Target Beta Portfolio Construction

Prof. Danny Tan

Session: Thursday 8pm

Kaizhen Tan

Shuining Yuan

Lu Jiang

Jingtao Yang

1.Project Scope

This project is conducted to construct a 10-stock portfolio from the stock pool which consists of 3000 cusips using portfolio management methods. After data cleaning and processing, we got our sample base -- stock database. We first split the stock database into train(from 1/1/2013 to 1/1/2014) and test(from 1/1/2014 to 1/1/2015) subsets. On the train subset we computed beta value for each stock and picked 10 stocks so that their equally weighted average beta is 4.8. Then we used four methods to determine the weights of the 10 stocks: Equal Weights, Weighted by Beta Value, Minimum Variance Portfolio and Efficient Frontier Theory(EF). We tested our portfolio and got P&L, volatility, Sharpe Ratio to evaluate 4 portfolios' performances. The results show that among all methods, EF portfolio performed the best with highest return and sharpe ratio 1.07 but it had the biggest difference of train beta and test beta. Therefore the efficient frontier portfolio is the most stable portfolio by considering the relatively same P&L of the 4 portfolios.

2.Procedure

The ***formulae*** we used to calculate beta value is:

$$beta = cov(Rm, Rc)/var(Rc)$$

Our ***approach*** to choose 10 stocks:

Our 10 stocks are chosen from the training set in that 5 stocks are closely above the target beta(4.8) and 5 stocks are closely below the target beta in order to have their average beta equal to 4.8.

We used 4 methods to weigh our stocks for the portfolio; in each of them we tested their P&L, Sharpe Ratios and volatilities in order to evaluate our portfolio performance. The assumption we made for this project is that among all four methods that we stated above, Efficient Frontier method is better than others in terms of beta, Profit and Loss, Sharpe Ratio and stable volatility combination. The parameters that we utilize throughout the process contain equity return and beta.

2.1 Equal Weights

The first method that we utilize is equal weighted method --- betas are equally weighted when computing parameters. We set each stock's weight as 0.1, assuming every stock is equally important.

➢ *Result:*

train beta is 4.815967202770259

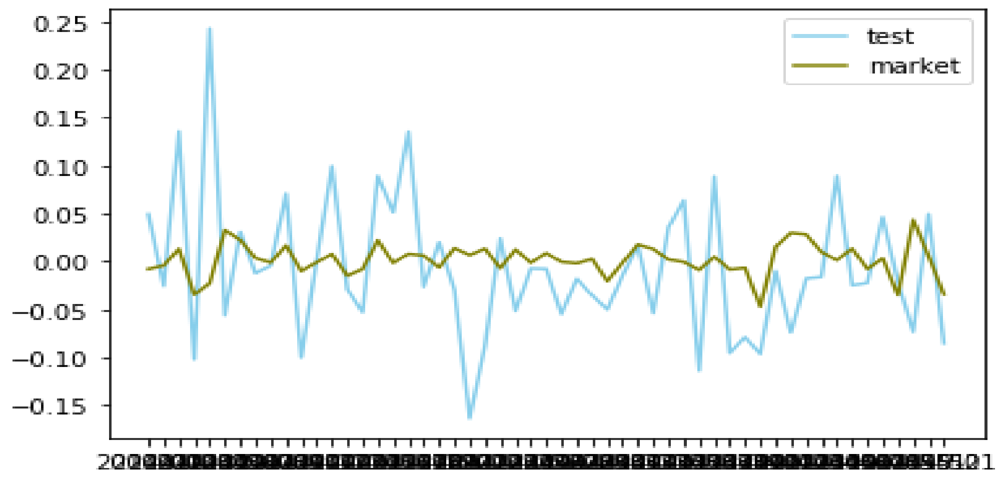
test beta is 0.4988343713080283

Difference between train beta and test beta: 4.31713283

pnl is -0.006878921854455629

volatility is 0.0731434914285682

sharpe ratio is -0.6758549173039377



2.2 Beta weighted-average weights

We set the weights based on the theory that high-beta stocks tend to be riskier but provide the potential for higher returns and low-beta stocks pose less riskier but typically yield lower returns. This is the profit-based strategy to construct the portfolio.

➢ *Result:*

train beta is 4.8369590536956455

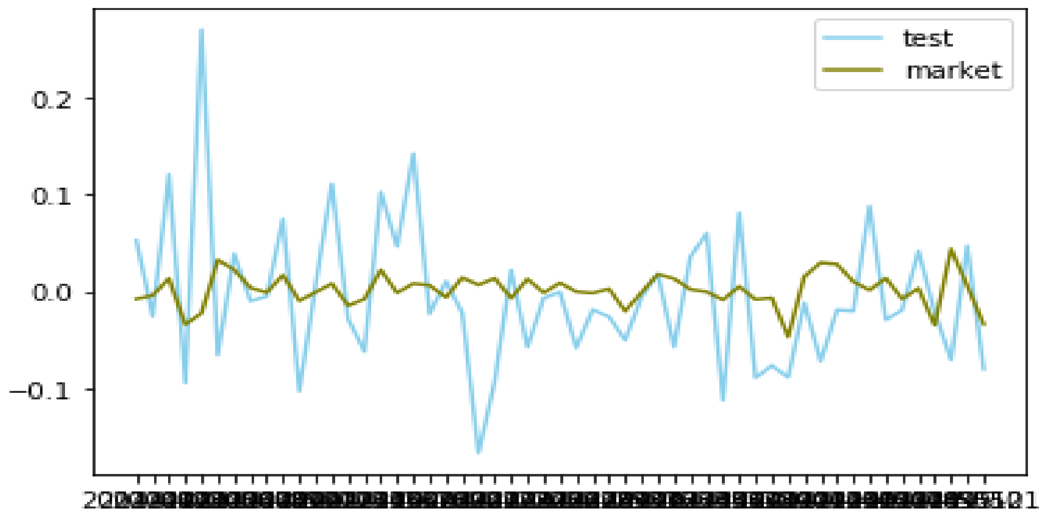
test beta is 0.4180481619955334

Difference between train beta and test beta: 4.4189108

pnl is -0.006048012942512105

volatility is 0.07530653147299314

sharpe ratio is -0.5773897784487156



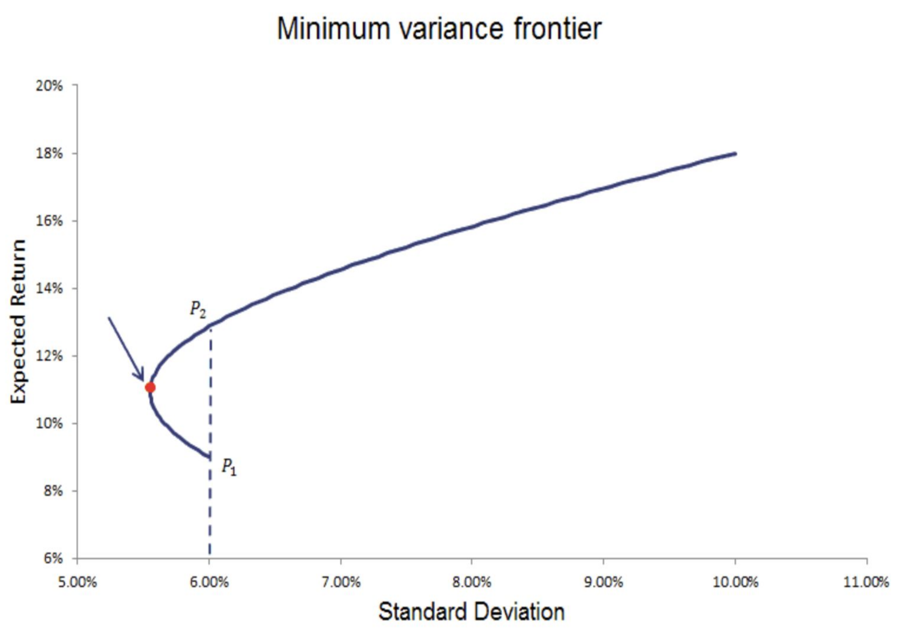
2.3 Minimum variance portfolio

MVP is a risk-based approach to portfolio construction which uses only the risk measure

