Fundamental Factor Model Research
for the Chinese Equity Market

LI Meiyi
BBA in Economics and minor in Actuarial Mathematics

Supervised by:
Dr. David Rossiter
Department of Computer Science and Engineering

Hong Kong University of Science and Technology
2016 summer
# Table of Contents

1. Abstract...........................................................................................................................................3
2. Introduction........................................................................................................................................4  
   a. Background....................................................................................................................................4  
   b. Purpose.........................................................................................................................................4  
   c. Scope...........................................................................................................................................4  
3. Methodology......................................................................................................................................4  
   a. Programming language...............................................................................................................4  
   b. Data Source..................................................................................................................................5  
   c. Fundamental Factor Model.........................................................................................................5  
   d. Factors.........................................................................................................................................6  
   e. Calculation for Risk Contribution...............................................................................................7  
4. Results and Interpretation.................................................................................................................8  
   a. Result sample display...................................................................................................................8  
   b. Interpretation of results...............................................................................................................11  
5. Issues................................................................................................................................................13  
6. Future Plans......................................................................................................................................13  
7. Codes...............................................................................................................................................13
1. **Abstract**

China is for no doubt one of the major equity market of this world with Shanghai Stock Exchange and Shenzhen Stock Exchange ranking world no. 5 and 8 in terms of market capitalization. However, the academic study over Chinese stock market is very limited, for mostly two reasons: the limitation of availability of consistent, clean data; the language that some most useful data portal uses is Chinese. These issues placed a lot of hindrance for international researchers to get a deep insight into Chinese stock market. One of the most desired information for both equity researchers and institutional investors is always the risk structure of assets. Yet because of the difficulty for obtaining good data resource for Chinese equity market, the risk structure of Chinese stock markets has been long wanted. The aim of this project is therefore to solve this problem by building a simple, convenient, and intuitive research tool on Chinese stock market. Based on fundamental factor model, the tool breaks down equity risks into multiple origins, thus providing a powerful and insightful picture into risk structure of Chinese equities.
2. Introduction
   a. Background
   Factor model as a pricing method on equity market is well-grounded in academic area. Since early 1960s with the introduction of capital asset pricing model (CAPM), arbitrage pricing theory (APT) in 1976, and Fama-French three factor model in 1992, 1993, factor model has been well developed and used by research and investment institutions. One of the major benefit of fundamental factor model, is that it intuitively shows the contribution of risk form different factors to the risk of a single stock or portfolio. Therefore factor model is an excellent tool for understanding the risk composition of equity market and other security markets.

   b. Purpose
   This project aims to provide insights on Chinese stock market based on the theory of fundamental multi-factor model and particularly the Fama-French three factor model. The purpose of the program is to offer a convenient, simple and informative tool for equity researchers to gain insight into the risk composition of stocks listed on two Chinese stock markets: Shanghai Stock Exchange and Shenzhen Stock Exchange. For inputs, the program requires ticker, start date, end date, and calculation unit. Then the program regress the returns of the designated stock over several factors: market excess return, industry excess return, concept excess return, SMB (big minus small), HML (high minus low), UMD (up minus down) using ordinary least square method. Shown in results are betas for different factors, statistical inference to each factor, and a graph presenting the risk composition of the investigated stock.

   c. Scope
   The time frame of data that can be investigated is 3 years, and the scope of data covers both Shenzhen Stock Exchange and Shanghai Stock Exchange.

3. Methodology
   a. Programming language
   This project is programmed in Python 3. Several most frequently used libraries are Pandas1, Numpy2, Statsmodel3, Matplotlib4, SciPy5 and Sympy6. This project is compiled and visualized in jupyter notebook7, which is a web application for creating, visualizing, and sharing coded programs.

---

1 http://pandas.pydata.org/
2 http://www.numpy.org/
3 http://statsmodels.sourceforge.net/
4 http://matplotlib.org/
5 https://www.scipy.org/
7 http://jupyter.org/

4
b. Data Source
Data used in the project are from tushare[^8] and Datayes[^9]. The former is an open source, free data portal based on Python 2 and 3. The drawback of this resource is its website is solely in Chinese. The latter is a paid data platform. The price used for calculation is daily closing price. The price has been adjusted for stock splits and dilutions.

c. Fundamental Factor Model
i. Fundamental Factor Model
A fundamental factor model of asset returns explains compositions of security returns using exogenously decided factors, such as market return, industry return and price-earning ratio. The factors are historical returns of portfolios that replicate the performance of desired explanatory variables. For example, we use historical return of CSI 300 index to resemble market return. After a robust linear regression of asset return over factor returns, the coefficient of each factor is called beta, representing the sensitivity of asset return in reaction to each unit change in factor return.

For this project, the factors are market excess return, industry excess return, concept excess return, SMB, HML, and UMD. The choice of factors originates from Mark Cahart’s four-factor model[^10] with industry excess return and concept excess return added for a more comprehensive picture for Chinese stock market. The robust linear regression can be expressed in the following form:

\[ r_a - r_f = \alpha_a + \beta_1 \times (r_m - r_f) + \beta_2 \times (r_i - r_f) + \beta_3 \times (r_c - r_f) + \beta_4 \times SMB + \beta_5 \times HML + \beta_6 \times UMD + \varepsilon_a \]

- \( r_a \) : historical return of asset of investigation
- \( r_a - r_f \) : asset excess return
- \( r_f \) : risk free return, taken to be 0.035 according to J.P. Morgan practice
- \( \beta \) : beta/sensitivity for 6 different factors, representing the level of change in investigated stock resulting from unit change in factors
- \( r_m - r_f \) : market excess return
- \( r_i - r_f \) : industry excess return
- \( r_c - r_f \) : concept excess return
- \( SMB, HML, UMD \) : the returns of portfolio of SMB, HML, UMD, representing the risk factor of size, value, and momentum
- \( \alpha_a \) : the intercept of the regression, representing excess return, which is the abnormal return usually generated by inefficiency of market
- \( \varepsilon_a \) : error term, which is the vertical distance between data points and the regression line, representing the return not explained by existing factors

[^8]: http://tushare.org/index.html
[^9]: https://m.datayes.com/
[^10]: Mark Cahart (1997), ‘On Persistence in Mutual Fund Performance’
d. Factors
   i. Market Excess Return
      The goal of this factor is to capture the overall market behavior as comprehensively as possible. Thus, the data used to calculate market return is CSI 300 index, a capitalization-weighted index reflecting the combined performance of Shenzhen and Shanghai stock exchange.

