《机器学习驱动的基本面量化投资研究》

程序列表

李 斌，邵新月，李玥阳

武汉大学经济与管理学院

1. 数据及代码说明

1） 数据说明：

a) factor 文件夹中包含初始 96 项因子数据

b) factorselect 文件夹中包含筛选完成的 16 项因子数据

c) returnseries 文件夹中包含 3/12/24/36 个月滑动窗口下各算法构建投资组合月度收益序列

d) ff3/ff5 分别为 Fama-French3 因子数据，RF 为月度无风险利率数据，final_return 为股票月度收益数据

e) ff30 为去掉市值最小的 30%股票后分别根据市值（size)和收益价格比（EP）分组后构建 MKT/SMB/VMG 3 因子数据

f) factorEW/factorVW 分别为单因子检验 10-1 多空组合（等权重/市值加权）月度收益序列

2） 代码说明：

a) MainFile.py

主程序，运行该函数即可得到各主要结果

b) DataTransfrom.py

导入基础数据并进行预处理，最后将原始数据转换成每个截面一个 stocknumfactornum 的 Dataframe,列名为[各因子名称+’stock'+'ret']，'stock'为股票代码，‘ret’为对应截面股票月度收益

c) NWttest.py

定义对某一序列是否异于 0 进行 Newey and West (1987) t 检验函数

d) ReturnSeriesTest.py

在获取各个算法构建多空组合月度收益序列后，对各个收益序列与 OLS 回归（benchmark）和 DFN（表现最好的深度算法）序列是否存在显著差异进行 NW-T 检验

e) StrategyConstruct.py

i. 投资组合构建通用函数（output),

ii. FC 和 ensemble 无内置算法包，故单独构建 FC 和 ensemblenn

f) FactorTest.py

因子检验补充结果，内容包含：

i. 在去掉市值最小的 30%股票后分别根据市值（size)和收益价格比（EP）分组后构建 MKT/SMB/VMG 3 因子

ii. 单因子 10 分组 10-1/1-10 多空组合因子调整收益

iii. 单因子 10 分组检验各组因子调整收益

iv. 各因子与 size 因子独立双变量分组检验结果

v. 各因子与 BM 因子独立双变量分组检验结果

vi. 6.96 项因子 fama macbeth 回归检验结果

g) DFN.py

深度前馈网络核心库文件，包含深度前馈网络核心函数 DFN()，需在 GPU 环境下运行

h) RNNMODEL.py

循环神经网络库文件，包含如下内容：

i. 基础循环神经网络单元：BaseRnn()

ii. 可用于训练的循环神经网络整体架构：lstmmodule()，该架构基于 BaseRnn()构建循环神经网络。

该文件需在 GPU 环境下运行

i) Ensembleall.py

集成深度学习模型库文件，包含集成模型：Ensemblelr()。

因其中集成深度学习模型，需在 GPU 环境下使用，如不集成深度学习模型，则无 GPU 限制。

j) transecfee.py

计算不同交易成本下的投资组合绩效变化，包含核心函数 transecfee()以及 showtransecfee()

直接调用 showtransecfee()即可

k) selectFactor.py

用于筛选重要因子，包含：

i. 非循环神经网络模型所用的 dropimportant()函数

ii. 循环神经网络模型所用的 dropimportant2()函数

iii. FC 方法的筛选函数 FCselect()函数

注：获取结果需要运行的 python 文件仅为：MainFile.py 和 FactorTest.py

2. 代码整理

a) MainFile.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
@description:
    构建机器学习驱动多因子投资策略主函数
    1.首先输入各算法参数（参数根据第一个滑动窗口网格调参确定，此处直接输入）
    2.全样本 3/12/24/36 个月滑动窗口函数运行,最终直接输出多空组合月度收益, FF3/5-alpha,sharpe
ratio, 并将收益序列保存在' output’文件夹内
    3.全样本 3/12/24/36 个月滑动窗口各个算法多空组合月度收益序列是否存在显著差异 NW-T 检验
    4.不同交易费率下的多空组合绩效结果
    5.全样本 12 个月华东窗口下各个算法的特征筛选
    6.特征筛选后 16 项因子 12 个月滑动窗口函数运行,最终输出多空组合月度收益, FF3/5-
alpha,sharpe ratio, 并将收益序列保存在' output’文件夹内
    注：深度学习算法要在使有 GPU 的环境下进行训练
"""
from StrategyConstruct import FC, output, output2, comboutput, ensemblenn
from selectFactor import dropimportant, dropimportant2, FCselect
from DataTransfrom import datatransfrom, datatransfrom2
from xgboost.sklearn import XGBRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import LinearRegression,Lasso,ElasticNet,Ridge
from sklearn.cross_decomposition import PLSRegression
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
import DFN
import RNNMODEL as rm
import Ensembleall as ea
import warnings
from mxnet import gpu
import os
from transecfee import showtrasecfee
from ReturnSeriesTest import returnseriestest
warnings.filterwarnings('ignore')


#各个算法参数（根据第一个窗口网格调参确定）
window=[3,12,24,36]
PLS_params=[2,2,1,1]
```

```
lasso_params=[1e-3,5e-4,0.01,0.01]
ridge_params=[0.1,0.005,0.01,0.005]
elasticnet_params={’alpha’:[0.01,1e-3,0.01,0.1],’l1_ratio’:[0.3,0.3,0.7,0.3]}
SVR_params={’kernel’:[’linear’,’linear’,’rbf’,’rbf’],’gamma’:[1e-3,1e-3,1e-3,1e-
4],’C’:[0.01,0.001,0.01,1e-4]}
GBDT_params={’learning_rate’:[0.1,0.1,0.1,0.1],’maxdepth’:[2,3,2,2],’n_estimators’:[100,100,
100,100]}#XGBOOST 与 GBDT 相同 此处共用
ENANN_params = {’max_iter’: [100, 100, 200, 300], ’p’: [0.3, 0.5, 0.7, 0.5]}
DFN_params = {’learning_rate’:[0.1, 0.1, 0.1, 0.001], ’batch’: [300, 400, 300, 400]}
LSTM_params = {’learning_rate’:[1e-4, 1e-5, 1e-4, 1e-6], ’depth’: [2, 2, 1, 2],
’hidden_number’: [256]*4}
RNN_params = {’learning_rate’:[0.1, 0.1, 0.1, 0.001], ’depth’: [1, 1, 2, 1],
’hidden_number’: [256]*4}


#***********************2. 全样本 3/12/24/36 个月滑动窗口函数运行
***********************************#
path = r’..\DataBase\factor’ #96 项因子所在路径
factorname = [x[1:-4] for x in os.listdir(path)]
riskfree, timeseries, factor, timeseries2, index = datatransfrom(path)[0],
datatransfrom(path)[1], datatransfrom(path)[2], datatransfrom2(path)[0],
datatransfrom2(path)[1]
for i in range(4):
    i= 0
    output(window[i],LinearRegression(),’OLS’+str(window[i]),riskfree[i], timeseries)
    FC(window[i], riskfree[i], timeseries, 96,’FC’)
    output(window[i], PLSRegression(PLS_params[i]), ’PLS’ + str(window[i]), riskfree[i],
timeseries)
    output(window[i],Lasso(alpha=lasso_params[i]),’Lasso’+ str(window[i]), riskfree[i],
timeseries)
    output(window[i],Ridge(alpha=ridge_params[i]),’Ridge’+str(window[i]),riskfree[i],
timeseries)
    output(window[i],ElasticNet(alpha= elasticnet_params[’alpha’] [i],l1_ratio=
elasticnet_params[’l1_ratio’][i]),’ElasticNet’+str(window[i]),riskfree[i], timeseries)
    output(window[i],SVR(kernel=SVR_params[’kernel’][i],gamma= SVR_params [’gamma’][i],C=
SVR_params [’C’][i] ),’SVR’+str(window[i]),riskfree[i], timeseries)
    output(window[i],
GradientBoostingRegressor(n_estimators=GBDT_params[’n_estimators’][i],max_depth=GBDT_params[
’maxdepth’][i],learning_rate=GBDT_params[’learning_rate’][i]), ’GBDT’ +
str(window[i]),riskfree[i], timeseries)
    output(window[i],
XGBRegressor(n_estimators=GBDT_params[’n_estimators’][i],max_depth=GBDT_params[’maxdepth’][i
], learning_rate=GBDT_params['learning_rate'][i]), 'XGBOOST' + str(window[i]), riskfree[i],
timeseries)
    output(window[i], ensemblenn(5,modeluse = MLPRegressor(solver = 'lbfgs',
max_iter=ENANN_params['max_iter'][i]), pickpercent=ENANN_params['p'][i]), 'ENANN' +
str(window[i]), riskfree[i], timeseries)
    output(window[i], DFN.DFN(outputdim=1, neuralset=[96, 50, 25, 10, 5, 2], ctx=gpu(0),
epoch=10, batch_size=DFN_params['batch'][i], lr=DFN_params['learning_rate'][i]), 'DFN' +
str(window[i]), riskfree[i], timeseries)
    output2(window[i], rm.lstmmodule(96, LSTM_params['hidden_number'][i],
LSTM_params['depth'][i], 100, 3571, lr=LSTM_params['learning_rate'][i]), 'LSTM'+
str(window[i]) ,riskfree[i], timeseries2)
    output2(window[i], rm.lstmmodule(96,  RNN_params['hidden_number'][i],
RNN_params['depth'][i], 100, 3571, lr=RNN_params['learning_rate'][i], ntype='RNN'), 'RNN'+
str(window[i]), riskfree[i], timeseries2)
    modellist = [DFN.DFN(outputdim=1, neuralset=[96, 50, 25, 10, 5, 2], ctx=gpu(0),
epoch=10, batch_size=DFN_params['batch'][i], lr=DFN_params['learning_rate'][i]),
                 ensemblenn(5,modeluse = MLPRegressor(solver = 'lbfgs',
max_iter=ENANN_params['max_iter'][i]), pickpercent=ENANN_params['p'][i]),