The weights $\omega_{MV} = (\omega_{MV,i}, \ldots, \omega_{MV,N})'$ *of the min variance portfolio are given as*

$$\omega_{MV} = \frac{\Sigma^{-1}1}{1\Sigma^{-1}1}$$

Where Σ *is the covariance matrix.*



➢ *Result:*

train beta is 4.5696231233626206

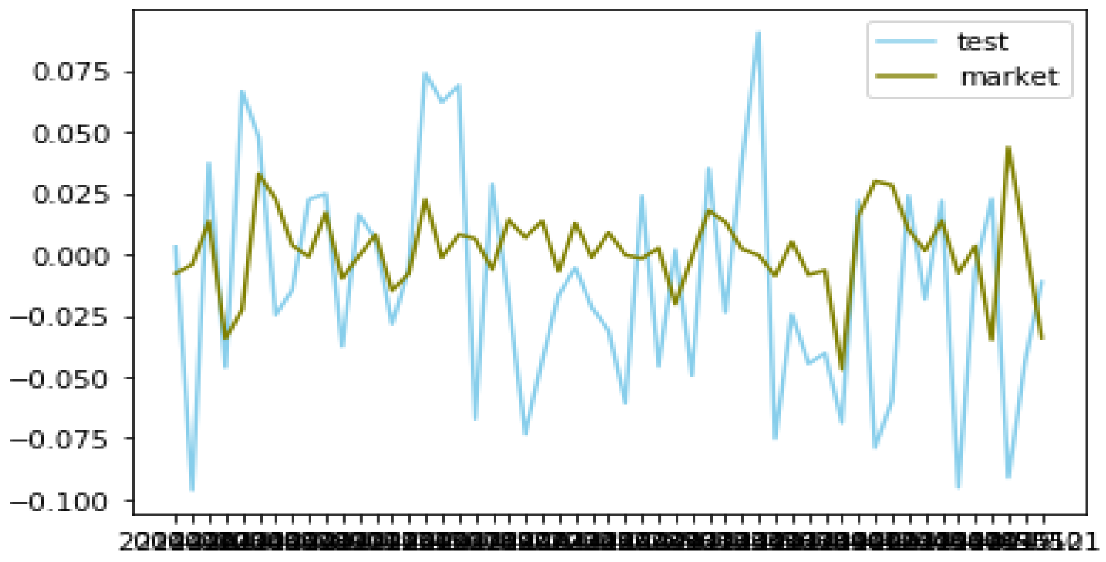
test beta is 0.11371787640326166

Difference between train beta and test beta: 4.45590525

pnl is -0.011932924741322610

volatility is 0.045858049410900780

sharpe ratio is -1.865281447055294



2.4 Efficient Frontier

We use the efficient frontier method to construct portfolios in order to achieve the lowest risk for a given level of expected return. We first use market return to calculate weights and then use the expected return of 0.8 to calculate weights. In addition, we set various parameter 'μ' as market return and we adjust this parameter to get the results we desire.

2.4.1 Set parameter 'μ' as 0.0033 market return calculated based on test set

➢ *Result:*

train beta is 4.7431669160125045

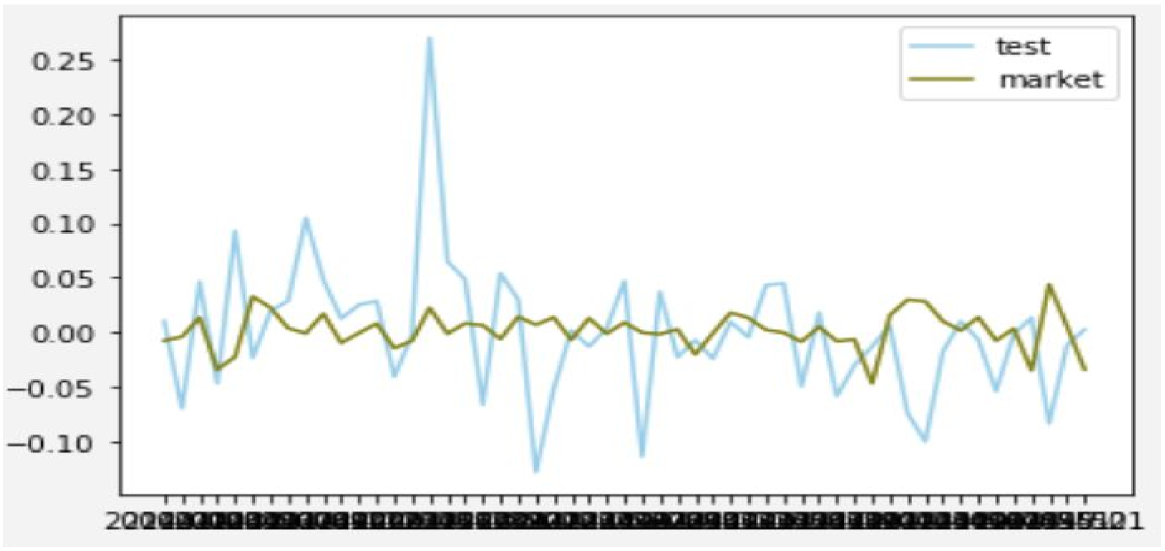
test beta is -0.051297257622065585

Difference between train beta and test beta: 4.79296616

pnl is 6.264150943396 84e-05

volatility is 0.06118879229976737

sharpe ratio is 0.007382536569612222



2.4.2 Set parameter 'μ' as 0.8 which we are expected to reach

➢ *Result:*

train beta is 4.960635424459918

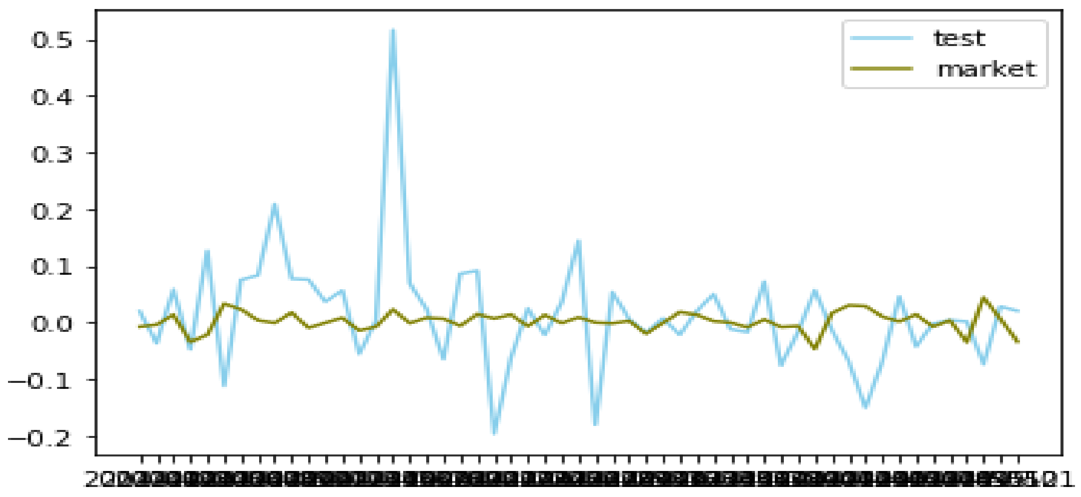
test beta is -0.25807846552233926

Difference between train beta and test beta: 5.21871388

pnl is 0.015094339622641518

volatility is 0.10214700435140785

sharpe ratio is 1.073672850959399



3.Table design

3.1 Database Processing

Firstly, we use data from “symbol.csv” which contains 3,000 stocks’ names. We download 3,000 stocks from YahooFinance using python. When we download, we extract the data dated from 2010-01-01 to 2017-01-01 with daily frequency and stock has enough time contained in the market. However, many stocks cannot be accessed from YahooFinancials by their names.Then we print the name list which is contained in the original 3000 cusips. This is the first step to drop data from raw data. Now we obtain ‘ticker_list’ to be used for later.