   ii. Industry Excess Return
      The industry return is obtained by summing up the capitalization-weighted return of every stock under the same industry category. The industry categorization is provided by Sina finance\textsuperscript{11}, an extensively used finance data website for both individual investors and investment institutions in China.

   iii. Concept Excess Return
      Besides overall market and industry, a single stock or portfolio’s return can be also explained by significant macro event, such as government policy, regional economic shock, technology advancement. ‘Concept’ factor accounts for these effect by categorizing stocks according to the most prominent image people relate each stock to. It could be a recent government policy, a specific geographical area, or a new technology. For example, in early 2015 3D printing technology caught public’s attention. For a period of time after that, 3D printing related stocks became ‘trendy’ to invest in. As a result, from March 2015 to August 2015, the concept of 3D printing, in average, increased by more than 50%, with several top runners reaching up to more than 100% increase. Isn’t all companies in technology industry are related with 3D printing, thus, industry category does not apply in this case. The categorization of concept is obtained from Sina finance.

   iv. SMB & HML
      In the 1992 paper Fama and French disclosed the finding that stock returns in the US market is positively related to book-to-market ratio (value effect) and inversely related to market capitalization of stocks (size effect)\textsuperscript{12}. For value effect of book-to-market ratio, the logic behind is that stocks with higher book-to-market ratios are more likely to have been undervalued relative to the company’s book size and its stock price has higher potential to rise in the future. To account for the size effect of market capitalization, usually companies with smaller size are more fragile against economical movements and shocks, and therefore have higher risks, which is interpreted as expected higher potential

\textsuperscript{11} http://finance.sina.com.cn/stock/
\textsuperscript{12} Eugene Fama & Kenneth French (1992), ‘The Cross Section of Expected Stock Returns’
return in stock market. In the subsequent 1993 paper, Fama and French proved that the risk from capitalization can be concluded from returns of SMB portfolio, and book-to-market ratio from HML portfolio\textsuperscript{13}. Thus, there is enough ground to use SMB and HML portfolio to replicate return effect from capitalization and book-to-market ratio.

SMB represents ‘small minus big’, small stands for the returns of small-capitalized companies, and vice versa. To construct a SMB portfolio of time t, all stocks across the two stock exchanges are ranked by their market capitalization at time t. Simple averages of the return for top 30% and bottom 30% are calculated. The SMB factor is obtained by subtracting the simple average of bottom 30% from the top 30%.

HML stands for ‘high minus low’, and high means the return for high book-to-market ratio companies. To construct a HML portfolio of time t, all stocks are ranked by their P/B ratio (price-to-book ratio), which is the inverse of book-to-market ratio, of time t. The rank is split into top half and bottom half. The HML factor is obtained by simple average return of bottom half (that is, higher book-to-market ratio) minus the simple average return of the top half.

v. UMD

According to Mark Cahart’s 1997 publishing on mutual fund return, abnormal return is found from buying last year’s winner and selling last year’s losers\textsuperscript{14}. To account for this abnormal return, Cahart came up with the fourth factor UMD adding up to the existing three Fama-French factors. UMD stands for up minus down, in which up means returns of last period’s best performing stocks while down means returns of last period’s worst performing stocks. This factor explains the part of return of asset resulting from ‘persistence’ quality, and thus sometimes named as momentum factor.

To construct an UMD portfolio, last period’s return of all stocks are ranked. Top 30% performing stock is taken and so are bottom 30% performing stocks. Capitalization-weighted returns are calculated for both top and bottom 30%, becoming the return for ‘up’ and ‘down’ portfolio. And then the factor is obtained by subtracting return of ‘down’ from ‘up’.

e. Calculation for Risk Contribution

Risk contribution from a specific factor to stock return is calculated by taking partial derivative of the stock risk, and integrating it with respect to the weight of the factor from 0 to the designated factor weight:

\textsuperscript{13} Eugene Fama & Kenneth French (1993), ‘Common Risk Factors in the Returns on Stocks and Bonds’

\textsuperscript{14} Mark Cahart (1997), ‘On Persistence in Mutual Fund Performance’
\[ \sigma_{stock} = \sqrt{(w_1 \sigma_1)^2 + (w_2 \sigma_2)^2 + \ldots + 2w_1w_2\sigma_{12} + \ldots} \]

contribution of risk from factor 1  \[ = \int w_1 \frac{\partial \sigma_{stock}}{\partial w_1} dw_1 / \sigma_{stock} \]

\( w_1 \): weight of factor 1 asset, which is equal to factor return coefficient \( \beta \)

\( \sigma_1 \): standard deviation, or risk of factor 1

4. Results and Interpretation
   a. Result sample display

<table>
<thead>
<tr>
<th>end date of week</th>
<th>Risk Free Rate</th>
<th>Market Return</th>
<th>Industry Return</th>
<th>Concept Return</th>
<th>SMB</th>
<th>HML</th>
<th>UMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-07-29</td>
<td>0.035</td>
<td>-0.66</td>
<td>-0.014010</td>
<td>-0.028755</td>
<td>-0.037384</td>
<td>-0.020068</td>
<td>-0.000080</td>
</tr>
<tr>
<td>2016-07-22</td>
<td>0.035</td>
<td>-1.56</td>
<td>-0.015982</td>
<td>-0.004466</td>
<td>0.010336</td>
<td>-0.007806</td>
<td>0.008120</td>
</tr>
<tr>
<td>2016-07-15</td>
<td>0.035</td>
<td>2.63</td>
<td>0.024653</td>
<td>0.020110</td>
<td>-0.020732</td>
<td>0.008714</td>
<td>0.008489</td>
</tr>
<tr>
<td>2016-07-08</td>
<td>0.035</td>
<td>2.11</td>
<td>0.016549</td>
<td>0.028480</td>
<td>0.008117</td>
<td>-0.010108</td>
<td>0.003014</td>
</tr>
<tr>
<td>2016-07-01</td>
<td>0.035</td>
<td>2.50</td>
<td>0.031132</td>
<td>0.034485</td>
<td>0.022671</td>
<td>-0.019018</td>
<td>-0.003076</td>
</tr>
<tr>
<td>2016-06-24</td>
<td>0.035</td>
<td>-1.07</td>
<td>0.002695</td>
<td>-0.007865</td>
<td>0.020978</td>
<td>-0.021483</td>
<td>-0.000770</td>
</tr>
<tr>
<td>2016-06-17</td>
<td>0.035</td>
<td>-1.70</td>
<td>-0.019611</td>
<td>-0.010581</td>
<td>0.010794</td>
<td>-0.011356</td>
<td>-0.001271</td>
</tr>
<tr>
<td>2016-06-08</td>
<td>0.035</td>
<td>-0.79</td>
<td>-0.005842</td>
<td>0.001193</td>
<td>0.016763</td>
<td>-0.013134</td>
<td>-0.009937</td>
</tr>
<tr>
<td>2016-06-03</td>
<td>0.035</td>
<td>4.14</td>
<td>0.056489</td>
<td>0.050286</td>
<td>0.035771</td>
<td>-0.030616</td>
<td>0.002277</td>
</tr>
<tr>
<td>2016-05-27</td>
<td>0.035</td>
<td>-0.51</td>
<td>-0.006429</td>
<td>0.001684</td>
<td>0.030567</td>
<td>-0.020015</td>
<td>-0.001558</td>
</tr>
<tr>
<td>2016-05-20</td>
<td>0.035</td>
<td>0.11</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-05-13</td>
<td>0.035</td>
<td>-1.77</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