XGBRegressor(n_estimators=GBDT_params['n_estimators'][i],max_depth=GBDT_params['maxdepth'][i
], learning_rate=GBDT_params['learning_rate'][i]),

GradientBoostingRegressor(n_estimators=GBDT_params['n_estimators'][i],max_depth=GBDT_params[
'maxdepth'][i],learning_rate=GBDT_params['learning_rate'][i]),
                 PLSRegression(PLS_params[i]),
                 Ridge(alpha=ridge_params[i]),
                 SVR(kernel=SVR_params['kernel'][i],gamma= SVR_params ['gamma'][i],C=
SVR_params ['C'][i])]# PLS 一定要放在倒数第三个（PLS 输出形式为 list，故进行了进一步处理）
    nmolist = [rm.lstmmodule(96, LSTM_params['hidden_number'][i], LSTM_params['depth'][i],
100, 3571, lr=LSTM_params['learning_rate'][i]),
               rm.lstmmodule(96,  RNN_params['hidden_number'][i], RNN_params['depth'][i],
100, 3571, lr=RNN_params['learning_rate'][i], ntype='RNN')]# 循环神经网络模型
    modelname = ['DFN', 'En-ann', 'xgboost', 'GBDT', 'lasso', 'Elasticnet', 'pls', 'Ridge',
'svm', 'LSTM', 'RNN']
    ensemblemodel = ea.Ensemblelr(modellist, nmolist, modelname)
    comboutput(window[i],ensemblemodel, 'Ensemble'+str(window[i]),riskfree[i], timeseries2,
index)
#**********************************3..各算法收益序列差异 NW-t 检验
*****************************************#

for i in window:
    returnseriestest(i)
```

```

#***********************************4.不同交易费率情形
******************************************#

showtrasecfee(0.005)
showtrasecfee(0.0075)
showtrasecfee(0.01)

#******************************5.全样本 12 个月特征筛选过程
**************************************#

i = 1#选取 12 个月滑动窗口筛选因子
dropimportant(window[i] ,LinearRegression(), ’OLS’+str(window[i]), factorname,
timeseries,0.0201)
FCselect(factorname, timeseries)
dropimportant(window[i], PLSRegression(PLS_params[i]), ’PLS’, factorname, timeseries,
0.0230)
dropimportant(window[i], Lasso(alpha=lasso_params[i]), ’Lasso’, factorname, timeseries,
0.0208)
dropimportant(window[i], Ridge(alpha=ridge_params[i]), ’Ridge’, factorname, timeseries,
0.0208)
dropimportant(window[i], ElasticNet(alpha= elasticnet_params[’alpha’] [i],l1_ratio=
elasticnet_params[’l1_ratio’][i]), ’ElasticNet’, factorname, timeseries, 0.0212)
dropimportant(window[i], SVR(kernel=SVR_params[’kernel’][i],gamma= SVR_params
[’gamma’][i],C= SVR_params [’C’][i] ), ’SVR’, factorname, timeseries, 0.0225)
dropimportant(window[i],
GradientBoostingRegressor(n_estimators=GBDT_params[’n_estimators’][i],max_depth=GBDT_params[
’maxdepth’][i],learning_rate=GBDT_params[’learning_rate’][i]), ’GBDT’, factorname,
timeseries, 0.0268)
dropimportant(window[i],XGBRegressor(n_estimators=GBDT_params[’n_estimators’][i],max_depth=G
BDT_params[’maxdepth’][i], learning_rate=GBDT_params[’learning_rate’][i]), ’XGBOOST’,
factorname, timeseries, 0.0273)
dropimportant(window[i], ensemblenn(5,modeluse = MLPRegressor(solver = ’lbfgs’,
max_iter=ENANN_params[’max_iter’][i]), pickpercent=ENANN_params[’p’][i]), ’ENANN’,
factorname, timeseries, 0.0234)
dropimportant(window[i], DFN.DFN(outputdim=1, neuralset=[96, 50, 25, 10, 5, 2], ctx=gpu(0),
epoch=10, batch_size=DFN_params[’batch’][i], lr=DFN_params[’learning_rate’][i]), ’DFN’,
factorname, timeseries, 0.0278)
dropimportant2(window[i], rm.lstmmodule(95, LSTM_params[’hidden_number’][i],
LSTM_params[’depth’][i], 100, 3571, lr=LSTM_params[’learning_rate’][i]), ’LSTM’, factorname,
timeseries2, 0.0257)
dropimportant2(window[i], rm.lstmmodule(95,  RNN_params[’hidden_number’][i],
RNN_params[’depth’][i], 100, 3571, lr=RNN_params[’learning_rate’][i], ntype=’RNN’), ’RNN’,
factorname, timeseries2, 0.0210)


#*****************************6.特征筛选后 16 项因子 12 个月滑动窗口函数运行
*************************************#
path = r’..\DataBase\factorselect’ #经过筛选后因子集合所在路径
riskfree,timeseries,factor,timeseries2=datatransfrom(path)[0],datatransfrom(path)[1],datatra
nsfrom(path)[2],datatransfrom2(path,after=True)[0]
i=1 #选取 12 个月滑动窗口测试筛选后因子集合绩效表现
output(window[i],LinearRegression(),’OLS’+str(window[i]),riskfree[i], timeseries)
FC(window[i], riskfree[i], timeseries, 11,’FC’)
output(window[i], PLSRegression(PLS_params[i]), ’PLS’ + str(window[i]), riskfree[i],
timeseries)
output(window[i],Lasso(alpha=lasso_params[i]),’Lasso’+ str(window[i]), riskfree[i],
timeseries)
output(window[i],Ridge(alpha=ridge_params[i]),’Ridge’+str(window[i]),riskfree[i],
timeseries)
output(window[i],ElasticNet(alpha= elasticnet_params[’alpha’] [i],l1_ratio=
elasticnet_params[’l1_ratio’][i]),’ElasticNet’+str(window[i]),riskfree[i], timeseries)
output(window[i],SVR(kernel=SVR_params[’kernel’][i],gamma= SVR_params [’gamma’][i],C=
SVR_params [’C’][i] ),’SVR’+str(window[i]),riskfree[i], timeseries)
output(window[i],
GradientBoostingRegressor(n_estimators=GDBT_params[’n_estimators’][i],max_depth=GBDT_params[
’maxdepth’][i],learning_rate=GBDT_params[’learning_rate’][i]), ’GBDT’ +
str(window[i]),riskfree[i], timeseries)
output(window[i],
XGBRegressor(n_estimators=GBDT_params[’n_estimators’][i],max_depth=GBDT_params[’maxdepth’][i
], learning_rate=GBDT_params[’learning_rate’][i]), ’XGBOOST’ + str(window[i]), riskfree[i],
timeseries)
output(window[i], ensemblenn(5,modeluse = MLPRegressor(solver = ’lbfgs’,
max_iter=ENANN_params[’max_iter’][i]), pickpercent=ENANN_params[’p’][i]), ’ENANN’ +
str(window[i]), riskfree[i], timeseries)
output(window[i], DFN.DFN(outputdim=1, neuralset=[16, 50, 25, 10, 5, 2], ctx=gpu(0),
epoch=10, batch_size=DFN_params[’batch’][i], lr=DFN_params[’learning_rate’][i]), ’DFN’ +
str(window[i]), riskfree[i], timeseries)
output2(window[i], rm.lstmmodule(11, LSTM_params[’hidden_number’][i],
LSTM_params[’depth’][i], 100, 3571, lr=LSTM_params[’learning_rate’][i]), ’LSTM’+
str(window[i]) ,riskfree[i], timeseries2)
output2(window[i], rm.lstmmodule(11,  RNN_params[’hidden_number’][i],
RNN_params[’depth’][i], 100, 3571, lr=RNN_params[’learning_rate’][i], ntype=’RNN’), ’RNN’+
str(window[i]), riskfree[i], timeseries2)
```

b) DataTransfrom.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
@description:
    导入基础数据并进行一些预处理，最后变成每个截面一个Dataframe,列名为[各因子名称+'
stock'+'ret']，index代表单只股票
"""
import glob,os
import pandas as pd
import warnings

#**************************1.导入因子数据，无风险利率，股票月度收益数据
******************************#
warnings.filterwarnings('ignore')
def datatransfrom(datapath):
    path=datapath
    file = glob.glob(os.path.join(path, "*.csv"))
    k=[]
    for i in range(len(file)):
        k.append(pd.read_csv(file[i]))
    #股票月度收益
    ret=pd.read_csv('..\DataBase\\final_return.csv')
    #无风险利率
    rf=pd.read_csv('..\DataBase\\RF.csv')
    rf3=rf.iloc[3:-1,:]
    rf12=rf.iloc[12:-1,:]
    rf24=rf.iloc[24:-1,:]
    rf36=rf.iloc[36:-1,:]
    riskfree = [rf3, rf12, rf24, rf36]
    #因子名称
    factor=[]
    for i in range(len(file)):
        factor.append(file[i][20:-4])
    factor.append('stock')


    #*****对原始数据进行预处理，每个截面一个Dataframe,列名为{96因子名称+' stock'+'ret'},
index代表单只股票******#
    timeseries=[]
    for i in range(len(ret.columns)-1):
        kl=pd.concat([k[j].iloc[:,i+1] for j in range(len(file))],axis=1)
        kl[’stock’] = ret.iloc[:,0]
        kl.columns = factor
        kl=kl.iloc[:-2,:]
        timeseries.append(kl)
    #删除月度收益不存在的数据条
    for i in range(len(timeseries)):
        timeseries[i][’ret’]=ret.iloc[:,i+1]
        timeseries[i][’ret’]=timeseries[i][’ret’].fillna(’null’)
        timeseries[i]=timeseries[i][~timeseries[i][’ret’].isin([’null’])]
    return riskfree,timeseries,factor

## 为LSTM\RNN设计的数据读取函数
def datatransfrom2(datapath, after=False):
    path=datapath
    file = glob.glob(os.path.join(path, ”*.csv”))
    k=[]
    for i in range(len(file)):
        k.append(pd.read_csv(file[i]))
    #股票月度收益
    ret=pd.read_csv(’..\DataBase\\final_return.csv’)
    #因子名称
    factor=[]
    for i in range(len(file)):
        factor.append(file[i][20:-4])
    factor.append(’stock’)

    #*****对原始数据进行预处理，每个截面一个Dataframe,列名为{96因子名称+’ stock’+’ret’},
index代表单只股票******#
    timeseries2=[]
    index = []
    for i in range(len(ret.columns)-1):
        kl=pd.concat([k[j].iloc[:,i+1] for j in range(len(file))],axis=1)
        kl[’stock’] = ret.iloc[:,0]
        kl.columns = factor
        if after:# 保证筛选后因子个数为3571个
            kl = kl.iloc[:, :]
        else:
            kl = kl.iloc[:-2,:]
        timeseries2.append(kl)
    # 加入月度收益，令月度收益不存在为null，方便下一步函数处理
    for i in range(len(timeseries2)):
        timeseries2[i][’ret’] = ret.iloc[:, i + 1]
        timeseries2[i][’ret’] = timeseries2[i][’ret’].fillna(’null’)
        index.append(timeseries2[i][’ret’].isin([’null’]))
    return timeseries2, index
```

