Chinese Equity Market Analysis System
Version 2

LI Meiyi
BBA in Economics and minor in Mathematics

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

Hong Kong University of Science and Technology
2016-2017 winter term

Table of Contents

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, for single stock investigation. To investigate a portfolio, the program requires tickers of all assets included in the portfolio as well as their weights. If the total weight of stock asset does not add up to 1, the remaining weight will be allocated to cash. Then the program regress the returns of the designated stock or portfolio 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. Combinations of different lags in factors are also investigated to search for the most fitting model. 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. Areas of Improvement

Compared to the first version finished in summer 2016[1], the second version of the research tool solved several problems and added a few new features:

a. Updated cache function. Stated in section 5 of the report of first version research tool, one major problem is the time required for fetching data. The updated cache function works roughly 30% faster than the past cache function.

[1] http://www.cse.ust.hk/~rossiter/independent_studies_projects/chinese_market_analysis/chinese_market_analysis.pdf

b. Upgraded functionality. The last version of the research tool only supported research on a single stock. The new version allows research on portfolio return with user providing stock tickers, weights, and date for rebalancing. A rebalancing algorithm is created to support this new functionality.

c. New optimization method. The old version research tool generates the model through regressing dependent variable against factors of the same time period. The new version investigates the possibility of 'lead-lag' relationship by allowing factors to have lags. That is, the new version regresses the dependent variable and factors of different time periods. After all permutations of lags in each factor, the program will generate the optimal combination of factors with the best prediction power.

d. Terminal application. The whole program is transformed into a terminal application with simpler operation and interface. For details of using instruction, please see the following section.

4. The Analysis System with a Command Line Interface

a. Terminal Application

In consideration of a lack of interface of Jupyter notebook and the difficulty in its operation, a terminal application is created for easier usage and better interface. Instructions to use the applications are following:

i. Install Anaconda[2] and tushare[3].

ii. Download ResearchTool.py and the cache folder from https://github.com/tinali0923/StockAnalysisTool

iii. Open a command prompt and navigate to the program folder. Alternatively, open a command prompt from the package folder. The user may do this on windows by right clicking the folder in Explorer while holding down the Shift key. The submenu should have an item that can open a prompt there.

iv. To see the helper message, input command 'python ResearchTool.py -h'. Variable explanation:

1. 'StartDate': The start date of stock data to research. Should be in form 'yyyy-mm-dd'. Required input.

2. 'EndDate': The end date of stock data to research. Should be in form 'yyyy-mm-dd'. Required input.

3. 'Tickers': The ticker of the stock to research, should be a six-digit number, for example, '000001' for Ping An Insurance (Group) Company of China (中国平安银行). Required input.

[2] https://www.continuum.io/downloads

[3] https://pypi.python.org/pypi/tushare/

4. 'unit': The unit used for return calculation. Input ‘d’ for daily, ‘w’ for weekly, and ‘m’ for monthly return. Required input. The user should add ‘-u‘ before inputting the unit. For example, input ‘-u w’ to indicate weekly return.

5. 'rebalance': If the object of research is a portfolio, the user should indicate the date for rebalancing. Input 'd', 'w', 'm' to indicate the rebalancing date to be daily, end of the week, or end of the monthly. Optional input.

6. 'weights': If the object of research is a portfolio, the user should indicate the the weight for each asset in the portfolio. Input a list of weights used on each ticker separated by commas. The total must sum up to less than 1. If not indicated, each asset will be set to equal weight. Optional input.

See the example below:

1. To research on stock ‘000001’, with daily return starting from 2016.Oct.1 to 2017.Jan.1, input:

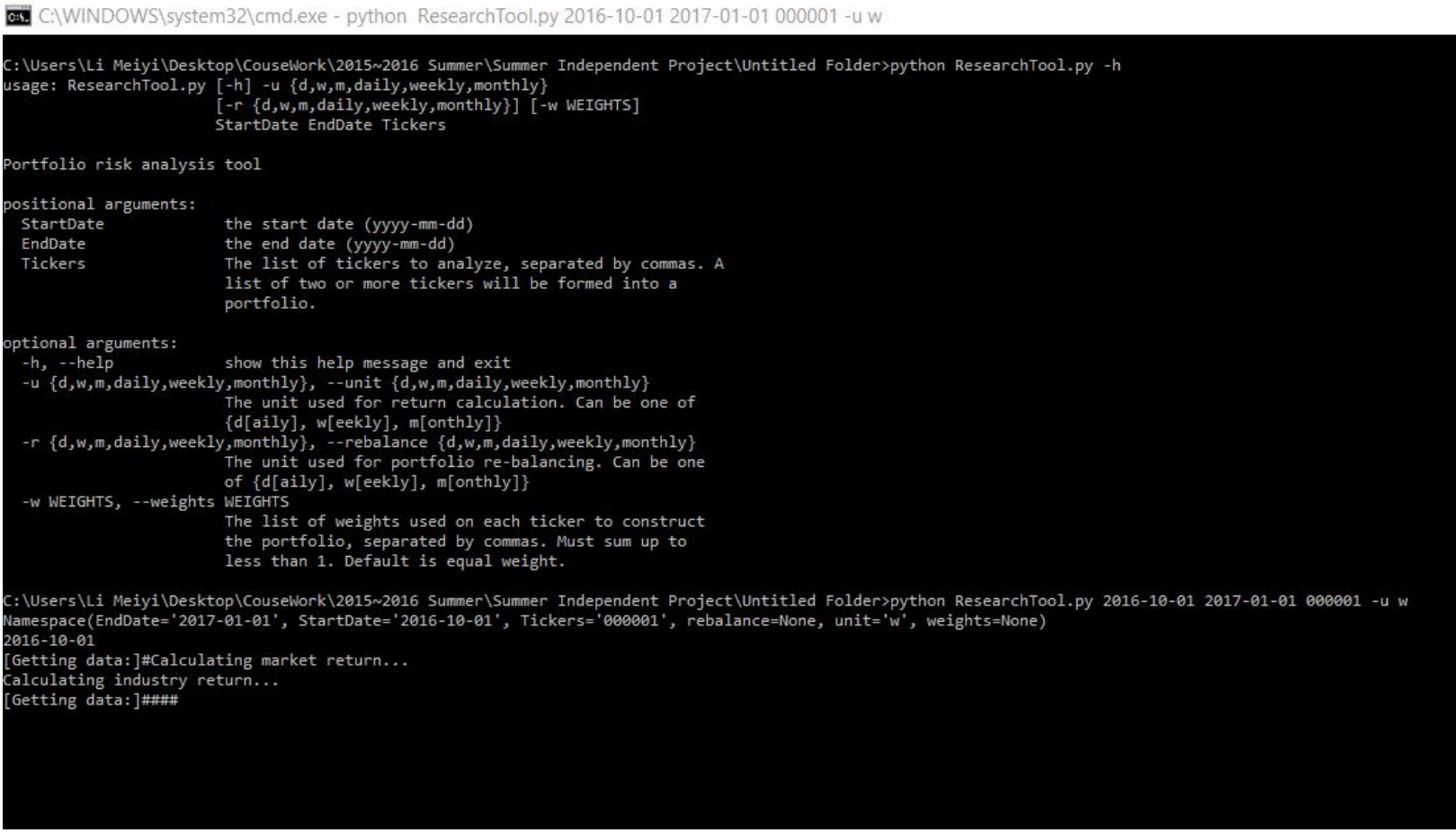
‘python ResearchTool.py 2016-10-01 2017-01-01 000001 -u d’

```
ResearchTool.py 2016-10-01 2017-01-01 000001 -u d
```

