ValueWeighted

November 12, 2018

1 Portfolio Management Project

1.1 Objective:

In this project, we aim to build stylized long-short equity factor mimicking portfolios using different fundamental variables from Ken French's data library and explore empirically their univariate efficacy over time and across different size segments. We then go on to build multi-factor strategies using alternative weighting schemes and compare them to the static equally weighted multi-factor strategy. Two alternative top-down factor weighting schemes will be considered: 1) Equal risk contribution across factors 2) Weighting based on factor persistence

1.2 Data:

Attached with the project description are 6 csv files containing the monthly time series of value- and equal-weighted returns for portfolios formed on size and different fundamental variables consisting of book-to-price, cashflow-to-price, dividend yield, investment, profitability, prior 1-month return and 12-1 price momentum. In addition, there is a csv file named "F-F_Research_Data_Factors" which houses the Fama-French 3 factor model returns.

1.3 Project description:

1.For each of the six fundamental variables, construct long-short factor mimicking portfolios and plot their historical performance across different size segments. Taking the market return from Fama-French's 3-factor model, calculate and plot the rolling 3-year market beta for these stylized portfolios. Considering both size segments, construct a beta-neutral factor mimicking portfolio for each fundamental variable. Comment on your results.

2.Calculate the full sample correlation matrix of unadjusted factor returns (i.e. not the beta-neutral version) derived from 1. Comment on your findings. Using a lookback period of 5 years, employ an equal risk contribution factor weighting strategy with monthly rebalancing. The monthly resultant portfolios should be dollar neutral with a long leg exposure of 100%. Plot the monthly factor weights over time and evaluate the strategy performance against the static equally weighted factor portfolio. Comment on your results.

3.Using different lookback periods of 1,12 and 36 months to determine factor persistence, build adaptive multi-factor models that appropriately reflect your view on each factor. For example, you may want to consider a factor weighting approach such that the factor allocation is proportional to the historical Sharpe ratio for a given lookback period. Comment on your results.

1.3.1 Data Source:

Fama-Frence Libary

1.3.2 Factors:

BP: book value / stock price
CFP: cash flow / stock price
DP: dividend / stock price
INV: investment value
MOMENTUM_PRIOR_1: short-term reversal, return for prior 1 month
MOMENTUM_PRIOR_12_2: short-term momentum, return for prior 12 month to prior 2 month

1.4 Code

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from scipy.optimize import linprog
from scipy.optimize import minimize
import pymprog
from statsmodels import regression
import statsmodels.api as sm
import statsmodels.formula.api as smf


def long_short(filename,buy_high=True):
    data = pd.read_csv(filename)
    data['Month'] = pd.to_datetime(data['Month'],format ='%Y%m')
    data = data.set_index('Month')

    if buy_high:
        data['SMALL'] = (data.iloc[:,2] - data.iloc[:,0])
        data['BIG'] = (data.iloc[:,5] - data.iloc[:,3])

    else:
        data['SMALL'] = (-data.iloc[:,2] + data.iloc[:,0])
        data['BIG'] = (-data.iloc[:,5] + data.iloc[:,3])

    return data


def beta(data,market_data):
    data['MARKET'] = market_data['MARKET']
    column_beta = [data.columns[i] for i in [0,2,3,5,6,7]]
```

```
    a=[]
    for i in column_beta:
        a.append(i+'_BETA')
        data[i+'_BETA'] = pd.rolling_cov(data[i],data['MARKET'],window=36)/pd.rolling_v

    for i in column_beta[:4]:
        data[i+'_ALPHA'] = data[i] -data[i+'_BETA']* data['MARKET']

    return data.dropna(),a[:4]

def beta_neutral_2(data,a,buy_high=True):

    b = [j[:-5]+'_ALPHA' for j in a]
    c = [j[:-5] for j in a]
    performance=[]
    for i in range(len(data)-1):
        beta1,beta2,beta3,beta4= data.iloc[i][a]
        alpha1,alpha2,alpha3,alpha4 = data.iloc[i][b]
        r1,r2,r3,r4 = data.iloc[i+1][c]

        target = np.array([alpha1,alpha2,alpha3,alpha4])

        if buy_high:

            results = linprog(-target,
                              A_eq=[[beta1,beta2,beta3,beta4],[1,0,1,0],[0,1,0,1]],
                              b_eq=[0,-1,1],
                              bounds=((-1,0),(0,1),(-1,0),(0,1)))

            if not 'successfully' in results.message:
                cons = ({'type': 'eq', 'fun': lambda p:beta1*p[0]+beta2*p[1]+beta3*p[2]
                        {'type': 'eq', 'fun': lambda p:p[1]+p[3]-1},
                        {'type': 'eq', 'fun': lambda p:p[0]+p[2]+1},
                        {'type': 'ineq', 'fun': lambda p:p[0]+1},
                        {'type': 'ineq', 'fun': lambda p:p[2]+1},
                        {'type': 'ineq', 'fun': lambda p:1-p[1]},
                        {'type': 'ineq', 'fun': lambda p:1-p[3]})

                results = minimize(fun=alpha_sum,
                                   x0=[-0.5,0.5,-0.5,0.5],
                                   method='SLSQP',
                                   constraints=cons,
                                   args=(alpha1,alpha2,alpha3,alpha4))



        else:
            results = linprog(-target,
```

```
                            A_eq=[[beta1,beta2,beta3,beta4],[1,0,1,0],[0,1,0,1]],
                            b_eq=[0,1,-1],
                            bounds=((0,1),(-1,0),(0,1),(-1,0)))
            if not 'successfully' in results.message:
                cons = ({'type': 'eq', 'fun': lambda p:beta1*p[0]+beta2*p[1]+beta3*p[2]
                        {'type': 'eq', 'fun': lambda p:p[0]+p[2]-1},
                        {'type': 'eq', 'fun': lambda p:p[1]+p[3]+1},
                        {'type': 'ineq', 'fun': lambda p:p[1]+1},
                        {'type': 'ineq', 'fun': lambda p:p[3]+1},
                        {'type': 'ineq', 'fun': lambda p:1-p[0]},
                        {'type': 'ineq', 'fun': lambda p:1-p[2]})


                results = minimize(fun=alpha_sum,
                                   x0=[0.5,-0.5,0.5,-0.5],
                                   method='SLSQP',
                                   constraints=cons,
                                   args=(alpha1,alpha2,alpha3,alpha4))



        rho1,rho2,rho3,rho4 = list(results.x)

        performance.append(r1*rho1+r2*rho2+r3*rho3+r4*rho4)

    output=pd.DataFrame()
    output['Beta_Neutral_Performance'] = pd.Series(performance)
    output.index = data.index[1:]

    return output


def alpha_sum(p,alpha1,alpha2,alpha3,alpha4):
    return -(alpha1*p[0]+alpha2*p[1]+alpha3*p[2]+alpha4*p[3])


def erc(df,look_back):
    origin_columns = list(df.columns)

    cov_name = []
    data=pd.DataFrame()
    data[origin_columns] = df[origin_columns]
    for a in origin_columns:
        for b in origin_columns:
            data[a+'_'+b+'_cov'] = df[a].rolling(window=look_back).cov(df[b])
            cov_name.append(a+'_'+b+'_cov')

    data = data.dropna()
```

```
erc_df = data[origin_columns]

x1=[]
x2=[]
x3=[]
x4=[]
x5=[]
x6=[]
x7=[]


for i in range(len(data)):
    cov_array = np.array(data.iloc[i][cov_name])
    cons = ({'type': 'eq', 'fun': lambda x:x[0]+x[1]+x[2]+x[3]+x[4]+x[5]+x[6]-1})
    results = minimize(fun=erc_target, x0=[0,0,0,0,0,0,0],  method='SLSQP',
                        constraints=cons,args=cov_array,bounds = ((0,1),(0,1),(0,1)

    x1.append(list(results.x)[0])
    x2.append(list(results.x)[1])
    x3.append(list(results.x)[2])
    x4.append(list(results.x)[3])
    x5.append(list(results.x)[4])
    x6.append(list(results.x)[5])
    x7.append(list(results.x)[6])