c) NWttest.py

```python
# -*- coding:utf-8 -*-
’’’
@description:
NW-t 检验所用包
’’’

import numpy as np
from collections import namedtuple
from scipy.stats import distributions

def _ttest_finish(df, t):
    ’’’
    :param df:自由度
    :param t: t 值
    :return: 输出 t 和对应 p 值
    ’’’
    prob = distributions.t.sf(np.abs(t), df) * 2  # use np.abs to get upper tail
    if t.ndim == 0:
        t = t[()]
    return t, prob

NWt_1sampleResult = namedtuple(’NWT_1sampResult’, (’statistic’, ’pvalue’))
def nwttest_1samp(a, popmean, axis=0,L=1):
    ’’’
    主函数
    :param a: 数据列表
    :param popmean: 原假设值 u0
    :param axis: 行还是列, 默认行
    :param L: lag,  滞后多少, 默认 1
    :return: 输出 nw-t 和对应 p 值
    ’’’
    a = np.array(a)
    N = len(a)
    df = N-1
    e = a - np.mean(a)
    residuals = np.sum(e**2)
    Q = 0
    for i in range(L):
        w_l = 1 - (i+1)/(1+L)
        for j in range(1,N):
            Q += w_l*e[j]*e[j-(i+1)]
    S = residuals + 2*Q
    nw_var = S/N
    d = np.mean(a,axis) - popmean
    nw_sd = np.sqrt(nw_var / float(df))
    with np.errstate(divide='ignore', invalid='ignore'):
        t = np.divide(d, nw_sd)
    t,prob = _ttest_finish(df,t)

    return NWt_1sampleResult(t,prob)
```

d) ReturnSeriesTest.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
@description:
    在获取各个算法构建多空组合月度收益序列后，对各个收益序列与OLS回归（benchmark）
    和DFN（表现最好的深度算法）序列是否存在显著差异进行NW-T检验
"""

import glob,os
import pandas as pd
from NWttest import nwttest_1samp
import warnings
warnings.filterwarnings(’ignore’)