A. Calculated factor returns
B. Regression analysis

<table>
<thead>
<tr>
<th></th>
<th>Market Excess Return</th>
<th>Industry Excess Return</th>
<th>Concept Excess Return</th>
<th>SMB</th>
<th>HML</th>
<th>UMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market Excess Return</td>
<td>1.000000</td>
<td>0.967117</td>
<td>0.891409</td>
<td>0.138886</td>
<td>-0.209613</td>
<td>0.249429</td>
</tr>
<tr>
<td>Industry Excess Return</td>
<td>0.967117</td>
<td>1.000000</td>
<td>0.921257</td>
<td>0.313690</td>
<td>-0.400449</td>
<td>0.181979</td>
</tr>
<tr>
<td>Concept Excess Return</td>
<td>0.891409</td>
<td>0.921257</td>
<td>1.000000</td>
<td>0.485833</td>
<td>-0.531329</td>
<td>0.163091</td>
</tr>
<tr>
<td>SMB</td>
<td>0.138886</td>
<td>0.313690</td>
<td>0.485833</td>
<td>1.000000</td>
<td>-0.86599</td>
<td>0.316509</td>
</tr>
<tr>
<td>HML</td>
<td>-0.209613</td>
<td>-0.400449</td>
<td>-0.531329</td>
<td>-0.86599</td>
<td>1.000000</td>
<td>0.294711</td>
</tr>
<tr>
<td>UMD</td>
<td>0.249429</td>
<td>0.181979</td>
<td>0.163091</td>
<td>-0.316509</td>
<td>0.294711</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

C. Factor correlation matrix
D. Covariance chart

*MER is short for Market Excess Return, IER for Industry Excess Return, CER for Concept Excess Return
b. Interpretation of results
   i. Regression analysis
      1. Coefficients/Betas
         Coefficient is the percentage of change in factor return that leads to the change in asset excess return. For example, coefficient of 1.2 represents for 10% increase in the according factor, there will be 12% increase in the asset excess return.
      2. z (z-stats) and P (p-values)
         z-stats measures the deviation of the estimated coefficient from its hypothetical value, which is 0 in this case, in unit of standard error. In the graph, z-stats of market excess return is 1.97, meaning that the coefficient of market excess return is 1.97 standard error from 0. The more z-stats deviate from 0, the more significant the coefficient is.
P-values is another way of interpreting the significance of coefficient besides z-stats, measuring how extreme the coefficient is. According to common statistical practice, a p-value equal to or less than 0.05 means high significance while 0.01 means very high significance.

ii. Correlation Matrix
Linear correlation coefficient represents the extent to which two variables are linearly related. It ranges from -1 to 1. -1 indicates a perfect negative linear relationship, 1 indicates a perfect positive linear relation while 0 indicates no correlation between two variables. However, correlation is not a robust measurement of linear relationship, and that makes visual inspection of the scatter plot crucial. Here is an example:

![Scatter plots showing different correlation relationships](image)

These are four sets of data with the same correlation 0.816, yet x2 does not show a linear relationship, while x3 and x4 have problem of extreme outlier. Thus, a covariance chart is provided for visual inspection.

iii. Covariance Chart
Covariance chart presents scatter plots showing the relationship between factor returns. On the diagonal line are the density plots.

---

iv. Risk Contribution Charts
The risk contribution chart explains the composition of risk from each factor. Risk is defined as the standard deviation of historical return of assets. The horizontal scale presents different factors and the vertical scale represents the asset risk contributed by each factor out of the total risk of the investigated stock. Note that the sum of all risk contributions is not necessarily 1. The discrepancy between the sum and 1 represents the risk not explained by existing factors.

5. Issues
One of the most outstanding issues of this program is the time required for fetching data. The total time spent on transporting data from the portal can be lengthy, since some factors require getting data for all stocks over the market of one day. Fetching 8 days of factor data can take up to 3 hrs to finish, which is not ideal. One way to solve this problem is by having a cache program. However, the current cache program in this system still needs the whole program to at least fetch the specific set of data once before storing those data. Thus, this is also a major potential for improvement.

6. Future Plans
Allowing for more customization will be the next step of this project. A list a factors will be provided for program users to choose from. A recommendation of factors will pop up when users have inputted their stock to research on. The investigation subject will no longer be a single stock, but can be an industry or a portfolio. For the next step of the program, users can input their portfolio by indicating which stock and what percentage of the total capital they want to invest on the stock. The program will do a auto-rebalancing each time the capital distribution deviates from the desired distribution. The model will be able to auto-test for the explanatory power of factors. Suggested by Connor and Korajczyk\textsuperscript{16}, if adding a factor doesn’t not contribute to less variance in average asset specific variance, then the factor is redundant. Using this method, the program will be able to identify the set of factors with the most explanatory power.

