituentRIC.change','TR.IndexJLConstituentRIC'], {'Sdate':'2015-
01-01', 'EDate':today, 'IC':'B'})
s_all_d = del_all[del_all["Change"]=="Leaver"]["Constituent RIC"].to_list()

# Combine
s_all_d.insert(0,s_all_on)
s_all = s_all_d


# ### Time periods

# In[8]:


# Yearly
date_y = {'Sdate':'2014-12-31', 'EDate':'2021-12-31', 'FRQ': 'FY',"Curn":"USD"}
date_y_ta = {'Sdate':'2013-12-31', 'EDate':'2021-12-31', 'FRQ': 'FY',"Curn":"USD"} # Inv

# Monthly
date_m = {'Sdate':'2014-12-31', 'EDate':'2021-12-31', 'FRQ': 'm',"Curn":"USD"}


# ### Company Data

# In[9]:


# Code OK

f_com =
["TR.CommonName","TR.ExchangeName","TR.TRBCBusinessSector","TR.HQCountryCode","TR.IPODate"]

fin_com, e = ek.get_data(s_all, f_com)

for i in np.arange(len(fin_com)):
    if "OL^" in fin_com["Instrument"][i]: # Norway
        fin_com["Exchange Name"][i] = "OSLO BORS ASA"
    if "ST^" in fin_com["Instrument"][i]: # Sweden
        fin_com["Exchange Name"][i] = "NASDAQ STOCKHOLM AB"
    if "CO^" in fin_com["Instrument"][i]: # Denmark
        fin_com["Exchange Name"][i] = "NASDAQ COPENHAGEN A/S"
    if "HE^" in fin_com["Instrument"][i]: # Finland
        fin_com["Exchange Name"][i] = "NASDAQ HELSINKI LTD"

fin_com.drop(fin_com.index[fin_com['Exchange Name'] == "NO MARKET (E.G. UNLISTED)"],inplace=True)
fin_com.drop(fin_com.index[fin_com['Exchange Name'] == ""],inplace=True)
fin_com.drop_duplicates(inplace=True)

fin_com.to_csv("fin_com.csv")


# ### Market Cap
```

In[11]:

f_mc = ['TR.CompanyMarketCap.date',"TR.CompanyMarketCap"]

Monthly
fin_mc_m, e = ek.get_data(s_all, f_mc, date_m)
fin_mc_m["Instrument"] = [word.split("^")[0] for word in fin_mc_m["Instrument"].to_list()]
fin_mc_m["Date"] = pd.to_datetime(fin_mc_m["Date"])
fin_mc_m["Date"] = fin_mc_m["Date"].dt.strftime("%Y-%m-%d")
fin_mc_m.drop_duplicates(inplace=True)

Yearly
fin_mc_y, e = ek.get_data(s_all, f_mc, date_y)
fin_mc_y["Instrument"] = [word.split("^")[0] for word in fin_mc_y["Instrument"].to_list()]
fin_mc_y["Date"] = pd.to_datetime(fin_mc_y["Date"])
fin_mc_y["Date"] = fin_mc_y["Date"].dt.strftime("%Y-%m-%d")
fin_mc_y.drop_duplicates(inplace=True)

# To CSV
fin_mc_m.to_csv("fin_mc_m.csv")
fin_mc_y.to_csv("fin_mc_y.csv")

Profitability

In[12]:

# Revenye
f_rev = ["TR.F.TotRevenue","TR.F.TotRevenue.date"]

fin_rev, e = ek.get_data(s_all, f_rev ,date_y)
fin_rev["Instrument"] = [word.split("^")[0] for word in fin_rev["Instrument"].to_list()]
fin_rev["Date"] = pd.to_datetime(fin_rev["Date"])
fin_rev["Date"] = fin_rev["Date"].dt.strftime("%Y-%m-%d")
fin_rev.drop_duplicates(inplace=True)

# Operating expence
f_oe = ["TR.F.OpExpnTot", "TR.F.OpExpnTot.date"]

fin_oe, e = ek.get_data(s_all, f_oe, date_y)
fin_oe["Instrument"] = [word.split("^")[0] for word in fin_oe["Instrument"].to_list()]
fin_oe["Date"] = pd.to_datetime(fin_oe["Date"])
fin_oe["Date"] = fin_oe["Date"].dt.strftime("%Y-%m-%d")
fin_oe.drop_duplicates(inplace=True)

# Interest expence
f_ie = ["TR.F.IntrExpnNetOfIntrInc", "TR.F.IntrExpnNetOfIntrInc.date"]

fin_ie, e = ek.get_data(s_all, f_ie, date_y)
fin_ie["Instrument"] = [word.split("^")[0] for word in fin_ie["Instrument"].to_list()]
fin_ie["Date"] = pd.to_datetime(fin_ie["Date"])
fin_ie["Date"] = fin_ie["Date"].dt.strftime("%Y-%m-%d")

```
# Book Euity
f_be = ["TR.F.ShHoldEqParentShHoldTot", "TR.F.ShHoldEqParentShHoldTot.date"]

fin_be, e = ek.get_data(s_all, f_be, date_y)
fin_be["Instrument"] = [word.split("^")[0] for word in fin_be["Instrument"].to_list()]
fin_be["Date"] = pd.to_datetime(fin_be["Date"])
fin_be["Date"] = fin_be["Date"].dt.strftime("%Y-%m-%d")

# To CSV
fin_rev.to_csv("fin_rev.csv")
fin_oe.to_csv("fin_oe.csv")
fin_ie.to_csv("fin_ie.csv")
fin_be.to_csv("fin_be.csv")


# ### Investment

# In[13]:


# Total Assets
f_ta = ["TR.TotalAssetsReported", "TR.TotalAssetsReported.date"]

fin_ta, e = ek.get_data(s_all, f_ta, date_y_ta)
fin_ta["Instrument"] = [word.split("^")[0] for word in fin_ta["Instrument"].to_list()]
fin_ta["Date"] = pd.to_datetime(fin_ta["Date"])
fin_ta["Date"] = fin_ta["Date"].dt.strftime("%Y-%m-%d")

fin_ta.to_csv("fin_ta.csv")


# ### Value

# In[14]:


# Book to Market
f_bm = ["TR.PriceToBVPerShare", "TR.PriceToBVPerShare.date"]

fin_bm, e = ek.get_data(s_all, f_bm, date_y)
fin_bm["Instrument"] = [word.split("^")[0] for word in fin_bm["Instrument"].to_list()]
fin_bm["Date"] = pd.to_datetime(fin_bm["Date"])
fin_bm["Date"] = fin_bm["Date"].dt.strftime("%Y-%m-%d")

fin_bm.to_csv("fin_bm.csv")


# # Abstractions

# ## Load Data

# ### Load all accounting data from CSV, Calculate Factor Values
#
```

```
# In[15]:


def get_fin_data():
    """
    This code retunr all relevant financial data:
    Company data (com), Book Equity (be), Book to Market (bm), Interest Expenses (ie), Market Cap (mc), Operating Expenses (oe), Revenue (re), Total Assets (ta)
    """
    # Get data
    com = pd.read_csv("fin_com.csv")
    be = pd.read_csv("fin_be.csv")
    bm = pd.read_csv("fin_bm.csv")
    ie = pd.read_csv("fin_ie.csv")
    mc = pd.read_csv("fin_mc_y.csv")
    oe = pd.read_csv("fin_oe.csv")
    re = pd.read_csv("fin_rev.csv")
    ta = pd.read_csv("fin_ta.csv")

    # remove unwanted column
    com.drop("Unnamed: 0",axis=1, inplace=True)
    be.drop("Unnamed: 0",axis=1, inplace=True)
    bm.drop("Unnamed: 0",axis=1, inplace=True)
    ie.drop("Unnamed: 0",axis=1, inplace=True)
    mc.drop("Unnamed: 0",axis=1, inplace=True)
    oe.drop("Unnamed: 0",axis=1, inplace=True)
    re.drop("Unnamed: 0",axis=1, inplace=True)
    ta.drop("Unnamed: 0",axis=1, inplace=True)

    # company set index
    com.set_index("Instrument", inplace=True)

    # pivot table
    be = pd.pivot_table(be, values="Shareholders' Equity - Attributable to Parent ShHold - Total", index="Date", columns=['Instrument']) #dropna=False
    bm = pd.pivot_table(bm, values="Price To Book Value Per Share (Daily Time Series Ratio)", index="Date", columns=['Instrument'])
    ie = pd.pivot_table(ie, values="Interest Expense - Net of (Interest Income)",index="Date", columns=['Instrument'], dropna=False) # Some banks don't have ie
    mc = pd.pivot_table(mc, values="Company Market Cap",index="Date", columns=['Instrument'])
    oe = pd.pivot_table(oe, values="Operating Expenses - Total",index="Date", columns=['Instrument'])
    re = pd.pivot_table(re, values="Revenue from Business Activities - Total",index="Date", columns=['Instrument'])
    ta = pd.pivot_table(ta, values="Total Assets, Reported", index="Date", columns=['Instrument'])

    # set index to datetime
    be.index = pd.to_datetime(be.index)
    bm.index = pd.to_datetime(bm.index)
    ie.index = pd.to_datetime(ie.index)
    mc.index = pd.to_datetime(mc.index)
    oe.index = pd.to_datetime(oe.index)
    re.index = pd.to_datetime(re.index)
    ta.index = pd.to_datetime(ta.index)
```

```
    # combine yearly data
    be = be.groupby(be.index.year).mean()
    bm = bm.groupby(bm.index.year).mean()
    ie = ie.groupby(ie.index.year).mean()
    mc = mc.groupby(mc.index.year).mean()
    oe = oe.groupby(oe.index.year).mean()
    re = re.groupby(re.index.year).sum() # lots of missing data that should be zero
    ta = ta.groupby(ta.index.year).mean()

    ie.fillna(0,inplace=True)

    # mach reÃ¥porting year wiht total retunr, Data is reported at the start of the calender year, instead of exit
    be = be.shift(periods=1)
    bm = bm.shift(periods=1)
    ie = ie.shift(periods=1)
    mc = mc.shift(periods=1)
    oe = oe.shift(periods=1)
    re = re.shift(periods=1)
    ta = ta.shift(periods=1)


    # Keep full (_f)
    be_f = be.copy()
    bm_f = bm.copy()
    ie_f = ie.copy()
    mc_f = mc.copy()
    oe_f = oe.copy()
    re_f = re.copy()
    ta_f = ta.copy()

    com_f = com.copy()