2. To research on portfolio with stock ‘000001’ and ‘000002’, with weekly return from 2015.Jan.1 to 2016.Jan.1, and with weight 0.3 and 0.4, as well as rebalancing date at the end of every month, input:

‘python ResearchTool.py 2015-01-01 2016-01-01 000001,000002 -u w -r m -w 0.3,0.4’

```
python ResearchTool.py 2015-01-01 2016-01-01 000001,000002 -u w -r m -w 0.3,0.4
```



5. Methodology

a. Programming language

This project is programmed in Python 3. Several most frequently used libraries are Pandas[4], Numpy[5], Statsmodel[6], Matplotlib[7], SciPy[8] and Sympy[9]. This project is compiled and visualized in jupyter notebook[10], which is a web application for creating, visualizing, and sharing coded programs.

[4] http://pandas.pydata.org/
[5] http://www.numpy.org/
[6] http://statsmodels.sourceforge.net/
[7] http://matplotlib.org/
[8] https://www.scipy.org/
[9] http://www.sympy.org/en/index.html
[10] http://jupyter.org/

b. Data Source

Data used in the project are from tushare[11] and Datayes[12]. 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. Rebalancing Algorithm

i. Introduction

If the object of investigation is a portfolio, the program applies a default algorithm to rebalance the portfolio at the designated time. The aim of rebalancing is to re-designate capital into given assets so that the weight of each asset is the same as the specified weight. The general logic of the rebalancing algorithm is that each rebalancing operation is divided into multiple mini steps. For each mini step, the algorithm deals with the largest deviation from designated weight. In this way, for each step the amount of cap reallocation is smaller and smaller, and finally the target weight can be achieved. During the process, cash is allowed to be negative, but finally the algorithm will make sure the cash reaches the designated weight.

ii. Methodology

1. First of all, total cash asset is set to 10000. This number doesn't influence the return and is for the purpose of convenient calculation.
2. A weight chart is built, which includes designated weight, weight after rebalancing, capital reallocation, total capital, new unit for each asset. This weight chart is updated each mini step and is shown for tracking. See example weight chart below.

	000001	000002	cash
column			
designated weight	0.300000	0.400000	0.3
weight after rebalancing	0.300000	0.400000	0.3
capital reallocation	0.000000	0.000000	0
total capital	9028.810456	12038.413942	9028.81
new unit	8.698611	4.747608	N/A

Weight Chart Example

*Note that the 'new unit' is in unit digit because the default total capital is small. This number will get bigger and the rebalancing action would cost less relatively when total capital is bigger.

[11] http://tushare.org/index.html

[12] https://m.datayes.com/

3. Weight rebalancing algorithm:
a) delta weight=new weight - supposed weight
b) Identify asset with the highest absolute value of delta weight
c) Identify the asset with largest delta weight of different direction
d) Allocate all the capital from asset in (c) to asset in (b)
e) Update the cumulative cap allocation in weight chart.
f) Repeat the above steps until weight allocation is precise to 0.1%, and cash isn't negative.

d. 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[13] 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
β : 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
α_a : the intercept of the regression, representing excess return, which is the abnormal return usually generated by inefficiency of market

[13] Mark Cahart (1997), 'On Persistence in Mutual Fund Performance'

ε_a : error term, which is the vertical distance between data points and the regression line, representing the return not explained by existing factors

e. 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[14], 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)[15]. 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.

[14] http://finance.sina.com.cn/stock/

[15] Eugene Fama & Kenneth French (1992), 'The Cross Section of Expected Stock Returns'

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 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[16]. 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[17]. 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'.

[16] Eugene Fama & Kenneth French (1993), 'Common Risk Factors in the Returns on Stocks and Bonds'

[17] Mark Cahart (1997), 'On Persistence in Mutual Fund Performance'

f. Optimization of Regression Factors

The program also allows users to regress factors against dependent variables with ‘lags’ in factors. That is, the user could regress factors against dependent variables not only of the same period, but of different period. For example, if now we have the daily return data over 2016-12-01 to 2016-12-05, and we want to investigate stock ‘000001’, then regressing ‘UMD’ return of 2016-12-02 with stock ‘000001’ return of 2016-12-01 means ‘UMD’ of one day lag.

The program would generate linear regress of all permutations of lag combination, and identify the lag combination which generates the largest coefficient of correlation value (R-squred), a measure of fit of the model. Which is calculated as follows:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

y is the observed dependent variable.

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$$

Total sum of squares.

$$SS_{\text{reg}} = \sum_i (f_i - \bar{y})^2$$

Explained sum of squares. f_i is the estimated result from regression model.

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

Residual sum of squares.

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}.$$

Coefficient of determination.

The lag combination that maximises r-square is 1 , 0 , 0 , 1 , 1 , 0 .

OLS Regression Results

Dep. Variable:	y	R-squared:	0.999
Model:	OLS	Adj. R-squared:	0.994
Method:	Least Squares	F-statistic:	235.3
Date:	Tue, 31 Jan 2017	Prob (F-statistic):	0.00424
Time:	23:34:06	Log-Likelihood:	50.841
No. Observations:	9	AIC:	-87.68
Df Residuals:	2	BIC:	-86.30
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
const	-0.0013	0.003	-0.406	0.724	-0.015	0.013
x1	0.0037	0.000	8.868	0.012	0.002	0.006
x2	1.4250	0.130	10.943	0.008	0.865	1.985
x3	-0.5444	0.189	-2.874	0.103	-1.359	0.271
x4	-0.5909	0.057	-10.324	0.009	-0.837	-0.345
x5	-0.5501	0.070	-7.915	0.016	-0.849	-0.251
x6	1.6329	0.351	4.647	0.043	0.121	3.145

Omnibus:	0.497	Durbin-Watson:	2.393
Prob(Omnibus):	0.780	Jarque-Bera (JB):	0.220
Skew:	-0.320	Prob(JB):	0.896
Kurtosis:	2.580	Cond. No.	1.24e+03