7. Codes

import tushare as ts
import numpy as np
import pandas as pd
import math
import os
from datetime import datetime
from pandas.stats.api import ols
from pandas.tools.plotting import scatter_matrix
from __future__ import print_function
import statsmodels.api as sm
import matplotlib.pyplot as plt
import matplotlib
from statsmodels.sandbox.regression.predstd import wls_prediction_std
from scipy.integrate import quad
from sympy import *
tset = ts.set_token('11ca1e622c4b228baa547fa4762dc8f28d97ef0decbc575c2f69aacf8a7ad')
mkt = ts.Market()

#Cache downloaded data
CACHE_FOLDER = './cache/

def CacheConstructor(function):
    def CachedFunction(*args, **kargs):
        file_name = CACHE_FOLDER + function.__module__ + '.' + function.__name__ + '().hdf5'
        storage = pd.HDFStore(file_name, format='table')
        key = 'h' + str(hash(str(args) + str(kargs))).replace('-','n')
        if key not in storage:
            storage.put(key, function(*args, **kargs))
        return storage.get(key)
    return CachedFunction
Cashed_get_h = CacheConstructor(ts.get_h_data)
Cashed_get_hist = CacheConstructor(ts.get_hist_data)

#Set basic variables
unit=''
ticker=''
start_date=''
end_date=''
#start
market_excess=0
SMB=0
HML=0
UMD=0
REV=0
return_mom=0
industry_mom=0

data=pd.DataFrame()
# A program to receive input

```python
def program():
    global unit, ticker, start_date, end_date
    unit=input('Please fill in the question, input q for quitting')
    unit=unit[0].lower()
    if unit=='q':
        return
    if unit=='d' or unit=='w' or unit=='m':
        start_date=input('Please indicate the start date (yyyy-mm-dd):')
        start=pd.to_datetime(start_date)
        if start.isoweekday()==6 or start.isoweekday()==7:
            print('The date you entered is not a business day.')
            program()
        else:
            end_date=input('Please indicate the end date (yyyy-mm-dd):')
            end=pd.to_datetime(end_date)
            if end.isoweekday()==6 or end.isoweekday()==7:
                print('The date you entered is not a business day.')
                program()
            else:
                ticker=input('Please indicate the stock to investigate on, input ticker for one stock, industry name, or market code:')
    else:
        print('There is an error in your input.')
        program()

program()
```

Please fill in the question, input q for quitting
Please indicate the time unit for calculating return, enter d for daily return, w for weekly return, m for monthly return:
Please indicate the start date (yyyy-mm-dd):2016-05-15
The date you entered is not a business day.
Please fill in the question, input q for quitting
Please indicate the time unit for calculating return, enter d for daily return, w for weekly return, m for monthly return:
Please indicate the start date (yyyy-mm-dd):2016-05-13
Please indicate the end date (yyyy-mm-dd):2016-08-02
Please indicate the stock to investigate on, input ticker for one stock, industry name, or market code:000001
In [7]:

```python
# Calculate return for indicated stock

def cal_return(ticker, start_date, end_date, unit, date_index_string_hyphen):
    data=Cashed_get_h(ticker, start_date, end_date, autype='hfq')
    try:
        data=data.loc[data.index.isin(date_index_string_hyphen)]
        if unit == 'd':
            data_day=data
            data_day['daily_return']=data_day['close'].pct_change(-1)
            return data_day
        elif unit == 'w':
            data_week=data
            data_week['weekly_return']=data_week['close'].pct_change(-1)
            data_week['end date of week']=pd.to_datetime(data_week.index.to_series())
            data_week['weekly_return'].apply(lambda x: x.strftime('%Y-%m-%d')).to_list()
            return data_week
        elif unit == 'm':
            data_month=data
            data_month['monthly_return']=data_month['close'].pct_change(-1)
            data_month['end date of month']=pd.to_datetime(data_month.index.to_series())
            data_month['monthly_return'].apply(lambda x: x.strftime('%Y-%m-%d')).to_list()
            return data_month
    except:
        print('No data available for the indicated period.')

data_copy=ts.get_hist_data(ticker,start_date,end_date,ktype=unit)
data_copy['datetime']=pd.to_datetime(data_copy.index.to_series())
data_copy=data_copy.set_index('datetime')
date_index_string_hyphen=data_copy.index.to_series().apply(lambda x: x.strftime('%Y-%m-%d')).tolist()

return_dependent_variable=cal_return(ticker, start_date, end_date, unit, date_index_string_hyphen)
return_dependent_variable
```

Out[7]:

<table>
<thead>
<tr>
<th>end date of week</th>
<th>open</th>
<th>high</th>
<th>close</th>
<th>low</th>
<th>volume</th>
<th>amount</th>
<th>weekly_return</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-07-29</td>
<td>1035.68</td>
<td>1053.93</td>
<td>1049.37</td>
<td>1029.98</td>
<td>67142534</td>
<td>614972669</td>
<td>0.029077</td>
</tr>
<tr>
<td>2016-07-22</td>
<td>1025.42</td>
<td>1025.42</td>
<td>1019.72</td>
<td>1017.43</td>
<td>29554964</td>
<td>264379250</td>
<td>-0.005559</td>
</tr>
<tr>
<td>2016-07-15</td>
<td>1020.86</td>
<td>1026.56</td>
<td>1025.42</td>
<td>1016.29</td>
<td>35203395</td>
<td>31555254</td>
<td>0.028609</td>
</tr>
<tr>
<td>2016-07-08</td>
<td>1002.61</td>
<td>1002.61</td>
<td>996.90</td>
<td>995.76</td>
<td>26134229</td>
<td>228778674</td>
<td>0.003442</td>
</tr>
<tr>
<td>2016-07-01</td>
<td>991.20</td>
<td>995.76</td>
<td>993.48</td>
<td>990.06</td>
<td>34893019</td>
<td>303652375</td>
<td>0.016337</td>
</tr>
<tr>
<td>2016-06-24</td>
<td>985.50</td>
<td>992.34</td>
<td>977.51</td>
<td>971.81</td>
<td>42766409</td>
<td>367771054</td>
<td>-0.001175</td>
</tr>
<tr>
<td>2016-06-17</td>
<td>977.51</td>
<td>982.08</td>
<td>978.66</td>
<td>974.09</td>
<td>31517300</td>
<td>270236698</td>
<td>-0.004830</td>
</tr>
<tr>
<td>2016-06-08</td>
<td>987.16</td>
<td>987.16</td>
<td>983.41</td>
<td>979.67</td>
<td>26818824</td>
<td>281651741</td>
<td>-0.000010</td>
</tr>
<tr>
<td>2016-06-03</td>
<td>980.61</td>
<td>985.29</td>
<td>983.42</td>
<td>974.99</td>
<td>42338929</td>
<td>443039581</td>
<td>0.022404</td>
</tr>
<tr>
<td>2016-05-27</td>
<td>957.19</td>
<td>963.75</td>
<td>961.87</td>
<td>954.38</td>
<td>22281892</td>
<td>228377555</td>
<td>-0.002913</td>
</tr>
<tr>
<td>2016-05-20</td>
<td>959.06</td>
<td>967.49</td>
<td>964.68</td>
<td>954.38</td>
<td>20670377</td>
<td>212277642</td>
<td>-0.005559</td>
</tr>
<tr>
<td>2016-05-13</td>
<td>969.37</td>
<td>974.05</td>
<td>969.37</td>
<td>967.49</td>
<td>18426611</td>
<td>191007256</td>
<td>NaN</td>
</tr>
</tbody>
</table>