weights_columns = [f+'_weight' for f in origin_columns]
x_all = [x1,x2,x3,x4,x5,x6,x7]
for num in range(7):
    weights_columns = [f+'_weight' for f in origin_columns]

    erc_df[weights_columns[num]] = pd.Series(x_all[num],index=erc_df.index)

simple_list = []
erc_list=[]

for i in range(len(erc_df)-1):
    simple_list.append(sum(erc_df.iloc[i+1][origin_columns])/7)

    w1 = x1[i]
    w2 = x2[i]
    w3 = x3[i]
    w4 = x4[i]
    w5 = x5[i]
    w6 = x6[i]
    w7 = x7[i]

    r1,r2,r3,r4,r5,r6,r7 = erc_df.iloc[i+1][origin_columns]
```

```
        erc_list.append(w1*r1+w2*r2+w3*r3+w4*r4+w5*r5+w6*r6+w7*r7)

    return erc_df,simple_list,erc_list


def erc_target(x,cov_array):
    x_array = x.reshape(7,1)
    #cov_array = cov_array*(10**14)
    cov_matrix = cov_array.reshape(7,7)

    total_risk = np.dot(cov_matrix,x_array)

    x2=[]
    for i in range(7):
        x2.append((total_risk[i][0])*x_array[i][0])

    '''
    diff_sum=0
    for i in range(7):
        for j in range(7):
            diff_sum += (x[i]-x[j])**2


    return diff_sum'''

    var_sum = np.var(np.array(x2))
    return var_sum*(10**14)

def total_risk(x,cov_array):
    x_array = x.reshape(7,1)
    #cov_array = cov_array*(10**14)
    cov_matrix = cov_array.reshape(7,7)

    total_risk = np.dot(cov_matrix,x_array)

    x2=[]
    for i in range(7):
        x2.append((total_risk[i][0])*x_array[i][0])

    '''
    diff_sum=0
    for i in range(7):
        for j in range(7):
            diff_sum += (x[i]-x[j])**2


    return diff_sum'''
```

```
        return x2
```

```
In [2]: def sharpe_ratio_equal(df,market_data,look_back,long_only=False):
            origin_columns = list(df.columns)
            simple_list=[]

            data=pd.DataFrame()
            data[origin_columns] = df[origin_columns]

            data['RF'] =  market_data['RF']
            sr_columns = [factor+'_SR' for factor in origin_columns]
            for factor in origin_columns:
                data[factor+'_RF'] = data[factor]-data['RF']
                data[factor+'_SR'] = data[factor+'_RF'].rolling(window=look_back).mean()\
                /data[factor].rolling(window=look_back).std()

            data=data.dropna()

            performance = []
            for i in range(len(data)-1):
                if not long_only:
                    sr_list = list(data.iloc[i][sr_columns])
                else:
                    sr_list = [i if i>0  else 0  for i in data.iloc[i][sr_columns]]

                sr_sum = np.array(sr_list).sum()

                simple_list.append(sum(data.iloc[i+1][origin_columns])/7)
                if sr_sum == 0:
                    performance.append(0)

                else:
                    w1,w2,w3,w4,w5,w6,w7= np.array(sr_list)/sr_sum
                    r1,r2,r3,r4,r5,r6,r7 = data.iloc[i+1][origin_columns]

                    performance.append(w1*r1+w2*r2+w3*r3+w4*r4+w5*r5+w6*r6+w7*r7)



            return data,performance,simple_list


        def average_mean_equal(df,market_data,look_back,long_only=False):
            origin_columns = list(df.columns)
            simple_list=[]

            data=pd.DataFrame()
```

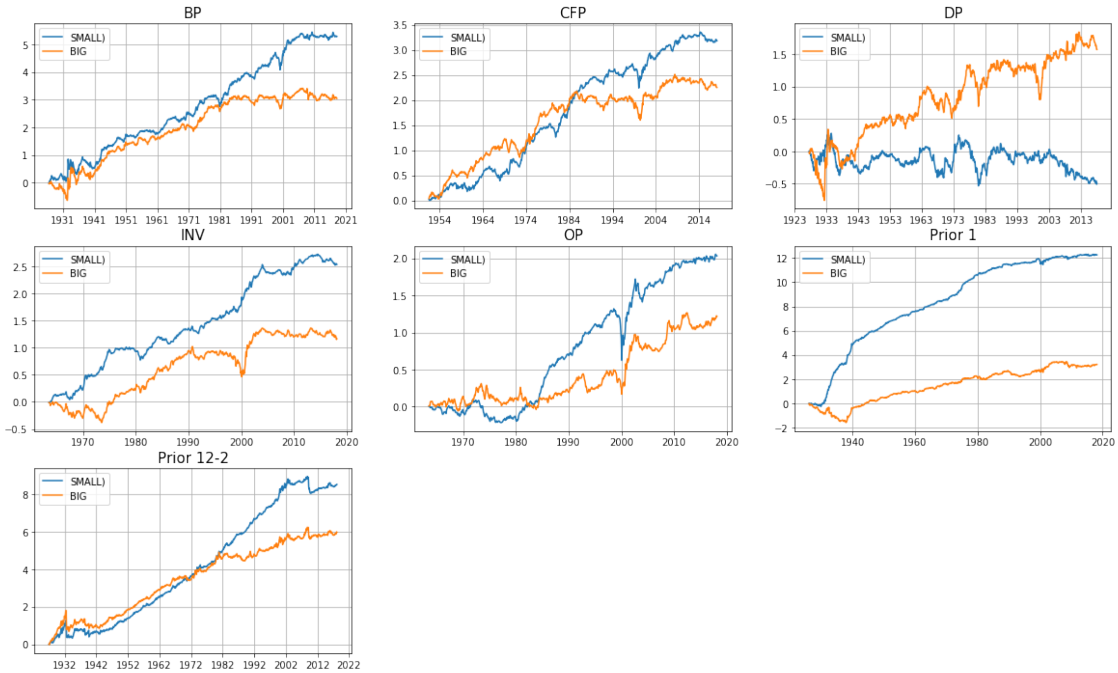
```
    data[origin_columns] = df[origin_columns]

    data['RF'] =  market_data['RF']
    sr_columns = [factor+'_MEAN' for factor in origin_columns]

    for factor in origin_columns:
        data[factor+'_RF'] = data[factor]-data['RF']
        data[factor+'_MEAN'] = data[factor].rolling(window=look_back).mean()

    data=data.dropna()

    performance = []
    for i in range(len(data)-1):
        if not long_only:
            sr_list = list(data.iloc[i][sr_columns])
        else:
            sr_list = [i if i>0  else 0  for i in data.iloc[i][sr_columns]]

        sr_sum = np.array(sr_list).sum()

        if sr_sum == 0 :
            performance.append(0)
        else:
            w1,w2,w3,w4,w5,w6,w7= np.array(sr_list)/sr_sum
            r1,r2,r3,r4,r5,r6,r7 = data.iloc[i+1][origin_columns]

            performance.append(w1*r1+w2*r2+w3*r3+w4*r4+w5*r5+w6*r6+w7*r7)

    return performance


def information_ratio_equal(df,market_data,look_back,long_only=False):
    origin_columns = list(df.columns)
    simple_list=[]

    data=pd.DataFrame()
    data[origin_columns] = df[origin_columns]

    data['MARKET'] =  market_data['MARKET']
    sr_columns = [factor+'_IR' for factor in origin_columns]
    for factor in origin_columns:
        data[factor+'_M'] = data[factor]-data['MARKET']
        data[factor+'_IR'] = data[factor+'_M'].rolling(window=look_back).mean()\
        /data[factor+'_M'].rolling(window=look_back).std()

    data=data.dropna()

    performance = []
```

```
    for i in range(len(data)-1):
        if not long_only:
            sr_list = list(data.iloc[i][sr_columns])
        else:
            sr_list = [i if i>0  else 0  for i in data.iloc[i][sr_columns]]

        sr_sum = np.array(sr_list).sum()

        if sr_sum == 0:
            performance.append(0)

        else:

            w1,w2,w3,w4,w5,w6,w7= np.array(sr_list)/sr_sum
            r1,r2,r3,r4,r5,r6,r7 = data.iloc[i+1][origin_columns]

            performance.append(w1*r1+w2*r2+w3*r3+w4*r4+w5*r5+w6*r6+w7*r7)

    return  performance
```

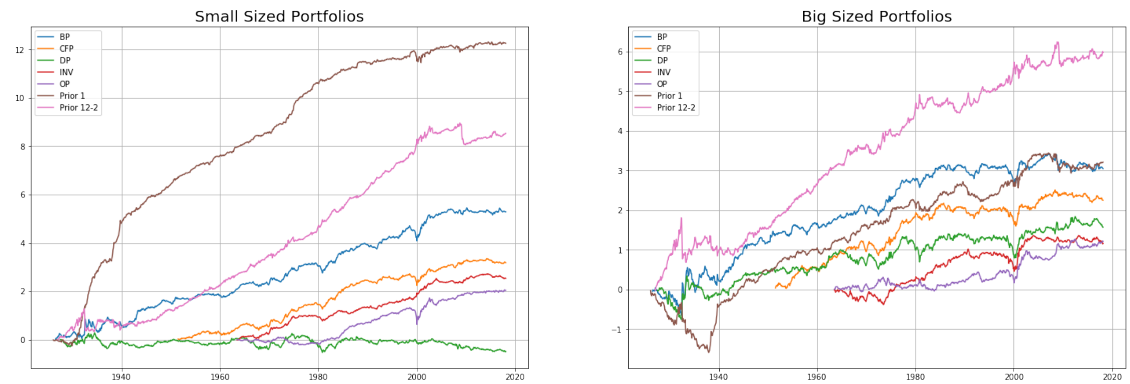
1.5 1. Performances for Small and Big Risk Premia Factors (Value-Weighted Factors

In [3]:
```
data1 = long_short('D:/PortfolioManagement/VW/BP_VW.csv')
data2 = long_short('D:/PortfolioManagement/VW/CFP_VW.csv')
data3 = long_short('D:/PortfolioManagement/VW/DP_VW.csv')
data4 = long_short('D:/PortfolioManagement/VW/INV_VW.csv',False)
data5 = long_short('D:/PortfolioManagement/VW/OP_VW.csv')
data6 = long_short('D:/PortfolioManagement/VW/MOMENTUM_PRIOR_1_VW.csv',False)
data7 = long_short('D:/PortfolioManagement/VW/MOMENTUM_PRIOR_12_2_VW.csv')



data_list = [data1,data2,data3,data4,data5,data6,data7]
factor_list = ['BP','CFP','DP','INV','OP','Prior 1','Prior 12-2']

plt.figure(figsize=(20,12))

for i in range(7):
    data = data_list[i]
    plt.subplot(3,3,i+1)
    plt.plot(data.index,data['SMALL'].cumsum()/100)
    plt.plot(data.index,data['BIG'].cumsum()/100)
    plt.legend(['SMALL)','BIG'])
    plt.title(factor_list[i],fontsize = 15)
```

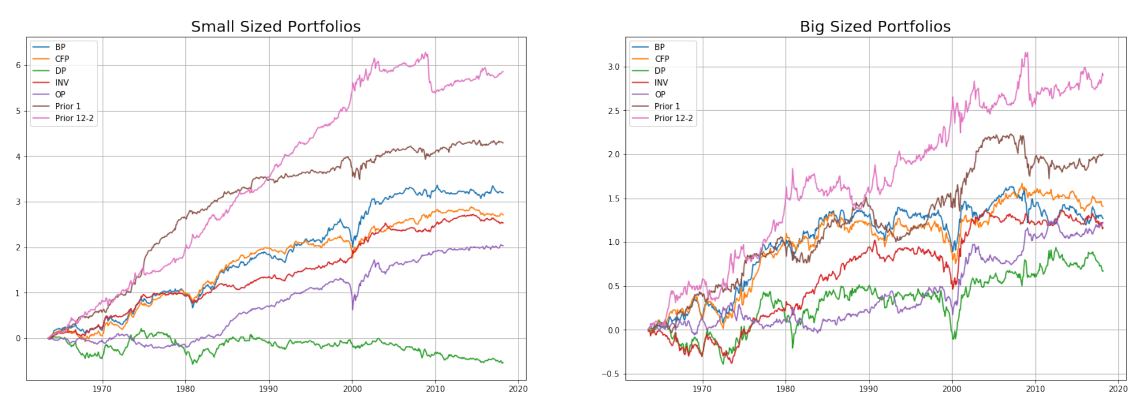
```
        plt.grid(True)

    plt.show()
```




```
In [4]: plt.figure(figsize=(25,8))
        plt.subplot(1,2,1)
        for i in range(7):
            data = data_list[i]
            plt.plot(data.index,data['SMALL'].cumsum()/100)
        plt.legend(factor_list)
        plt.title('Small Sized Portfolios',fontsize=20)
        plt.grid(True)

        plt.subplot(1,2,2)
        for i in range(7):
            data = data_list[i]
            plt.plot(data.index,data['BIG'].cumsum()/100)
        plt.legend(factor_list)
        plt.title('Big Sized Portfolios',fontsize=20)
        plt.grid(True)

        plt.show()
```



In [5]:
```
index_min = data4.index
plt.figure(figsize=(25,8))
plt.subplot(1,2,1)
for i in range(7):
    data = data_list[i].loc[index_min,:]
    plt.plot(data.index,data['SMALL'].cumsum()/100)
plt.legend(factor_list)
plt.grid(True)
plt.title('Small Sized Portfolios',fontsize=20)

plt.subplot(1,2,2)
for i in range(7):
    data = data_list[i].loc[index_min,:]
    plt.plot(data.index,data['BIG'].cumsum()/100)
plt.legend(factor_list)
plt.grid(True)
plt.title('Big Sized Portfolios',fontsize=20)

plt.show()
```



1.6 2. Statistics for Small and Big Risk Premia Factors (Value-Weighted Factors

In [6]:
```
big = pd.DataFrame()
small = pd.DataFrame()
market_data = pd.read_csv('D:/PortfolioManagement/F-F.csv')
market_data['MARKET'] = market_data['Mkt-RF'] + market_data['RF']
market_data['Month'] = pd.to_datetime(market_data['Month'],format ='%Y%m')
market_data = market_data.set_index('Month')

import numpy as np

index_min_1 = data4.index

for i in range(7):
    data=data_list[i].loc[index_min_1,:]
    data['RF'] =  market_data['RF']
    data['MARKET'] = market_data['MARKET']
    data['RM_RF'] = market_data['Mkt-RF']
    data['SMB'] = market_data['SMB']
    data['HML'] = market_data['HML']
    data['BIG_RF'] = data['BIG'] - data['RF']
    data['SMALL_RF'] = data['SMALL'] - data['RF']

    SMB = np.array(data['SMB'])
    HML = np.array(data['HML'])
    RM_RF = np.array(data['RM_RF'])

    X = np.column_stack((RM_RF,SMB,HML))

    Y_BIG = np.array(data['BIG_RF'] ).T
    Y_SMALL = np.array(data['SMALL_RF'] ).T

    X = sm.add_constant(X)
    MODEL_BIG = regression.linear_model.OLS(Y_BIG, X).fit()
    MODEL_SMALL = regression.linear_model.OLS(Y_SMALL, X).fit()

    data_big = data['BIG']
    data_small = data['SMALL']
    data_big_2 = data['BIG']-data['RF']
    data_small_2 = data['SMALL']-data['RF']
    data_big_3 = data['BIG']-data['MARKET']
    data_small_3 = data['SMALL']-data['MARKET']

    factor = factor_list[i]

    rm_rf_mean = RM_RF.mean()
    rf_mean = data['RF'].mean()
```

```
    big.loc['Average Returns',factor] = data_big.mean()
    big.loc['Std',factor] = np.std(data_big)
    big.loc['Sharpe Ratio',factor] = data_big_2.mean()/np.std(data_big)
    big.loc['Alpha',factor] = MODEL_BIG.params[0]
    big.loc['Beta',factor] = MODEL_BIG.params[1]
    big.loc['Treynor Ratio',factor] = data_big_2.mean()/MODEL_BIG.params[1]
    big.loc['Jensen Measure',factor] = data_big.mean()-rf_mean-MODEL_BIG.params[1]*rm_
    big.loc['Information Ratio',factor] = data_big_3.mean()/np.std(data_big_3)

    small.loc['Average Returns',factor] = data_small.mean()
    small.loc['Std',factor] = np.std(data_small)
    small.loc['Sharpe Ratio',factor] = data_small_2.mean()/np.std(data_small)
    small.loc['Alpha',factor] = MODEL_SMALL.params[0]
    small.loc['Beta',factor] = MODEL_SMALL.params[1]
    small.loc['Treynor Ratio',factor] = data_small_2.mean()/MODEL_SMALL.params[1]
    small.loc['Jensen Measure',factor] = data_small.mean()-rf_mean-MODEL_SMALL.params[
    small.loc['Information Ratio',factor] = data_small_3.mean()/np.std(data_small_3)
```

In [7]: small

Out[7]:

```
                         BP       CFP        DP       INV         OP  \
Average Returns    0.487443  0.412557 -0.081096  0.385586   0.309592
Std                3.196657  2.640228  2.741534  1.975922   2.640304
Sharpe Ratio       0.032011  0.010394 -0.170054  0.000239  -0.028604
Alpha             -0.136975 -0.124430 -0.411403 -0.080154  -0.060452
Beta              -0.091394 -0.073078 -0.250515 -0.096941   0.005921
Treynor Ratio     -1.119644 -0.375531  1.861006 -0.004863 -12.755923
Jensen Measure     0.150577  0.066022 -0.333959  0.051648  -0.078648
Information Ratio -0.067197 -0.085294 -0.156221 -0.098803  -0.111334

                    Prior 1  Prior 12-2
Average Returns    0.653470    0.891446
Std                3.125325    4.159649
Sharpe Ratio       0.085865    0.121725
Alpha              0.099987    0.713121
Beta               0.211523   -0.172492
Treynor Ratio      1.268686   -2.935389
Jensen Measure     0.156690    0.597393
Information Ratio -0.057061   -0.003369
```

In [8]: big

Out[8]:

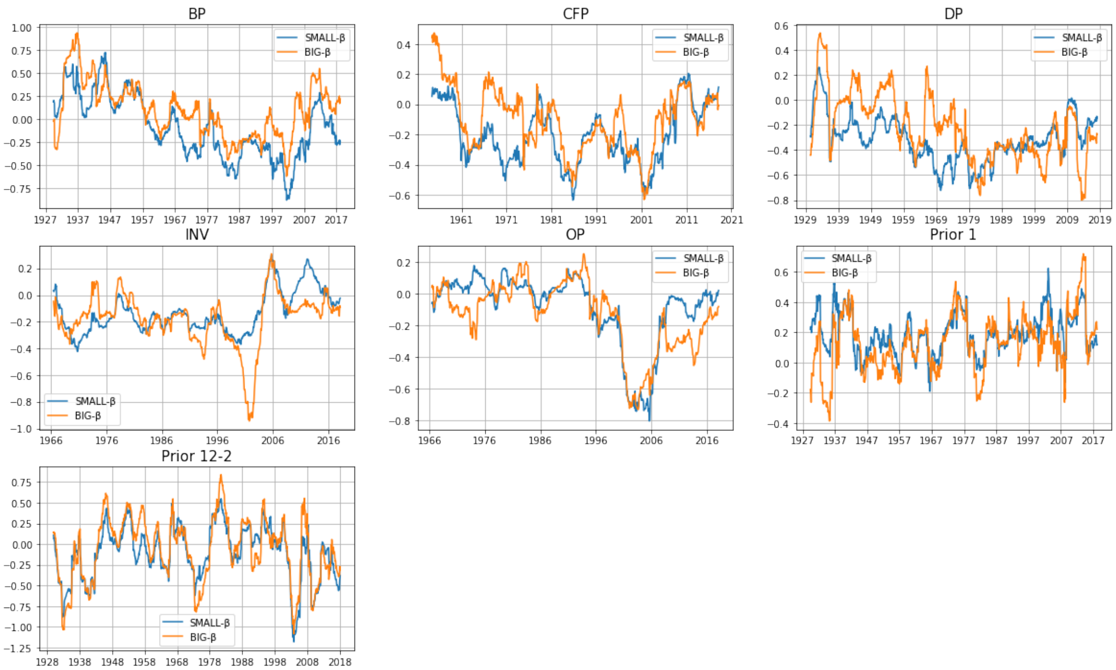
```
                         BP       CFP        DP       INV        OP   Prior 1  \
Average Returns    0.192981  0.214094  0.101796  0.176987  0.185499  0.304414
Std                3.090911  3.029134  3.501705  2.694617  2.489473  3.543958
Sharpe Ratio      -0.062161 -0.056458 -0.080909 -0.077238 -0.080184 -0.022771
Alpha             -0.634544 -0.426072 -0.402126 -0.337063 -0.008821 -0.234937
Beta               0.099215 -0.038761 -0.222807 -0.102530 -0.132317  0.191279
Treynor Ratio     -1.936537  4.412174  1.271586  2.029920  1.508611 -0.421898
```

```
Jensen Measure    -0.244510 -0.150557 -0.165695 -0.154000 -0.129763 -0.181679
Information Ratio -0.130737 -0.119420 -0.121244 -0.126006 -0.130788 -0.123314

                   Prior 12-2
Average Returns      0.440959
Std                  4.642842
Sharpe Ratio         0.012028
Alpha                0.277264
Beta                -0.191074
Treynor Ratio       -0.292267
Jensen Measure       0.156716
Information Ratio   -0.070061
```

1.7 3. Beta of SMALL and BIG for Market Model

In [9]:
```
market_data = pd.read_csv('D:/PortfolioManagement/F-F.csv')
market_data['MARKET'] = market_data['Mkt-RF'] + market_data['RF']
market_data['Month'] = pd.to_datetime(market_data['Month'],format ='%Y%m')
market_data = market_data.set_index('Month')

data1,a1 = beta(data1,market_data)
data2,a2 = beta(data2,market_data)
data3,a3 = beta(data3,market_data)
data4,a4 = beta(data4,market_data)
data5,a5 = beta(data5,market_data)
data6,a6 = beta(data6,market_data)
data7,a7 = beta(data7,market_data)

a_list = [a1,a2,a3,a4,a5,a6,a7]
data_list=[data1,data2,data3,data4,data5,data6,data7]

plt.figure(figsize=(20,12))

for i in range(7):
    data = data_list[i]
    plt.subplot(3,3,i+1)
    plt.plot(data.index,data['SMALL_BETA'])
    plt.plot(data.index,data['BIG_BETA'])
    plt.legend(['SMALL-','BIG-'])
    plt.title(factor_list[i],fontsize = 15)
    plt.grid(True)