    # Comanies with complete data, all columns at the same location
    comp_full_data =
set(com.index).intersection(set(be.columns).intersection(set(bm.columns).intersection(set(ie.columns).intersection(set(
mc.columns).intersection(set(oe.columns).intersection(set(re.columns).intersection(ta)))))))

    be = be[comp_full_data]
    bm = bm[comp_full_data]
    ie = ie[comp_full_data]
    mc = mc[comp_full_data]
    oe = oe[comp_full_data]
    re = re[comp_full_data]
    ta = ta[comp_full_data]

    com = com.loc[set(com.index).intersection(set(ta.columns))]

    # Calculations
       # Profitability, Operating Profitability (op)
    op = (re - oe + ie)/be
       # Investments (iv)
    iv = (ta-ta.shift(periods=1))/(ta.shift(periods=1))

    # Cut data to size
```

```
    be = be.loc[2015:2021]
    bm = bm.loc[2015:2021]
    ie = ie.loc[2015:2021]
    mc = mc.loc[2015:2021]
    oe = oe.loc[2015:2021]
    re = re.loc[2015:2021]
    ta = ta.loc[2015:2021]
    op = op.loc[2015:2021]
    iv = iv.loc[2015:2021]

    return com, be, bm, ie, mc, oe, re, ta, op, iv, com_f, be_f, bm_f, ie_f, mc_f, oe_f, re_f, ta_f


# ### Load Total return and Risk Free Rate from CSV, Calculatet Excess Return

# In[16]:


def get_tr_data():
    """
    This code retunr all relevant return data:
    total return (tr)
    """

    # Get data
        # Total Return
    tr = pd.read_csv("total_ret_all.csv")
        # Risk free rate
    rf = pd.read_csv("risk_free_rate.csv")


    # drop ["contry","Unnamed: 0"]
#     tr.drop(["Country","Unnamed: 0"],axis=1, inplace=True)
    tr.drop(["Unnamed: 0"],axis=1, inplace=True)

    # PPivot data
    tr = pd.pivot_table(tr, values='Total Return',index="Date", columns=['Instrument'])

    # Set index
    rf.set_index("Date",inplace=True)

    # set index to datetime
    tr.index = pd.to_datetime(tr.index)
    rf.index = pd.to_datetime(rf.index)

    #combine monthly data
    tr =  tr.groupby([lambda x: x.year, lambda x: x.month]).sum()

    # reconstruct index, monly data is reported here on first day, is for the whole month
        # seperate year and month into columns
    tr.reset_index(level=[0,1],inplace=True)
        # combine year and month to date
    tr["date"] = tr["level_0"].astype(str)+ "-" + (("0" + tr["level_1"].astype(str)).str[-2:])
        # Set date to index
```

```
    tr.set_index("date",inplace = True)
        # drop unwanted columns
    tr.drop(["level_0","level_1"], axis=1, inplace=True)
        # set date index to datetime
    tr.index = pd.to_datetime(tr.index)

    # Maxh datasets
    rf = rf.loc[tr.index]

    # excess return
    er = tr.sub(rf["Fixing Value"],axis=0)

    return tr, er, rf


# ### Load Monthly Marct Cap from CSV

# In[17]:


def get_mcm_data():
    """
    This code retunr monthly market cap:
    Marcet Cap Monthly (mcm)
    """

    # Get data, monthly market cap
    mcm = pd.read_csv("fin_mc_m.csv")

    # drop ["contry","Unnamed: 0"]
    mcm.drop("Unnamed: 0",axis=1, inplace=True)

    # PPivot data
    mcm = pd.pivot_table(mcm, values="Company Market Cap",index="Date", columns=['Instrument'])

    # Set index
    mcm.index = pd.to_datetime(mcm.index)



    #combine monthly data
    mcm =  mcm.groupby([lambda x: x.year, lambda x: x.month]).sum()

    # reconstruct index, monly data is reported here on first day, is for the whole month
        # seperate year and month into columns
    mcm.reset_index(level=[0,1],inplace=True)
        # combine year and month to date
    mcm["date"] = mcm["level_0"].astype(str)+ "-" + (("0" + mcm["level_1"].astype(str)).str[-2:])
        # Set date to index
    mcm.set_index("date",inplace = True)
        # drop unwanted columns
    mcm.drop(["level_0","level_1"], axis=1, inplace=True)
        # set date index to datetime
    mcm.index = pd.to_datetime(mcm.index)
```

```
    return mcm


# ### Load Referance Index from CSV

# In[18]:


def get_ref_ind():
    ref_ind = pd.read_csv("ref_ind.csv")
    ref_ind.set_index("Date",inplace=True)
    ref_ind.index = pd.to_datetime(ref_ind.index)

    ref_ind = ref_ind.groupby([lambda x: x.year, lambda x: x.month]).sum()
    # reconstruct index, monly data is reported here on first day, is for the whole month
        # seperate year and month into columns
    ref_ind.reset_index(level=[0,1],inplace=True)
        # combine year and month to date
    ref_ind["date"] = ref_ind["level_0"].astype(str)+ "-" + (("0" + ref_ind["level_1"].astype(str)).str[-2:])
        # Set date to index
    ref_ind.set_index("date",inplace = True)
        # drop unwanted columns
    ref_ind.drop(["level_0","level_1"], axis=1, inplace=True)
        # set date index to datetime
    ref_ind.index = pd.to_datetime(ref_ind.index)
    return(ref_ind)


# ## Data Manipulation

# ### Value weighted Return

# In[19]:


def vw(mc):
    """
    Turn market cap values into percentage
    """
    mc_w = mc.div(mc.sum(axis=1), axis=0)
    return mc_w


# ### Portfolio percentile sorting

# In[20]:


def sort_port(data, n=[0.5]):
    """
    Returns the list of the tickers in each percentile group per year.
    self specified percentiles
    """
```

```
    # Len data
    p = len(data.index)

    # percentile array
    per = np.array(n)
    per = np.insert(per,0,0)
    per = np.insert(per,len(n)+1,1)

    # Quantiles
    data_p = data.quantile(q=per, axis=1).transpose()

    # emplty lists to fill with ticker grouping
    data_p_g = [[0 for i in range(len(n)+1)] for j in range(p)]

    # assigns tickers
    for i in np.arange(p): # 0:2016, 1: 2017, 2:2018,....
        for j in np.arange(len(n)+1): # 0: 0-20%, 1: 20-40%, or simmilar based on the number of bins
            data_p_g[i][j] = data.iloc[i][data.iloc[i].between(data_p.iloc[i].iloc[j],data_p.iloc[i].iloc[j+1])].index
    return data_p_g


# ### Portfolio Return

# In[21]:


# 1 Factor
def port_ret_1_fac(p_g_1, str_1, ret, mcm):
    # mtk.port_ret_2_fac(p_g_1=mc_p_g_2, p_g_2=bm_p_g_3, ret = er, mcm = mcm, str_1 = "Size", str_2 = "Book")
    n = len(p_g_1[0])

    str_1_1 = [str(x) for x in np.arange(n)+1]

    port_ret = pd.DataFrame(index=ret.index)
    fill = [None]*len(ret.index)

    for i in np.arange(n): #Size
        for y in np.arange(len(p_g_1)):
            for k in np.arange(12):
                fill[k+y*12] = ((ret[set(p_g_1[y][i])].iloc[k+12*y]*vw(mcm[set(p_g_1[y][i])]).iloc[k+12*y]).sum())
        port_ret[str_1+" "+str_1_1[i]] = fill
    return port_ret

# 2 Factors
def port_ret_2_fac(p_g_1, p_g_2, str_1, str_2, ret, mcm):
    # mtk.port_ret_2_fac(p_g_1=mc_p_g_2, p_g_2=bm_p_g_3, ret = er, mcm = mcm, str_1 = "Size", str_2 = "Book")
    n = len(p_g_1[0])
    m = len(p_g_2[0])

    str_1_1 = [str(x) for x in np.arange(n)+1]
    str_1_2 = [str(x) for x in np.arange(m)+1]

    port_ret = pd.DataFrame(index=ret.index)
    fill = [None]*len(ret.index)
```

```
    for i in np.arange(n): #Size
        for j in np.arange(m): #Other
            for y in np.arange(len(p_g_1)):
                for k in np.arange(12):
                    fill[k+y*12] = ((ret[set(p_g_2[y][j]).intersection(set(p_g_1[y][i]))].iloc[k+12*y]*vw(mcm[set(p_g_2[y]
[j]).intersection(set(p_g_1[y][i]))]).iloc[k+12*y]).sum())
            port_ret[str_1+" "+str_1_1[i]+" - "+str_2+" "+str_1_2[j]] = fill
    return port_ret

# 3 Factors
def port_ret_3_fac(p_g_1, p_g_2, p_g_3, ret, mcm, str_1, str_2, str_3):
    n = len(p_g_1[0])
    m = len(p_g_2[0])
    o = len(p_g_3[0])


    str_1_1 = [str(x) for x in np.arange(n)+1]
    str_1_2 = [str(x) for x in np.arange(m)+1]
    str_1_3 = [str(x) for x in np.arange(o)+1]

    port_ret = pd.DataFrame(index=ret.index)
    fill = [None]*len(ret.index)

    for i in np.arange(n): #Size
        for j in np.arange(m): #Other 1
            for l in np.arange(o): #Other 2
                for y in np.arange(len(p_g_1)):
                    for k in np.arange(12):
                        fill[k+y*12] = ((ret[set(p_g_1[y][i]).intersection(set(set(p_g_2[y][j]).intersection(set(p_g_3[y]
[l])))))].iloc[k+12*y]*vw(mcm[set(p_g_1[y][i]).intersection(set(set(p_g_2[y][j]).intersection(set(p_g_3[y]
[l])))))]).iloc[k+12*y]).sum())
                port_ret[str_1+" "+str_1_1[i]+" - "+str_2+" "+str_1_2[j]+" - "+str_3+" "+str_1_3[l]] = fill
    return port_ret

# 4 Factors
def port_ret_4_fac(p_g_1, p_g_2, p_g_3, p_g_4, ret, mcm, str_1, str_2, str_3, str_4):
    n = len(p_g_1[0])
    m = len(p_g_2[0])
    o = len(p_g_3[0])
    p = len(p_g_4[0])

    str_1_1 = [str(x) for x in np.arange(n)+1]
    str_1_2 = [str(x) for x in np.arange(m)+1]
    str_1_3 = [str(x) for x in np.arange(o)+1]
    str_1_4 = [str(x) for x in np.arange(p)+1]

    port_ret = pd.DataFrame(index=ret.index)
    fill = [None]*len(ret.index)

    for i in np.arange(n): #Size
        for j in np.arange(m): #Other 1
            for l in np.arange(o): #Other 2
                for h in np.arange(p): #Other 3
```