Secondly, we normalized datetime and got the date from 2012-12-31 to 2015-01-06 from the downloaded documents. There are two items we need to explain. First of all, downloading is time-consuming and space-wasting for computers. However if we need to debug and want everyone in the group to use the same coding method, the fastest way is to download all data at the beginning. Furthermore, as we want to use week dates, we need to transform daily data to weekly data. (1) We cannot divide the figures by each other every 4 data from our downloaded document because some holidays occur in the year. We similarly cannot define the trading week the same way as we did for nature week in terms of equal 5 days. (2) As we know, a certain year might have 252 trading days but not every year does . For instance, 252 trading days take place from 2012 to 2014, but other years, such as 2015, may have different trading days. Due to this, we wish to apply our method to universal years. Accordingly, we cannot calculate weekly return by assuming there come 252 trading days in a year. (3) Thus, in our method, we calculate weekly figures by using daily data.

Thirdly, we need to merge Market return with the whole dataframe return. Because we use python function set() at the beginning, we cannot be certain whether the last dataframe return is from the market or not. In order to solve the problem, we merge market return when we output all stocks return.

Fourthly, we seperated train data and test data. Before separating, we need to test again if there is any NaN value. We apply 2013 data to train and 2014 data to test.

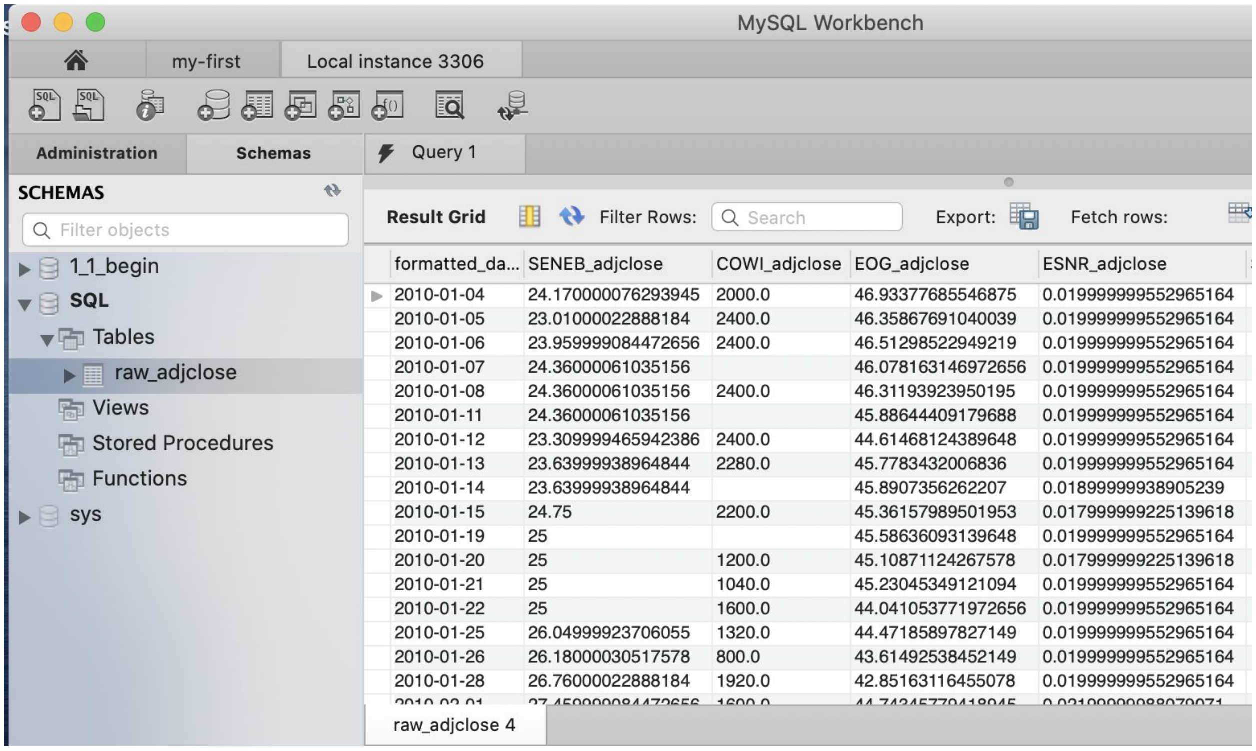
Fifthly, we acquired the covariance between every stock by calculating market covariance of dataframe and getting its last row. In addition, the last number of last row is var(market). Thus, we have beta now.

3.2 Database construction

Depending on our stock pool, the data we really need and the result of our portfolio, we constructed 3 tables to save our important sample base and output which are fundamental and derivable for analysis.

➢ **3.2.1 Raw-adjclose**

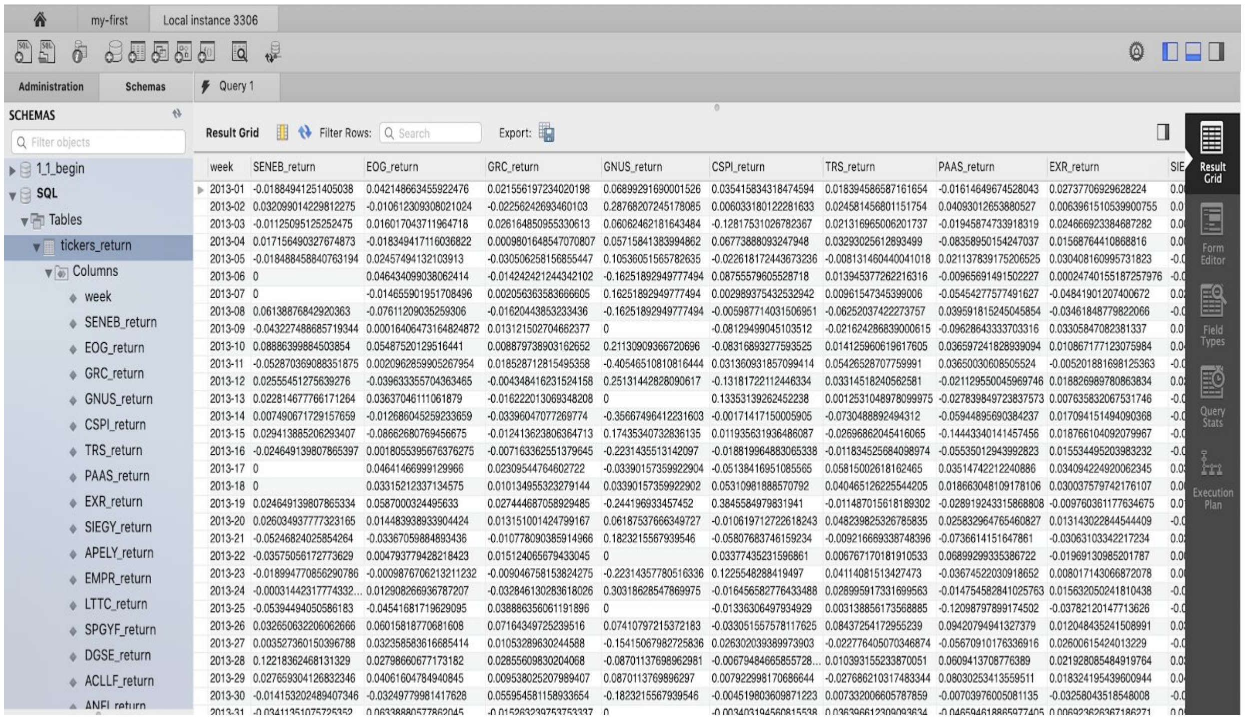
Because we only need the ‘adjclose’, ‘date’ and ‘cusip-name’ in the table downloaded from yahoo finance to conduct analysis, we built the original sample database ‘Raw-adjclose’ containing stock name, trade date and adjusted close price information.