#
def returnseriestest(length):
    path = r’..\DataBase\returnseries’+’\\’+str(length)
    file = glob.glob(os.path.join(path, ”*.csv”))
    ols = pd.read_csv(path+’\\OLS’+str(length)+’.csv’)
    dfn=pd.read_csv(path+’\\DFN’+str(length)+’.csv’)
    k = []  # 每个算法一个df
    for i in range(len(file)):
        k.append(pd.read_csv(file[i]))
    #OLS与其他算法区别
    for i in range(len(k)):
        t = []
        t1 = nwttest_1samp(k[i].iloc[:, 1] - ols[’long-short’], 0)
        t.append(t1.statistic)
        t2 = nwttest_1samp(k[i].iloc[:, 2] - ols[’long’], 0)
        t.append(t2.statistic)
        t3 = nwttest_1samp(k[i].iloc[:, 3] - ols[’short’], 0)
        t.append(t3.statistic)
        print(’ols-’+file[i][27:-4], t)
    #DFN与其他算法区别
    for i in range(len(k)):
        t = []
        t1 = nwttest_1samp(-k[i].iloc[:, 1] + dfn[’long-short’], 0)
        t.append(t1.statistic)
        t2 = nwttest_1samp(-k[i].iloc[:, 2] + dfn[’long’], 0)
        t.append(t2.statistic)
        t3 = nwttest_1samp(-k[i].iloc[:, 3] + dfn[’short’], 0)
        t.append(t3.statistic)
        print('dfn-'+file[i][27:-4], t)
    return
```

e) StrategyConstruct.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
@description:
    1.投资组合构建通用函数（output),函数最终输出多空组合月度收益, FF3/5-alpha,sharpe ratio
    2.FC 和 ensemble 无内置算法包，此处单独构建 FC 和 ensemblenn

"""
import glob,os
import pandas as pd
import numpy as np
from scipy import stats
import statsmodels.api as sm
from xgboost.sklearn import XGBRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import scale
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression
from sklearn.linear_model import Lasso,ElasticNet,Ridge
from NWttest import nwttest_1samp
from sklearn.svm import SVR
import warnings
from sklearn.neural_network import MLPRegressor
import gc


#************************3.EN-ANN、FC 算法函数******************************#
#python 里没有 EN-ANN 和 FC 的对应算法包 此处先定义算法计算方式
#EN-ANN
class ensemblenn(object):
    def __init__(self,ensemblenumbers,modeluse = MLPRegressor(solver = 'lbfgs'),pickpercent 
= 0.5):
        self.ensemblenumbers = ensemblenumbers
        self.modellist = []
        self.score = [0]*ensemblenumbers
        for i in range(self.ensemblenumbers):
            self.modellist.append(modeluse)
        self.pickpercent = pickpercent
    def fit(self,X,Y):
        for i in range(self.ensemblenumbers):
            self.modellist[i].fit(X,Y)
            self.score[i] = self.modellist[i].loss_
    def predict(self,xtest):
        usemodel = np.array(self.modellist)[np.argsort(np.array(self.score))]
        usemodel = usemodel[0:self.ensemblenumbers//2]
        predict = []
        for i in range(self.ensemblenumbers//2):
            predict.append(usemodel[i].predict(xtest))
        return list(np.mean(predict,axis=0))
#FC
def FC(length,rf, timeseries, lenn=96, na=’FC’):
    #length为滑动窗口长度：取值{3, 12, 24, 36}
    #na为输出文件名称
    #rf为无风险利率，取值与length对应{rf3, rf12, rf24, rf36}
    Long_Short = []
    Long = []
    Short = []
    for i in range(len(timeseries) - length):
        print(i)
        FINALm = pd.concat(timeseries[i:i + (length +1)], axis=0)
        FINALm = FINALm.fillna(0)
        FINAL_X = FINALm.iloc[:, :-2]
        FINAL_x = scale(FINAL_X)
        final = pd.concat(timeseries[i:i + length], axis=0)
        x_train = FINAL_x[:len(final)]
        x_test = FINAL_x[len(final):]
        y_train = final.iloc[:, -1]
        test = timeseries[i + length]
        clf = LinearRegression()
        k = []
        for i in range(lenn):
            x = x_train[:, i].reshape(-1, 1)
            clf.fit(x, y_train)
            k.append(clf.coef_[0])
        PREDICTION = []
        for i in range(len(x_test)):
            test0 = np.array(x_test[i])
            y = 0
            for j in range(lenn):
                y = y + test0[j] * k[j]
            PREDICTION.append(y)
        y_test = test.iloc[:, -1]
        # 构建投资组合
        r_predict = pd.DataFrame(PREDICTION, columns=[’predict’])
        r_ture = pd.DataFrame(y_test)
        r_ture.columns = [’ture’]
        r_ture.index = r_predict.index
        FINAL = pd.concat([r_predict, r_ture], axis=1)
        FINAL_sort = FINAL.sort_values(by=’predict’, axis=0)
        r_final = np.array(FINAL_sort[’ture’])
        m = int(len(r_final) * 0.1) + 1
        r_final = r_final.tolist()
        long = r_final[-m:]
        short = r_final[:m]
        r_end = (np.sum(long) - np.sum(short)) / m
        Long_Short.append(r_end)
        Long.append(np.average(long))
        Short.append(np.average(short))
    T_value = []
    Mean = []
    p_value = []
    sharpratio = []
    Std = []
    TO = [Long_Short, Long, Short]
    for l in TO:
        t_test = nwttest_1samp(l, 0)
        mean = np.average(l) * 12
        STD = np.std(l) * np.sqrt(12)
        sharp = (mean - rf.mean().tolist()[0] * 12 / 100) / STD
        T_value.append(t_test.statistic)
        p_value.append(t_test.pvalue)
        Mean.append(mean)
        Std.append(STD)
        sharpratio.append(sharp)
    name = na
    length = length
    print(name, ’long-short’, ’long’, ’short’)
    print(’mean’, Mean[0] / 12, Mean[1] / 12, Mean[2] / 12)
    print(’t-statistic’, ’(’ + str(round(T_value[0], 4)) + ’)’, ’(’ + str(round(T_value[1],
4)) + ’)’,
          ’(’ + str(round(T_value[2], 4)) + ’)’)
    A = pd.DataFrame(Long_Short, columns=[’long-short’])
    B = pd.DataFrame(Long, columns=[’long’])
    C = pd.DataFrame(Short, columns=[’short’])
    M = pd.concat([A, B, C], axis=1)
    M.to_csv(’..\output\\’+name + ’.csv’)
    ff3 = pd.read_csv(’..\DataBase\\ff3.csv’)
    ff5 = pd.read_csv(’..\DataBase\\ff5.csv’)
    alpha3 = []
    t3 = []
    t5 = []
    alpha5 = []
    for i in range(3):
        X1 = ff3.iloc[length:, 1:]
        X2 = ff5.iloc[length:, 1:]
        Y = M.iloc[:-2, i]
        Y.index = X1.index
        Y = Y - rf.RF[:-1] / 100
        x1 = sm.add_constant(X1)
        reg = sm.OLS(Y, x1).fit()
        t3.append(reg.tvalues[0])
        alpha3.append(reg.params[0] * 12)
        x2 = sm.add_constant(X2)
        reg = sm.OLS(Y, x2).fit()
        t5.append(reg.tvalues[0])
        alpha5.append(reg.params[0] * 12)
    print('alpha-FF3', alpha3[0] / 12, alpha3[1] / 12, alpha3[2] / 12)
    print('t-statistic', '(' + str(round(t3[0], 4)) + ')', '(' + str(round(t3[1], 4)) + ')',
          '(' + str(round(t3[2], 4)) + ')')
    print('alpha-FF5', alpha5[0] / 12, alpha5[1] / 12, alpha5[2] / 12)
    print('t-statistic', '(' + str(round(t5[0], 4)) + ')', '(' + str(round(t5[1], 4)) + ')',
          '(' + str(round(t5[2], 4)) + ')')
    print('sharpe', sharpratio[0], sharpratio[1], sharpratio[2])


#************************4.投资组合构建主函数*****************************************#
def output(length,CLF,name,rf,timeseries):
    #length 为滑动窗口长度：取值{3,12,24,36}
    #CLF 为预测选取的机器学习模型
    #name 为输出文件名称 (type:string)
    #rf 为无风险利率, 取值与 length 对应{rf3,rf12,rf24,rf36}
    Long_Short = []
    Long = []
    Short = []
    for i in range(len(timeseries) - (length)):
        print(i)
        FINALm = pd.concat(timeseries[i:i + (length+1)], axis=0)
        FINALm = FINALm.fillna(0)#因子缺失值以 0 填充
        FINAL_X = FINALm.iloc[:, :-2]
        FINAL_x = scale(FINAL_X)
        final = pd.concat(timeseries[i:i + length], axis=0)
        x_train = FINAL_x[:len(final)]
        x_test = FINAL_x[len(final):]
        y_train = final.iloc[:, -1]
        test = timeseries[i + length]
        y_test = test.iloc[:, -1]
        # 基准-linear
        clf = CLF
        clf.fit(x_train, y_train)
        PREDICTION = clf.predict(x_test)
        # 构建投资组合
        prediction = pd.DataFrame(PREDICTION)
        r_predict = pd.DataFrame(PREDICTION, columns=['predict'])
        r_ture = pd.DataFrame(y_test)
        r_ture.columns = ['ture']
        r_ture.index = r_predict.index
        FINAL = pd.concat([r_predict, r_ture], axis=1)
        FINAL_sort = FINAL.sort_values(by='predict', axis=0)
        r_final = np.array(FINAL_sort['ture'])
        m = int(len(r_final) * 0.1) + 1
        r_final = r_final.tolist()
        long = r_final[-m:]
        short = r_final[:m]
        r_end = (np.sum(long) - np.sum(short)) / m
        Long_Short.append(r_end)
        Long.append(np.average(long))
        Short.append(np.average(short))
    T_value = []
    Mean = []
    p_value = []
    sharpratio = []
    Std = []
    TO = [Long_Short, Long, Short]
    for l in TO:
        t_test = nwttest_1samp(l, 0)
        mean = np.average(l) * 12
        STD = np.std(l) * np.sqrt(12)
        sharp = (mean- rf.mean().tolist()[0]*12/100 ) / STD
        T_value.append(t_test.statistic)
        p_value.append(t_test.pvalue)
        Mean.append(mean)
        Std.append(STD)
        sharpratio.append(sharp)
    print(name, 'long-short', 'long', 'short')
    print('mean', Mean[0]/12, Mean[1]/12, Mean[2]/12)
    print('t-statistic', '('+str(round(T_value[0],4))+')','('+str(round(T_value[1],4))+')',
'('+str(round(T_value[2],4))+')')
    A = pd.DataFrame(Long_Short, columns=['long-short'])
    B = pd.DataFrame(Long, columns=[’long’])
    C = pd.DataFrame(Short, columns=[’short’])
    M = pd.concat([A, B, C], axis=1)
    M.to_csv(’..\output\\’+name+’.csv’)
    ff3 = pd.read_csv(’..\DataBase\\ff3.csv’)
    ff5 = pd.read_csv(’..\DataBase\\ff5.csv’)
    alpha3 = []
    t3 = []
    t5 = []
    alpha5 = []
    for i in range(3):