Example of optimization result

g. 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:

$$\sigma_{stock} = \sqrt{(w_1\sigma_1)^2 + (w_2\sigma_2)^2 + ... + 2w_1w_2\sigma_{12} + ...}$$

$$\text{contribution of risk from factor 1} = \int_0^{w1} \frac{\partial \sigma_{stock}}{\partial w_1} dw_1 / \sigma_{stock}$$

w_1 : weight of factor 1 asset, which is equal to factor return coefficient β

σ_1 : standard deviation, or risk of factor 1

6. Results and Interpretation

a. Result sample display

	000001 close	000002 close	000001 asset capital	000002 asset capital	cash	total capital	portfolio return
end date of week							
2016-12-30	1037.96	2535.68	9028.810456	12038.413942	9028	30096.034855	0.005585
2016-12-23	1035.68	2504.84	8978.661926	11971.549235	8978	29928.873086	-0.043981
2016-12-16	1055.07	2771.36	9391.720443	12522.293924	9391	31305.734809	-0.050322
2016-12-09	1100.69	3061.33	9889.374809	13185.833079	9889	32964.582697	-0.021662
2016-12-02	1089.29	3263.69	10108.338766	13477.785021	10108	33694.462554	-0.013212
2016-11-25	1097.27	3356.24	10243.678024	13658.237365	10243	34145.593413	0.006874
2016-11-18	1047.09	3420.40	10173.746017	13564.994689	10173	33912.486723	0.026626
2016-11-11	1047.09	3206.93	9909.884560	13213.179413	9909	33032.948532	0.025075
2016-11-04	1039.10	3034.19	9648.064108	12889.968081	9686	32224.920203	-0.004550
2016-10-28	1045.95	3053.93	9711.666494	12948.888659	9711	32372.221646	-0.006605
2016-10-21	1041.38	3115.63	9776.236867	13034.982490	9776	32587.456225	-0.022376
2016-10-14	1036.82	3311.82	10000.000000	13333.333333	10000	33333.333333	NaN

A. Calculated portfolio return (if to investigate a portfolio)

	open	high	close	low	volume	amount	weekly_return
end date of week							
2016-12-30	1035.68	1037.96	1037.96	1033.40	30260736	274882698	0.002201
2016-12-23	1042.52	1042.52	1035.68	1034.54	38291216	348140451	-0.018378
2016-12-16	1053.93	1059.63	1055.07	1050.51	39681397	366925349	-0.041447
2016-12-09	1083.58	1112.10	1100.69	1081.30	151419915	1460778328	0.010466
2016-12-02	1094.99	1094.99	1089.29	1076.74	82968650	790044442	-0.007273
2016-11-25	1081.30	1097.27	1097.27	1079.02	101367499	968127766	0.047923
2016-11-18	1050.51	1050.51	1047.09	1044.80	51759734	475289973	0.000000
2016-11-11	1042.52	1047.09	1047.09	1039.10	81226969	742969897	0.007689
2016-11-04	1039.10	1045.94	1039.10	1037.96	50811757	463831939	-0.006549
2016-10-28	1044.81	1052.79	1045.95	1042.52	54199774	497615051	0.004388
2016-10-21	1035.68	1041.38	1041.38	1033.40	68389677	621549845	0.004398
2016-10-14	1033.40	1036.82	1036.82	1031.12	33541176	304218677	NaN

B. Calculated single stock return (if to investigate a single stock)

	Risk Free Rate	Market Return	Industry Return	Concept Return	SMB	HML	UMD
end date of week							
2016-07-29	0.035	-0.66	-0.014010	-0.028755	-0.037364	0.020066	-0.000080
2016-07-22	0.035	-1.56	-0.015982	-0.004466	0.010336	-0.007896	0.008120
2016-07-15	0.035	2.63	0.024653	0.020110	-0.020732	0.008714	0.008489
2016-07-08	0.035	1.21	0.016549	0.028480	0.008117	-0.010108	0.003014
2016-07-01	0.035	2.50	0.031132	0.034485	0.022671	-0.019018	-0.003076
2016-06-24	0.035	-1.07	0.002595	-0.007865	0.020978	-0.021483	-0.000770
2016-06-17	0.035	-1.70	-0.019611	-0.010581	0.010794	-0.011356	-0.001271
2016-06-08	0.035	-0.79	-0.005842	0.001193	0.016763	-0.013134	-0.009937
2016-06-03	0.035	4.14	0.056489	0.050286	0.035771	-0.030616	0.002277
2016-05-27	0.035	-0.51	-0.006429	0.001684	0.030567	-0.020015	-0.001558
2016-05-20	0.035	0.11	NaN	NaN	NaN	NaN	NaN
2016-05-13	0.035	-1.77	NaN	NaN	NaN	NaN	NaN