formatted_da...	SENEB_adjclose	COWI_adjclose	EOG_adjclose	ESNR_adjclose
2010-01-04	24.170000076293945	2000.0	46.93377685546875	0.019999999552965164
2010-01-05	23.01000022888184	2400.0	46.35867691040039	0.019999999552965164
2010-01-06	23.959999084472656	2400.0	46.51298522949219	0.019999999552965164
2010-01-07	24.36000061035156		46.078163146972656	0.019999999552965164
2010-01-08	24.36000061035156	2400.0	46.31193923950195	0.019999999552965164
2010-01-11	24.36000061035156		45.88644409179688	0.019999999552965164
2010-01-12	23.309999465942386	2400.0	44.61468124389648	0.019999999552965164
2010-01-13	23.63999938964844	2280.0	45.7783432006836	0.019999999552965164
2010-01-14	23.63999938964844		45.8907356262207	0.018999999389052391
2010-01-15	24.75	2200.0	45.36157989501953	0.017999999225139618
2010-01-19	25		45.58636093139648	0.019999999552965164
2010-01-20	25	1200.0	45.10871124267578	0.017999999225139618
2010-01-21	25	1040.0	45.23045349121094	0.019999999552965164
2010-01-22	25	1600.0	44.041053771972656	0.019999999552965164
2010-01-25	26.04999923706055	1320.0	44.47185897827149	0.019999999552965164
2010-01-26	26.18000030517578	800.0	43.61492538452149	0.019999999552965164
2010-01-28	26.76000022888184	1920.0	42.85163116455078	0.019999999552965164

➢ **3.2.2 Stock-return**

We calculated the weekly returns for our stock database and saved it into ‘Tickers-return’ database which consists of cusips weekly return and trade week date