```
                for y in np.arange(len(p_g_2)):
                    for k in np.arange(12):
                        fill[k+y*12] = ((ret[set(p_g_1[y][i]).intersection(set(p_g_2[y][j]).intersection(set(p_g_3[y]
[l]).intersection(set(p_g_4[y][h])))))].iloc[k+12*y]*vw(mcm[set(p_g_1[y][i]).intersection(set(p_g_2[y]
[j]).intersection(set(p_g_3[y][l]).intersection(set(p_g_4[y][h])))))]).iloc[k+12*y]).sum())
                port_ret[str_1+" "+str_1_1[i]+" - "+str_2+" "+str_1_2[j]+" - "+str_3+" "+str_1_3[l]+" - "+str_4+"
"+str_1_4[h]] = fill
    return port_ret

# Full function
def port_ret(port, sort, name, sort_dim, ret, mcm):
    """
    Returns the monthly portfolio returns "port_ret" of the portfolios defined here.
    Size is a constant factor
    Port: data to be sorted into portfolios (size, book, profitability, investments). Sort: selected method of sorting the
portfolios [[0.5],[0.3,0.7],[0.3,0.7],[0.3,0.7]] 2x3 sort
    Name: Name of the sorts. Sort_dim: number of portfolios to be sorted togeather, (2x2, 2x2x2,...,).
    Ret: Excess return. MCM: Market capitalization monthly.
    """
    sorted_port = [None]*len(port)

    # Port Sort
        # Sorts companies into prortfolios based on data and sort percentiles
    for i in np.arange(len(port)):
        sorted_port[i] = sort_port(port[i], sort[i]) # Sort_port function

    # port ret
        # calculates the averave return of the sorted portfolios
    if sort_dim==1: # N  sort
        port_ret = [None]*(len(port))
        for i in np.arange(len(port)):
            port_ret = port_ret_1_fac(p_g_1=sorted_port[0], ret = ret, mcm = mcm, str_1 = name[0])

    if sort_dim==2: # N x N sort
        port_ret = [None]*(len(port)-1)
        for i in np.arange((len(port)-1)):
            port_ret[i] = port_ret_2_fac(p_g_1=sorted_port[0], p_g_2=sorted_port[i+1], ret = ret, mcm = mcm, str_1 =
name[0], str_2 = name[i+1])

    if sort_dim==3: # N  N x N sort
        if len(port)==3:
            port_ret = port_ret_3_fac(p_g_1=sorted_port[0], p_g_2=sorted_port[1], p_g_3=sorted_port[2], ret = ret, mcm =
mcm, str_1=name[0], str_2=name[1], str_3=name[2])
        if len(port)==4:
            fi = np.arange(3) # first
            se = np.roll(fi,2) # second
            port_ret = [None]*(len(port)-1)
            for i in np.arange(len(port)-1):
                port_ret[i] = port_ret_3_fac(p_g_1=sorted_port[0], p_g_2=sorted_port[fi[i]+1], p_g_3=sorted_port[se[i]+1],
ret = ret, mcm = mcm, str_1=name[0], str_2=name[fi[i]+1], str_3=name[se[i]+1])

    if sort_dim == 4: # N x N x N x N sort
        port_ret = port_ret_4_fac(p_g_1 = sorted_port[0], p_g_2 = sorted_port[1], p_g_3 = sorted_port[2], p_g_4 =
sorted_port[3], ret = ret, mcm = mcm, str_1 = name[0], str_2 = name[1], str_3 = name[2], str_4 = name[3])
```

```
    return port_ret


# ### Number of LHS Portfolios

# In[22]:


def port_num_2_fac(p_g_1, p_g_2, str_1, str_2):
    """
    Return number of firms in each port per year, for 2 facor sort
    """
    # mtk.port_ret_2_fac(p_g_1=mc_p_g_2, p_g_2=bm_p_g_3, ret = er, mcm = mcm, str_1 = "Size", str_2 = "Book")
    n = len(p_g_1[0])
    m = len(p_g_2[0])

    str_1_1 = [str(x) for x in np.arange(n)+1]
    str_1_2 = [str(x) for x in np.arange(m)+1]

    port_ret = pd.DataFrame(index=np.arange(len(p_g_1)))
    fill = [None]*len(port_ret.index)

    for i in np.arange(n): #Size
        for j in np.arange(m): #Other
            for y in np.arange(len(p_g_1)):
                fill[y] = len(set(p_g_2[y][j]).intersection(set(p_g_1[y][i])))
            port_ret[str_1+" "+str_1_1[i]+" - "+str_2+" "+str_1_2[j]] = fill
    return port_ret


def port_num_3_fac(p_g_1, p_g_2, p_g_3, str_1, str_2, str_3):
    """
    Return number of firms in each port per year, for 3 facor sort
    """
    n = len(p_g_1[0])
    m = len(p_g_2[0])
    o = len(p_g_3[0])


    str_1_1 = [str(x) for x in np.arange(n)+1]
    str_1_2 = [str(x) for x in np.arange(m)+1]
    str_1_3 = [str(x) for x in np.arange(o)+1]

    port_ret = pd.DataFrame(index=np.arange(len(p_g_1)))
    fill = [None]*len(port_ret.index)


    for i in np.arange(n): #Size
        for j in np.arange(m): #Other 1
            for l in np.arange(o): #Other 2
                for y in np.arange(len(p_g_1)):
                    fill[y] = len(set(p_g_1[y][i]).intersection(set(set(p_g_2[y][j]).intersection(set(p_g_3[y][l])))))
                port_ret[str_1+" "+str_1_1[i]+" - "+str_2+" "+str_1_2[j]+" - "+str_3+" "+str_1_3[l]] = fill
    return port_ret
```

```
def port_num_4_fac(p_g_1, p_g_2, p_g_3, p_g_4, str_1, str_2, str_3, str_4):
    n = len(p_g_1[0])
    m = len(p_g_2[0])
    o = len(p_g_3[0])
    p = len(p_g_4[0])

    str_l_1 = [str(x) for x in np.arange(n)+1]
    str_l_2 = [str(x) for x in np.arange(m)+1]
    str_l_3 = [str(x) for x in np.arange(o)+1]
    str_l_4 = [str(x) for x in np.arange(p)+1]

    port_ret = pd.DataFrame(index=np.arange(len(p_g_1)))
    fill = [None]*len(port_ret.index)

    for i in np.arange(n): #Size
        for j in np.arange(m): #Other 1
            for l in np.arange(o): #Other 2
                for h in np.arange(p): #Other 3
                    for y in np.arange(len(p_g_2)):
                        fill[y] = len(set(p_g_1[y][i]).intersection(set(p_g_2[y][j]).intersection(set(p_g_3[y]
[l]).intersection(set(p_g_4[y][h])))))
                    port_ret[str_1+" "+str_l_1[i]+" - "+str_2+" "+str_l_2[j]+" - "+str_3+" "+str_l_3[l]+" - "+str_4+"
"+str_l_4[h]] = fill
    return port_ret


def port_num(port, sort, name, sort_dim):
    """
    Return the number of firms per portfolio per yaar
    Size is a constant factor
    Port: data to be sorted into portfolios (size, book, profitability, investments). Sort: selected method of sorting the
portfolios [[0.5],[0.3,0.7],[0.3,0.7],[0.3,0.7]] 2x3 sort
    Name: Name of the sorts. Sort_dim: number of portfolios to be sorted togeather, (2x2, 2x2x2,...,).
    """
    sorted_port = [None]*len(port)
    # Port Sort
        # Sorts companies into prortfolios based on data and sort percentiles
    for i in np.arange(len(port)):
        sorted_port[i] = sort_port(port[i], sort[i]) # Sort_port function

    if sort_dim==2: # N x N sort
        port_ret = [None]*(len(port)-1)
        for i in np.arange((len(port)-1)):
            port_ret[i] = port_num_2_fac(p_g_1=sorted_port[0], p_g_2=sorted_port[i+1], str_1 = name[0], str_2 =
name[i+1])

    if sort_dim==3: # N  N x N sort
        if len(port)==3:
            port_ret = port_num_3_fac(p_g_1=sorted_port[0], p_g_2=sorted_port[1], p_g_3=sorted_port[2], str_1=name[0],
str_2=name[1], str_3=name[2])
        if len(port)==4:
            fi = np.arange(3) # first
            se = np.roll(fi,2) # second
```

```
        port_ret = [None]*(len(port)-1)
        for i in np.arange(len(port)-1):
            port_ret[i] = port_num_3_fac(p_g_1=sorted_port[0], p_g_2=sorted_port[fi[i]+1], p_g_3=sorted_port[se[i]+1], str_1=name[0], str_2=name[fi[i]+1], str_3=name[se[i]+1])
    if sort_dim == 4: # N x N x N x N sort
        port_ret = port_num_4_fac(p_g_1 = sorted_port[0], p_g_2 = sorted_port[1], p_g_3 = sorted_port[2], p_g_4 = sorted_port[3], str_1 = name[0], str_2 = name[1], str_3 = name[2], str_4 = name[3])

    return port_ret


# ### Factor Calculation

# In[23]:


def fac_cal(port, sort, name, fac_name, sort_dim, ret, mcm):
    """
    Use port_ret to calcuate the factors from the input data
    Use the same input data +
    fac_name: the name of the factors to be calculated
    """

    # Portfolio return
    port_rets = port_ret(port, sort, name, sort_dim, ret, mcm)

    # Empty DataFrame to fill with factors
    if isinstance(port_rets, pd.DataFrame):
        factor = pd.DataFrame(index=port_rets.index)
    else:
        factor = pd.DataFrame(index=port_rets[0].index)

    # Combine port ret
    if isinstance(port_rets, pd.DataFrame):
        all_port_ret = port_rets
    if len(port_rets)==1:
        all_port_ret = port_rets[0]
    if len(port_rets)==2:
        all_port_ret = pd.concat([port_rets[0],port_rets[1]],axis=1)
    if len(port_rets)==3:
        all_port_ret = pd.concat([port_rets[0],port_rets[1],port_rets[2]],axis=1)
    if len(port_rets)==4:
        all_port_ret = pd.concat([port_rets[0],port_rets[1],port_rets[2],port_rets[3]],axis=1)