        X1 = ff3.iloc[length:, 1:]
        X2 = ff5.iloc[length:, 1:]
        Y = M.iloc[:-2, i]
        Y.index = X1.index
        Y = Y - rf.RF[:-1] / 100
        x1 = sm.add_constant(X1)
        reg = sm.OLS(Y, x1).fit()
        t3.append(reg.tvalues[0])
        alpha3.append(reg.params[0] * 12)
        x2 = sm.add_constant(X2)
        reg = sm.OLS(Y, x2).fit()
        t5.append(reg.tvalues[0])
        alpha5.append(reg.params[0] * 12)
    print(’alpha-FF3’, alpha3[0]/12, alpha3[1]/12, alpha3[2]/12)
    print(’t-statistic’, ’(’+str(round(t3[0],4))+’)’,’(’+str(round(t3[1],4))+’)’,
’(’+str(round(t3[2],4))+’)’)
    print(’alpha-FF5’, alpha5[0]/12, alpha5[1]/12, alpha5[2]/12)
    print(’t-statistic’, ’(’+str(round(t5[0],4))+’)’,’(’+str(round(t5[1],4))+’)’,
’(’+str(round(t5[2],4))+’)’)
    print(’sharpe’, sharpratio[0], sharpratio[1], sharpratio[2])

# 因为 LSTM 与 RNN 一个步长内所用数据形状必须一致，设置专用的主函数供使用
def output2(length,CLF,name,rf,timeseries2):
    # length 为滑动窗口长度：取值{3, 12, 24, 36}
    # CLF 为预测选取的机器学习模型
    # name 为输出文件名称 (type:string)
    # rf 为无风险利率，取值与 length 对应{rf3, rf12, rf24, rf36}
    Long_Short = []
    Long = []
    Short = []
    for i in range(len(timeseries2) - length):
        FINALm = pd.concat(timeseries2[i:(i + length + 1)], axis=0)
        FINALm[~FINALm[’ret’].isin([’null’])] =
FINALm[~FINALm[’ret’].isin([’null’])].fillna(0)
        FINAL_X = FINALm.iloc[:, :-2]
        FINAL_x = FINAL_X
        FINAL_x[~FINALm[’ret’].isin([’null’])] =
scale(FINAL_X[~FINALm[’ret’].isin([’null’])])
        FINAL_x[FINALm[’ret’].isin([’null’])] = 0
        FINALm[FINALm[’ret’].isin([’null’])] = 0
        x_train = [FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in range(length)]
        y_train = [FINALm.iloc[j * 3571:(j + 1) * 3571, -1].values for j in range(length)]
        x_test = np.array([FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in range(1,
length + 1)])
        y_test = list(FINALm.iloc[(length) * 3571:(length + 1) * 3571, -1].values)
        # 基准-linear
        clf = CLF
        clf.fit(x_train, y_train)
        PREDICTION = clf.predict(x_test)
        PREDICTION = [PREDICTION[m][0] for m in range(3571 * (length - 1), 3571 * length)]
        # 构建投资组合
        r_predict = pd.DataFrame(PREDICTION, columns=[’predict’])
        r_ture = pd.DataFrame(y_test)
        r_ture.columns = [’ture’]
        r_ture.index = r_predict.index
        FINAL = pd.concat([r_predict, r_ture], axis=1)
        FINAL = FINAL[~timeseries2[i + length][’ret’].isin([’null’])]
        FINAL_sort = FINAL.sort_values(by=’predict’, axis=0)
        r_final = np.array(FINAL_sort[’ture’])
        m = int(len(r_final) * 0.1) + 1
        r_final = r_final.tolist()
        long = r_final[-m:]
        short = r_final[:m]
        r_end = (np.sum(long) - np.sum(short)) / m
        Long_Short.append(r_end)
        Long.append(np.average(long))
        Short.append(np.average(short))
        gc.collect()
    T_value = []
    Mean = []
    p_value = []
    sharpratio = []
    Std = []
    TO = [Long_Short, Long, Short]
    for l in TO:
        t_test = nwttest_1samp(l, 0, L=1)
        mean = np.average(l) * 12- rf.mean().tolist()[0] * 12 / 100
        STD = np.std(l) * np.sqrt(12)
        sharp = (mean ) / STD
        T_value.append(t_test.statistic)
        p_value.append(t_test.pvalue)
        Mean.append(mean)
        Std.append(STD)
        sharpratio.append(sharp)
    print(name, ’long-short’, ’long’, ’short’)
    print(’mean’, Mean[0] / 12 + rf.mean().tolist()[0] / 100, Mean[1] / 12 +
rf.mean().tolist()[0] / 100, Mean[2] / 12 + rf.mean().tolist()[0] / 100)
    print(’t-statistic’, ’(’+str(round(T_value[0],4))+’)’, ’(’+str(round(T_value[1],4))+’)’,
’(’+str(round(T_value[2],4))+’)’)
    A = pd.DataFrame(Long_Short, columns=[’long-short’])
    B = pd.DataFrame(Long, columns=[’long’])
    C = pd.DataFrame(Short, columns=[’short’])
    M = pd.concat([A, B, C], axis=1)
    M.to_csv(’..\output\\’ + name + ’.csv’)
    ff3 = pd.read_csv(’..\DataBase\\ff3.csv’)
    ff5 = pd.read_csv(’..\DataBase\\ff5.csv’)
    alpha3 = []
    t3 = []
    t5 = []
    alpha5 = []
    for i in range(3):
        X1 = ff3.iloc[length:, 1:]
        X2 = ff5.iloc[length:, 1:]
        Y = M.iloc[:-2, i]
        Y.index = X1.index
        Y = Y - rf.RF[:-1] / 100
        x1 = sm.add_constant(X1)
        reg = sm.OLS(Y, x1).fit()
        t3.append(reg.tvalues[0])
        alpha3.append(reg.params[0] * 12)
        x2 = sm.add_constant(X2)
        reg = sm.OLS(Y, x2).fit()
        t5.append(reg.tvalues[0])
        alpha5.append(reg.params[0] * 12)
    print(’alpha-FF3’, alpha3[0] / 12, alpha3[1] / 12, alpha3[2] / 12)
    print(’t-statistic’, ’(’ + str(round(t3[0], 4)) + ’)’, ’(’ + str(round(t3[1], 4)) + ’)’,
          ’(’ + str(round(t3[2], 4)) + ’)’)
    print(’alpha-FF5’, alpha5[0] / 12, alpha5[1] / 12, alpha5[2] / 12)
    print(’t-statistic’, ’(’ + str(round(t5[0], 4)) + ’)’, ’(’ + str(round(t5[1], 4)) + ’)’,
          ’(’ + str(round(t5[2], 4)) + ’)’)
    print(’sharpe’, sharpratio[0], sharpratio[1], sharpratio[2])
```

```python

# 针对所有机器学习模型集成所设置的主函数
def comboutput(length, clf, name, rf,timeseries2, index):
    Long_Short = []
    Long = []
    Short = []
    for i in range(len(timeseries2) - length):
        print(i)
        # LSTM 数据
        FINALm = pd.concat(timeseries2[i:(i + length + 1)], axis=0)
        FINALm[~FINALm[’ret’].isin([’null’])] =
FINALm[~FINALm[’ret’].isin([’null’])].fillna(0)
        FINAL_X = FINALm.iloc[:, :-2]
        FINAL_x = FINAL_X
        FINAL_x[~FINALm[’ret’].isin([’null’])] =
scale(FINAL_X[~FINALm[’ret’].isin([’null’])])
        FINAL_x[FINALm[’ret’].isin([’null’])] = 0
        FINALm[FINALm[’ret’].isin([’null’])] = 0
        Nx_train = [FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in range(length)]
        Ny_train = [FINALm.iloc[j * 3571:(j + 1) * 3571, -1].values for j in range(length)]
        Nx_test = [FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in range(1, length
+ 1)]
        ## 传统数据
        dl = timeseries2[i:i + (length + 1)]
        p = [dl[j][~index[i+j]] for j in range(len(dl))]
        TTX = pd.concat(p, axis=0)
        TTX = TTX.fillna(0)
        TXX = TTX.iloc[:, :-2]
        TXx = scale(TXX)
        final = pd.concat(p[:length], axis=0)
        Tx_train = TXx[:len(final)]
        Tx_test = TXx[len(final):]
        Ty_train = final.iloc[:, -1]
        test = p[-1]
        Ty_test = test.iloc[:, -1]
        # 基准-linear
        clf = clf
        clf.fit(Tx_train, Ty_train, Nx_train, Ny_train)
        PREDICTION = clf.predict(Tx_test, Nx_test, index[i+length], length)
        r_predict = pd.DataFrame(PREDICTION, columns=[’predict’])
        r_ture = pd.DataFrame(Ty_test)
        r_ture.columns = [’ture’]
        r_ture.index = r_predict.index
        FINAL = pd.concat([r_predict, r_ture], axis=1)
        FINAL_sort = FINAL.sort_values(by=’predict’, axis=0)
        r_final = np.array(FINAL_sort[’ture’])
        m = int(len(r_final) * 0.1) + 1
        r_final = r_final.tolist()
        long = r_final[-m:]
        short = r_final[:m]
        r_end = (np.sum(long) - np.sum(short)) / m
        Long_Short.append(r_end)
        Long.append(np.average(long))
        Short.append(np.average(short))
        gc.collect()
    T_value = []
    Mean = []
    p_value = []
    sharpratio = []
    Std = []
    TO = [Long_Short, Long, Short]
    for l in TO:
        t_test = nwttest_1samp(l, 0, L=1)
        mean = np.average(l) * 12 - rf.mean().tolist()[0] * 12 / 100
        STD = np.std(l) * np.sqrt(12)
        sharp = (mean) / STD
        T_value.append(t_test.statistic)
        p_value.append(t_test.pvalue)
        Mean.append(mean)
        Std.append(STD)
        sharpratio.append(sharp)
    print(name, ’long-short’, ’long’, ’short’)
    print(’mean’, Mean[0] / 12, Mean[1] / 12, Mean[2] / 12)
    print(’t-statistic’, ’(’ + str(round(T_value[0], 4)) + ’)’, ’(’ + str(round(T_value[1],
4)) + ’)’,
          ’(’ + str(round(T_value[2], 4)) + ’)’)
    A = pd.DataFrame(Long_Short, columns=[’long-short’])
    B = pd.DataFrame(Long, columns=[’long’])
    C = pd.DataFrame(Short, columns=[’short’])
    M = pd.concat([A, B, C], axis=1)
    M.to_csv(’..\output\\’ + name + ’.csv’)
    ff3 = pd.read_csv(’..\DataBase\\ff3.csv’)
    ff5 = pd.read_csv(’..\DataBase\\ff5.csv’)
    alpha3 = []
    t3 = []
    t5 = []
    alpha5 = []
    for i in range(3):
        X1 = ff3.iloc[length:, 1:]
        X2 = ff5.iloc[length:, 1:]
        Y = M.iloc[:-2, i]
        Y.index = X1.index
        Y = Y - rf.RF[:-1] / 100
        x1 = sm.add_constant(X1)
        reg = sm.OLS(Y, x1).fit()
        t3.append(reg.tvalues[0])
        alpha3.append(reg.params[0] * 12)
        x2 = sm.add_constant(X2)
        reg = sm.OLS(Y, x2).fit()
        t5.append(reg.tvalues[0])
        alpha5.append(reg.params[0] * 12)
    print('alpha-FF3', alpha3[0] / 12, alpha3[1] / 12, alpha3[2] / 12)
    print('t-statistic', '(' + str(round(t3[0], 4)) + ')', '(' + str(round(t3[1], 4)) + ')',
          '(' + str(round(t3[2], 4)) + ')')
    print('alpha-FF5', alpha5[0] / 12, alpha5[1] / 12, alpha5[2] / 12)
    print('t-statistic', '(' + str(round(t5[0], 4)) + ')', '(' + str(round(t5[1], 4)) + ')',
          '(' + str(round(t5[2], 4)) + ')')
    print('sharpe', sharpratio[0], sharpratio[1], sharpratio[2])
```