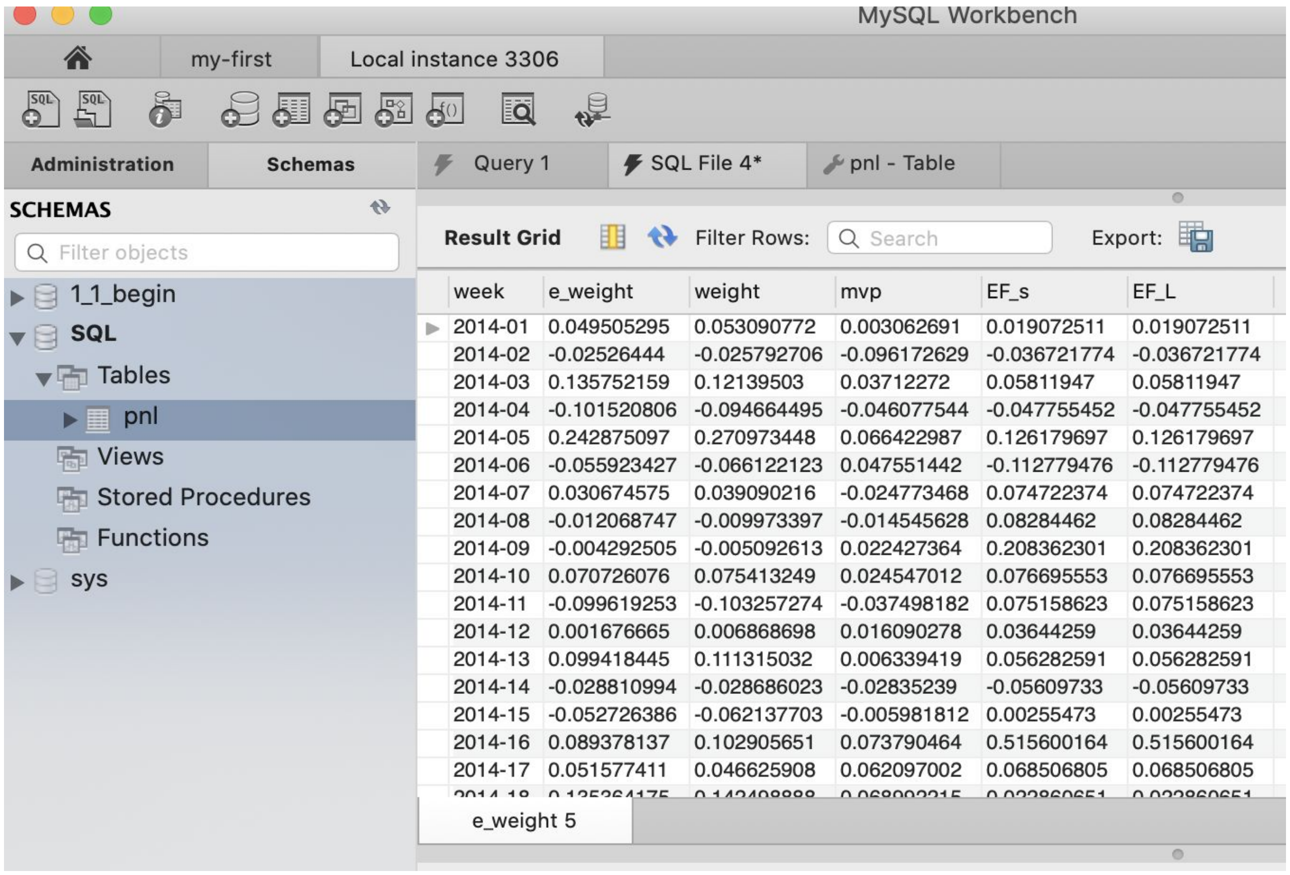


week	SENEB_return	EOG_return	GRC_return	GNUS_return	CSPI_return	TRS_return	PAAS_return	EXR_return	SIE
2013-01	-0.01884941251405038	0.042148663455922476	0.021556197234020198	0.06899291690001526	0.035415834318474594	0.018394586587161654	-0.01614649674528043	0.02737706929628224	0.0
2013-02	0.032099014229812275	-0.010612309308021024	-0.02256242693460103	0.28768207245178085	0.006033180122281633	0.024581456801151754	0.04093012653880527	0.0063961510539900755	0.0
2013-03	-0.01125095125252475	0.016017043711964718	0.026164850955330613	0.06062462181643484	-0.12817531026782367	0.021316965006201737	-0.01945874733918319	0.024666923384687282	0.0
2013-04	0.017156490327674873	-0.018349417116036822	0.0009801648547070807	0.05715841383994862	0.06773888093247948	0.03293025612893499	-0.08358950154247037	0.01568764410868816	0.0
2013-05	-0.018488458840763194	0.02457494132103913	-0.030506258156855447	0.10536051565782635	-0.022618172443673236	-0.008131460440041018	0.021137839175206525	0.030408160995731823	-0.0
2013-06	0	0.046434099038062414	-0.014242421244342102	-0.16251892949777494	0.08755579605528718	0.013945377262216316	-0.00965691491502227	0.00024740155187257976	-0.0
2013-07	0	-0.014655901951708496	0.002056363583666605	0.16251892949777494	0.002989375432532942	0.00961547345399006	-0.05454277577491627	-0.04841901207400672	0.0
2013-08	0.06138876842920363	-0.07611209035259306	-0.01620443853233436	-0.16251892949777494	-0.005987714031506951	-0.06252037422273757	0.039591815245045854	-0.03461848779822066	-0.0
2013-09	-0.043227488685719344	0.00016406473164824872	0.013121502704662377	0	-0.08129499045103512	-0.021624286839000615	-0.09628643333703316	0.03305847082381337	0.0
2013-10	0.08886399884503854	0.05487520129516441	0.008879738903162652	0.21130909366720696	-0.08316893277593525	0.014125960619617605	0.036597241828939094	0.010867177123075984	0.0
2013-11	-0.052870369088351875	0.0020962859905267954	0.018528712815495358	-0.40546510810816444	0.031360931857099414	0.05426528707759991	0.03650030608505524	-0.005201881698125363	-0.0
2013-12	0.02555451275639276	-0.039633355704363465	-0.004348416231524158	0.25131442828090617	-0.13181722112446334	0.03314518240562581	-0.021129550045969746	0.018826989780863834	0.0
2013-13	0.022814677766171264	0.03637046111061879	-0.016222013069348208	0	0.13353139262452238	0.0012531048978099975	-0.027839849723837573	0.007635832067531746	-0.0
2013-14	0.007490671729157659	-0.012686045259233659	-0.03396047077269774	-0.35667496412231603	-0.00171417150005905	-0.0730488892494312	-0.05944895690384237	0.017094151494090368	-0.0
2013-15	0.029413885206293407	-0.08662680769456675	-0.012413623806364713	0.17435340732836135	0.011935631936486087	-0.02696862045416065	-0.14443340141457456	0.018766104092079967	-0.0
2013-16	-0.024649139807865397	0.0018055395676376275	-0.007163362551379645	-0.2231435513142097	-0.018819964883065338	-0.011834525684098974	-0.05535012943992823	0.015534495203983232	-0.0
2013-17	0	0.04641466999129966	0.02309544764602722	-0.03390157359922904	-0.05138416951085565	0.05815002618162465	0.03514742212240886	0.034094224920062345	0.0
2013-18	0	0.03315212337134575	0.010134955323279144	0.03390157359922902	0.05310981888570792	0.040465126225544205	0.018663048109178106	0.030037579742176107	0.0
2013-19	0.024649139807865334	0.0587000324495633	0.027444687058929485	-0.244196933457452	0.3845584979831941	-0.011487015618189302	-0.028919243315868808	-0.009760361177634675	0.0
2013-20	0.026034937777323165	0.014483938933904424	0.013151001424799167	0.06187537666349727	-0.010619712722618243	0.048239825326785835	0.025832964765460827	0.013143022844544409	-0.0
2013-21	-0.05246824025854264	-0.03367059884893436	-0.010778090385914966	0.1823215567939546	-0.05807683746159234	-0.009216669338748396	-0.0736614151647861	-0.03063103342217234	0.0
2013-22	-0.03575056172773629	0.004793779428218423	0.015124065679433045	0	0.03377435231596861	0.006767170181910533	0.06899299335386722	-0.01969130985201787	0.0
2013-23	-0.018994770856290786	-0.0009876706213211232	-0.009046758153824275	-0.22314357780516336	0.1225548288419497	0.04114081513427473	-0.03674522030918652	0.008017143066872078	0.0
2013-24	-0.00031442317774332...	0.012908266936787207	-0.032846130283618026	0.30318628547869975	-0.016456582776433488	0.028995917331699563	-0.014754582841025763	0.015632050241810438	-0.0
2013-25	-0.05394494050586183	-0.04541681719629095	0.038886356061191896	0	-0.01336306497934929	0.003138856173568885	-0.12098797899174502	-0.03782120147713626	-0.0
2013-26	0.032650632206062666	0.06015818770681608	0.07164349725239516	0.07410797215372183	-0.033051557578117625	0.08437254172955239	0.09420794941327379	0.012048435241508991	0.0
2013-27	0.003527360150396788	0.032358583616685414	0.01053289630244588	-0.15415067982725836	0.026302039389973903	-0.022776405070346874	-0.05670910176336916	0.02600615424013229	-0.0
2013-28	0.12218362468131329	0.02798660677173182	0.02855609830204068	-0.08701137698962981	-0.00679484665855728...	0.010393155233870051	0.0609413708776389	0.021928085484919764	0.0
2013-29	0.027659304126832346	0.04061604784940845	0.009538025207989407	0.0870113769896297	0.007922998170686644	-0.027686210317483344	0.08030253413559511	0.018324195439600944	0.0
2013-30	-0.014153202489407346	-0.03249779981417628	0.055954581158933654	-0.1823215567939546	-0.004519803609871223	0.007332006605787859	-0.00703976005081135	-0.03258043518548008	-0.0
2013-31	-0.03411351075725352	0.06338880577862045	-0.01526323975375337	0	-0.003403194560815538	0.036396612309093634	-0.046594618865977405	0.006923626367186271	0.0

.