plt.show()
```

```
E:\Anaconda3\lib\site-packages\ipykernel_launcher.py:36: FutureWarning: pd.rolling_cov is depre
        Series.rolling(window=36).cov(other=<Series>)
E:\Anaconda3\lib\site-packages\ipykernel_launcher.py:36: FutureWarning: pd.rolling_var is depre
        Series.rolling(window=36,center=False).var()
```



1.8 4. Performances for Beta-Neutral Portfolios of 7 Risk Premia Factors

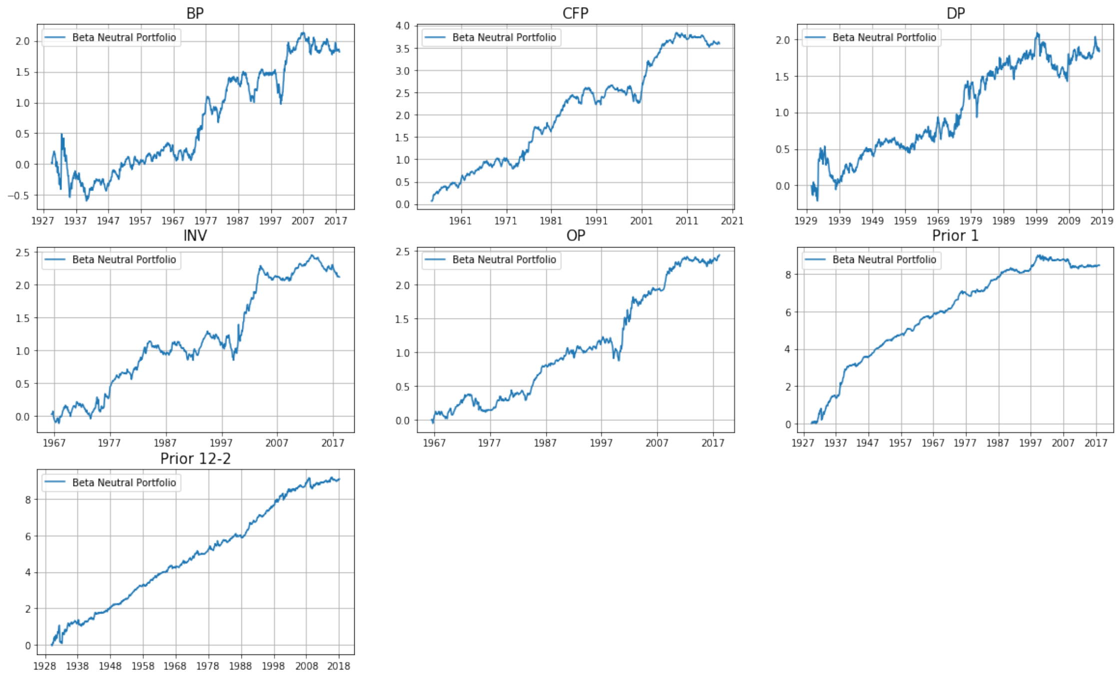
1.8.1 Beta-Neutral Portfolios are based on rolling dynamic Betas.

1.8.2 Holding a beta-neutral portfolio implies the investor can get an alpha without being exposed to any systematic risk.

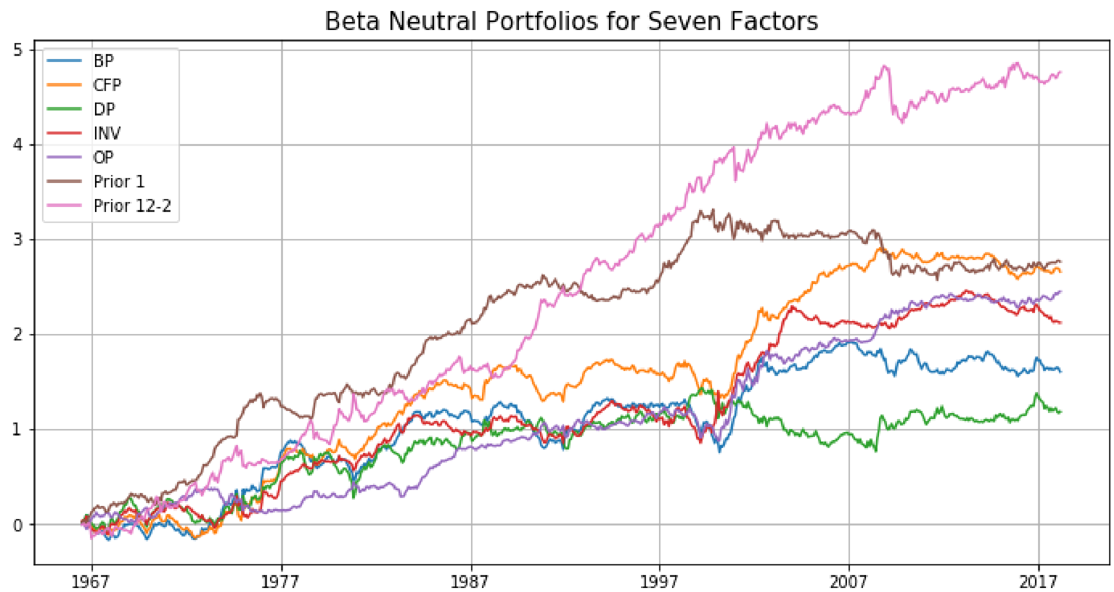
In [10]:
```
output1 = beta_neutral_2(data1,a1)
output2= beta_neutral_2(data2,a2)
output3 = beta_neutral_2(data3,a3)
output4 = beta_neutral_2(data4,a4,False)
output5 = beta_neutral_2(data5,a5)
output6 = beta_neutral_2(data6,a6,False)
output7 = beta_neutral_2(data7,a7)

data_list = [data1,data2,data3,data4,data5,data6,data7]
output_list = [output1,output2,output3,output4,output5,output6,output7]
```

In [11]:
```
plt.figure(figsize=(20,12))

for i in range(7):
    output=output_list[i]
    plt.subplot(3,3,i+1)
    plt.plot(output.index,output['Beta_Neutral_Performance'].cumsum()/100)
    plt.legend(['Beta Neutral Portfolio'])
```

```
        plt.title(factor_list[i],fontsize = 15)
        plt.grid(True)

    plt.show()
```




```
In [12]: plt.figure(figsize=(12,6))
         index_min_2 = data4.index[1:]
         for i in range(7):
             output = output_list[i].loc[index_min_2,:]
             plt.plot(output.index,output['Beta_Neutral_Performance'].cumsum()/100)
             plt.legend(factor_list)
             plt.title('Beta Neutral Portfolios for Seven Factors',fontsize = 15)
         plt.grid(True)
         plt.show()
```




```
In [13]: output_2 = pd.DataFrame()


         import numpy as np

         for i in range(7):
             data = output_list[i].loc[index_min_2,:]
             data['RF'] =  market_data['RF']
             data['MARKET'] = market_data['MARKET']
             data['RM_RF'] = market_data['Mkt-RF']
             data['SMB'] = market_data['SMB']
             data['HML'] = market_data['HML']
             data['BIG_RF'] = data['Beta_Neutral_Performance'] - data['RF']

             SMB = np.array(data['SMB'])
             HML = np.array(data['HML'])
             RM_RF = np.array(data['RM_RF'])

             X = np.column_stack((RM_RF,SMB,HML))

             Y = np.array(data['Beta_Neutral_Performance'] ).T
             X = sm.add_constant(X)
             MODEL = regression.linear_model.OLS(Y, X).fit()

             data_p = data['Beta_Neutral_Performance']
             data_p_2 = data['Beta_Neutral_Performance']-data['RF']
             data_p_3 = data['Beta_Neutral_Performance']-data['MARKET']
```

```
            factor = factor_list[i]

            rm_rf_mean = RM_RF.mean()
            rf_mean = data['RF'].mean()

            output_2.loc['Average Returns',factor] = data_p.mean()
            output_2.loc['Std',factor] = np.std(data_p)
            output_2.loc['Sharpe Ratio',factor] = data_p_2.mean()/np.std(data_p)
            output_2.loc['Alpha',factor] = MODEL.params[0]
            output_2.loc['Beta',factor] = MODEL.params[1]
            output_2.loc['Treynor Ratio',factor] = data_p_2.mean()/MODEL.params[1]
            output_2.loc['Jensen Measure',factor] = data_p.mean()-rf_mean-MODEL.params[1]*rm_r
            output_2.loc['Information Ratio',factor] = data_p_3.mean()/np.std(data_p_3)
```

In [14]:
```
output_2
```

Out[14]:
```
                          BP       CFP        DP       INV        OP   Prior 1  \
Average Returns     0.257475  0.427392  0.189936  0.340926  0.393490  0.444016
Std                 3.330820  3.244216  3.791230  3.224033  2.812513  3.529391
Sharpe Ratio       -0.039609  0.011709 -0.052613 -0.015037  0.001453  0.015474
Alpha              -0.114322  0.174560  0.046123  0.198108  0.459893  0.474572
Beta               -0.040876 -0.124881 -0.118699 -0.179811 -0.087021  0.162911
Treynor Ratio       3.227528 -0.304192  1.680458  0.269606 -0.046957  0.335226
Jensen Measure     -0.110437  0.103650 -0.137057  0.046066  0.049841 -0.031046
Information Ratio -0.110182 -0.080274 -0.108627 -0.095730 -0.091795 -0.087932