C. Calculated factor returns

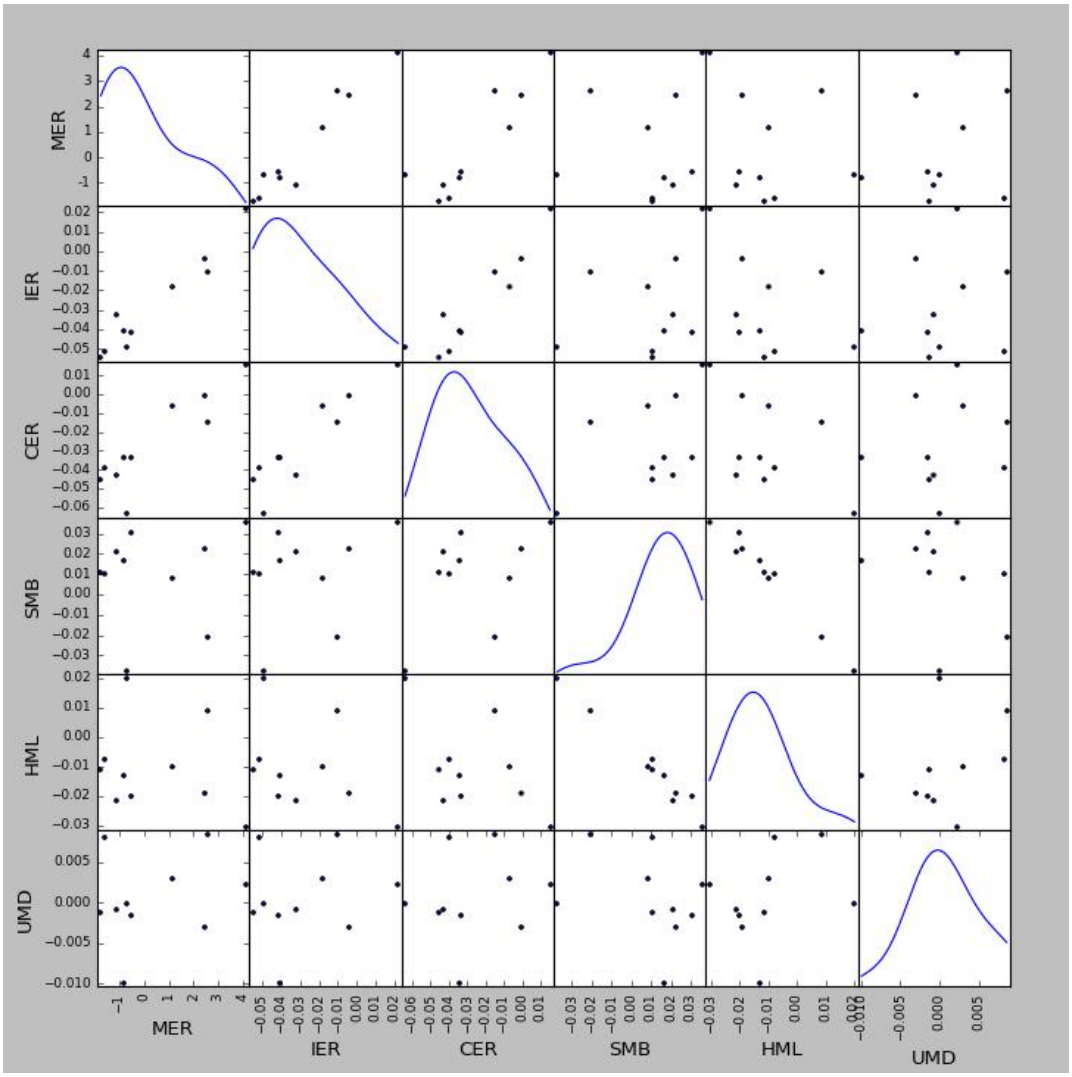
```
                    Robust linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   10
Model:                            RLM   Df Residuals:                       3
Method:                          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 .
```

D. Regression analysis

	Market Excess Return	Industry Excess Return	Concept Excess Return	SMB	HML	UMD
Market Excess Return	1.000000	0.967117	0.891409	0.138886	-0.209613	0.246429
Industry Excess Return	0.967117	1.000000	0.921257	0.313690	-0.400449	0.181979
Concept Excess Return	0.891409	0.921257	1.000000	0.485833	-0.531329	0.163691
SMB	0.138886	0.313690	0.485833	1.000000	-0.986599	-0.316509
HML	-0.209613	-0.400449	-0.531329	-0.986599	1.000000	0.294711
UMD	0.246429	0.181979	0.163691	-0.316509	0.294711	1.000000

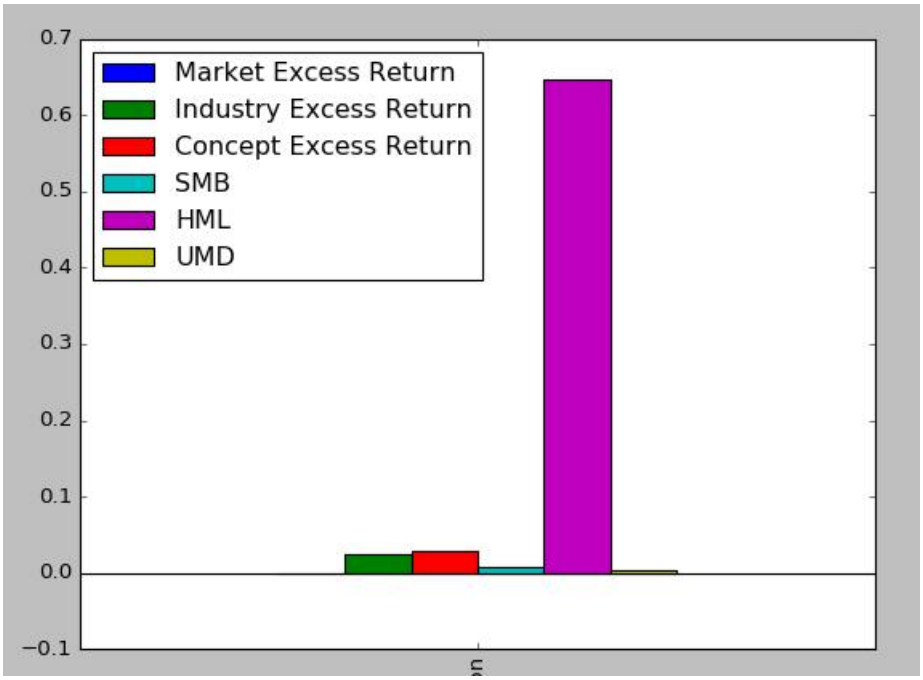
E. Factor correlation matrix



F. Covariance chart

*MER is short for Market Excess Return, IER for Industry Excess Return, CER for Concept Excess Return

	Market Excess Return	Industry Excess Return	Concept Excess Return	SMB	HML	UMD
risk contribution	-4.348519e-07	0.024082	0.029314	0.007396	0.646692	0.003979



G. Risk contribution charts

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)

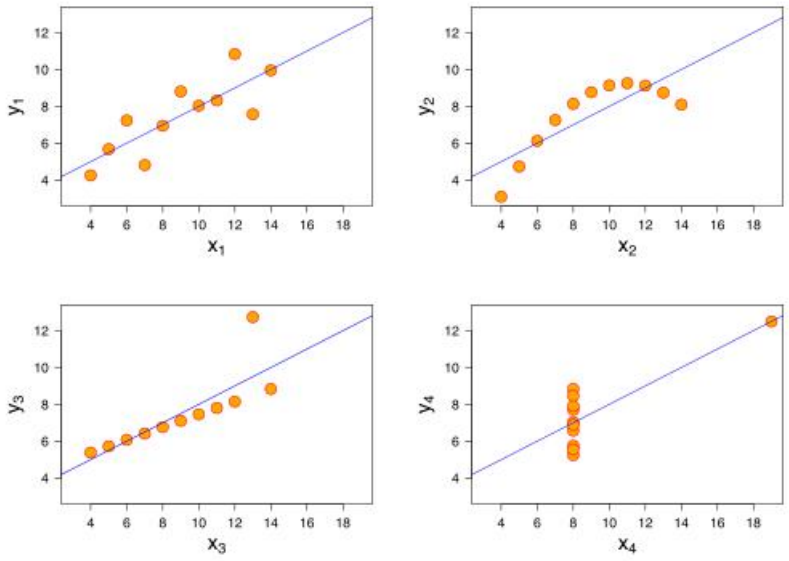
z-stats measures the deviation of the estimated coefficient from its hypothetical value, which is 0 in this case, in unit of standard error. In the graph, z-stats of market excess return is 1.97, meaning that the coefficient of market excess return is 1.97 standard error from 0. The more z-stats deviate 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:



[18]

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.