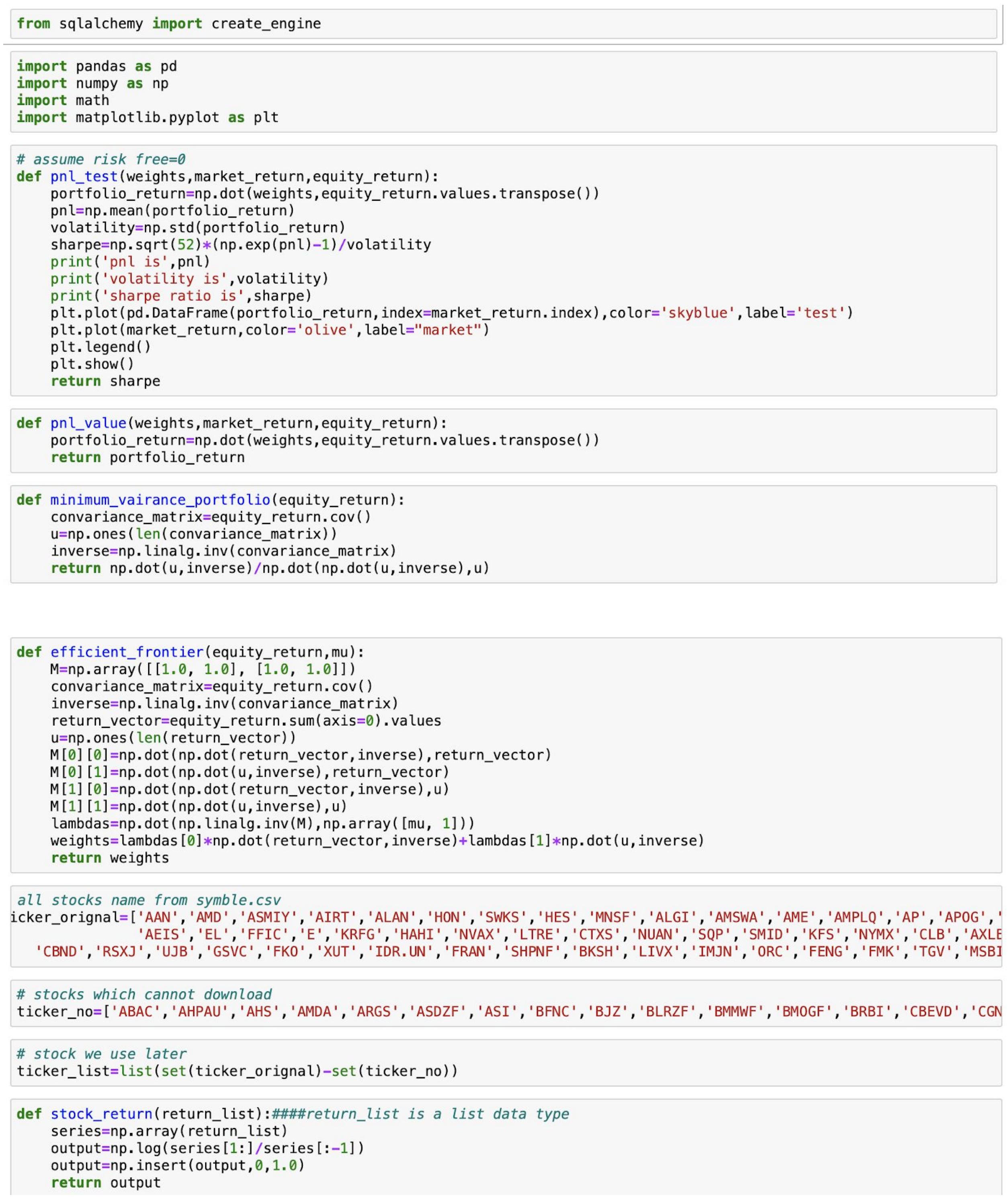
➢ 3.3.3 P&L

We calculated each portfolio’s P&L on test set together with the market return and input them into database ‘pnl’ which contains 4 portfolios’ weekly returns and market return with specific week date.



week	e_weight	weight	mvp	EF_s	EF_L
2014-01	0.049505295	0.053090772	0.003062691	0.019072511	0.019072511
2014-02	-0.02526444	-0.025792706	-0.096172629	-0.036721774	-0.036721774
2014-03	0.135752159	0.12139503	0.03712272	0.05811947	0.05811947
2014-04	-0.101520806	-0.094664495	-0.046077544	-0.047755452	-0.047755452
2014-05	0.242875097	0.270973448	0.066422987	0.126179697	0.126179697
2014-06	-0.055923427	-0.066122123	0.047551442	-0.112779476	-0.112779476
2014-07	0.030674575	0.039090216	-0.024773468	0.074722374	0.074722374
2014-08	-0.012068747	-0.009973397	-0.014545628	0.08284462	0.08284462
2014-09	-0.004292505	-0.005092613	0.022427364	0.208362301	0.208362301
2014-10	0.070726076	0.075413249	0.024547012	0.076695553	0.076695553
2014-11	-0.099619253	-0.103257274	-0.037498182	0.075158623	0.075158623
2014-12	0.001676665	0.006868698	0.016090278	0.03644259	0.03644259
2014-13	0.099418445	0.111315032	0.006339419	0.056282591	0.056282591
2014-14	-0.028810994	-0.028686023	-0.02835239	-0.05609733	-0.05609733
2014-15	-0.052726386	-0.062137703	-0.005981812	0.00255473	0.00255473
2014-16	0.089378137	0.102905651	0.073790464	0.515600164	0.515600164
2014-17	0.051577411	0.046625908	0.062097002	0.068506805	0.068506805
2014-18	0.135364175	0.142498888	0.068992215	0.022860651	0.022860651

4.Screen shot of how code operates (code results)



```
from sqlalchemy import create_engine
```

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
```

```
# assume risk free=0
def pnl_test(weights,market_return,equity_return):
    portfolio_return=np.dot(weights,equity_return.values.transpose())
    pnl=np.mean(portfolio_return)
    volatility=np.std(portfolio_return)
    sharpe=np.sqrt(52)*(np.exp(pnl)-1)/volatility
    print('pnl is',pnl)
    print('volatility is',volatility)
    print('sharpe ratio is',sharpe)
    plt.plot(pd.DataFrame(portfolio_return,index=market_return.index),color='skyblue',label='test')
    plt.plot(market_return,color='olive',label="market")
    plt.legend()
    plt.show()
    return sharpe
```

```
def pnl_value(weights,market_return,equity_return):
    portfolio_return=np.dot(weights,equity_return.values.transpose())
    return portfolio_return
```

```
def minimum_vairance_portfolio(equity_return):
    convariance_matrix=equity_return.cov()
    u=np.ones(len(convariance_matrix))
    inverse=np.linalg.inv(convariance_matrix)
    return np.dot(u,inverse)/np.dot(np.dot(u,inverse),u)
```

```
def efficient_frontier(equity_return,mu):
    M=np.array([[1.0, 1.0], [1.0, 1.0]])
    convariance_matrix=equity_return.cov()
    inverse=np.linalg.inv(convariance_matrix)
    return_vector=equity_return.sum(axis=0).values
    u=np.ones(len(return_vector))
    M[0][0]=np.dot(np.dot(return_vector,inverse),return_vector)
    M[0][1]=np.dot(np.dot(u,inverse),return_vector)
    M[1][0]=np.dot(np.dot(return_vector,inverse),u)
    M[1][1]=np.dot(np.dot(u,inverse),u)
    lambdas=np.dot(np.linalg.inv(M),np.array([mu, 1]))
    weights=lambdas[0]*np.dot(return_vector,inverse)+lambdas[1]*np.dot(u,inverse)
    return weights
```

```
 all stocks name from symble.csv
icker_orignal=['AAN','AMD','ASMIY','AIRT','ALAN','HON','SWKS','HES','MNSF','ALGI','AMSWA','AME','AMPLQ','AP','APOG','
               'AEIS','EL','FFIC','E','KRFG','HAHI','NVAX','LTRE','CTXS','NUAN','SQP','SMID','KFS','NYMX','CLB','AXLE
   'CBND','RSXJ','UJB','GSVC','FKO','XUT','IDR.UN','FRAN','SHPNF','BKSH','LIVX','IMJN','ORC','FENG','FMK','TGV','MSBI
```

```
# stocks which cannot download
ticker_no=['ABAC','AHPAU','AHS','AMDA','ARGS','ASDZF','ASI','BFNC','BJZ','BLRZF','BMMWF','BMOGF','BRBI','CBEVD','CGN
```

```
# stock we use later
ticker_list=list(set(ticker_orignal)-set(ticker_no))
```

```
def stock_return(return_list):####return_list is a list data type
    series=np.array(return_list)
    output=np.log(series[1:]/series[:-1])
    output=np.insert(output,0,1.0)
    return output
```

In [34]:
```python
# We first create a column named week to store the daily comes from which week of year.
# Then we use drop week ‘00’ which means the first week of a new year because it is not completely.
# we have to know the first day of the week,because we decide to use weekly mini day to work out return and drop NaN
df_return=pd.DataFrame()
for i in ticker_list:
    df=pd.read_csv(i)
    df['formatted_date']=pd.to_datetime(df['formatted_date']).dt.strftime('%Y-%m-%d')
    df=df[(df['formatted_date']>='2012-12-31')&(df['formatted_date']<'2015-01-06')]
    df['week']=pd.to_datetime(df['formatted_date']).dt.strftime('%Y-%U')
    df=df[~df['week'].str.contains('00')]
    df['weekdays']=pd.to_datetime(df['formatted_date']).dt.dayofweek
    df=df[['week','adjclose']].loc[df.groupby(['week'])['weekdays'].idxmin()].set_index('week')
    if df['adjclose'].isnull().values.any():
        df=pd.DataFrame()
    print(i,'is undering the iteration')
    if len(df)>=106: # total number of two years
        print('not empty')
        df[(i+'_return')]=stock_return(df['adjclose'])
        df=df[(i+'_return')]
        df_return=df_return.merge(df,left_index=True,right_index=True,how='outer')
        if df_return.isna().iloc[0,0]:
            print('error')
            break
```