In [8]:

```python
#Set factor variables
date_index=return_dependent_variable.index
date_index_string=return_dependent_variable.index.to_series().apply(lambda x: x.strftime('%Y%m%d')).tolist()

Factor_Data=pd.DataFrame(index=date_index)
Factor_Data['Risk Free Rate']=py.nan
Factor_Data['Market Return']=py.nan
Factor_Data['Industry Return']=py.nan
Factor_Data['Concept Return']=py.nan
Factor_Data['SMB']=py.nan
Factor_Data['HML']=py.nan
Factor_Data['UMD']=py.nan
```
# Calculate factor returns

```
# Market Return & Risk Free Rate

market_return = ts.get_hist_data('hs300', start_date, end_date, ktype='unit')
market_return['date'] = pd.to_datetime(market_return.index)
market_return['date'] = market_return.set_index('date')

for mr in range(len(date_index)):
    Factor_Data['Market Return'][date_index[mr]] = market_return['p_change'][date_index[mr]]

rf = 0.035  # the number used by Morgan Stanley
Factor_Data['Risk Free Rate'] = [rf * len(date_index)]
```

```
<table>
<thead>
<tr>
<th>end date of week</th>
<th>Risk Free Rate</th>
<th>Market Return</th>
<th>Industry Return</th>
<th>Concept Return</th>
<th>SMB</th>
<th>HML</th>
<th>UMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-07-29</td>
<td>0.035</td>
<td>-0.66</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-07-22</td>
<td>0.035</td>
<td>-1.56</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-07-15</td>
<td>0.035</td>
<td>2.63</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-07-08</td>
<td>0.035</td>
<td>1.21</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-07-01</td>
<td>0.035</td>
<td>2.50</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-06-24</td>
<td>0.035</td>
<td>-1.07</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-06-17</td>
<td>0.035</td>
<td>-1.70</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-06-08</td>
<td>0.035</td>
<td>-0.79</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-06-03</td>
<td>0.035</td>
<td>4.14</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-05-27</td>
<td>0.035</td>
<td>-0.51</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-05-20</td>
<td>0.035</td>
<td>0.11</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-05-13</td>
<td>0.035</td>
<td>-1.77</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>
```
```python
In [11]:

def add_zeros(tickers):
    for k in range(len(tickers)):
        tickers[k]=str(tickers[k])
    if len(tickers[k])!=6:
        tickers[k]='0'*((6-len(tickers[k]))+ticks[k]
    return tickers

#Getting industry information
industry_classification=t.get_industry_classified()
industry_name=industry_classification.loc[industry_classification['code']==ticker]['c_name'].to_string()[-4:]
all_companies_same_industry=industry_classification.loc[industry_classification['code']==industry_name]

###Industry return
for each_date in range(len(date_index_string)-2):
    all_companies_same_industry['return '+date_index_string[each_date]]=py.nan
    all_companies_same_industry['market cap '+date_index_string[each_date]]=py.nan
    all_companies_same_industry['percentage '+date_index_string[each_date]]=py.nan
    all_companies_same_industry['weighted return '+date_index_string[each_date]]=py.nan

number_industry=all_companies_same_industry.count()['code']
all_companies_same_industry=all_companies_same_industry.set_index(pd.Series(list(range(number_industry))))
list_of_company=all_companies_same_industry['code'].tolist()

#Getting returns, market cap
for each_company in range(number_industry):
    try:
        company_return=cal_return(all_companies_same_industry['code'][each_company],start_date,end_date,unit,
        date_index_string_hyphen)
    except TypeError:
        continue
    for each_date in range(len(date_index_string)-2):
        try:
            all_companies_same_industry['return '+date_index_string[each_date]][each_company]=company_return
        except KeyError:
            continue

for each_date in range(len(date_index_string)-2):
    try:
        industry_stock_cap_one_day=mkt.StockFactorsOneDay(tradeDate=date_index_string[each_date],field='ASS
I,ticker')
    except:
        industry_stock_cap_one_day=mkt.StockFactorsOneDay(tradeDate=date_index_string[each_date],field='ASS
I,ticker')
        industry_stock_cap_one_day['ticker']=pd.Series(add_zeros(industry_stock_cap_one_day['ticker'].tolist()))
        for each_company in range(number_industry):
            try:
                market_cap_one_company=industry_stock_cap_one_day.loc[industry_stock_cap_one_day['ticker'][1]=list
                of_company][each_company]['ASSI'].__float__()
            except (ValueError,TypeError):
                continue
                number_industry=all_companies_same_industry.count()['code']
                all_companies_same_industry['market cap '+date_index_string[each_date]][each_company]=market_cap
                one_company
    except

#Calculating weighted return
for each_date in range(len(date_index_string)-2):
    all_companies_same_industry['market cap '+date_index_string[each_date]]=all_companies_same_industry['market cap '+date_index_string[each_date]].sum()
    all_companies_same_industry['weighted return '+date_index_string[each_date]]=all_companies_same_industry['weighted return '+date_index_string[each_date]].sum()
```
In [13]:

### Getting concept information

```python
# Getting concept information
concept_classification=ts.get_concept_classified()
concept_name=concept_classification.loc[concept_classification['code']==ticker]['c_name'].to_string()[-4:]
all_companies_same_concept=concept_classification.loc[concept_classification['c_name']==concept_name].__float__()[0]
```

### Concept return

```python
# Concept return
for each_date in range(len(date_index_string)-2):
    all_companies_same_concept['market cap'+date_index_string[each_date]]=py.nan
    all_companies_same_concept['percentage'+date_index_string[each_date]]=py.nan
    all_companies_same_concept['weighted return'+date_index_string[each_date]]=py.nan
```

```python
number_concept=all_companies_same_concept.count()]['code']
all_companies_same_concept=all_companies_same_concept.set_index(pd.Series(list(range(number_concept))))
list_of_company_concept=all_companies_same_concept['code'].tolist()
```