f) FactorTest.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
@description:
1.在去掉市值最小的 30%股票后分别根据市值 (size)和收益价格比 (EP) 分组后构建 MKT/SMB/VMG 3 因
子
2.单因子 10 分组 10-1/1-10 多空组合因子调整收益
3.单因子 10 分组检验各组因子调整收益
4.各因子与 size 因子独立双变量分组检验结果
5.各因子与 BM 因子独立双变量分组检验结果
6.96 项因子 fama macbeth 回归检验结果

"""
#*****1.在去掉市值最小的 30%股票后分别根据市值 (size)和收益价格比 (EP) 分组后构建 MKT/SMB/VMG
3 因子*****#

import pandas as pd
import numpy as np
ret=pd.read_csv('..\DataBase\\final_return.csv')
size=pd.read_csv('..\DataBase\\factor\\01size.csv')
EP=pd.read_csv('..\DataBase\\factor\\32EP.csv')

faceq=pd.DataFrame(columns=['MKT','SMB','VMG'],index=ret.columns[1:])
facvw=pd.DataFrame(columns=['MKT','SMB','VMG'],index=ret.columns[1:])
rf=pd.read_csv('RF.csv')
for i in range(len(ret.columns)-1):
    total=pd.concat([ret.iloc[:,i+1],size.iloc[:,i+1],EP.iloc[:,i+1]],axis=1)
    total.columns=['ret','size','EP']
    total=total.dropna()
    total = total.sort_values(by='size')
    final = total.iloc[int(len(total) * 0.3):, :]#去掉 30%小市值
    #MKT
    faceq.iloc[i,0] = final.ret.mean()-rf.RF[i]/100
    final['VW'] = final.apply(lambda x: x['size'] * x['ret'], axis=1)
    facvw.iloc[i,0] = final.VW.sum()/final.iloc[:,1].sum()-rf.RF[i]/100
    #SMB/VMG
    SIZE1=final.iloc[:int(len(final)/2),:]#小
    SIZE1=SIZE1.sort_values(by='EP')
    SV=SIZE1.iloc[int(2*len(SIZE1)/3):,:]
    sveq=SV['ret'].mean()
    svvw=SV.VW.sum()/SV.iloc[:,1].sum()
    SM=SIZE1.iloc[int(1*len(SIZE1)/3):int(2*len(SIZE1)/3),:]
    smeq=SM['ret'].mean()
    smvw=SM.VW.sum()/SM.iloc[:,1].sum()
    SG=SIZE1.iloc[:int(1*len(SIZE1)/3),:]
    sgeq=SG['ret'].mean()
    sgvw=SG.VW.sum()/SG.iloc[:,1].sum()

    SIZE2 = final.iloc[int(len(final) / 2):,:]#大
    SIZE2 = SIZE2.sort_values(by='EP')
    BV=SIZE2.iloc[int(2*len(SIZE2)/3):,:]
    bveq=BV['ret'].mean()
    bvvw=BV.VW.sum()/BV.iloc[:,1].sum()
    BM=SIZE2.iloc[int(1*len(SIZE2)/3):int(2*len(SIZE2)/3),:]
    bmeq=BM['ret'].mean()
    bmvw=BM.VW.sum()/BM.iloc[:,1].sum()
    BG=SIZE2.iloc[:int(1*len(SIZE2)/3),:]
    bgeq=BG['ret'].mean()
    bgvw=BG.VW.sum()/BG.iloc[:,1].sum()
    faceq.iloc[i, 1] = (sveq +smeq +sgeq )/3-(bveq +bmeq +bgeq )/3
    facvw.iloc[i, 1] = (svvw +smvw +sgvw )/3-(bvvw +bmvw +bgvw )/3
    faceq.iloc[i, 2] = (sveq +bveq)/2-(sgeq+bgeq )/2
    facvw.iloc[i, 2] = (svvw +bvvw)/2-(sgvw+bgvw )/2
    print(i)

#*******************2.单因子 10 分组 10-1 多空组合因子调整收益**********************8#
import pandas as pd
import numpy as np
import statsmodels.api as sm
from scipy import stats
from NWttest import nwttest_1samp
factoreq=pd.read_csv('..\DataBase\\factorEW.csv')
factorvw=pd.read_csv('..\DataBase\\factorVW.csv')
resultew=pd.DataFrame(columns=factoreq.columns[1:],index=['ret','t','exret','t','capmret','t
','ff3ret','t','ff30ret','t'] )
resultvw=pd.DataFrame(columns=factoreq.columns[1:],index=['ret','t','exret','t','capmret','t
','ff3ret','t','ff30ret','t'] )
rf=pd.read_csv('RF.csv')
for i in range(len(factoreq.columns)-1):
    if factoreq.iloc[:,i+1].mean()>0:
        faceq=factoreq.iloc[:,i+1]
    else:
        faceq = -factoreq.iloc[:, i + 1]
    if factorvw.iloc[:,i+1].mean()>0:
        facvw=factorvw.iloc[:,i+1]
    else:
        facvw = -factorvw.iloc[:, i + 1]
    ####return
    reteq0 = faceq.dropna()
    resultew.iloc[0, i] = reteq0.mean()
    ttest = nwttest_1samp(reteq0, 0)
    resultew.iloc[1, i] = ttest.statistic
    retvw0 = facvw.dropna()
    resultvw.iloc[0, i] = retvw0.mean()
    ttest = nwttest_1samp(retvw0, 0)
    resultvw.iloc[1, i] = ttest.statistic

    ##excess return
    exeq=faceq-rf.RF/100
    exvw=facvw-rf.RF/100
    exeq0=exeq.dropna()
    resultew.iloc[2, i] = exeq0.mean()
    #ttest=stats.ttest_1samp(exeq0,0)
    ttest=nwttest_1samp(exeq0, 0)
    resultew.iloc[3, i]=ttest.statistic
    exvw0 = exvw.dropna()
    resultvw.iloc[2, i] = exvw0.mean()
    # ttest=stats.ttest_1samp(exvw0,0)
    ttest = nwttest_1samp(exvw0, 0)
    resultvw.iloc[3, i] = ttest.statistic
    ##CAPM
    ff3 = pd.read_csv(’ff3.csv’)
    capm=ff3.iloc[:,1]
    x1 = sm.add_constant(capm)
    X1=pd.concat([exeq[:-2],x1],axis=1)
    X1 = X1.dropna()
    X2 = pd.concat([exvw[:-2], x1], axis=1)
    X2 = X2.dropna()
    regeq = sm.OLS(X1.iloc[:,0],X1.iloc[:,1:]).fit()
    regvw = sm.OLS(X2.iloc[:,0],X2.iloc[:,1:]).fit()
    resultew.iloc[4, i]=regeq.params[0]
    resultew.iloc[5, i]=regeq.tvalues[0]
    resultvw.iloc[4, i] = regvw.params[0]
    resultvw.iloc[5, i] = regvw.tvalues[0]

    x1 = sm.add_constant(ff3.iloc[:,1:])
    X1 = pd.concat([exeq[:-2], x1], axis=1)
    X1 = X1.dropna()
    X2 = pd.concat([exvw[:-2], x1], axis=1)
    X2 = X2.dropna()
    regeq = sm.OLS(X1.iloc[:, 0], X1.iloc[:, 1:]).fit()
    regvw = sm.OLS(X2.iloc[:, 0], X2.iloc[:, 1:]).fit()
    resultew.iloc[6, i] = regeq.params[0]
    resultew.iloc[7, i] = regeq.tvalues[0]
    resultvw.iloc[6, i] = regvw.params[0]
    resultvw.iloc[7, i] = regvw.tvalues[0]

    ff30=pd.read_csv(’..\DataBase\\ff30.csv’)
    x1 = sm.add_constant(ff30.iloc[:, 1:])
    X1 = pd.concat([exeq, x1], axis=1)
    X1 = X1.dropna()
    X2 = pd.concat([exvw, x1], axis=1)
    X2 = X2.dropna()
    regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
    regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
    resultew.iloc[8, i] = regeq.params[0]
    resultew.iloc[9, i] = regeq.tvalues[0]
    resultvw.iloc[8, i] = regvw.params[0]
    resultvw.iloc[9, i] = regvw.tvalues[0]
    print(i)