                   Prior 12-2
Average Returns      0.765828
Std                  4.439253
Sharpe Ratio         0.084794
Alpha                0.918476
Beta                -0.150576
Treynor Ratio       -2.499886
Jensen Measure       0.455596
Information Ratio   -0.022190
```

1.9 5. Performances for Equal-Risk Contribution Portfolios of 7 Risk Premia Factors

How to get a best portfolio is a popular and important question to answer.In the Efficient-Frontier problem, the tangency portfolio has a maximum Sharpe Ratio and is generally seen as the best portfolio, given expected returns and expected Correlation matrix. However, there is a large drawdown of the traditional tangency portfolio. The weights are highly dependent on inputed expected returns. Once there is a slight change in expected returns, it will lead to a very large change in outputed weights.

The reason why this happens is because the tangency portfolio is to minimize the total portfolio variance, and the stock which has large variance and large marginal covariance will always have a small weight.

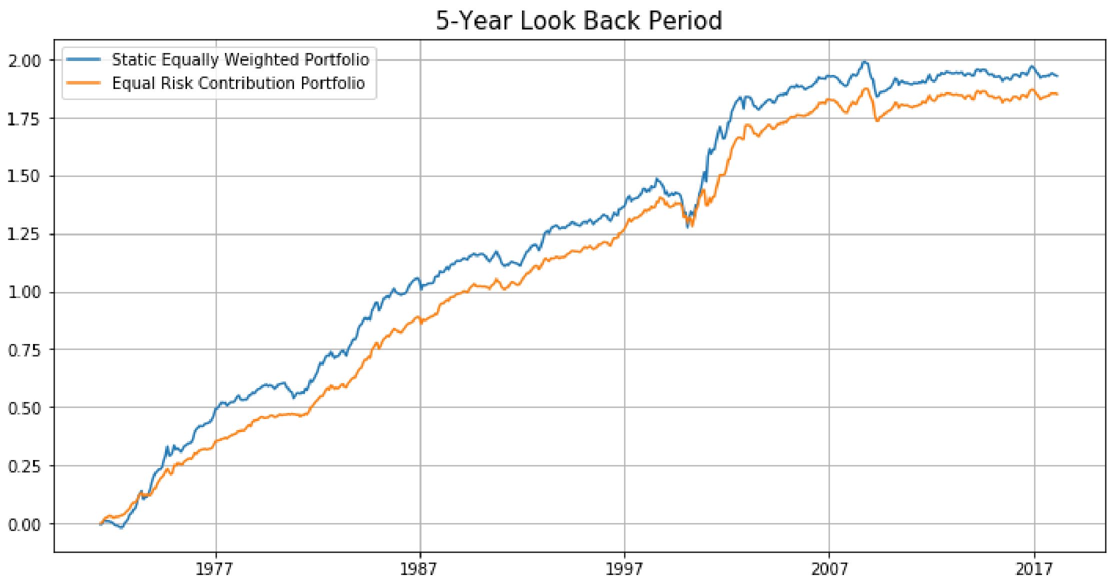
To solve this problem, the Equal-Risk Contribution portfolio is introduced. All stocks have not negative weights. Besides, all stocks have equl risk contribution to the portfolio.

To get the ERC portfolio, optimization technique is applied to minimize the sum of differences of stocks' marginal risk contributions.

In [15]:
```
df = pd.DataFrame()

for i in range(7):
    df[factor_list[i]] =( data_list[i]['SMALL']+data_list[i]['BIG'])/2
df = df.dropna()
```

In [16]:
```
erc_df_1,simple_1,erc_list_1 = erc(df,60)
look_back = [1,2,3,4,5]
erc_df_all=[]
simple_all=[]
erc_all=[]

plt.figure(figsize=(12,6))
idx = erc_df_1.index[1:]
plt.plot(idx,pd.Series(simple_1).cumsum()/100)
plt.plot(idx,pd.Series(erc_list_1).cumsum()/100)
plt.legend(['Static Equally Weighted Portfolio','Equal Risk Contribution Portfolio'])
plt.title('5-Year Look Back Period',fontsize = 15)
plt.grid(True)
plt.show()
```

```
E:\Anaconda3\lib\site-packages\ipykernel_launcher.py:160: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

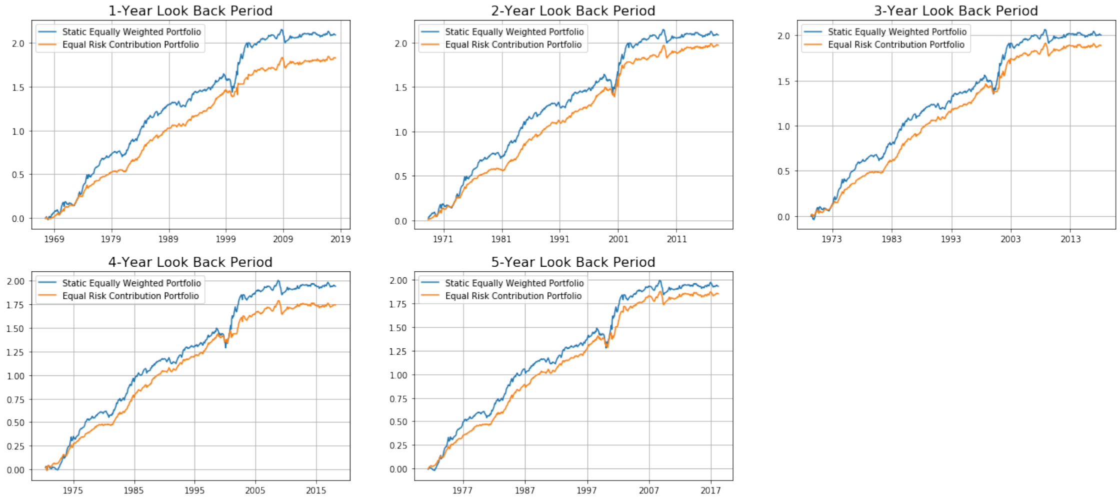


1.10 Performances for ERC Portfolios with Varies Look-Back Periods

```
In [17]: for i in range(5):
             year = look_back[i]
             erc_df,simple,erc_list = erc(df,12*year)
             erc_df_all.append(erc_df)
             simple_all.append(simple)
             erc_all.append(erc_list)


         plt.figure(figsize=(23,10))

         for i in range(5):
             year = look_back[i]
             erc_df = erc_df_all[i]
             simple = simple_all[i]
             erc_list = erc_all[i]
             plt.subplot(2,3,i+1)
             plt.plot(erc_df.index[1:],pd.Series(simple).cumsum()/100)
             plt.plot(erc_df.index[1:],pd.Series(erc_list).cumsum()/100)
             plt.legend(['Static Equally Weighted Portfolio','Equal Risk Contribution Portfoli
             plt.title(str(year)+'-Year Look Back Period',fontsize = 16)
             plt.grid(True)

         plt.show()
```