### Getting returns, market cap

```python
# Getting returns, market cap
for each_company in range(number_concept):
    try:
        company_return=cal_return(all_companies_same_concept['code'][each_company],start_date,end_date,unit,
        date_index_string_byphen)
    except TypeError:
        continue
    except ValueError:
        continue
    for each_date in range(len(date_index_string)-2):
        try:
            all_companies_same_concept['return'+date_index_string[each_date]][each_company]=company_return.
            loc[date_index[each_date]][-1]
        except KeyError:
            continue
```

```python
all_companies_same_concept[concept_name[0]+date_index_string[each_date]][each_company]=py.nan
for each_date in range(len(date_index_string)-2):
    try:
        concept_stock_cap_one_day=mkt.StockFactorsOneDay(tradeDate=date_index_string[each_date],field='ASSI
, ticker')
    except:  
        concept_stock_cap_one_day=mkt.StockFactorsOneDay(tradeDate=date_index_string[each_date],field='ASSI
, ticker')
    concept_stock_cap_one_day['ticker']=pd.Series(add_zeros(concept_stock_cap_one_day['ticker'].tolist()))
    for each_company in range(number_concept):
        try:
            market_cap_one_company=concept_stock_cap_one_day.loc[concept_stock_cap_one_day['ticker']==list_o
            f_company_concept[each_company]]['ASSI!']._float_()
                all_companies_same_concept['market cap'+date_index_string[each_date]][each_company]=market_cap
            _one_company
        except (ValueError,TypeError):
            continue
```

### Calculating weighted return

```python
# Calculating weighted return
for each_date in range(len(date_index)-2):
    all_companies_same_concept['percentage'+date_index_string[each_date]]=all_companies_same_concept['mark
et cap'+date_index_string[each_date]]/all_companies_same_concept['market cap'+date_index_string[each_dat
e]].sum()
    all_companies_same_concept['weighted return'+date_index_string[each_date]]=all_companies_same_concept[ 'percentage'+date_index_string[each_date]]/all_companies_same_concept['return'+date_index_string[each_d
ate]]
```

### Concept Return

```python
# Concept Return
for each_date in range(len(date_index)-2):
    Concept_Return=all_companies_same_concept['weighted return'+date_index_string[each_date]].sum()
    Factor_Data['Concept Return'][each_date]=Concept_Return
```

[Getting data]:################################################################################################
In [14]:

```
# Create Sorting index and Sum
sorted_stock_cap['sorting index']=number_x['A']+(number-2*number_x)[' ']+number_x['B']

# Get tickers
small_tickers_cap=sorted_stock_cap.loc[list(range(number_x))]['ticker'].tolist()
big_tickers_cap=sorted_stock_cap.loc[list(range(number-x,number))]['ticker'].tolist()

# Get Returns
sorted_stock_cap['return']=py.nan

for y in range(0,number_x):
    try:
        sorted_stock_cap['return'][y]=cal_return(small_tickers_cap[y], start_date, end_date, unit, date_index_string_hyphen).loc[date_index[k]][-1]
    except:
        sorted_stock_cap['return'][y]=py.nan

for z in range(number-x,number):
    try:
        sorted_stock_cap['return'][z]=cal_return(big_tickers_cap[z-(number-x)], start_date, end_date, unit, date_index_string_hyphen).loc[date_index[k]][-1]
    except:
        sorted_stock_cap['return'][z]=py.nan

# Weighted Return
grouped_cap=sorted_stock_cap.groupby("sorting index")
Small_Index=grouped_cap.get_group('A')['return'].mean()
Big_Index=grouped_cap.get_group('B')['return'].mean()
SMB=Small_Index-Big_Index
Factor_Data['SMB'][date_index[k]]=SMB
```

try:
    sorted_stock_PB['return'][y]=cal_return(small_tickers_PB[y],start_date,end_date,unit,date_index_string_hyphen).loc[date_index[k]][:1]
except:
    sorted_stock_PB['return'][y]=np.nan

for z in range(number-int(number*0.5),number):
    if math.isnan(sorted_stock_PB['return'][y]):
        try:
            sorted_stock_PB['return'][z]=cal_return(big_tickers_PB[z-(number-number_x)],start_date,end_date,unit,date_index_string_hyphen).loc[date_index[k]][:1]
        except:
            sorted_stock_PB['return'][z]=np.nan

#Calculating weighted return
grouped_PB=sorted_stock_PB.groupby("sorting index")

High_Index=grouped_PB.get_group('A')['return'].mean()
Low_Index=grouped_PB.get_group('B')['return'].mean()