    # factor calculation
    for i in np.arange(len(port)):
        name_low = name[i]+" 1"
        name_high = name[i]+" "+ str(len(sort[i])+1)
        if i == 0: # SMB
            factor[fac_name[i]] = all_port_ret.filter(regex='Size 1').sum(axis=1)/len(all_port_ret.filter(regex='Size 1').columns)-all_port_ret.filter(regex=name_high).sum(axis=1)/len(all_port_ret.filter(regex=name_high).columns)
        else: # other factors
            factor[fac_name[i]] = all_port_ret.filter(regex=name_high).sum(axis=1)/len(all_port_ret.filter(regex=name_high).columns)-
```

```
all_port_ret.filter(regex=name_low).sum(axis=1)/len(all_port_ret.filter(regex=name_low).columns)
    # RM-RF
    er_mk = (vw(mcm)*ret).sum(axis=1)/100
    factor.insert(loc = 0, column='RM - RF', value=er_mk)
    return factor


# ## Factor and regression tests

# ### Factor Mean, Standard Deviation, and t-stat

# In[24]:


def factor_mean_std_t(factors,factor_name):
    """
    Return mean, standard deviation and T-stat of factors
    """
    ### Factor Mean
    factor_mean = pd.DataFrame()
    for i in np.arange(len(factors)):
        factor_mean = factor_mean.append(pd.DataFrame(factors[i].mean()).T,ignore_index=True)
    factor_mean.index = factor_name

    ### Factor std
    factor_std = pd.DataFrame()
    for i in np.arange(len(factors)):
        factor_std = factor_std.append(pd.DataFrame(factors[i].std()).T,ignore_index=True)
    factor_std.index = factor_name

    ### Factor t-stat
    factor_tstat = pd.DataFrame()
    for i in np.arange(len(factors)):
        factor_tstat =
factor_tstat.append(pd.DataFrame(factor_mean.iloc[i]/(factor_std.iloc[i]/((len(factors[i]))**0.5))).T,ignore_index=True)
    factor_tstat.index = factor_name

    return factor_mean, factor_std, factor_tstat


# ### Factor Spanning

# In[25]:


def fac_span(data):
    """
    Preform a Factor spanning test on given factors.
    Return regression coefficients, t-value, R2
    """
    # Get Data
    fac = data.copy()
    l = len(fac.columns)
```

```
    # RM x100
#     fac["RM - RF"]=fac["RM - RF"]*100

    # Col names
    col_f = fac.columns.to_list()
        # Constant Term CT
    col_f.insert(0,"CT")
        # T-stat
    col_t = ["t_"+ s for s in col_f]
        # R2
    col_r = ["R2"]
        # Combine
    col = col_f + col_t + col_r

    # Empty DataFrame
    fac_span = pd.DataFrame(columns=col)

    # for loop
    for i in np.arange(l):
        # x, y
        reg = fac.drop(col_f[i+1],axis=1)
        reg.insert(0, 'CT', 1)

        y = fac[col_f[i+1]]
        x = reg

        # Regresion
        model = sm.OLS(y,x,hasconst=bool)
        result = model.fit()

            # Parameters
        par =  result.params
            # T-test
        t_test = result.tvalues
        col_t_t = col_t.copy()
        col_t_t.remove(col_t[i+1])
        t_test.index = col_t_t
            # R2
        r_2 = pd.Series(result.rsquared,index=["R2"])
            # NaN
        nan = pd.Series([np.nan,np.nan],index=[col_f[i+1],col_t[i+1]])

        # All results
        res = pd.concat([par,t_test,r_2,nan])

        # Data Frame
        par_df = pd.DataFrame(res).transpose()
        par_df = par_df[col]
        par_df["Index"] = col_f[i+1]
        par_df.set_index(["Index"],inplace=True)

        # append df
        fac_span = fac_span.append(par_df)
    return fac_span
```

```
# ### Asset Pricing Test

# In[26]:


def asset_price_test(fac,port_ret,port_name,factor_name):
    for i in np.arange(len(fac)):
        fac[i].insert(0, 'CT', 1)
    for i in np.arange(len(port_ret)):
        port_ret[i] = port_ret[i][((port_ret[i].abs()).sum())[((port_ret[i].abs()).sum()!=0)].index]

    lhs = pd.DataFrame()
    lhs_1 = []

    a_mean = pd.DataFrame(columns = port_name,index=factor_name)
    at_mean = pd.DataFrame(columns = port_name,index=factor_name)
    r_squared = pd.DataFrame(columns = port_name,index=factor_name)
    ri_squared = pd.DataFrame(columns = port_name,index=factor_name)
    std_mean = pd.DataFrame(columns = port_name,index=factor_name) # std2 / mean alpha



    for k in np.arange(len(fac)):# k = 24 # per factor set

        lhs = pd.DataFrame()
        lhs_1 = []

        a_s = []
        ria_s = []
        r2_s = []
        ri_s = []
        ari_s = []
        std_s = []
        std2_s = []

        for j in np.arange(len(port_ret)): # per portfolio ret combination
            a = np.empty(0)
            r2 = np.empty(0)
            ri = np.empty(0)
            std = np.empty(0)

            for i in np.arange(len(port_ret[j].columns)): # per portfolio ret
                lhs = port_ret[j].iloc[:,i]
                rhs = fac[k]

                model = sm.OLS(lhs,rhs,hasconst=bool)
                result = model.fit()

                a = np.append(a,result.params[0])
                r2 = np.append(r2,result.rsquared_adj)
                ri = np.append(ri,lhs.mean()-rhs["RM - RF"].mean())
                std = np.append(std,result.bse[0]**2)
```

```
            a_s.append(np.absolute(a).mean())
            r2_s.append(np.absolute(r2).mean())
            ria_s.append(np.absolute(a**2).mean())
            ri_s.append(np.absolute(ri**2).mean())
            ari_s.append(np.absolute(ria_s[j]/ri_s[j]))
            std2_s.append((std[j]**2).mean())
            std_s.append(np.absolute(std2_s[j]/ria_s[j]).mean())

        for h in np.arange(len(port_ret)): # per portfolio ret combination
            a_mean.iloc[k][port_name[h]] = a_s[h]
            r_squared.iloc[k][port_name[h]] = r2_s[h]
            ri_squared.iloc[k][port_name[h]] = ari_s[h]
            std_mean.iloc[k][port_name[h]] = std_s[h]


    return(a_mean,r_squared,ri_squared,std_mean)


# ### Portfolio alphas

# In[76]:


def asset_price_test_alpha(fac,port_ret,port_name,factor_name):

    for i in np.arange(len(fac)):
        fac[i].insert(0, 'CT', 1)
    for i in np.arange(len(port_ret)):
        port_ret[i] = port_ret[i][((port_ret[i].abs()).sum())[((port_ret[i].abs()).sum()!=0)].index]

    lhs = pd.DataFrame()
    lhs_1 = []

    a_list = []
    at_list = []

    for k in np.arange(len(fac)):# k = 25 # per factor set

        lhs = pd.DataFrame()
        lhs_1 = []

        a_s = []
        at_s = []

        for j in np.arange(len(port_ret)): # per portfolio ret combination
            a = np.empty(0)
            at = np.empty(0)

            for i in np.arange(len(port_ret[j].columns)): # per portfolio ret
                lhs = port_ret[j].iloc[:,i]
                rhs = fac[k]
```

```
            model = sm.OLS(lhs,rhs,hasconst=bool)
            result = model.fit()

            a = np.append(a,result.params[0])
            at = np.append(at,result.tvalues[0])

#           a_s.append(pd.concat([pd.concat([pd.DataFrame(a.reshape((4,4)))], keys=["Size"], names=[''])], keys=
[port_name[j]], names=[''],axis=1))
#           at_s.append(pd.concat([pd.concat([pd.DataFrame(at.reshape((4,4)))], keys=["Size"], names=[''])], keys=
[port_name[j]], names=[''],axis=1))

          a_s.append(pd.DataFrame(a.reshape((4,4))))
          at_s.append(pd.DataFrame(at.reshape((4,4))))

      a_list.append(a_s)
      at_list.append(at_s)
   return a_list, at_list


# ### Asset Pricing Results Table

# In[28]:


def asset_pricing_fin_tab(res,ind_1,ind_2,ind_3,col,m_ind):
   bm_2x3 = res.loc[ind_1][col]
   bm_2x2 = res.loc[ind_2][col]
   bm_2x2x2x2 = res.loc[ind_3][col]

   bm_2x2.loc[" "] = " "
   bm_2x3.loc[" "] = " "
   bm_2x2x2x2.loc[" "] = " "

   bm_2x3.index = [i.split(" - ")[0] for i in bm_2x3.index.to_list()]


   bm_2x2 = bm_2x2.set_index(bm_2x3.index)
   bm_2x2x2x2 = bm_2x2x2x2.set_index(bm_2x3.index)

   bm_2x2[" "] = " "
   bm_2x3[" "] = " "
   bm_2x2x2x2[" "] = " "


   df1 = pd.concat([pd.concat([bm_2x3], keys=m_ind, names=['Test Portfolirs'])], keys=["2x3"], names=
['Sorting'],axis=1)
   df2 = pd.concat([pd.concat([bm_2x2], keys=m_ind, names=['Test Portfolirs'])], keys=["2x2"], names=
['Sorting'],axis=1)
   df3 = pd.concat([pd.concat([bm_2x2x2x2], keys=m_ind, names=['Test Portfolirs'])], keys=["2x2x2x2"], names=
['Sorting'],axis=1)

   size_bm = df1.join(df2.join(df3))

   return df1
```

Main Code

Load Data From CSV, and Clean data

Financial Data

In[29]:

```
com, be, bm, ie, mc, oe, re, ta, op, iv, com_f, be_f, bm_f, ie_f, mc_f, oe_f, re_f, ta_f = get_fin_data()

# Remove Negative Book values
bm[bm<0]=np.nan

# mach all data
ident = mc/mc*bm/bm*op/op*iv/iv

# # ident
mc_i = mc*ident
bm_i = bm*ident
op_i = op*ident
iv_i = iv*ident
```

Total Return

In[30]:

```
tr,er,rf = get_tr_data()

# Cap return to 300% monthly
tr[tr>300] = 300
er[er>300] = 300


# Keep only complete firms "sample"
tr_f = tr.copy()
er_f = er.copy()

tr = tr[be.columns]
er = er[be.columns]

tr_i = tr*ident
er_i = er*ident
```

Marcet cap - monthly

In[31]:

```
mcm = get_mcm_data()
# Only tickers with data
mcm = mcm[be.columns]
# Mach MC and TR
mcm = mcm.shift(periods=1)
mcm = mcm.loc[tr.index]


# ### Reference indicies

# In[32]:


ref_ind = get_ref_ind()


# ## Data due diligence

# ### Reference indicies correlation

# In[33]:


ref_ch = ref_ind.pct_change()*100

all_tic = (tr.columns).to_list()
nor_tic = [s for s in all_tic if ".OL" in s]
swe_tic = [s for s in all_tic if ".ST" in s]
fin_tic = [s for s in all_tic if ".HE" in s]
den_tic = [s for s in all_tic if ".CO" in s]
data_tic = [nor_tic,den_tic,fin_tic,swe_tic]

cor_l = []
for i in np.arange(len(data_tic)):
    cor_df = pd.DataFrame(ref_ch.iloc[:,i])
    cor_df["VW_nor"] = (vw(mcm[data_tic[i]])*tr[data_tic[i]]).sum(axis=1)[1:]
    cor_l.append(cor_df.corr().iloc[1][0])

cor_df = pd.DataFrame(cor_l,index=["Nor","Den","Fin","Swe"],columns=["Ref-Data Cor"]).T


# ## Firm/data samples sorting

# In[34]:


# Green
green_firms =
['AEGA.OL','AFK.OL','ASA.OL','BONHR.OL','CADLR.OL','CARBN.OL','CLOUD.OL','EAM.OL','ELO.OL','HEX.O
L','HYARD.OL','MGN.OL','NEL.OL','RECSI.OL','SCATC.OL','TOM.OL','VOLUE.OL','VOW.OL','AGLX.OL','AND
F.OL','BCS.OL','BWIDL.OL','CAMBI.OL','DSRT.OL','EFUEL.OL','GNP.OL','HPUR.OL','HRGI.OL','HYPRO.OL','H
YN.OL','KYOTO.OL','NSOL.OL','OSUN.OL','OTOVO.OL','PROXI.OL','PRYME.OL','QFUEL.OL','SKAND.OL','ZA
P.OL','AZELIO.ST','HEXI.ST','MINEST.ST','NILAR.ST','PCELL.ST','SAVOS.ST','BIOGAS.ST','SOLT.ST','STRLNG
.ST','ORSTED.CO','VWS.CO','GREENH.CO','GREENM.CO','FORTUM.HE','NESTE.HE','SAGAS.OL','MVWM.OL','
```

```
ABTA.OL','STWA.ST','ALTA.OL','COEN.ST','ELOSSb.ST','OX2SE.ST','MIDSU.ST','HAVH.OL','STWA.ST']

# Half-Green
half_green_firms =
['ELOP.OL','HTRO.ST','KAHOT.OL','HUDL.OL','HDLY.OL','KIT.OL','CRAYN.OL','ATEA.OL','BOUV.OL','ITERA.
OL','KNOW.ST','RIVER.OL','EMGS.OL','PEXIP.OL','DETEC.HE','B3.ST','ANOT.ST','GARO.ST','KNOW.ST','SHAP
E.CO','PREC.ST','SENS.ST','TOBII.ST','NOD.OL','PLT.OL','QFR.OL','STRO.OL','TECH.OL','WSTEP.OL','INDCT.O
L','NCOD.OL','HTRO.ST','MYCR.ST','BALCO.ST','BETCO.ST','GARO.ST','KNOW.ST','TOBII.ST','ANOT.ST','EW
RK.ST','FPIP.ST','HANZA.ST','QLIRO.ST','SEMC.ST','SEZI.ST','TRAD.ST','CHEMM.CO','ENALYZ.CO','IMPERO.
CO','ODICO.CO','SLXIT.CO','ADVBOX.ST','BRIG.ST','CLEMO.ST','DIAH.ST','EKOBOT.ST','POLYG.ST','QAIR.S
T','RENEW.ST','UNIBAP.ST','UPSALE.ST','WPAY.ST','XMR.ST','ZAPLOX.ST','QTCOM.HE','TIETO.HE','BAS1V.
HE','ENENTO.HE','GOFORE.HE','SITOWS.HE','ENEDO.HE','EXL1V.HE','HONBS.HE','SOLTEQ.HE','TLT1V.HE','
ADMCM.HE','BILOT.HE','NXTMH.HE','VINCIT.HE','AEGA.OL','AFK.OL','ASA.OL','BONHR.OL','CADLR.OL','C
ARBN.OL','CLOUD.OL','EAM.OL','ELO.OL','HEX.OL','HYARD.OL','MGN.OL','NEL.OL','RECSI.OL','SCATC.OL','
TOM.OL','VOLUE.OL','VOW.OL','AGLX.OL','ANDF.OL','BCS.OL','BWIDL.OL','CAMBI.OL','DSRT.OL','EFUEL.
OL','GNP.OL','HPUR.OL','HRGI.OL','HYPRO.OL','HYN.OL','KYOTO.OL','NSOL.OL','OSUN.OL','OTOVO.OL','PR
OXI.OL','PRYME.OL','QFUEL.OL','SKAND.OL','ZAP.OL','AZELIO.ST','HEXI.ST','MINEST.ST','NILAR.ST','PCEL
L.ST','SAVOS.ST','BIOGAS.ST','SOLT.ST','STRLNG.ST','ORSTED.CO','VWS.CO','GREENH.CO','GREENM.CO','F
ORTUM.HE','NESTE.HE','SAGAS.OL','MVWM.OL','ABTA.OL','STWA.ST','ALTA.OL','COEN.ST','ELOSSb.ST','O
X2SE.ST','MIDSU.ST','HAVH.OL','STWA.ST']

## Green, g
tr_g = tr[green_firms]
er_g = er[green_firms]
mcm_g = mcm[green_firms]

mc_g = mc_i[green_firms]
bm_g = bm_i[green_firms]
op_g = op_i[green_firms]
iv_g = iv_i[green_firms]

# Half-Green, hg
tr_hg = tr[half_green_firms]
er_hg = er[half_green_firms]
mcm_hg = mcm[half_green_firms]

mc_hg = mc_i[half_green_firms]
bm_hg = bm_i[half_green_firms]
op_hg = op_i[half_green_firms]
iv_hg = iv_i[half_green_firms]


# ## Factor and Portfolio sorting and calculation

# ### Settup Factor Specifications

# In[35]:


# factors 3 - 5
columns_3 = ["SMB, HML","SMB, RMW","SMB, CMA"]
columns_4 = ["SMB, HML, RMW","SMB, HML,CMA","SMB, RMW, CMA"]
columns_5 = ["SMB, HML, RMW, CMA"]
```

```
sort_3 = [[[0.5],[0.5]], [[0.5],[0.3,0.7]],[[0.5],[0.2,0.8]]]
sort_4 = [[[0.5],[0.5],[0.5]], [[0.5],[0.3,0.7],[0.3,0.7]],[[0.5],[0.5],[0.5]], [[0.5],[0.3,0.7],[0.3,0.7]]]
sort_5 = [[[0.5],[0.5],[0.5],[0.5]], [[0.5],[0.3,0.7],[0.3,0.7],[0.3,0.7]],[[0.5],[0.5],[0.5],[0.5]], [[0.5],[0.3,0.7],[0.3,0.7],
[0.3,0.7]],[[0.5],[0.5],[0.5],[0.5]]]

sortn_3 = ["2x2","2x3"]
sortn_4 = ["2x2","2x3","2x2x2","2x3x3"]
sortn_5 = ["2x2","2x3","2x2x2","2x3x3","2x2x2x2"]

name_3 = ["Size","Book"],["Size","PRO"],["Size","INV"]
name_4 = ["Size","Book","PRO"],["Size","Book","INV"],["Size","PRO","INV"]
name_5 = ["Size","Book","PRO","INV"],["Size","Book","PRO","INV"],["Size","Book","PRO","INV"],
["Size","Book","PRO","INV"]

fac_name_3 = ["SMB","HML"],["SMB","RMW"],["SMB","CMA"]
fac_name_4 = ["SMB","HML","RMW"],["SMB","HML","CMA"],["SMB","RMW","CMA"]
fac_name_5 = ["SMB","HML","RMW","CMA"],["SMB, HML, RMW, CMA"],["SMB, HML, RMW, CMA"],["SMB,
HML, RMW, CMA"]

sort_deph_3 = [2,2,2]
sort_deph_4 = [2,2,3,3]
sort_deph_5 = [2,2,3,3,4]

columns_345 = [columns_3, columns_4, columns_5]
sortn_345 = [sortn_3, sortn_4, sortn_5]
name_345 = [name_3,name_4,name_5]
fac_name_345 = [fac_name_3,fac_name_4,fac_name_5]
sorts_345 = [sort_3,sort_4,sort_5]
sort_deph_345 = [sort_deph_3,sort_deph_4,sort_deph_5]


# ### Portfolio settup

# In[36]:


port_all = [mc_i,bm_i,op_i,iv_i] # all firms
port_g = [mc_g,bm_g,op_g,iv_g] # g, green
port_hg = [mc_hg,bm_hg,op_hg,iv_hg] # hg, half-green

# All
port_3_all = [[mc_i,bm_i],[mc_i,op_i],[mc_i,iv_i]]
port_4_all = [[mc_i,bm_i,op_i],[mc_i,bm_i,iv_i],[mc_i,op_i,iv_i]]
port_5_all = [[mc_i,bm_i,op_i,iv_i]]
port_345_all = [port_3_all, port_4_all, port_5_all]

# Green
port_3_g = [[mc_g,bm_g],[mc_g,op_g],[mc_g,iv_g]]
port_4_g = [[mc_g,bm_g,op_g],[mc_g,bm_g,iv_g],[mc_g,op_g,iv_g]]
port_5_g = [[mc_g,bm_g,op_g,iv_g]]
port_345_g = [port_3_g, port_4_g, port_5_g]

# Half Green
port_3_hg = [[mc_hg,bm_hg],[mc_hg,op_hg],[mc_hg,iv_hg]]
```

```
port_4_hg = [[mc_hg,bm_hg,op_hg],[mc_hg,bm_hg,iv_hg],[mc_hg,op_hg,iv_hg]]
port_5_hg = [[mc_hg,bm_hg,op_hg,iv_hg]]
port_345_hg = [port_3_hg, port_4_hg, port_5_hg]