[18] Anscombe, Francis J. (1973) Graphs in statistical analysis. American Statistician, 27, 17–21

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.

7. 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.

8. Codes

```
import tushare as ts
import numpy as py
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 *

ts.set_token('a76443ee238b9549846083d5535bae10b278fa59daddba4bb6dbae6b9f2b6d1a')
mkt=ts.Market()
```

```
CACHE_FOLDER = "./cache/"

def CacheConstructor(function):
    def CachedFunction(*args, **kargs):
        cached = True
        file_name = CACHE_FOLDER + function.__module__ + '.' + function.__name__ + '().hdf5'
        with pd.HDFStore(file_name, format='table') as storage:
            #key = 'h'+ str(args) + '___' + str(kargs).replace('-','_')
            #Build key and check for alpha_numerics:
            karg_list = []
            for k in sorted(kargs.keys()):
                karg_list.append(k)
                karg_list.append(kargs[k])
            key = 'k'
            for args_list in [args, karg_list]:
                for v in args_list:
                    if (not type(v) is str) or (not v.replace('-','').isalnum()):
                        raise NotImplementedError('None alphanumeric in the argument list.')
                    key += v.replace('-','_d') + '__' #FIXME
                key = key[:-1] + 'k'
            #import pdb; pdb.set_trace()
            if '/' + key not in storage.keys():
                    cached = False
        if not cached:
            data = function(*args, **kargs)
            with pd.HDFStore(file_name, format='table') as storage:
                storage.put(key, data)
        with pd.HDFStore(file_name, format='table') as storage:
            return storage.get(key)
    return CachedFunction

cached_get_h=CacheConstructor(ts.get_h_data)
cached_get_hist=CacheConstructor(ts.get_hist_data)
```

```
def program():
    global unit, ticker, start_date, end_date, ticker_list, weight_list, mode, frequency_for_rebalancing
    print('Please fill in the question, input q for quitting')
    mode=input('input y for portfolio, n for single stock')
    if mode=='n':
        unit=input('APlease indicate the time unit for calculating return, enter d for daily return, w for weekly return, m for monthly 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_date != 'q':
                end_date=input('Please indicate the end date (yyyy-mm-dd):')
                end=pd.to_datetime(end_date)
                if end_date!= 'q':
                    ticker=input('Please indicate the stock to investigate on, input ticker for one stock, industry name, or market code:')
    if mode == 'y' :
        unit=input('Please indicate the time unit for calculating return, enter d for daily return, w for weekly return, m for monthly return:')
        frequency_for_rebalancing=input('Please indicate the frequency for reblancing (d, w, m): ')
        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)
            end_date=input('Please indicate the end date (yyyy-mm-dd):')
            end=pd.to_datetime(end_date)
            if end_date!= 'q':
                ticker='0'
                while True:
                    ticker=input('Please indicate the stock to investigate on through ticker, input n for end of the portfolio ')
                    if ticker=='n':
                        break
                    else:
                        weight=input('Please indicate the weight allocated to the stock in decimal form, leave the weight for cash: ')
                        ticker_list.append(ticker)
                        weight_list.append(float(weight))

                #check if the weight is more than 1
                total_weight=0
                for every_weight in weight_list:
                    total_weight+=every_weight
                if total_weight>1:
                    print('Total weight is more than one, please resubmit the information.')
                    weight_list=[]
                    ticker_list=[]
```

```
#Setting variables

unit=''
ticker=''
start_date=''
end_date=''
mode=''
#start
#end

market_excess=0
SMB=0
HML=0
UMD=0
REV=0
return_mom=0
industry_mom=0

ticker_list=[]
weight_list=[]
frequency_for_rebalancing=''

data=pd.DataFrame()
```

```
def cal_return(ticker, start_date, end_date, unit, date_index_string_hyphen):
    data=ts.get_h_data(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=data_week.set_index('end date of week')
            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=data_month.set_index('end date of month')
            return data_month
    except:
        print('No data available for the indicated period.')
```

```
program()
```

```
Please fill in the question, input q for quitting
input y for portfolio, n for single stocky
Please indicate the time unit for calculating return, enter d for daily return, w for weekly return, m for monthly return:w
Please indicate the frequency for reblancing (d, w, m): w
Please indicate the start date (yyyy-mm-dd):2016-10-01
Please indicate the end date (yyyy-mm-dd):2017-01-01
Please indicate the stock to investigate on through ticker, input n for end of the portfolio 000001
Please indicate the weight allocated to the stock in decimal form, leave the weight for cash: 0.3
Please indicate the stock to investigate on through ticker, input n for end of the portfolio 000002
Please indicate the weight allocated to the stock in decimal form, leave the weight for cash: 0.4
Please indicate the stock to investigate on through ticker, input n for end of the portfolio n
```

```
#construct the main chart
if mode =='n':
    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)
    date_index=return_dependent_variable.index
    date_index_string=return_dependent_variable.index.to_series().apply(lambda x: x.strftime('%Y%m%d')).tolist()


if mode=='y':
    ticker=ticker_list[0]
    portfolio_close_price=pd.DataFrame()

    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_first=cal_return(ticker, start_date, end_date, unit,date_index_string_hyphen)
    date_index=return_dependent_variable_first.index
    date_index_string=return_dependent_variable_first.index.to_series().apply(lambda x: x.strftime('%Y%m%d')).tolist()
```

```
#Setting Variables
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
```

```
########################  Market Return & Risk Free Rate ##########################
print('Calculating market return...')
market_return=ts.get_hist_data('hs300',start_date,end_date,ktype=unit)
market_return['date']=pd.to_datetime(market_return.index)
market_return=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
Factor_Data['Risk Free Rate']=[rf]*len(date_index)
```

```
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]))+tickers[k]
    return tickers
```

```
################## Industry ####################
print('Calculating industry return...')
#Getting industry information
industry_classification=ts.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['c_name']==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.loc[date_index[each_date]][-1]
        except KeyError:
            continue
```

```
for each_date in range(len(date_index)-2):