HML=High_Index-Low_Index

Factor_Data['HML'][date_index[k]]=HML
In [15]:

```
#Getting a chart of all stocks and their return for all dates
all_stock_basics=ts.get_stock_basics()
all_tickers=all_stock_basics.index.tolist()
Momentum_Data=pd.DataFrame(index=all_stock_basics.index)
Momentum_Data['market cap']=py.na

for each_date in date_index_string[:-1]:
    Momentum_Data[each_date]=py.na

for each in all_tickers:
    try:
        each_stock_record=cal_return(each,start_date,end_date,unit,each_index_string_hyphen)
    except:
        continue
    for a_number in range(len(date_index)-1):
        try:
            Momentum_Data.loc[each][date_index_string[a_number]]=each_stock_record.loc[date_index[a_number]]
        except:
            continue

Momentum_Data_clean=Momentum_Data

for each_date in range(len(date_index_string)-1):
    Momentum_Data_clean=Momentum_Data_clean[Momentum_Data_clean[date_index_string[each_date]].notnull()]

def cal_UDM_one_period(k, Momentum_Data_clean, date_index, date_index_string):
    Momentum_Data_clean=all_tickers.sort_values(by=date_index_string[k],ascending=False)

    try:
        all_stock_cap_one_day_momentum=mkt.StockFactorsOneDay(tradeDate=date_index_string[k],field='ASSI,ticker')
    except:
        all_stock_cap_one_day_momentum=mkt.StockFactorsOneDay(tradeDate=date_index_string[k],field='ASSI,ticker')

    for each in all_tickers:
        Momentum_Data_clean['market cap'][each]=all_stock_cap_one_day_momentum.loc[all_stock_cap_one_day_momentum['ticker']==int(each)]['ASSI'].__float__()

    Momentum_Data_clean=Momentum_Data_clean[Momentum_Data_clean['market cap'].notnull()]
    number_momentum=Momentum_Data_clean.count()[date_index_string[k]]
    number_momentum=x=int(0.3*number_momentum)

    #adding sorting index
    Momentum_Data_clean[sorting_index]=number_momentum_x['A']+(number_momentum-2*number_momentum_x)/*/number_momentum_x['B']
    grouped_momentum=Momentum_Data_clean.groupby(sorting_index)
    big_cap=grouped_momentum.get_group('A').sum()['market cap']
    small_cap=grouped_momentum.get_group('B').sum()['market cap']
    Momentum_Data_clean['sum']=[big_cap]*number_momentum_x+(number_momentum-2*number_momentum_x)*[py.na]+[small_cap]*number_momentum_x
    Momentum_Data_clean[percentage]=Momentum_Data_clean['market cap']/Momentum_Data_clean['sum']

    #Getting weighted return
    Momentum_Data_clean[weighting_return]=Momentum_Data_clean[date_index_string[k]]*Momentum_Data_clean[percentage]

    #Getting Final Result
    grouped_momentum_again=Momentum_Data_clean.groupby(sorting_index)
    top_30=grouped_momentum_again.get_group('A').sum()['weighted return']
    bottom_30=grouped_momentum_again.get_group('B').sum()['weighted return']
    Momentum_one_period=top_30-bottom_30

    return Momentum_one_period

for k in range(1,len(date_index)-1):
    Factor_Data[UMD][date_index[k-1]]=cal_UDM_one_period(k, Momentum_Data_clean, date_index, date_index_string)
```
```python
Out[153]:

<table>
<thead>
<tr>
<th>end date of week</th>
<th>Risk Free Rate</th>
<th>Market Return</th>
<th>Industry Return</th>
<th>Concept Return</th>
<th>SMB</th>
<th>HML</th>
<th>UMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-07-29</td>
<td>0.035</td>
<td>-0.66</td>
<td>-0.014010</td>
<td>-0.028755</td>
<td>0.037364</td>
<td>0.020066</td>
<td>0.000080</td>
</tr>
<tr>
<td>2016-07-22</td>
<td>0.035</td>
<td>-1.56</td>
<td>-0.015982</td>
<td>-0.004466</td>
<td>0.010336</td>
<td>-0.007896</td>
<td>0.008120</td>
</tr>
<tr>
<td>2016-07-15</td>
<td>0.035</td>
<td>2.63</td>
<td>0.024653</td>
<td>0.020110</td>
<td>0.020732</td>
<td>0.008714</td>
<td>0.008489</td>
</tr>
<tr>
<td>2016-07-08</td>
<td>0.035</td>
<td>1.21</td>
<td>0.016549</td>
<td>0.028480</td>
<td>0.008714</td>
<td>-0.007896</td>
<td>0.000080</td>
</tr>
<tr>
<td>2016-07-01</td>
<td>0.035</td>
<td>2.50</td>
<td>0.031132</td>
<td>0.022671</td>
<td>-0.019018</td>
<td>-0.007896</td>
<td>0.003076</td>
</tr>
<tr>
<td>2016-06-24</td>
<td>0.035</td>
<td>-1.07</td>
<td>0.002595</td>
<td>-0.007865</td>
<td>0.020978</td>
<td>-0.021483</td>
<td>0.000770</td>
</tr>
<tr>
<td>2016-06-17</td>
<td>0.035</td>
<td>-1.70</td>
<td>-0.019611</td>
<td>-0.010581</td>
<td>0.010794</td>
<td>-0.011356</td>
<td>0.001271</td>
</tr>
<tr>
<td>2016-06-08</td>
<td>0.035</td>
<td>-0.79</td>
<td>-0.005842</td>
<td>0.001193</td>
<td>0.016763</td>
<td>-0.013134</td>
<td>0.009937</td>
</tr>
<tr>
<td>2016-06-03</td>
<td>0.035</td>
<td>4.14</td>
<td>0.056489</td>
<td>0.050286</td>
<td>0.035771</td>
<td>-0.030616</td>
<td>0.002277</td>
</tr>
<tr>
<td>2016-05-27</td>
<td>0.035</td>
<td>-0.51</td>
<td>-0.006429</td>
<td>0.001684</td>
<td>0.030567</td>
<td>-0.020015</td>
<td>0.001558</td>
</tr>
<tr>
<td>2016-05-20</td>
<td>0.035</td>
<td>0.11</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2016-05-13</td>
<td>0.035</td>
<td>-1.77</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

In [57]:

```

# Combining data
Factor_Data['Market Excess Return']=Factor_Data['Market Return']-Factor_Data['Risk Free Rate']
Factor_Data['Industry Excess Return']=Factor_Data['Industry Return']-Factor_Data['Risk Free Rate']
Factor_Data['Concept Excess Return']=Factor_Data['Concept Return']-Factor_Data['Risk Free Rate']
Complete_Data=Factor_Data.copy()[0:-2]
Complete_Data['stock return']=return_dependent_variable.iloc[:,1]
Complete_Data['dependent variable']=Complete_Data['stock return']-Complete_Data['Risk Free Rate']

# regression=sm.ols('stock return- Market Return + SMB', data=Complete_Data)
regression=ols(y=Complete_Data['stock return']-Complete_Data['Risk Free Rate'],x=Complete_Data[['Market Excess Return','Industry Excess Return','Concept Excess Return','SMB','HML','UMD']])
regression
```
Out[57]:
-------------------------Summary of Regression Analysis-------------------------
Formula: Y ~ <Market Excess Return> + <Industry Excess Return>  
+ <Concept Excess Return> + <SMB> + <HML> + <UMD> + <intercept>
Number of Observations:         10
Number of Degrees of Freedom:   7
R-squared:         0.9882
Adj R-squared:     0.9645
Rmse:              0.0026
F-stat (6, 3):    41.8014, p-value:     0.0055
Degrees of Freedom: model 6, resid 3 
-----------------------Summary of Estimated Coefficients------------------------
<table>
<thead>
<tr>
<th>Variable</th>
<th>Coef</th>
<th>Std Err</th>
<th>t-stat</th>
<th>p-value</th>
<th>CI 2.5%</th>
<th>CI 97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market Excess Return</td>
<td>0.0099</td>
<td>0.0044</td>
<td>2.24</td>
<td>0.1109</td>
<td>0.0012</td>
<td>0.0186</td>
</tr>
<tr>
<td>Industry Excess Return</td>
<td>0.2215</td>
<td>0.4100</td>
<td>0.54</td>
<td>0.6265</td>
<td>-0.5820</td>
<td>1.0250</td>
</tr>
<tr>
<td>Concept Excess Return</td>
<td>-0.6771</td>
<td>0.1477</td>
<td>-4.59</td>
<td>0.0195</td>
<td>-0.9665</td>
<td>-0.3877</td>
</tr>
<tr>
<td>SMB</td>
<td>0.1036</td>
<td>0.4873</td>
<td>0.21</td>
<td>0.8453</td>
<td>-0.8516</td>
<td>1.0588</td>
</tr>
<tr>
<td>HML</td>
<td>0.4634</td>
<td>0.8398</td>
<td>0.55</td>
<td>0.6195</td>
<td>-1.1825</td>
<td>2.1094</td>
</tr>
<tr>
<td>UMD</td>
<td>-0.1932</td>
<td>0.1919</td>
<td>-1.01</td>
<td>0.3881</td>
<td>-0.5693</td>
<td>0.1828</td>
</tr>
<tr>
<td>intercept</td>
<td>-0.0731</td>
<td>0.0174</td>
<td>-4.19</td>
<td>0.0247</td>
<td>-0.1072</td>
<td>-0.0389</td>
</tr>
</tbody>
</table>

In [86]:
```
Method 2, Robust analysis