```
CNO is undering the iteration
not empty
FUGMF is undering the iteration
not empty
WDC is undering the iteration
not empty
CPSH is undering the iteration
not empty
MRIC is undering the iteration
not empty
SCIL is undering the iteration
not empty
```

Out[36]:

week	SENEB_return	EOG_return	GRC_return	GNUS_return	CSPI_return	TRS_return	PAAS_return	EXR_return	SIEGY_return	APELY_return	...	MLFNF_retu
2013-01	-0.018849	0.042149	0.021556	0.068993	0.035416	0.018395	-0.016146	0.027377	0.001369	-0.010704	...	0.0000
2013-02	0.032099	-0.010612	-0.022562	0.287682	0.006033	0.024581	0.040930	0.006396	0.017991	-0.073853	...	0.0000
2013-03	-0.011251	0.016017	0.026165	0.060625	-0.128175	0.021317	-0.019459	0.024667	0.007053	0.010638	...	0.0225
2013-04	0.017156	-0.018349	0.000980	0.057158	0.067739	0.032930	-0.083590	0.015688	0.006628	-0.055293	...	-0.0116
2013-05	-0.018488	0.024575	-0.030506	0.105361	-0.022618	-0.008131	0.021138	0.030408	-0.044860	0.166982	...	0.0551

5 rows × 1798 columns

In [37]:
```python
##test it nan data exists in the return dataframe
for i in range(len(df_return)):
    if len(df_return.iloc[i][df_return.iloc[i].isnull()==True])>=1:
        print(df_return.iloc[i][df_return.iloc[i].isnull()==True])
```

```
SVNDY_return   NaN
Name: 2013-09, dtype: float64
ITRN_return   NaN
Name: 2014-13, dtype: float64
IAUFF_return   NaN
Name: 2014-23, dtype: float64
TISCY_return   NaN
Name: 2014-39, dtype: float64
```

In [38]:
```
####adjusted close tended to be negative which is insane, thus we couldn't find log return
df_return=df_return.drop(['SVNDY_return','ITRN_return','IAUFF_return','TISCY_return'],axis=1)
```

In [43]:
```
#####read sp500 as market return ,then store it into the overall return dataframe
df=pd.read_csv('GSPC.csv')
df['formatted_date']=pd.to_datetime(df['formatted_date']).dt.strftime('%Y-%m-%d')
df=df[(df['formatted_date']>='2012-12-31')&(df['formatted_date']<'2015-01-06')]
df['week']=pd.to_datetime(df['formatted_date']).dt.strftime('%Y-%U')
df=df[~df['week'].str.contains('00')]
df['weekdays']=pd.to_datetime(df['formatted_date']).dt.dayofweek
df=df[['week','adjclose']].loc[df.groupby(['week'])['weekdays'].idxmin()].set_index('week')
if df['adjclose'].isnull().values.any():
    df=pd.DataFrame()

df[('SP_return')]=stock_return(df['adjclose'])
df=df[('SP_return')].iloc[1:]
df_return=df_return.merge(df,left_index=True,right_index=True,how='outer')
```

In [49]:
```
# the first line is 1 which we set, now we drop it.
df_return=df_return[1:]
df_return
```

Out[49]:

	SENEB_return	EOG_return	GRC_return	GNUS_return	CSPI_return	TRS_return	PAAS_return	EXR_return	SIEGY_return	APELY_return	...	TIPNF_retur
week												
2013-01	-0.018849	0.042149	0.021556	0.068993	0.035416	0.018395	-0.016146	0.027377	0.001369	-0.010704	...	0.00000
2013-02	0.032099	-0.010612	-0.022562	0.287682	0.006033	0.024581	0.040930	0.006396	0.017991	-0.073853	...	0.00000
2013-03	-0.011251	0.016017	0.026165	0.060625	-0.128175	0.021317	-0.019459	0.024667	0.007053	0.010638	...	0.00000
2013-04	0.017156	-0.018349	0.000980	0.057158	0.067739	0.032930	-0.083590	0.015688	0.006628	-0.055293	...	0.00000

In [50]:
```
##test it nan data exists in the return dataframe
for i in range(len(df_return)):
    if len(df_return.iloc[i][df_return.iloc[i].isnull()==True])>=1:
        print(df_return.iloc[i][df_return.iloc[i].isnull()==True])
```

In [51]:
```
# train and test
df_train=df_return[:'2013-52']
df_test=df_return['2014-01':]
```

In [52]:
```
df_train.tail()
```

Out[52]:

eturn	APELY_return	...	TIPNF_return	HCBN_return	PCRFY_return	MAYS_return	PCEF_return	PDCO_return	FCT_return	SP_return_x	SP_return_y	SP_return
1140	-0.032760	...	0.0	0.000000	0.030252	-0.010676	0.003777	-0.014648	0.003078	-0.000877	-0.000877	-0.000877
1086	0.044044	...	0.0	0.000000	0.014789	0.000000	-0.002517	0.004585	-0.020146	0.004139	0.004139	0.004139
4350	-0.054890	...	0.0	-0.013038	-0.033366	-0.016231	-0.000525	-0.028577	-0.021278	-0.012145	-0.012145	-0.012145
28392	0.058027	...	0.0	-0.026597	0.052186	0.053110	0.028382	0.010353	0.025479	0.022936	0.022936	0.022936
17525	0.039479	...	0.0	0.007033	-0.012793	0.098440	0.000000	0.009761	0.000698	0.007130	0.007130	0.007130

In [53]:
```
df_test.head()
```

Out[53]:

	SENEB_return	EOG_return	GRC_return	GNUS_return	CSPI_return	TRS_return	PAAS_return	EXR_return	SIEGY_return	APELY_return	...	TIPNF_retur
week												
2014-01	0.024535	-0.020099	0.009703	-0.127833	-0.005000	-0.034074	0.031804	-0.000946	-0.023065	-0.047565	...	0.00000
2014-02	-0.057658	-0.003420	-0.013060	0.000000	-0.002509	-0.002071	0.036549	0.024075	-0.012073	0.019652	...	0.05406
2014-03	0.000000	0.035806	0.067096	-0.046520	0.024815	0.009545	0.067033	0.018761	0.022096	0.090029	...	0.00000
2014-04	0.000333	-0.043045	-0.081258	-0.182322	-0.029853	-0.075986	-0.071939	-0.010710	-0.029924	0.010454	...	0.00000
2014-05	-0.000667	-0.002223	-0.044705	0.028171	-0.003795	-0.084647	0.016261	0.016811	-0.034121	0.049171	...	0.00000

5 rows × 1797 columns

In [54]:
```
#####covariance matrix
covs=df_train.cov()
covs_test=df_test.cov()
```

```
In [55]: covs
```

9069	-0.003771	0.000112	0.000294	-0.000932	0.000029	-0.000628	...	0.000436	-0.000261	-0.000030	-0.000203	0.000039	0.000059	-3
3771	0.006647	-0.000444	-0.000947	-0.000288	-0.000305	0.000199	...	0.000539	-0.000168	0.000878	0.000059	-0.000066	-0.000083	-1
0112	-0.000444	0.001266	0.000582	0.000416	0.000401	-0.000457	...	-0.000098	0.000087	0.000162	-0.000066	0.000215	0.000168	1
0294	-0.000947	0.000582	0.003101	0.000099	0.000405	-0.000204	...	-0.000233	0.000107	0.000118	0.000206	0.000253	0.000147	2
0932	-0.000288	0.000416	0.000099	0.000865	0.000280	0.000045	...	0.000076	-0.000152	0.000575	0.000158	0.000216	0.000205	6
0029	-0.000305	0.000401	0.000405	0.000280	0.000742	-0.000261	...	0.000014	-0.000105	0.000076	0.000030	0.000175	0.000242	9
0628	0.000199	-0.000457	-0.000204	0.000045	-0.000261	0.002614	...	-0.000161	0.000160	0.000996	0.000772	0.000009	0.000067	1
2552	-0.003000	0.000793	-0.000839	0.001199	-0.000626	-0.002407	...	0.000228	0.000036	0.000173	-0.001100	-0.000042	-0.000371	-5