# ### Montly average return per portfolio & number of firms per portfolio per year, 4x4

# In[37]:


# All
p = [0.25,0.5,0.75]
port_ret_4x4_all = port_ret(port = port_all, sort = [p,p,p,p], name = ["Size","Book","PRO","INV"], sort_dim = 2, ret =
er, mcm = mcm)
port_num_4x4_all = port_num(port = port_all, sort = [p,p,p,p], name = ["Size","Book","PRO","INV"], sort_dim = 2)

# Green
p = [0.25,0.5,0.75]
port_ret_4x4_g = port_ret(port = port_g, sort = [p,p,p,p], name = ["Size","Book","PRO","INV"], sort_dim = 2, ret = er,
mcm = mcm)
port_num_4x4_g = port_num(port = port_g, sort = [p,p,p,p], name = ["Size","Book","PRO","INV"], sort_dim = 2)

# Half-Green
p = [0.25,0.5,0.75]
port_ret_4x4_hg = port_ret(port = port_hg, sort = [p,p,p,p], name = ["Size","Book","PRO","INV"], sort_dim = 2, ret =
er, mcm = mcm)
port_num_4x4_hg = port_num(port = port_hg, sort = [p,p,p,p], name = ["Size","Book","PRO","INV"], sort_dim = 2)


# ### Factor Calculation

# In[38]:


# All
   # CAPM
factor_capm = pd.DataFrame(index=er.index)
factor_capm["RM - RF"] = (vw(mcm)*er).sum(axis=1)/100
   # CAPM + SMB
factor_capm_smb = fac_cal(port = [mc_i], sort = [[0.2,0.4,0.6,0.8]], name = ["Size"], sort_dim = 1,fac_name=["SMB"],
ret = er , mcm = mcm)
factor_12_all = [factor_capm,factor_capm_smb]
factor_name_12 =["CAPM","SMB - 5"]
   # 3, 4, 5 - Factor
sum_t  = 0
factor_345_all = []
factor_name_345 =[]
for i in np.arange(len(columns_345)): # num fac
   for j in np.arange(len(columns_345[i])): # num fac comb
      for k in np.arange(len(sortn_345[i])): # num sort
         factor_345_all.append(fac_cal(port = port_345_all[i][j], sort = sorts_345[i][k], name = name_345[i][j],
fac_name = fac_name_345[i][j], sort_dim = sort_deph_345[i][k], ret = er, mcm = mcm))
         factor_name_345.append((columns_345[i][j]+ " - " +sortn_345[i][k]))
   # Combining
```

```
factor_all = factor_12_all + factor_345_all
factor_name = factor_name_12 + factor_name_345

# Green
   # CAPM
factor_capm = pd.DataFrame(index=er_g.index)
factor_capm["RM - RF"] = (vw(mcm_g)*er_g).sum(axis=1)/100
   # CAPM + SMB
factor_capm_smb = fac_cal(port = [mc_g], sort = [[0.2,0.4,0.6,0.8]], name = ["Size"], sort_dim = 1,fac_name=["SMB"],
ret = er_g , mcm = mcm_g)
factor_12_g = [factor_capm,factor_capm_smb]
   # 3, 4, 5 - Factor
sum_t  = 0
factor_345_g = []
for i in np.arange(len(columns_345)): # num fac
    for j in np.arange(len(columns_345[i])): # num fac comb
        for k in np.arange(len(sortn_345[i])): # num sort
            factor_345_g.append(fac_cal(port = port_345_g[i][j], sort = sorts_345[i][k], name = name_345[i][j], fac_name =
fac_name_345[i][j], sort_dim = sort_deph_345[i][k], ret = er_g, mcm = mcm_g))
   # Combining
factor_g = factor_12_g + factor_345_g

# Half Green
   # CAPM
factor_capm = pd.DataFrame(index=er_hg.index)
factor_capm["RM - RF"] = (vw(mcm_hg)*er_hg).sum(axis=1)/100
   # CAPM + SMB
factor_capm_smb = fac_cal(port = [mc_hg], sort = [[0.2,0.4,0.6,0.8]], name = ["Size"], sort_dim = 1,fac_name=
["SMB"], ret = er_hg , mcm = mcm_hg)
factor_12_hg = [factor_capm,factor_capm_smb]
   # 3, 4, 5 - Factor
sum_t  = 0
factor_345_hg = []
for i in np.arange(len(columns_345)): # num fac
    for j in np.arange(len(columns_345[i])): # num fac comb
        for k in np.arange(len(sortn_345[i])): # num sort
            factor_345_hg.append(fac_cal(port = port_345_hg[i][j], sort = sorts_345[i][k], name = name_345[i][j],
fac_name = fac_name_345[i][j], sort_dim = sort_deph_345[i][k], ret = er_hg, mcm = mcm_hg))
   # Combining
factor_hg = factor_12_hg + factor_345_hg


# ## Summary Statistics

# ### Factor Mean, Std, and t-stat

# In[39]:


## ALL
factor_mean_all, factor_std_all, factor_tstat_all = factor_mean_std_t(factor_all,factor_name)

## Green
factor_mean_g, factor_std_g, factor_tstat_g = factor_mean_std_t(factor_g,factor_name)
```

```
## Half Green
factor_mean_hg, factor_std_hg, factor_tstat_hg = factor_mean_std_t(factor_hg,factor_name)


# ### Factor Correlation

# In[40]:


# Between Factors
factor = [factor_all[21],factor_g[21], factor_hg[21]]# Five - Factor, 2x3
factors_name = factor_all[21].columns.to_list()
sample = ['All', 'Green', 'Half Green']

factro_correlation = []
for i in np.arange(len(sample)):
    factor_cor = pd.concat([factor[i].corr()], keys = [sample[i]], names = ["Sample"])
    factro_correlation.append(factor_cor)

# Between Samples
factor = [factor_all[21],factor_g[21], factor_hg[21]] # Five - Factor, 2x3
factors_name = factor_all[21].columns.to_list()
sample = ['All', 'Green', 'Half Green']

factor_sample_correlation = []
for j in np.arange(len(factor_all[21].columns)):
    same_factor = pd.DataFrame(index=factor[0].index)
    for i in np.arange(len(sample)):
        same_factor[sample[i]] =  factor[i][factors_name[j]]
    factor_samp_cor = pd.concat([same_factor.corr()], keys = [factors_name[j]], names = ["Factor"])
    factor_sample_correlation.append(factor_samp_cor)



# ### Factor Spanning

# In[41]:


factor = [copy.deepcopy(factor_all[21]),copy.deepcopy(factor_g[21]), copy.deepcopy(factor_hg[21])] # Five - Factor,
2x3
factors_name = factor_all[21].columns.to_list()
sample = ['All', 'Green', 'Half Green']

factor_span = []
for i in np.arange(len(factor)):
    factor[i]["RM - RF"] = factor[i]["RM - RF"]
    fac_spen = fac_span(factor[i])
    fac_spen.loc[" "] = " "
    fac_spen = pd.concat([fac_spen], keys = [sample[i]], names = ["Factor"])
    factor_span.append(fac_spen)
```

```
# ## Export Data to R

# ### LHS Portfolios

# In[42]:


all_port_ret_4x4 = [port_ret_4x4_all, port_ret_4x4_g, port_ret_4x4_hg]
all_port_name_4x4 = ["port_ret_4x4_all", "port_ret_4x4_g", "port_ret_4x4_hg"]

for i in np.arange(len(all_port_ret_4x4)):
    for j in np.arange(len(all_port_ret_4x4[i])):
        port_ret = all_port_ret_4x4[i][j][((all_port_ret_4x4[i][j].abs()).sum())[((all_port_ret_4x4[i]
[j].abs()).sum()!=0)].index]
        port_ret.to_csv(all_port_name_4x4[i]+"_"+str(j+1)+".csv")


# ### RHS Factors

# In[61]:


all_factors = [factor_all, factor_g, factor_hg]
all_factors_names = ["factor_all", "factor_g", "factor_hg"]

for i in np.arange(len(all_factors)):
    for j in np.arange(len(all_factors[i])):
        all_factors[i][j].to_csv(all_factors_names[i]+"_"+str(j+1)+".csv")


# In[ ]:


# Code 1, Part 1 - END
################################################################################################


# In[78]:


# Run Code 2 in R


# In[ ]:


# Code 1, Part 2 - START
################################################################################################

# ## Inport GRS Results from R

# ### Import GRS f-statisitc and GRS - p-value from CSV

# In[46]:
```

```
# GRS-stat
    # ALl
grs_s_all_4x4 = pd.read_csv("grs_stat_4x4_all.csv",index_col=False).rename(columns={"V1": "BM", "V2": "OP",
"V3": "INV"})
grs_s_all_4x4.index = factor_name
    # GREEN
grs_s_g_4x4 = pd.read_csv("grs_stat_4x4_g.csv",index_col=False).rename(columns={"V1": "BM", "V2": "OP", "V3":
"INV"})
grs_s_g_4x4.index = factor_name
    # HALF GREEN
grs_s_hg_4x4 = pd.read_csv("grs_stat_4x4_hg.csv",index_col=False).rename(columns={"V1": "BM", "V2": "OP",
"V3": "INV"})
grs_s_hg_4x4.index = factor_name

# GRS p-value
    # ALl
grs_p_all_4x4 = pd.read_csv("grs_pval_4x4_all.csv",index_col=False).rename(columns={"V1": "BM", "V2": "OP",
"V3": "INV"})
grs_p_all_4x4.index = factor_name
    # GREEN
grs_p_g_4x4 = pd.read_csv("grs_pval_4x4_g.csv",index_col=False).rename(columns={"V1": "BM", "V2": "OP",
"V3": "INV"})
grs_p_g_4x4.index = factor_name
    # HALF GREEN
grs_p_hg_4x4 = pd.read_csv("grs_pval_4x4_hg.csv",index_col=False).rename(columns={"V1": "BM", "V2": "OP",
"V3": "INV"})
grs_p_hg_4x4.index = factor_name


# ## Asset Pricing Test

# ### Asset Pricing Regression

# In[44]:


# ALL
a_mean_all_4x4, r_squared_all_4x4, ri_squared_all_4x4, std_all_4x4 =
asset_price_test(copy.deepcopy(factor_all),copy.deepcopy(port_ret_4x4_all),["BM","OP","INV"],factor_name)

# GREEN
a_mean_g_4x4, r_squared_g_4x4, ri_squared_g_4x4, std_g_4x4 =
asset_price_test(copy.deepcopy(factor_g),copy.deepcopy(port_ret_4x4_g),["BM","OP","INV"],factor_name)