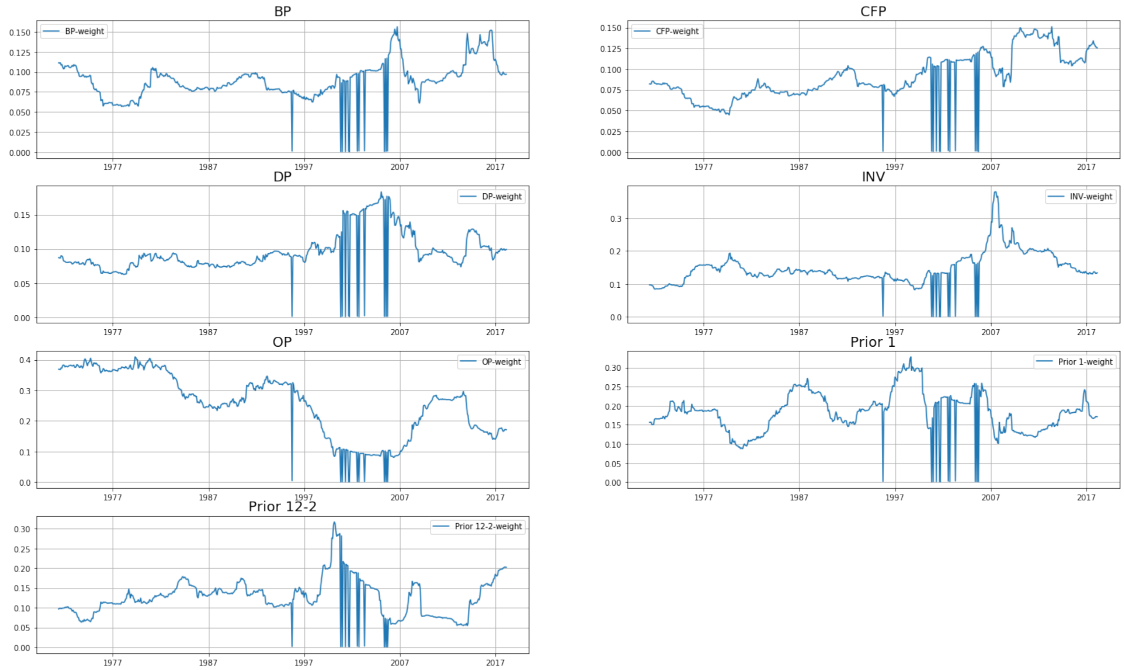
```
E:\Anaconda3\lib\site-packages\ipykernel_launcher.py:160: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```



1.11 6. Weights in ERC Portfolio

```
In [18]: weights_columns = [i+'_weight' for i in factor_list]

         plt.figure(figsize=(25,15))
         for i in range(7):
             w = weights_columns[i]
             plt.subplot(4,2,i+1)
             plt.plot(erc_df_1.index,erc_df_1[w])
             plt.legend([factor_list[i]+'-weight'])
             plt.title(factor_list[i],fontsize = 18)
             plt.grid(True)

         plt.show()
```



1.12 7 SAMPLE CORRELATION

In [19]:
```
df.corr()
```

Out[19]:

	BP	CFP	DP	INV	OP	Prior 1
BP	1.000000	0.849261	0.674611	0.693908	0.080970	0.002582
CFP	0.849261	1.000000	0.661933	0.608805	0.219449	-0.054607
DP	0.674611	0.661933	1.000000	0.611033	0.056343	-0.094664
INV	0.693908	0.608805	0.611033	1.000000	-0.036048	-0.125791
OP	0.080970	0.219449	0.056343	-0.036048	1.000000	-0.086440
Prior 1	0.002582	-0.054607	-0.094664	-0.125791	-0.086440	1.000000
Prior 12-2	-0.192215	-0.117848	-0.195671	-0.016750	0.114908	-0.283565

	Prior 12-2
BP	-0.192215
CFP	-0.117848
DP	-0.195671
INV	-0.016750
OP	0.114908
Prior 1	-0.283565
Prior 12-2	1.000000

1.13 8. Performances of Risk Premia Factor Based Portfolios

In [4]:
```
market_data = pd.read_csv('D:/PortfolioManagement/F-F.csv')
market_data['MARKET'] = market_data['Mkt-RF'] + market_data['RF']
```

```
        market_data['Month'] = pd.to_datetime(market_data['Month'],format ='%Y%m')
        market_data = market_data.set_index('Month')
```

```
In [5]: df = pd.DataFrame()

        for i in range(7):
            df[factor_list[i]] =( data_list[i]['SMALL']+data_list[i]['BIG'])/2
        df = df.dropna()
```

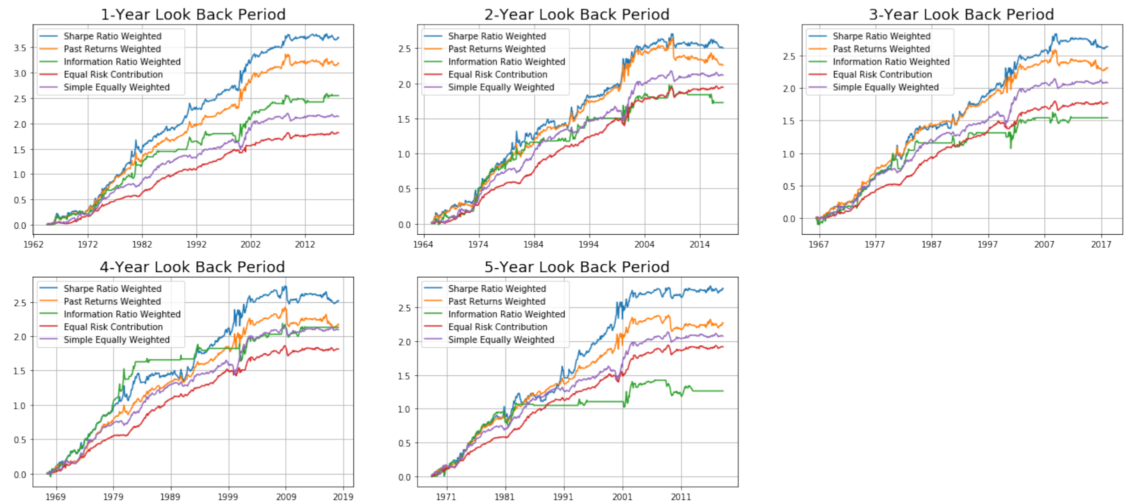
```
In [6]: look_back_periods = [1,2,3,4,5]

        sharpe_ratio_long_short = []
        sharpe_ratio_long_only = []

        average_mean_long_short = []
        average_mean_long_only = []

        information_ratio_long_short = []
        information_ratio_long_only = []

        simple_all = []
        df_all=[]
        erc_all_data=[]

        for year in look_back_periods:
            look_back =year*12
            erc_df,simple,erc_list = erc(df,12*year)
            erc_all_data.append(erc_list)


            data,x11,x12 = sharpe_ratio_equal(df,market_data,look_back,long_only=False)
            sharpe_ratio_long_short.append(x11)
            simple_all.append(x12)
            df_all.append(data)

            data,x21,x22 = sharpe_ratio_equal(df,market_data,look_back,long_only=True)

            sharpe_ratio_long_only.append(x21)

            average_mean_long_short.append(average_mean_equal(df,market_data,look_back,long_onl
            average_mean_long_only.append(average_mean_equal(df,market_data,look_back,long_only

            information_ratio_long_short.append(information_ratio_equal(df,market_data,look_bac
            information_ratio_long_only.append(information_ratio_equal(df,market_data,look_back
```

```
E:\Anaconda3\lib\site-packages\ipykernel_launcher.py:160: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [8]: plt.figure(figsize=(23,10))

        output_combine = []

        for i in range(5):
            year = look_back_periods[i]
            sr = sharpe_ratio_long_only[i]
            am = average_mean_long_only[i]
            ir = information_ratio_long_only[i]
            si = simple_all[i]
            new_df = df_all[i]
            idx = new_df.index[1:]
            erc_list = erc_all_data[i]

            plt.subplot(2,3,i+1)
            plt.plot(idx,pd.Series(sr).cumsum()/100)
            plt.plot(idx,pd.Series(am).cumsum()/100)
            plt.plot(idx,pd.Series(ir).cumsum()/100)
            plt.plot(idx,pd.Series(erc_list).cumsum()/100)
            plt.plot(idx,pd.Series(si).cumsum()/100)
            plt.title(str(year)+'-Year Look Back Period',fontsize = 18)

            plt.legend(['Sharpe Ratio Weighted','Past Returns Weighted',\
                        'Information Ratio Weighted','Equal Risk Contribution','Simple Equally

            plt.grid(True)

            data = pd.DataFrame()
            data['sr'] = pd.Series(sr,index=idx)
            data['am'] = pd.Series(am,index=idx)
            data['ir'] = pd.Series(ir,index=idx)
            data['erc'] = pd.Series(erc_list,index=idx)
            data['si'] = pd.Series(si,index=idx)



            data['RF'] =  market_data['RF']
            data['MARKET'] = market_data['MARKET']
            data['RM_RF'] = market_data['Mkt-RF']
            data['SMB'] = market_data['SMB']
            data['HML'] = market_data['HML']