#*****************3.单因子 10 分组检验各组因子调整收益***********************#
import pandas as pd
import numpy as np
import glob,os
import statsmodels.api as sm
from scipy import stats
from NWttest import nwttest_1samp
ret=pd.read_csv(’final_return.csv’)
size=pd.read_csv(’..\DataBase\\factor\\01size.csv’)
path = r’..\DataBase\\factor’
file = glob.glob(os.path.join(path, ”*.csv”))
rf=pd.read_csv(’RF.csv’)
k = []  # 每个因子一个表
for i in range(96):
    k.append(pd.read_csv(file[i]))
factor = []#因子名称
for i in range(len(file)):
    factor.append(file[i][29:-4])
for i in range(10):
    resultew=pd.DataFrame
(columns=factor,index=[’ret’,’t’,’exret’,’t’,’capmret’,’t’,’ff3ret’,’t’,’ff30ret’,’t’])
    resultvw = pd.DataFrame(columns=factor,index=[’ret’, ’t’, ’exret’, ’t’, ’capmret’, ’t’,
’ff3ret’, ’t’, ’ff30ret’, ’t’])
```

```
    for j in range(96):
        totalew =[]
        totalvw =[]
        for m in range(len(ret.columns)-1):
            final=pd.concat([size.iloc[:,m+1],k[j].iloc[:,m+1],ret.iloc[:,m+1]],axis=1)
            final.columns=['size','factor','ret']
            final=final.dropna()
            if len(final)==0:
                totalew.append(np.nan)
                totalvw.append(np.nan)
            else:
                final = final.sort_values(by='factor')
                final['VW'] = final.apply(lambda x: x['size'] * x['ret'], axis=1)
                total = final.iloc[int(len(final) / 10) * i:int(len(final) / 10) * (i +
1), :]
                totalew.append(total['ret'].mean())
                totalvw.append(total['VW'].sum() / total['size'].sum())
        faceq = pd.Series(totalew)
        facvw = pd.Series(totalvw)

        ####return
        reteq0 = faceq.dropna()
        resultew.iloc[0, j] = reteq0.mean()
        ttest = nwttest_1samp(reteq0, 0)
        resultew.iloc[1, j] = ttest.statistic
        retvw0 = facvw.dropna()
        resultvw.iloc[0, j] = retvw0.mean()
        ttest = nwttest_1samp(retvw0, 0)
        resultvw.iloc[1, j] = ttest.statistic

        ##excess return
        exeq = faceq - rf.RF / 100
        exvw = facvw - rf.RF / 100
        exeq0 = exeq.dropna()
        resultew.iloc[2, j] = exeq0.mean()
        # ttest=stats.ttest_1samp(exeq0,0)
        ttest = nwttest_1samp(exeq0, 0)
        resultew.iloc[3, j] = ttest.statistic
        exvw0 = exvw.dropna()
        resultvw.iloc[2, j] = exvw0.mean()
        # ttest=stats.ttest_1samp(exvw0,0)
        ttest = nwttest_1samp(exvw0, 0)
        resultvw.iloc[3, j] = ttest.statistic
        ##CAPM-alpha/FF3-alpha/去掉市值最小 30%股票后 adj-FF3-alpha
        ff3 = pd.read_csv(’ff3.csv’)
        capm = ff3.iloc[:, 1]
        x1 = sm.add_constant(capm)
        X1 = pd.concat([exeq[:-2], x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw[:-2], x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(X1.iloc[:, 0], X1.iloc[:, 1:]).fit()
        regvw = sm.OLS(X2.iloc[:, 0], X2.iloc[:, 1:]).fit()
        resultew.iloc[4, j] = regeq.params[0]
        resultew.iloc[5, j] = regeq.tvalues[0]
        resultvw.iloc[4, j] = regvw.params[0]
        resultvw.iloc[5, j] = regvw.tvalues[0]

        x1 = sm.add_constant(ff3.iloc[:, 1:])
        X1 = pd.concat([exeq[:-2], x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw[:-2], x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(X1.iloc[:, 0], X1.iloc[:, 1:]).fit()
        regvw = sm.OLS(X2.iloc[:, 0], X2.iloc[:, 1:]).fit()
        resultew.iloc[6, j] = regeq.params[0]
        resultew.iloc[7, j] = regeq.tvalues[0]
        resultvw.iloc[6, j] = regvw.params[0]
        resultvw.iloc[7, j] = regvw.tvalues[0]

        ff30 = pd.read_csv(’..\DataBase\\ff30.csv’)
        x1 = sm.add_constant(ff30.iloc[:, 1:])
        X1 = pd.concat([exeq, x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw, x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(X1.iloc[:, 0], X1.iloc[:, 1:]).fit()
        regvw = sm.OLS(X2.iloc[:, 0], X2.iloc[:, 1:]).fit()
        resultew.iloc[8, j] = regeq.params[0]
        resultew.iloc[9, j] = regeq.tvalues[0]
        resultvw.iloc[8, j] = regvw.params[0]
        resultvw.iloc[9, j] = regvw.tvalues[0]
    resultew.to_csv(’RESLew’+str(i)+’.csv’)
    resultvw.to_csv(’RESLvw’ + str(i) + ’.csv’)

#***********************4.各因子与size因子独立双变量分组检验结果
***********************************#
import pandas as pd
import numpy as np
import glob,os
import statsmodels.api as sm
from NWttest import nwttest_1samp
ret=pd.read_csv(’..\DataBase\\final_return.csv’)
size=pd.read_csv(’..\DataBase\\factor\\01size.csv’)
size=size.iloc[:,:-1]
size.columns=ret.columns
path = r’..\DataBase\\factor’
file = glob.glob(os.path.join(path, ”*.csv”))
rf=pd.read_csv(’..\DataBase\\RF.csv’)
k = []  # 每个因子一个表
for i in range(96):
    k.append(pd.read_csv(file[i]))
factor = []#因子名称
for i in range(len(file)):
    factor.append(file[i][29:-4])

def gendata(x1,x2,ret,size,m=5,n=5):
    x1 = x1[~np.isnan(ret)]
    x1=x1[~np.isnan(size)]
    x2 = x2[~np.isnan(ret)]
    x2 = x2[~np.isnan(size)]
    x1sort = x1.apply(lambda x: np.argsort(x), axis=0)
    x2sort = x2.apply(lambda x: np.argsort(x), axis=0)
    datacolumn = []
    for a in range(m):
        for b in range(n):
            datacolumn.append(str(a + 1) + ’X’ + str(b + 1))
    dfeq=pd.DataFrame(columns=datacolumn,index=[i for i in range(len(ret.columns)-1)])
    dfvw = pd.DataFrame(columns=datacolumn, index=[i for i in range(len(ret.columns) - 1)])
    for i in range(len(ret.columns)-1):
        truelistx1 = x1sort.iloc[:,i+1 ]
        truelistx2 = x2sort.iloc[:, i+1]
        truelistx1 = truelistx1[~(truelistx1 == -1)].sort_values()
        truelistx1=truelistx1.index
        truelistx2 = truelistx2[~(truelistx2 == -1)].sort_values()
        truelistx2 = truelistx2.index
        if len(truelistx1)<5 or len(truelistx2)<5:
            dfeq.iloc[i,:] =np.nan
            dfvw.iloc[i,:] =np.nan
        else:
            x1lines = np.array_split(np.array(truelistx1), m)
            x2lines = np.array_split(np.array(truelistx2), n)
            for a in range(m):
                x1use = x1lines[a]
                for b in range(n):
                    x2use = x2lines[b]
                    tempindex = np.intersect1d(x1use, x2use)
                    if len(tempindex)==0:
                        dfeq.iloc[i, a * m + b]=np.nan
                        dfvw.iloc[i, a * m + b] = np.nan
                    else:
                        dfeq.iloc[i, a * m + b] = ret.iloc[tempindex, i + 1].mean()
                        total = pd.concat([ret.iloc[tempindex, i + 1], size.iloc[tempindex,
i + 1]], axis=1)
                        total.columns = ['ret', 'size']
                        total['vw'] = total.apply(lambda x: x['size'] * x['ret'], axis=1)
                        dfvw.iloc[i, a * m + b] = total['vw'].sum() / total['size'].sum()
    return dfeq,dfvw


RETEQ,RETVQ=[],[]
for i in range(len(k)):
    k[i] = k[i].iloc[:, :264]
    k[i].columns = ret.columns
    reteq, retvw = gendata(size, k[i], ret, size)
    RETEQ.append(reteq)
    RETVQ.append(retvw)