    try:
        industry_stock_cap_one_day=mkt.StockFactorsOneDay(tradeDate=date_index_string[each_date],field='ASSI,ticker')
    except :
        industry_stock_cap_one_day=mkt.StockFactorsOneDay(tradeDate=date_index_string[each_date],field='ASSI,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']==list_of_company[each_company]]['ASSI'].__float__()
            all_companies_same_industry['market cap '+ date_index_string[each_date]][each_company]=market_cap_one_company
        except (ValueError,TypeError):
            continue
###!!!!!!!!! a lot of data of market cap is Nan, need to know the reason

#calculating weighted return
for each_date in range(len(date_index)-2):
    all_companies_same_industry['percentage '+ date_index_string[each_date]]=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['percentage '+ date_index_string[each_date]]
    *all_companies_same_industry['return '+ date_index_string[each_date]]

#Industry Return
for each_date in range(len(date_index)-2):
    Industry_Return=all_companies_same_industry['weighted return '+ date_index_string[each_date]].sum()
    Factor_Data['Industry Return'][each_date]=Industry_Return
```

```
################## Concept ####################
print('Calculating concept return...')

#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]

####concept return
for each_date in range(len(date_index_string)-2):
    all_companies_same_concept['return '+ date_index_string[each_date]]=py.nan
    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

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
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_hyphen)
    except TypeError:
        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:
            continue
```

```
for each_date in range(len(date_index)-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_of_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
###!!!!!!!! a lot of data of market cap is Nan, need to know the reason

#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['market cap '+ date_index_string[each_date]]/
    all_companies_same_concept['market cap '+ date_index_string[each_date]].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_date]]

#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
```

```
##################  SMB & HML  ###################
print('Calculating SMB & HML...')

x=0.3
for k in range(3,len(date_index_string)-2):
    try:
        all_stock_cap_PB_one_day=mkt.StockFactorsOneDay(tradeDate=date_index_string[k],field='ASSI,PB,ticker')
    except:
        all_stock_cap_PB_one_day=mkt.StockFactorsOneDay(tradeDate=date_index_string[k],field='ASSI,PB,ticker')


    all_stock_cap_PB_one_day=all_stock_cap_PB_one_day[all_stock_cap_PB_one_day['PB'].notnull()]
    all_stock_cap_PB_one_day=all_stock_cap_PB_one_day[all_stock_cap_PB_one_day['ASSI'].notnull()]

    number=all_stock_cap_PB_one_day.index.__len__()
    number_x=int(number*x)
    ###############  SMB  ###############
    sorted_stock_cap=all_stock_cap_PB_one_day.sort_values(by='ASSI',ascending=True).set_index(pd.Series(list(range(number))))

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

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

    small_tickers_cap=add_zeros(small_tickers_cap)
    big_tickers_cap=add_zeros(big_tickers_cap)

    #Getting Returns
    sorted_stock_cap['return']=py.nan
```

```
    #Getting 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-number_x,number):
        try:
            sorted_stock_cap['return'][z]=cal_return(big_tickers_cap[z-(number-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
```

```
    ############  HML  ############

    #Creating Sorting index and Sum
    sorted_stock_PB=sorted_stock_cap.sort_values(by='PB',ascending=True).set_index(pd.Series(list(range(number))))
    sorted_stock_PB['sorting index']=int(number*0.5)*['A']+(number-2*int(number*0.5))*[' ']+int(number*0.5)*['B']

    #Getting tickers
    small_tickers_PB=sorted_stock_PB.loc[list(range(int(number*0.5)))]['ticker'].tolist()
    big_tickers_PB=sorted_stock_PB.loc[list(range(number-int(number*0.5),number))]['ticker'].tolist()

    small_tickers_PB=add_zeros(small_tickers_PB)
    big_tickers_PB=add_zeros(big_tickers_PB)

    #Getting Returns
    for y in range(0,int(number*0.5)):
        if math.isnan(sorted_stock_PB['return'][y]):
            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]=py.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]=py.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
```

```
############## UMD #################
print('Calculating UMD...')
#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.nan

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

for each in all_tickers:
    try:
        each_stock_record=cal_return(each,start_date,end_date,unit,date_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]][-1].__float__()
        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_UMD_one_period(k, Momentum_Data_clean, date_index, date_index_string):
    Momentum_Data_clean.sort_values(by=date_index_string[k],ascending=False)

    all_tickers_new=Momentum_Data_clean.index.tolist()

    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_new:
        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.nan]+[small_cap]*number_momentum_x
    Momentum_Data_clean['percentage']=Momentum_Data_clean['market cap']/Momentum_Data_clean['sum']

    #Getting weighted return
    Momentum_Data_clean['weighted 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_UMD_one_period(k, Momentum_Data_clean, date_index, date_index_string)
```

```
if mode == 'y':
    print('Building portfolio returns...')
    for every_ticker in ticker_list:
        every_stock_in_portfolio=cal_return(every_ticker, start_date, end_date, unit,date_index_string_hyphen)
        portfolio_close_price[every_ticker+' close']=every_stock_in_portfolio['close']

    #set index
    if unit=='w':
        portfolio_close_price['end date of week']=date_index_string_hyphen
        portfolio_close_price=portfolio_close_price.set_index('end date of week')
    if unit=='m':
        portfolio_close_price['end date of month']=date_index_string_hyphen
        portfolio_close_price=portfolio_close_price.set_index('end date of month')

    #get the date for rebalancing
    rebalancing_date=ts.get_hist_data(ticker_list[0],start_date,end_date,ktype=frequency_for_rebalancing).index[::-1]

    #initialise the weight chart
    weight_chart=pd.DataFrame()
    weight_chart['column']=['designated weight','weight after rebalancing','capital reallocation','total capital','new unit']
    weight_chart=weight_chart.set_index('column')

    total_weight=0
    for every_weight in weight_list:
        total_weight+=every_weight

    cash_weight=1-total_weight

    for number in range(len(ticker_list)):
        weight_chart[ticker_list[number]]=[weight_list[number],weight_list[number],0,10000/cash_weight*weight_list[number],
                                           10000/cash_weight*weight_list[number]/portfolio_close_price[ticker_list[number]+' close'][rebalancing_date[0]]]

    weight_chart['cash']=[cash_weight, cash_weight,0,10000,'N/A']
```

```