            SMB = np.array(data['SMB'])
```

```
HML = np.array(data['HML'])
RM_RF = np.array(data['RM_RF'])

X = np.column_stack((RM_RF,SMB,HML))
X = sm.add_constant(X)

factor_list_2 = ['sr','am','ir','erc','si']
factor_full_name = ['Sharpe Ratio Weighted','Past Returns Weighted',\
                    'Information Ratio Weighted','Equal Risk Contribution','Simple

output_3 = pd.DataFrame()
for j in range(5):
    factor = factor_list_2[j]
    factor_full = factor_full_name[j]
    Y = np.array(data[factor] ).T

    MODEL = regression.linear_model.OLS(Y, X).fit()

    data_p = data[factor]
    data_p_2 = data[factor]-data['RF']
    data_p_3 = data[factor]-data['MARKET']

    rm_rf_mean = RM_RF.mean()
    rf_mean = data['RF'].mean()

    output_3.loc['Average Returns',factor_full] = data_p.mean()
    output_3.loc['Std',factor_full] = np.std(data_p)
    output_3.loc['Sharpe Ratio',factor_full] = data_p_2.mean()/np.std(data_p)
    output_3.loc['Alpha',factor_full] = MODEL.params[0]
    output_3.loc['Beta',factor_full] = MODEL.params[1]
    output_3.loc['Treynor Ratio',factor_full] = data_p_2.mean()/MODEL.params[1]
    output_3.loc['Jensen Measure',factor_full] = data_p.mean()-rf_mean-MODEL.params
    output_3.loc['Information Ratio',factor_full] = data_p_3.mean()/np.std(data_p_3

output_combine.append(output_3)
```




```
In [9]: output_combine[0]
```

```
Out[9]:                    Sharpe Ratio Weighted  Past Returns Weighted  \
        Average Returns                 0.571622               0.493093
        Std                             2.655556               2.469076
        Sharpe Ratio                    0.069497               0.042940
        Alpha                           0.568686               0.503211
        Beta                           -0.055650              -0.082969
        Treynor Ratio                  -3.316311              -1.277868
        Jensen Measure                  0.213217               0.148760
        Information Ratio              -0.063259              -0.077638

                           Information Ratio Weighted  Equal Risk Contribution  \
        Average Returns                      0.394564                 0.281275
        Std                                  2.382773                 1.041245
        Sharpe Ratio                         0.003145                -0.101604
        Alpha                                0.387407                 0.250353
        Beta                                -0.065432                -0.031098
        Treynor Ratio                       -0.114529                 3.401982
        Jensen Measure                       0.041198                -0.089776
        Information Ratio                   -0.099060                -0.130036

                           Simple Equally Weighted
        Average Returns                   0.330883
        Std                               1.382539
        Sharpe Ratio                     -0.040640
        Alpha                             0.260629
        Beta                             -0.065580
        Treynor Ratio                     0.856770
        Jensen Measure                   -0.022407
        Information Ratio                -0.111137
```

In [10]: output_combine[1]

```
Out[10]:                        Sharpe Ratio Weighted  Past Returns Weighted  \
         Average Returns                     0.395502               0.356521
         Std                                 2.697305               2.456542
         Sharpe Ratio                        0.002549              -0.013069
         Alpha                               0.425415               0.397981
         Beta                               -0.071298              -0.117230
         Treynor Ratio                      -0.096448               0.273863
         Jensen Measure                      0.043951               0.028855
         Information Ratio                  -0.096295              -0.101774

                                  Information Ratio Weighted  Equal Risk Contribution  \
         Average Returns                            0.272262                 0.306684
         Std                                        2.497338                 1.111815
         Sharpe Ratio                              -0.046595                -0.073700
         Alpha                                      0.272103                 0.274864
         Beta                                      -0.065494                -0.034750
         Treynor Ratio                              1.776693                 2.358001
         Jensen Measure                            -0.082306                -0.063871
         Information Ratio                         -0.121250                -0.123885

                                  Simple Equally Weighted
         Average Returns                         0.333875
         Std                                     1.393527
         Sharpe Ratio                           -0.039289
         Alpha                                   0.262222
         Beta                                   -0.066158
         Treynor Ratio                           0.827585
         Jensen Measure                         -0.020349
         Information Ratio                      -0.110971
```

In [26]: output_combine[2]

```
Out[26]:                        Sharpe Ratio Weighted  Past Returns Weighted  \
         Average Returns                     0.424730               0.368439
         Std                                 2.784460               2.343684
         Sharpe Ratio                        0.012969              -0.008610
         Alpha                               0.473404               0.410732
         Beta                               -0.094165              -0.113301
         Treynor Ratio                      -0.383500               0.178098
         Jensen Measure                      0.084935               0.038566
         Information Ratio                  -0.087051              -0.098412

                                  Information Ratio Weighted  \
         Average Returns                            0.250120
         Std                                        2.278296
         Sharpe Ratio                              -0.060790
```

```
Alpha                                     0.241276
Beta                                     -0.061522
Treynor Ratio                             2.251189
Jensen Measure                           -0.106599
Information Ratio                        -0.122674

                   Equal Risk Contribution Portfolio  Simple Equally Weighted
Average Returns                             0.322079                 0.341780
Std                                         1.096604                 1.428252
Sharpe Ratio                               -0.060677                -0.032794
Alpha                                       0.294572                 0.266634
Beta                                       -0.045176                -0.066563
Treynor Ratio                               1.472893                 0.703661
Jensen Measure                             -0.043116                -0.012326
Information Ratio                          -0.117645                -0.107184
```

In [27]: output_combine[3]

```
Out[27]:
                   Sharpe Ratio Weighted  Past Returns Weighted  \
Average Returns                 0.417721               0.354440
Std                             2.838203               2.260521
Sharpe Ratio                    0.011529              -0.013519
Alpha                           0.482189               0.406925
Beta                           -0.109343              -0.117801
Treynor Ratio                  -0.299253               0.259423
Jensen Measure                  0.097416               0.039139
Information Ratio              -0.100091              -0.114877

                   Information Ratio Weighted  \
Average Returns                      0.353950
Std                                  2.337596
Sharpe Ratio                        -0.013283
Alpha                                0.362989
Beta                                -0.057616
Treynor Ratio                        0.538911
Jensen Measure                       0.003040
Information Ratio                   -0.117559

                   Equal Risk Contribution Portfolio  Simple Equally Weighted
Average Returns                             0.303422                 0.338191
Std                                         1.070301                 1.424255
Sharpe Ratio                               -0.076220                -0.032865
Alpha                                       0.280580                 0.273511
Beta                                       -0.037267                -0.068169
Treynor Ratio                               2.189009                 0.686657
Jensen Measure                             -0.059528                -0.006475
Information Ratio                          -0.138222                -0.122171
```

In [11]: output_combine[4]

Out[11]:

```
                     Sharpe Ratio Weighted  Past Returns Weighted  \
Average Returns                   0.464865               0.379710
Std                               2.796618               2.123614
Sharpe Ratio                      0.026782              -0.004830
Alpha                             0.501467               0.393378
Beta                             -0.090989              -0.103402
Treynor Ratio                    -0.823163               0.099186
Jensen Measure                    0.121429               0.042623
Information Ratio                -0.078714              -0.097442

                     Information Ratio Weighted  Equal Risk Contribution  \
Average Returns                        0.211281                 0.320588
Std                                    1.785575                 1.075992
Sharpe Ratio                          -0.100072                -0.064479
Alpha                                  0.225383                 0.289631
Beta                                  -0.070168                -0.035413
Treynor Ratio                          2.546557                 1.959153
Jensen Measure                        -0.142803                -0.051269
Information Ratio                     -0.134619                -0.118522

                     Simple Equally Weighted
Average Returns                     0.347068
Std                                 1.419656
Sharpe Ratio                       -0.030218
Alpha                               0.266823
Beta                               -0.066504
Treynor Ratio                       0.645052
Jensen Measure                     -0.008889
Information Ratio                  -0.105558
```