# HALF - GREEN
a_mean_hg_4x4, r_squared_hg_4x4, ri_squared_hg_4x4, std_hg_4x4 =
asset_price_test(copy.deepcopy(factor_hg),copy.deepcopy(port_ret_4x4_hg),["BM","OP","INV"],factor_name)


# ### Asset Pricing Alphas

# In[45]:
```

```
# All
a_all_4x4, at_all_4x4 = asset_price_test_alpha(copy.deepcopy(factor_all),copy.deepcopy(port_ret_4x4_all),
["BM","OP","INV"],factor_name)

# GREEN
a_g_4x4, at_g_4x4 = asset_price_test_alpha(copy.deepcopy(factor_g),copy.deepcopy(port_ret_4x4_g),
["BM","OP","INV"],factor_name)

# HALF - GREEN
a_hg_4x4, at_hg_4x4 = asset_price_test_alpha(copy.deepcopy(factor_hg),copy.deepcopy(port_ret_4x4_hg),
["BM","OP","INV"],factor_name)


# # Results Output Tables

# ## 1 - Data Cleaning

# In[47]:


raw_data = [com_f, be_f, bm_f, ie_f, mc_f, oe_f, re_f, ta_f]

clean_step_0_s = []
for i in raw_data:
    clean_step_0_s.append(len([ x for x in i.columns.to_list() if "NGM" not in x ]))

clean_step_0 = max(clean_step_0_s)
clean_step_1 = clean_step_0-bm_f[bm_f<0].count().sum()
clean_step_2 = len(mc_i.columns)
clean_step_3 = mc_i.mean().count()

clean_steps = pd.DataFrame(data={"Number of Firms":[clean_step_0,clean_step_1,clean_step_2,clean_step_3]},index=
["Step 0","Step 1", "Step 2", "Step 3"])
clean_steps.to_excel("1 - Number of firms, cleaning data.xlsx")


# ## 2 - Sample size

# ### 2.1 - Firm Count per Year per Sample

# In[48]:


data = [mc_i, mc_g, mc_hg]
firm_count_y =[]
for j in np.arange(len(data)):
    firm_count_s = []
    for i in np.arange(len(mc_i)):
        firm_count_s.append(data[j].iloc[i].count())
    firm_count_y.append(firm_count_s)

sample = ["All","Green","Half Green"]
```

```
years = ["2015","2016","2017","2018","2019","2020","2021"]

firm_count = pd.DataFrame(firm_count_y,index=[sample],columns=[years]).T
firm_count.to_excel("2.1 - Firm count per year per sample.xlsx")


# ### 2.2 - Sample percentage og marcet-cap pre yer

# In[49]:


data = [mc_i,mc_g,mc_hg]
sample = ["All","Green","Half Green"]

mc_samp = []
for i in np.arange(len(data)):
    mc_samp.append(data[i].sum(axis=1))

mc_samples = pd.DataFrame(mc_samp,index=sample).T

for i in np.roll(np.arange(len(mc_samples.columns)),-1):
    mc_samples.iloc[:,i] = mc_samples.iloc[:,i]/mc_samples.iloc[:,0]

mc_samples.to_excel("2.2 - Firm sample, percentage of marcet cap.xlsx")


# ## 3 - Data - Reference Indicies Correlation

# In[50]:


cor_df.to_excel("3 - Reference-index, correlatios.xlsx")


# ## 4 - Average Number of firms per LHS portfolio

# In[51]:


data = [port_num_4x4_all,port_num_4x4_g, port_num_4x4_hg]
sample = ["All", "Green", "Half Green"]

port_num = []
for i in np.arange(len(data)):
    df_bm = pd.DataFrame(np.round(np.array(data[i][0].mean()),2).reshape((4,4)),columns=
["Low","2","3","High"],index=["Small","2","3","Big"])
    df_bm[" "] = " "
    df_bm.loc[" "] = " "
    df_bm = pd.concat([pd.concat([pd.concat([df_bm], keys=["Size"], names=[''])], keys=["B/M"], names=[''],axis=1)],
keys=[sample[i]], names=['"])


    df_op = pd.DataFrame(np.round(np.array(data[i][1].mean()),2).reshape((4,4)),columns=
["Low","2","3","High"],index=["Small","2","3","Big"])
```

```
    df_op[" "] = " "
    df_op.loc[" "] = " "
    df_op = pd.concat([pd.concat([pd.concat([df_op], keys=["Size"], names=[''])], keys=["OP"], names=[''],axis=1)],
keys=[sample[i]], names=[''])


    df_inv = pd.DataFrame(np.round(np.array(data[i][2].mean()),2).reshape((4,4)),columns=
["Low","2","3","High"],index=["Small","2","3","Big"])
    df_inv[" "] = " "
    df_inv.loc[" "] = " "
    df_inv = pd.concat([pd.concat([pd.concat([df_inv], keys=["Size"], names=[''])], keys=["INV"], names=[''],axis=1)],
keys=[sample[i]], names=[''])

    port_num.append(pd.concat([df_bm,df_op,df_inv],axis=1))

for i in np.arange(len(port_num)-1):
    port_num_per_samp = pd.concat([port_num[0],port_num[i+1]])
    port_num[0] = port_num_per_samp

port_num_per_samp.to_excel("4 - Average Number of Firms per Portfolio, 4x4.xlsx")


# ## 5 - Monthly Return and Standard deviation per Portfolio

# ### 5.1 - Monthly Excess Return per Portfolio

# In[52]:


data = [port_ret_4x4_all,port_ret_4x4_g, port_ret_4x4_hg]
sample = ["All", "Green", "Half Green"]

port_ret = []
for i in np.arange(len(data)):
    df_bm = pd.DataFrame(np.round(np.array(data[i][0].mean()),2).reshape((4,4)),columns=
["Low","2","3","High"],index=["Small","2","3","Big"])
    df_bm[" "] = " "
    df_bm.loc[" "] = " "
    df_bm = pd.concat([pd.concat([pd.concat([df_bm], keys=["Size"], names=[''])], keys=["B/M"], names=[''],axis=1)],
keys=[sample[i]], names=[''])

    df_op = pd.DataFrame(np.round(np.array(data[i][1].mean()),2).reshape((4,4)),columns=
["Low","2","3","High"],index=["Small","2","3","Big"])
    df_op[" "] = " "
    df_op.loc[" "] = " "
    df_op = pd.concat([pd.concat([pd.concat([df_op], keys=["Size"], names=[''])], keys=["OP"], names=[''],axis=1)],
keys=[sample[i]], names=[''])

    df_inv = pd.DataFrame(np.round(np.array(data[i][2].mean()),2).reshape((4,4)),columns=
["Low","2","3","High"],index=["Small","2","3","Big"])
    df_inv[" "] = " "
    df_inv.loc[" "] = " "
    df_inv = pd.concat([pd.concat([pd.concat([df_inv], keys=["Size"], names=[''])], keys=["INV"], names=[''],axis=1)],
keys=[sample[i]], names=[''])
```

```
    port_ret.append(pd.concat([df_bm,df_op,df_inv],axis=1))


for i in np.arange(len(port_ret)-1):
    prrt_ret_per_samp = pd.concat([port_ret[0],port_ret[i+1]])
    port_ret[0] = prrt_ret_per_samp

prrt_ret_per_samp.to_excel("5.1 - Average monthly return per portfoleo per sample, 4x4.xlsx")


# ### 5.2 - Standard deviation of returns

# In[53]:


data = [port_ret_4x4_all,port_ret_4x4_g, port_ret_4x4_hg]
sample = ["All", "Green", "Half Green"]

test = []
for i in np.arange(len(data)):
    test_1 = []
    for j in np.arange(len(data[i])):
        test_1.append(data[i][j].std().mean())
    test.append(test_1)

port_std = round(pd.DataFrame(test,columns=["B/M","OP","INV"],index=[sample]),2)
port_std.to_excel("5.2 - Average standard deviation of portfolio return, 4x4.xlsx")


# ## 6 - Factor summary, mean, std, t-stat

# In[54]:


factor_mean = [factor_mean_all, factor_mean_g, factor_mean_hg]
factor_std = [factor_std_all, factor_std_g, factor_std_hg]
factor_tstat = [factor_tstat_all, factor_tstat_g, factor_tstat_hg]


sample = ["All","Green","Half Green"]
model = "SMB, HML, RMW, CMA - 2x3"

factor_summary = []
for i in np.arange(len(sample)):
    mean = pd.DataFrame(np.array(factor_mean[i].loc["SMB, HML, RMW, CMA - 2x3"]), columns=
["Mean"],index=factor_mean_all.columns).T
    std = pd.DataFrame(np.array(factor_std[i].loc["SMB, HML, RMW, CMA - 2x3"]), columns=["Std
Dev"],index=factor_mean_all.columns).T
    tstat = pd.DataFrame(np.array(factor_tstat[i].loc["SMB, HML, RMW, CMA - 2x3"]), columns=["t-
Mean"],index=factor_mean_all.columns).T

    factor_summary.append(pd.concat([pd.concat([mean,std,tstat])],keys=[sample[i]], names=["Sample firms"]))
```

```
for i in np.arange(len(sample)-1):
    factor_summary_per_sample_2x3 = pd.concat([factor_summary[0],factor_summary[i+1]])
    factor_summary[0] = factor_summary_per_sample_2x3

###
factor_summary_per_sample_2x3["RM - RF"] = factor_summary_per_sample_2x3["RM - RF"]*100

factor_summary_per_sample_2x3.to_excel("6 - Factor summary per sample.xlsx")


# ## 7 - Correlation between factors within sample

# In[55]:


factro_correlation_1 = copy.deepcopy(factro_correlation)

for i in np.arange(len(factro_correlation_1)-1):
    factor_factor_correlation_2x3 = pd.concat([factro_correlation_1[0],factro_correlation_1[i+1]])
    factro_correlation_1[0] = factor_factor_correlation_2x3

factor_factor_correlation_2x3.to_excel("7 - Factor correlation between factors.xlsx")


# ## 8 - Correlation between factors between samples

# In[56]:


factor_sample_correlation_1 = copy.deepcopy(factor_sample_correlation)

for i in np.arange(len(factor_sample_correlation_1)-1):
    factor_sample_correlation_2x3 = pd.concat([factor_sample_correlation_1[0],factor_sample_correlation_1[i+1]])
    factor_sample_correlation_1[0] = factor_sample_correlation_2x3

factor_sample_correlation_2x3.to_excel("8 - Factor correlation between samples.xlsx")