#initialise captital & cash column in the main chart
total_cap=0
for every_ticker in ticker_list:
    portfolio_close_price[every_ticker+' asset capital']=weight_chart[every_ticker][3]
    total_cap+=weight_chart[every_ticker][3]
portfolio_close_price['cash']=10000
portfolio_close_price['total capital']=total_cap+10000

#rebalancing algo functions

#generate a list of delta weight
def cal_delta_weight(ticker_list, weight_chart):
    delta_weight=[]
    for every_ticker in ticker_list:
        individual_weight=weight_chart[every_ticker]['weight after rebalancing']-weight_chart[every_ticker]['designated weight']
        delta_weight.append(individual_weight)
    individual_weight=weight_chart['cash']['weight after rebalancing']-weight_chart['cash']['designated weight']
    delta_weight.append(individual_weight)
    return delta_weight
```

```
    #initilise the list
    ticker_cash_list=ticker_list+['cash']


    #rebalancing algo: for each rebalancing date
    for the_date in rebalancing_date[1::]:

    #update new weight and new capital in weight chart
        new_total_capital=0
        for every_ticker in ticker_list:
            weight_chart[every_ticker][3]=weight_chart[every_ticker][4]*portfolio_close_price[every_ticker+' close'][the_date]
            new_total_capital+=weight_chart[every_ticker][3]
        new_total_capital+=weight_chart['cash'][3]

        new_total_weight=0
        for every_ticker in ticker_list:
            weight_chart[every_ticker][1]=weight_chart[every_ticker][3]/new_total_capital
            new_total_weight+=weight_chart[every_ticker][1]
        weight_chart['cash'][1]=1-new_total_weight

    #update total capital in the main chart
        portfolio_close_price['total capital'][the_date]=new_total_capital

    #initilise the list
    ticker_cash_list=ticker_list+['cash']


    #rebalancing algo: for each rebalancing date
    for the_date in rebalancing_date[1::]:

    #update new weight and new capital in weight chart
        new_total_capital=0
        for every_ticker in ticker_list:
            weight_chart[every_ticker][3]=weight_chart[every_ticker][4]*portfolio_close_price[every_ticker+' close'][the_date]
            new_total_capital+=weight_chart[every_ticker][3]
        new_total_capital+=weight_chart['cash'][3]

        new_total_weight=0
        for every_ticker in ticker_list:
            weight_chart[every_ticker][1]=weight_chart[every_ticker][3]/new_total_capital
            new_total_weight+=weight_chart[every_ticker][1]
        weight_chart['cash'][1]=1-new_total_weight

    #update total capital in the main chart
        portfolio_close_price['total capital'][the_date]=new_total_capital
```

```
    #the rebalancing algo, update every item in weight chart
        delta_weight=cal_delta_weight(ticker_list, weight_chart)
        index_of_max=find_max(delta_weight,0)
        while py.fabs(delta_weight[index_of_max])>0.001:
            if delta_weight[index_of_max]>0:
                index_of_oppo_max=find_max(delta_weight,-1)
            if delta_weight[index_of_max]<0:
                index_of_oppo_max=find_max(delta_weight,1)
            weight_chart[ticker_cash_list[index_of_max]][1]+=delta_weight[index_of_oppo_max]
            weight_chart[ticker_cash_list[index_of_oppo_max]][1]=weight_chart[ticker_cash_list[index_of_oppo_max]][0]
            cap_to_reallocate=portfolio_close_price['total capital'][the_date]*delta_weight[index_of_oppo_max]
            weight_chart[ticker_cash_list[index_of_max]][2]+=cap_to_reallocate
            weight_chart[ticker_cash_list[index_of_oppo_max]][2]-=cap_to_reallocate
            weight_chart[ticker_cash_list[index_of_max]][3]+=cap_to_reallocate
            weight_chart[ticker_cash_list[index_of_oppo_max]][3]-=cap_to_reallocate
            if index_of_max!=len(delta_weight)-1:
                weight_chart[ticker_cash_list[index_of_max]][4]+=cap_to_reallocate/portfolio_close_price[ticker_cash_list[index_of_max]
                                                                                                        +' close'][the_date]
            if index_of_oppo_max!=len(delta_weight)-1:
                weight_chart[ticker_cash_list[index_of_oppo_max]][4]-=cap_to_reallocate/portfolio_close_price[ticker_cash_list[index_of_oppo_max]+
                                                                                                              ' close'][the_date]
            delta_weight[index_of_max]+=delta_weight[index_of_oppo_max]
            delta_weight[index_of_oppo_max]=0
            print(weight_chart)
            for every_item in ticker_cash_list:
                weight_chart[every_item][2]=0
            index_of_max=find_max(delta_weight,0)

    #update the main chart
        for every_ticker in ticker_list:
            portfolio_close_price[every_ticker+' asset capital'][the_date]=weight_chart[every_ticker][3]
        portfolio_close_price['cash'][the_date]=weight_chart['cash'][3]
```

```
    #transform the portfolio main chart into return_dependent_variable
    portfolio_close_price['portfolio return']=portfolio_close_price['total capital'].pct_change(-1)

    return_dependent_variable=pd.DataFrame(index=portfolio_close_price.index)
    return_dependent_variable['return']=py.nan
    return_dependent_variable['return']=portfolio_close_price['portfolio return']
```

```
#####################  Linear Regression  ####################
#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']=py.nan
#Complete_Data['stock return'][:]=return_dependent_variable.iloc[0:-2, -1]
#Complete_Data['dependent variable']=Complete_Data['stock return']-Complete_Data['Risk Free Rate']

# Method 2, Robust analysis
Complete_Data=Complete_Data[['Market Excess Return','Industry Excess Return','Concept Excess Return','SMB','HML','UMD']]
return_dependent_variable2=return_dependent_variable.copy()
return_dependent_variable2.iloc[:,-1]=return_dependent_variable2.iloc[:,-1]-rf
Dependent_Variable_array=return_dependent_variable2.iloc[:-2,-1].as_matrix()
```

```
print('Robust linear regression with no lag:')
Complete_Data_array=Complete_Data.as_matrix()
Complete_Data_array=sm.add_constant(Complete_Data_array)

return_dependent_variable2=return_dependent_variable.copy()
return_dependent_variable2.iloc[:,-1]=return_dependent_variable2.iloc[:,-1]-rf
Dependent_Variable_array=return_dependent_variable2.iloc[:-2,-1].as_matrix()

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

regression_result=regression_model.fit()
print(regression_result.summary())
```

```
##################  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')

print('Covariance Scatter Plot, robust linear regression')
matplotlib.pyplot.show()
```

```
################ Risk Contribution Chart  ########################
Covariance_Matrix=Complete_Data[['Market Excess Return','Industry Excess Return','Concept Excess Return','SMB','HML','UMD']]
Covariance_Matrix=Covariance_Matrix.cov()
se=regression_result.bse
coef=regression_result.params

#Calculate porfolio 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)):
    for column in range(row+1,len(Covariance_Matrix.iloc[row])):
        each=2*Covariance_Matrix.iloc[row,column]*coef[row+1]*coef[column+1]