```
In [170]: #betas
          betas=(covs['SP_return'][:-1]/covs['SP_return'].iloc[-1]).sort_values(ascending=False)
          betas_test=(covs_test['SP_return'][:-1]/covs_test['SP_return'].iloc[-1])
          stock_set=(betas).index.tolist()
```

```
In [171]: # our group beta
          group_target=4.8
```

```
In [172]: # equal weighted, pick up stock around which beta is near group_target
          target_index=len(betas[betas>group_target])
          betas.iloc[target_index-5:target_index+5]

Out[172]: USEI_return     5.444165
          CBRSF_return    5.112732
          NIHK_return     5.073672
          SMRL_return     4.938329
          GTII_return     4.875425
          SNVP_return     4.686555
          VIAP_return     4.626468
          GREW_return     4.599758
          GVSI_return     4.403463
          JAMN_return     4.399104
          Name: SP_return, dtype: float64
```

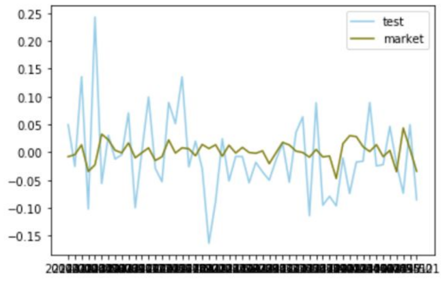
```
In [173]: # which stock decide to use
          stock_set[target_index-5:target_index+5]

Out[173]: ['USEI_return',
           'CBRSF_return',
           'NIHK_return',
           'SMRL_return',
           'GTII_return',
           'SNVP_return',
           'VIAP_return',
           'GREW_return',
           'GVSI_return',
           'JAMN_return']
```

```
In [174]: #store stock name
          picked_stock_train=df_test[stock_set[target_index-5:target_index+5]]
          picked_stock_test=df_test[stock_set[target_index-5:target_index+5]]
```

```
In [175]: ######equal weighted, train and test
          train_beta=betas.iloc[target_index-5:target_index+5].mean()
          test_beta=betas_test[stock_set[target_index-5:target_index+5]].mean()
          print('train beta is',train_beta,'test beta is ',test_beta)
          pnl_test(np.full((1,10),0.1)[0],df_test.iloc[:,-1],picked_stock_test) #portfolio_test
```

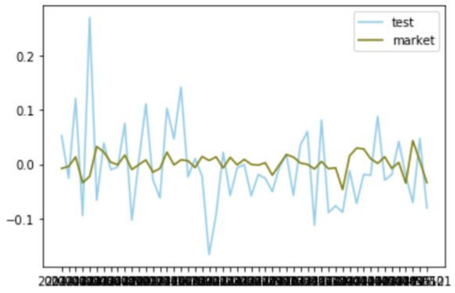
```
train beta is 4.815967202770261 test beta is  0.49883437130802816
pnl is -0.006878921854455629
volatility is 0.0731434914285682
sharpe ratio is -0.6758549173039377
```



```
Out[175]: -0.6758549173039377
```

```
In [177]: #####wieghted by beta
          weights=((betas.iloc[target_index-5:target_index+5])/betas.iloc[target_index-5:target_index+5].sum()).values
          train_beta=np.dot(weights,betas.iloc[target_index-5:target_index+5])
          test_beta=np.dot(weights,betas_test[stock_set[target_index-5:target_index+5]])
          print('train beta is',train_beta,'test beta is ',test_beta)
          pnl_test(weights,df_test.iloc[:,-1],picked_stock_test)
```

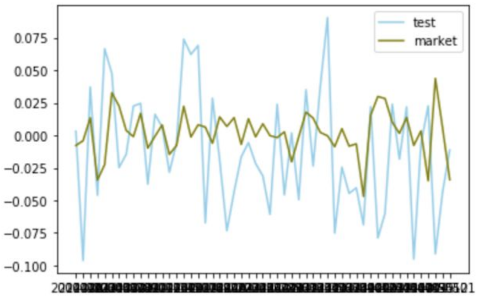
```
train beta is 4.836959053695647 test beta is  0.41804816199553296
pnl is -0.0060480129425121015
volatility is 0.07530653147299314
sharpe ratio is -0.5773897784487156
```



```
Out[177]: -0.5773897784487156
```

In [179]:
```
########mvp
weights_mvp=minimum_vairance_portfolio(picked_stock_train)
train_beta=np.dot(weights_mvp,betas.iloc[target_index-5:target_index+5])
test_beta=np.dot(weights_mvp,betas_test[stock_set[target_index-5:target_index+5]])####
print('train beta is',train_beta,'test beta is ',test_beta)
pnl_test(weights_mvp,df_test.iloc[:,-1],picked_stock_test)
```

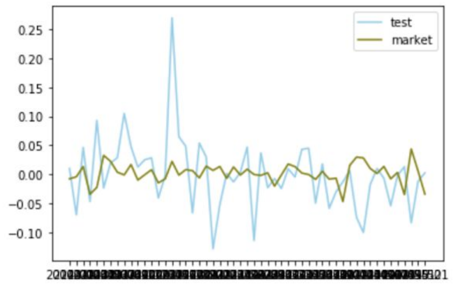
```
train beta is 4.5696231233626206 test beta is  0.11371787640326159
pnl is -0.01193292474132261
volatility is 0.04585804941090078
sharpe ratio is -1.865281447055294
```



Out[179]:
```
-1.865281447055294
```

In [181]:
```
#######EF mu=0.00332
weights_EL=efficient_frontier(picked_stock_train,0.00332)
train_beta=np.dot(weights_EL,betas.iloc[target_index-5:target_index+5])
test_beta=np.dot(weights_EL,betas_test[stock_set[target_index-5:target_index+5]])####
print('train beta is',train_beta,'test beta is ',test_beta)
pnl_test(weights_EL,df_test.iloc[:,-1],picked_stock_test)
```

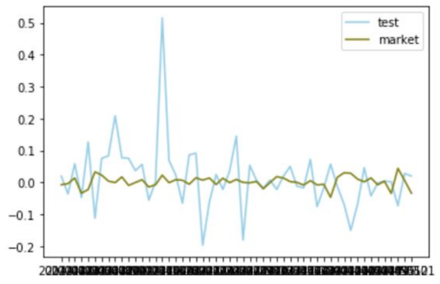
```
train beta is 4.7431669160125045 test beta is  -0.051297257622065585
pnl is 6.26415094339684e-05
volatility is 0.06118879229976737
sharpe ratio is 0.007382536569612222
```



Out[181]:
```
0.007382536569612222
```

```
In [184]: #######EF mu=0.8
          weights_EL=efficient_frontier(picked_stock_train,0.8)
          train_beta=np.dot(weights_EL,betas.iloc[target_index-5:target_index+5])
          test_beta=np.dot(weights_EL,betas_test[stock_set[target_index-5:target_index+5]])####
          print('train beta is',train_beta,'test beta is ',test_beta)
          pnl_test(weights_EL,df_test.iloc[:,-1],picked_stock_test)
```

```
train beta is 4.960635424459919 test beta is  -0.25807846552233926
pnl is 0.015094339622641518
volatility is 0.10214700435140785
sharpe ratio is 1.073672850959399
```



```
Out[184]: 1.073672850959399
```