# ## 9 - Factor Spanning

# In[58]:


factor_span_1 = copy.deepcopy(factor_span)

for i in np.arange(len(factor_span_1)-1):
    factor_spanning_2x3 = pd.concat([factor_span_1[0],factor_span_1[i+1]])
    factor_span_1[0] = factor_spanning_2x3

factor_spanning_2x3.to_excel("9 - Factor spanning.xlsx")


# ## 10 - Asset Prizing Results
```

```
# In[59]:


grs_s = [grs_s_all_4x4,grs_s_g_4x4,grs_s_hg_4x4]
grs_p = [grs_p_all_4x4,grs_p_g_4x4,grs_p_hg_4x4]
a_mean = [a_mean_all_4x4,a_mean_g_4x4,a_mean_hg_4x4]
ri_squared = [ri_squared_all_4x4,ri_squared_g_4x4,ri_squared_hg_4x4]
std = [std_all_4x4,std_g_4x4,std_hg_4x4]
r_squared = [r_squared_all_4x4,r_squared_g_4x4,r_squared_hg_4x4]

# ind = ["SMB, HML - 2x3","SMB, HML, RMW - 2x3","SMB, HML,CMA - 2x3","SMB, RMW, CMA - 2x3","SMB,
HML, RMW, CMA - 2x3"]
ind = ["CAPM","SMB, HML - 2x3","SMB, HML, RMW - 2x3","SMB, HML,CMA - 2x3","SMB, RMW, CMA -
2x3","SMB, HML, RMW, CMA - 2x3"]

fac_sort = ["2x3"]
sample = ["All","Green","Half Green"]
sort_ports = ["Size - BM","Size - OP","Size - INV"]


# for i in np.arange(len(grs_s)):
res = []
for i in np.arange(len(grs_s)):
    res_1 = []
    for j in np.arange(len(grs_s[0].columns)):
        df = pd.DataFrame([grs_s[i].iloc[:,j],grs_p[i].iloc[:,j],
a_mean[i].iloc[:,j],ri_squared[i].iloc[:,j],std[i].iloc[:,j],r_squared[i].iloc[:,j]],index=
["GRS","GRSp","A|a|","A|a|/Ar","As(a)/Aa","AR2"]).T.loc[ind]
        df.index = [i.split(" - ")[0] for i in df.index.to_list()]
        df.loc[" "] = " "
        df[" "] = " "
        df = pd.concat([pd.concat([df], keys=[sample[i]], names=['Sample firms'])], keys=[sort_ports[j]], names=
['Sorting'],axis=1)
        res_1.append(df)
    df = pd.concat([res_1[0],res_1[1],res_1[2]],axis=1)
    res.append(df)

for i in np.arange(len(res)-1):
    df = pd.concat([res[0],res[i+1]])
    res[0] = df

res_4x4 = copy.deepcopy(df)
res_4x4.to_excel("10 - Asset pricing model results, 3-4-5-facrtor model, 4x4.xlsx")


# ## 11 - Regression intercepts

# In[71]:


sort = ["BM","OP","INV"]

## ALL
```

```
# CAPM
data_a_capm_all = copy.deepcopy(a_all_4x4[0])
data_t_capm_all = copy.deepcopy(at_all_4x4[0])
# FF3
data_a_ff3_all = copy.deepcopy(a_all_4x4[3])
data_t_ff3_all = copy.deepcopy(at_all_4x4[3])
# FF5
data_a_ff5_all = copy.deepcopy(a_all_4x4[21])
data_t_ff5_all = copy.deepcopy(at_all_4x4[21])

data_a_all = [data_a_capm_all,data_a_ff3_all,data_a_ff5_all]
data_t_all = [data_t_capm_all,data_t_ff3_all,data_t_ff5_all]

for j in np.arange(len(data_a_all)):
    for i in np.arange(len(data_a_all[j])):
        data_a_all[j][i][" "] = " "
        data_a_all[j][i].loc[" "] = " "
        data_a_all[j][i] = pd.concat([data_a_all[j][i]],keys = "a", names = " ",axis=1)
        data_t_all[j][i].loc[" "] = " "
        data_t_all[j][i] = pd.concat([data_t_all[j][i]],keys = "t", names = " ",axis=1)

## Half Green
# CAPM
data_a_capm_hg = copy.deepcopy(a_hg_4x4[0])
data_t_capm_hg = copy.deepcopy(at_hg_4x4[0])
# FF3
data_a_ff3_hg = copy.deepcopy(a_hg_4x4[3])
data_t_ff3_hg = copy.deepcopy(at_hg_4x4[3])
# FF5
data_a_ff5_hg = copy.deepcopy(a_hg_4x4[21])
data_t_ff5_hg = copy.deepcopy(at_hg_4x4[21])

data_a_hg = [data_a_capm_hg,data_a_ff3_hg,data_a_ff5_hg]
data_t_hg = [data_t_capm_hg,data_t_ff3_hg,data_t_ff5_hg]

for j in np.arange(len(data_a_hg)):
    for i in np.arange(len(data_a_hg[j])):
        data_a_hg[j][i][" "] = " "
        data_a_hg[j][i].loc[" "] = " "
        data_a_hg[j][i] = pd.concat([data_a_hg[j][i]],keys = "a", names = " ",axis=1)
        data_t_hg[j][i].loc[" "] = " "
        data_t_hg[j][i] = pd.concat([data_t_hg[j][i]],keys = "t", names = " ",axis=1)

# Size - BM
df_a = pd.concat([pd.concat([data_a_capm_all[0]],keys=["All"]),pd.concat([data_a_capm_hg[0]],keys=["Half
Green"])])
df_t = pd.concat([pd.concat([data_t_capm_all[0]],keys=["All"]),pd.concat([data_t_capm_hg[0]],keys=["Half Green"])])
df_bm_capm = pd.concat([pd.concat([df_a,df_t],axis=1)], keys=["CAPM"], names=" ")

df_a = pd.concat([pd.concat([data_a_ff3_all[0]],keys=["All"]),pd.concat([data_a_ff3_hg[0]],keys=["Half Green"])])
df_t = pd.concat([pd.concat([data_t_ff3_all[0]],keys=["All"]),pd.concat([data_t_ff3_hg[0]],keys=["Half Green"])])
df_bm_ff3 = pd.concat([pd.concat([df_a,df_t],axis=1)], keys=["FF-3"], names=" ")

df_a = pd.concat([pd.concat([data_a_ff5_all[0]],keys=["All"]),pd.concat([data_a_ff5_hg[0]],keys=["Half Green"])])
```

```
df_t = pd.concat([pd.concat([data_t_ff5_all[0]],keys=["All"]),pd.concat([data_t_ff5_hg[0]],keys=["Half Green"])])
df_bm_ff5 = pd.concat([pd.concat([df_a,df_t],axis=1)], keys=["FF-5"], names=" ")

df_alpha_bm = pd.concat([pd.concat([df_bm_capm,df_bm_ff3,df_bm_ff5])],keys=["Size - B/M"],names=" ")


# Size - OP
df_a = pd.concat([pd.concat([data_a_capm_all[1]],keys=["All"]),pd.concat([data_a_capm_hg[1]],keys=["Half
Green"])])
df_t = pd.concat([pd.concat([data_t_capm_all[1]],keys=["All"]),pd.concat([data_t_capm_hg[1]],keys=["Half Green"])])
df_op_capm = pd.concat([pd.concat([df_a,df_t],axis=1)], keys=["CAPM"], names=" ")

df_a = pd.concat([pd.concat([data_a_ff3_all[1]],keys=["All"]),pd.concat([data_a_ff3_hg[1]],keys=["Half Green"])])
df_t = pd.concat([pd.concat([data_t_ff3_all[1]],keys=["All"]),pd.concat([data_t_ff3_hg[1]],keys=["Half Green"])])
df_op_ff3 = pd.concat([pd.concat([df_a,df_t],axis=1)], keys=["FF-3"], names=" ")

df_a = pd.concat([pd.concat([data_a_ff5_all[1]],keys=["All"]),pd.concat([data_a_ff5_hg[1]],keys=["Half Green"])])
df_t = pd.concat([pd.concat([data_t_ff5_all[1]],keys=["All"]),pd.concat([data_t_ff5_hg[1]],keys=["Half Green"])])
df_op_ff5 = pd.concat([pd.concat([df_a,df_t],axis=1)], keys=["FF-5"], names=" ")

df_alpha_op = pd.concat([pd.concat([df_op_capm,df_op_ff3,df_op_ff5])],keys=["Size - OP"],names=" ")


# Size - INV
df_a = pd.concat([pd.concat([data_a_capm_all[2]],keys=["All"]),pd.concat([data_a_capm_hg[2]],keys=["Half
Green"])])
df_t = pd.concat([pd.concat([data_t_capm_all[2]],keys=["All"]),pd.concat([data_t_capm_hg[2]],keys=["Half Green"])])
df_inv_capm = pd.concat([pd.concat([df_a,df_t],axis=1)], keys=["CAPM"], names=" ")

df_a = pd.concat([pd.concat([data_a_ff3_all[2]],keys=["All"]),pd.concat([data_a_ff3_hg[2]],keys=["Half Green"])])
df_t = pd.concat([pd.concat([data_t_ff3_all[2]],keys=["All"]),pd.concat([data_t_ff3_hg[2]],keys=["Half Green"])])
df_inv_ff3 = pd.concat([pd.concat([df_a,df_t],axis=1)], keys=["FF-3"], names=" ")

df_a = pd.concat([pd.concat([data_a_ff5_all[2]],keys=["All"]),pd.concat([data_a_ff5_hg[2]],keys=["Half Green"])])
df_t = pd.concat([pd.concat([data_t_ff5_all[2]],keys=["All"]),pd.concat([data_t_ff5_hg[2]],keys=["Half Green"])])
df_inv_ff5 = pd.concat([pd.concat([df_a,df_t],axis=1)], keys=["FF-5"], names=" ")

df_alpha_inv = pd.concat([pd.concat([df_inv_capm,df_inv_ff3,df_inv_ff5])],keys=["Size - INV"],names=" ")

# Export to excel
df_alpha_bm.to_excel("11.1 - Alphas, Size-BM.xlsx")
df_alpha_op.to_excel("11.2 - Alphas, Size-OP.xlsx")
df_alpha_inv.to_excel("11.3 - Alphas, Size-INV.xlsx")


# In[79]:


# Code 1, Part 2 - END
################################################################################
```