        portfolio_risk=portfolio_risk+each
portfolio_risk=portfolio_risk**0.5
```

```
# 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):
        risk_x=0
        li=list(range(1,len(se)))
        li.remove(k+1)

        for item in li:
            each=(se[item]*coef[item])**2
            risk_x=risk_x+each

        li2=list(range(len(Covariance_Matrix.columns)))
        li2.remove(k)

        for row in li2:
            for column in range(row+1,len(Covariance_Matrix.iloc[row])):
                each=2*Covariance_Matrix.iloc[row,column]*coef[row+1]*coef[column+1]
                risk_x=risk_x+each
        risk_x

        y=Symbol('y')

        summation=0
        for column in range(k+1,len(Covariance_Matrix.iloc[row])):
            each=2*Covariance_Matrix.iloc[row,column]*y*coef[column+1]
            summation=summation+each

        risk=(y**2*se[k+1]**2+summation+risk_x)**0.5
        return risk.diff(y).replace(y,x)
    I = quad(integrand, 0, coef[k+1], args=(k,se,coef,Covariance_Matrix,portfolio_risk))
    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)

print('Risk Contribution Chart, robust linear regression')

print(risk_contribution_chart)
```

```
##Risk Contribution Chart in graphics
risk_contribution_chart.plot.bar(); plt.axhline(0, color='k')
matplotlib.pyplot.show()
```

```
def find_max(a,b,c,d,e,f):
    li=[a,b,c,e,d,f]
    maximum=a
    for x in range(1,6):
        if li[x]>maximum:
            maximum=li[x]
    return maximum
```

```
lag_number=len(Complete_Data.index)-8
column_list=Complete_Data.columns
count=0
Complete_Data_2=Complete_Data.copy()

max_1=0
max_2=0
max_3=0
max_4=0
max_5=0
max_6=0
max_rsquared=0

for a in range(lag_number):
    for b in range(lag_number):
        for c in range(lag_number):
            for d in range(lag_number):
                for e in range(lag_number):
                    for f in range(lag_number):
                        Complete_Data_2=Complete_Data.copy()
                        Dependent_Variable_array_2=Dependent_Variable_array.copy()
                        Complete_Data_2[column_list[0]]=Complete_Data_2[column_list[0]].shift(-a)
                        Complete_Data_2[column_list[1]]=Complete_Data_2[column_list[1]].shift(-b)
                        Complete_Data_2[column_list[2]]=Complete_Data_2[column_list[2]].shift(-c)
                        Complete_Data_2[column_list[3]]=Complete_Data_2[column_list[3]].shift(-d)
                        Complete_Data_2[column_list[4]]=Complete_Data_2[column_list[4]].shift(-e)
                        Complete_Data_2[column_list[5]]=Complete_Data_2[column_list[5]].shift(-f)

                        max_lag=find_max(a,b,c,d,e,f)
```

```
                        if max_lag>0:
                            Complete_Data_2=Complete_Data_2[:-max_lag]
                            Dependent_Variable_array_2=Dependent_Variable_array_2[:-max_lag]

                        Complete_Data_array=Complete_Data_2.as_matrix()
                        Complete_Data_array=sm.add_constant(Complete_Data_array)
                        regression_model = sm.OLS(Dependent_Variable_array_2, Complete_Data_array, M=sm.robust.norms.HuberT())
                        regression_result=regression_model.fit()

                        if regression_result.rsquared>max_rsquared:
                            max_1=a
                            max_2=b
                            max_3=c
                            max_4=d
                            max_5=e
                            max_6=f
                            max_rsquared=regression_result.rsquared

print('The lag combination that maximises r-square is', max_1, ',' , max_2 , ',' , max_3 , ',', max_4 , ',' ,max_5 , ',' , max_6 ,'.')
Complete_Data_2=Complete_Data.copy()
Dependent_Variable_array_2=Dependent_Variable_array.copy()
Complete_Data_2[column_list[0]]=Complete_Data_2[column_list[0]].shift(-max_1)
Complete_Data_2[column_list[1]]=Complete_Data_2[column_list[1]].shift(-max_2)
Complete_Data_2[column_list[2]]=Complete_Data_2[column_list[2]].shift(-max_3)
Complete_Data_2[column_list[3]]=Complete_Data_2[column_list[3]].shift(-max_4)
Complete_Data_2[column_list[4]]=Complete_Data_2[column_list[4]].shift(-max_5)
Complete_Data_2[column_list[5]]=Complete_Data_2[column_list[5]].shift(-max_6)


max_lag=find_max(max_1,max_2,max_3,max_4,max_5,max_6)
```

```
if max_lag>0:
    Complete_Data_2=Complete_Data_2[:-max_lag]
    Dependent_Variable_array_2=Dependent_Variable_array_2[:-max_lag]

Complete_Data_array=Complete_Data_2.as_matrix()

Complete_Data_array=sm.add_constant(Complete_Data_array)
regression_model = sm.OLS(Dependent_Variable_array_2, Complete_Data_array, M=sm.robust.norms.HuberT())
regression_result=regression_model.fit()
print(regression_result.summary())
```

```
#####################   Correlation Matrix  ######################
Correlation_Matrix=Complete_Data_2[['Market Excess Return','Industry Excess Return','Concept Excess Return','SMB','HML','UMD']]
Correlation_Matrix=Correlation_Matrix.corr()

print('Correlation Matrix, with maximised r-squred value')
print(Correlation_Matrix)
```

```
##################  Covariance Scatter Plot  ######################
Complete_Data_Plot=Complete_Data_2.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')

print('Covariance Scatter Plot, with maximised r-squred value')
matplotlib.pyplot.show()
```

```
################ Risk Contribution Chart  ########################
Covariance_Matrix=Complete_Data_2[['Market Excess Return','Industry Excess Return','Concept Excess Return','SMB','HML','UMD']]
Covariance_Matrix=Covariance_Matrix.cov()
se=regression_result.bse
coef=regression_result.params

#Calculate porfolio 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)):
    for column in range(row+1,len(Covariance_Matrix.iloc[row])):
        each=2*Covariance_Matrix.iloc[row,column]*coef[row+1]*coef[column+1]
        portfolio_risk=portfolio_risk+each
portfolio_risk=portfolio_risk**0.5
```

```
# 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):
        risk_x=0
        li=list(range(1,len(se)))
        li.remove(k+1)

        for item in li:
            each=(se[item]*coef[item])**2
            risk_x=risk_x+each

        li2=list(range(len(Covariance_Matrix.columns)))
        li2.remove(k)

        for row in li2:
            for column in range(row+1,len(Covariance_Matrix.iloc[row])):
                each=2*Covariance_Matrix.iloc[row,column]*coef[row+1]*coef[column+1]
                risk_x=risk_x+each
        risk_x

        y=Symbol('y')

        summation=0
        for column in range(k+1,len(Covariance_Matrix.iloc[row])):
            each=2*Covariance_Matrix.iloc[row,column]*y*coef[column+1]
            summation=summation+each

        risk=(y**2*se[k+1]**2+summation+risk_x)**0.5
        return risk.diff(y).replace(y,x)
    I = quad(integrand, 0, coef[k+1], args=(k,se,coef,Covariance_Matrix,portfolio_risk))
    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)

print('Risk Contribution Chart, with maximised r-squred value')
print(risk_contribution_chart)
```

```
##Risk Contribution Chart in graphics
risk_contribution_chart.plot.bar(); plt.axhline(0, color='k')
matplotlib.pyplot.show()
```