Complete_Data=Complete_Data[['Market Excess Return','Industry Excess Return','Concept Excess Return','SMB','HML','UMD']]

Complete_Data_array=Complete_Data.as_matrix()
Complete_Data_array=sm.add_constant(Complete_Data_array)
Complete_Data_array

regression_model = sm.RLM(Dependent_Variable_array, Complete_Data_array, M=sm.robust.norms.HuberT())

regression_result=regression_model.fit()

print(regression_result.summary())

Robust linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   10
Model:                            RLM   Df Residuals:                        3
Method:                          IRLS   Df Model:                            6
Norm:                          HuberT
Scale Est.:                       mad
Cov Type:                          H1
Date:                Tue, 09 Aug 2016
Time:                        00:16:01
No. Iterations:                     2
==============================================================================
     coef    std err     z    P>|z|     [95.0% Conf. Int.]
const  -0.1081    0.017  -6.204    0.000   [ -0.142,  -0.074]
x1     0.0099    0.004   2.241    0.025    [ 0.001,   0.019]
x2     0.2215    0.410   0.540    0.589    [ -0.582,   1.025]
x3    -0.6771    0.148  -4.586    0.000   [- 0.967,  -0.388]
x4     0.1036    0.487   0.213    0.832    [ -0.852,   1.059]
x5     0.4634    0.840   0.552    0.581   [-1.182,    2.109]
x6    -0.1932    0.192  -1.007    0.314    [-0.569,   0.183]
```

If the model instance has been used for another fit with different fit parameters, then the fit options might not be the correct ones anymore.

In [37]:

```
# Correlation Matrix

Correlation_Matrix=Complete_Data[['Market Excess Return', 'Industry Excess Return', 'Concept Excess Return', 'SMB', 'HML', 'UMD']]
Correlation_Matrix=Correlation_Matrix.corr()
```

Out[37]:

```
<table>
<thead>
<tr>
<th></th>
<th>Market Excess Return</th>
<th>Industry Excess Return</th>
<th>Concept Excess Return</th>
<th>SMB</th>
<th>HML</th>
<th>UMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market Excess Return</td>
<td>1.000000</td>
<td>0.967117</td>
<td>0.891409</td>
<td>0.138886</td>
<td>-0.209613</td>
<td>0.246429</td>
</tr>
<tr>
<td>Industry Excess Return</td>
<td>0.967117</td>
<td>1.000000</td>
<td>0.921257</td>
<td>0.313690</td>
<td>0.400449</td>
<td>0.181979</td>
</tr>
<tr>
<td>Concept Excess Return</td>
<td>0.891409</td>
<td>0.921257</td>
<td>1.000000</td>
<td>0.485833</td>
<td>-0.531329</td>
<td>0.163691</td>
</tr>
<tr>
<td>SMB</td>
<td>0.138886</td>
<td>0.313690</td>
<td>0.485833</td>
<td>1.000000</td>
<td>-0.986599</td>
<td>0.316509</td>
</tr>
<tr>
<td>HML</td>
<td>-0.209613</td>
<td>-0.400449</td>
<td>-0.531329</td>
<td>-0.986599</td>
<td>1.000000</td>
<td>0.294711</td>
</tr>
<tr>
<td>UMD</td>
<td>0.246429</td>
<td>0.181979</td>
<td>0.163691</td>
<td>-0.316509</td>
<td>0.294711</td>
<td>1.000000</td>
</tr>
</tbody>
</table>
```

In [154]:
```
# Covariance Scatter Plot

Complete_Data_Plot=Complete_Data.copy()
Complete_Data_Plot.columns=['MER', 'IER', 'CER', 'SMB', 'HML', 'UMD']

Covariance_Plot=scatter_matrix(Complete_Data_Plot, alpha=1, figsize=(10, 10), diagonal='kde')
matplotlib.pyplot.show()
```
In [132]:

# Calculate portfolio risk
portfolio_risk=0
for item in range(1, len(se)):  
    each=(se[item]*coef[item])**2  
    portfolio_risk=portfolio_risk+each

for row in range(len(Covariance_Matrix.columns)):  
    each=Covariance_Matrix.iloc[row,column]*coef[row+1]*coef[column+1]  
    portfolio_risk=portfolio_risk+each
portfolio_risk=portfolio_risk**0.5
portfolio_risk

# Calculate individual risk contribution

def cal_risk_contribution(k, se, coef, Covariance_Matrix, portfolio_risk):
    def integrand(x, k, se, coef, Covariance_Matrix, portfolio_risk):
        # code
        li2=list(range(len(Covariance_Matrix.columns)))
        li2.remove(k)
        for row in li2:
            risk_x=0
            for column in range(k+1, len(Covariance_Matrix.iloc[row])):  
                each=Covariance_Matrix.iloc[row,column]*y*coef[column+1]  
                risk_x+=each
            contribution=I[0]/portfolio_risk
        return contribution

## Risk Contribution Chart in number

risk_contribution_chart=pd.DataFrame(py.nan,index=['risk contribution'],columns=Covariance_Matrix.columns)
for k in range(len(Covariance_Matrix.columns)):  
    risk_contribution_chart.iloc[0,k]=cal_risk_contribution(k, se, coef, Covariance_Matrix, portfolio_risk)
risk_contribution_chart

In [230]:

## Risk Contribution Chart in graphics

risk_contribution_chart.plot.bar(); plt.axhline(0, color='k')

matplotlib.pyplot.show()