#删除 SIZE 因子本身（不与自身分组）
del RETEQ[0]
del RETVQ[0]
del factor[0]

for l in range(5):
    longew=pd.concat([i.iloc[:, l*5+4] for i in RETEQ], axis=1)
    shortew = pd.concat([i.iloc[:, l * 5 ] for i in RETEQ], axis=1)
    longvw = pd.concat([i.iloc[:, l * 5 + 4] for i in RETVQ], axis=1)
    shortvw = pd.concat([i.iloc[:, l * 5] for i in RETVQ], axis=1)
    longew.columns = factor
    longvw.columns = factor
    shortew.columns = factor
    shortvw.columns = factor
    lsew=longew-shortew
    lsvw=longvw-shortvw
    resultew = pd.DataFrame(columns=factor,
                            index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',
’ff30ret’, ’t’])
    resultvw = pd.DataFrame(columns=factor,
                            index=[’ret’, ’t’, ’exret’, ’t’, ’capmret’, ’t’, ’ff3ret’, ’t’,
’ff30ret’, ’t’])
    for j in range(len(factor)):
        faceq, facvw = lsew.iloc[:, j], lsvw.iloc[:, j]
        reteq0 = faceq.dropna()
        resultew.iloc[0, j] = reteq0.mean()
        ttest = nwttest_1samp(reteq0, 0)
        resultew.iloc[1, j] = ttest.statistic
        retvw0 = facvw.dropna()
        resultvw.iloc[0, j] = retvw0.mean()
        ttest = nwttest_1samp(retvw0, 0)
        resultvw.iloc[1, j] = ttest.statistic
        ##excess return
        exeq = faceq - rf.RF / 100
        exvw = facvw - rf.RF / 100
        exeq0 = exeq.dropna()
        resultew.iloc[2, j] = exeq0.mean()
        # ttest=stats.ttest_1samp(exeq0,0)
        ttest = nwttest_1samp(exeq0, 0)
        resultew.iloc[3, j] = ttest.statistic
        exvw0 = exvw.dropna()
        resultvw.iloc[2, j] = exvw0.mean()
        # ttest=stats.ttest_1samp(exvw0,0)
        ttest = nwttest_1samp(exvw0, 0)
        resultvw.iloc[3, j] = ttest.statistic
        ##CAPM
        ff3 = pd.read_csv(’..\DataBase\\ff3.csv’)
        capm = ff3.iloc[:, 1]
        x1 = sm.add_constant(capm)
        X1 = pd.concat([exeq[:-2], x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw[:-2], x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[4, j] = regeq.params[0]
        resultew.iloc[5, j] = regeq.tvalues[0]
        resultvw.iloc[4, j] = regvw.params[0]
        resultvw.iloc[5, j] = regvw.tvalues[0]
        x1 = sm.add_constant(ff3.iloc[:, 1:])
        X1 = pd.concat([exeq[:-2], x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw[:-2], x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[6, j] = regeq.params[0]
        resultew.iloc[7, j] = regeq.tvalues[0]
        resultvw.iloc[6, j] = regvw.params[0]
        resultvw.iloc[7, j] = regvw.tvalues[0]
        ff30 = pd.read_csv(’..\DataBase\\ff30.csv’)
        x1 = sm.add_constant(ff30.iloc[:, 1:])
        X1 = pd.concat([exeq, x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw, x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[8, j] = regeq.params[0]
        resultew.iloc[9, j] = regeq.tvalues[0]
        resultvw.iloc[8, j] = regvw.params[0]
        resultvw.iloc[9, j] = regvw.tvalues[0]
    resultew.to_csv(’longshort55SIZEEW’ + str(l) + ’.csv’)
    resultvw.to_csv(’longshort55SIZEVW’ + str(l) + ’.csv’)

for u in range(25):
    DATAEQ=pd.concat([i.iloc[:,u] for i in RETEQ],axis=1)
    DATAVQ = pd.concat([i.iloc[:, u] for i in RETVQ],axis=1)
    DATAEQ.columns=factor
    DATAVQ.columns = factor
    resultew = pd.DataFrame(columns=factor,
                                index=[’ret’, ’t’, ’exret’, ’t’, ’capmret’, ’t’, ’ff3ret’,
’t’, ’ff30ret’, ’t’])
    resultvw = pd.DataFrame(columns=factor,
                                index=[’ret’, ’t’, ’exret’, ’t’, ’capmret’, ’t’, ’ff3ret’,
’t’, ’ff30ret’, ’t’])
    for j in range(len(factor)):
        faceq, facvw=DATAEQ.iloc[:,j],DATAVQ.iloc[:,j]
        reteq0 = faceq.dropna()
        resultew.iloc[0, j] = reteq0.mean()
        ttest = nwttest_1samp(reteq0, 0)
        resultew.iloc[1, j] = ttest.statistic
        retvw0 = facvw.dropna()
        resultvw.iloc[0, j] = retvw0.mean()
        ttest = nwttest_1samp(retvw0, 0)
        resultvw.iloc[1, j] = ttest.statistic

        ##excess return
        exeq = faceq - rf.RF / 100
        exvw = facvw - rf.RF / 100
        exeq0 = exeq.dropna()
        resultew.iloc[2, j] = exeq0.mean()
        # ttest=stats.ttest_1samp(exeq0,0)
        ttest = nwttest_1samp(exeq0, 0)
        resultew.iloc[3, j] = ttest.statistic
        exvw0 = exvw.dropna()
        resultvw.iloc[2, j] = exvw0.mean()
        # ttest=stats.ttest_1samp(exvw0,0)
        ttest = nwttest_1samp(exvw0, 0)
        resultvw.iloc[3, j] = ttest.statistic
        ##CAPM
        ff3 = pd.read_csv(’..\DataBase\\ff3.csv’)
        capm = ff3.iloc[:, 1]
        x1 = sm.add_constant(capm)
        X1 = pd.concat([exeq[:-2], x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw[:-2], x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[4, j] = regeq.params[0]
        resultew.iloc[5, j] = regeq.tvalues[0]
        resultvw.iloc[4, j] = regvw.params[0]
        resultvw.iloc[5, j] = regvw.tvalues[0]

        x1 = sm.add_constant(ff3.iloc[:, 1:])
        X1 = pd.concat([exeq[:-2], x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw[:-2], x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[6, j] = regeq.params[0]
        resultew.iloc[7, j] = regeq.tvalues[0]
        resultvw.iloc[6, j] = regvw.params[0]
        resultvw.iloc[7, j] = regvw.tvalues[0]

        ff30 = pd.read_csv(’C:\\Users\\15083\Desktop\\ff30.csv’)
        x1 = sm.add_constant(ff30.iloc[:, 1:])
        X1 = pd.concat([exeq, x1], axis=1)
```

```python
        X1 = X1.dropna()
        X2 = pd.concat([exvw, x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[8, j] = regeq.params[0]
        resultew.iloc[9, j] = regeq.tvalues[0]
        resultvw.iloc[8, j] = regvw.params[0]
        resultvw.iloc[9, j] = regvw.tvalues[0]
    resultew.to_csv('EW55size'+reteq.columns[u]+'.csv')
    resultvw.to_csv('VW55size' + reteq.columns[u] + '.csv')


#***********************5.各因子与BM因子独立双变量分组检验结果
***********************************#
import pandas as pd
import numpy as np
import glob, os
import statsmodels.api as sm
from NWttest import nwttest_1samp

ret = pd.read_csv('..\DataBase\\final_return.csv')
size = pd.read_csv('..\DataBase\\factor\\01size.csv')
BM = pd.read_csv('..\DataBase\\factor\\28BM.csv')
size = size.iloc[:, :-1]
size.columns = ret.columns
path = r'..\DataBase\\factor'
file = glob.glob(os.path.join(path, "*.csv"))
rf = pd.read_csv('..\DataBase\\RF.csv')
k = []  # 每个因子一个表
for i in range(96):
    k.append(pd.read_csv(file[i]))
factor = []  # 因子名称
for i in range(len(file)):
    factor.append(file[i][29:-4])

BM = BM.iloc[:, :-1]
BM.columns = ret.columns


def gendata(x1, x2, ret, size, m=5, n=5):
    x1 = x1[~np.isnan(ret)]
    x1 = x1[~np.isnan(size)]
    x2 = x2[~np.isnan(ret)]
    x2 = x2[~np.isnan(size)]
    x1sort = x1.apply(lambda x: np.argsort(x), axis=0)
    x2sort = x2.apply(lambda x: np.argsort(x), axis=0)
    datacolumn = []

    for a in range(m):
        for b in range(n):
            datacolumn.append(str(a + 1) + ’X’ + str(b + 1))
    dfeq = pd.DataFrame(columns=datacolumn, index=[i for i in range(len(ret.columns) - 1)])
    dfvw = pd.DataFrame(columns=datacolumn, index=[i for i in range(len(ret.columns) - 1)])
    for i in range(len(ret.columns) - 1):
        truelistx1 = x1sort.iloc[:, i + 1]
        truelistx2 = x2sort.iloc[:, i + 1]
        truelistx1 = truelistx1[~(truelistx1 == -1)].sort_values()
        truelistx1 = truelistx1.index
        truelistx2 = truelistx2[~(truelistx2 == -1)].sort_values()
        truelistx2 = truelistx2.index
        if len(truelistx1) < 5 or len(truelistx2) < 5:
            dfeq.iloc[i, :] = np.nan
            dfvw.iloc[i, :] = np.nan
        else:
            x1lines = np.array_split(np.array(truelistx1), m)
            x2lines = np.array_split(np.array(truelistx2), n)
            for a in range(m):
                x1use = x1lines[a]
                lsew=[]
                lsvw = []
                for b in range(n):
                    x2use = x2lines[b]
                    tempindex = np.intersect1d(x1use, x2use)
                    if len(tempindex) == 0:
                        dfeq.iloc[i, a * m + b] = np.nan
                        dfvw.iloc[i, a * m + b] = np.nan
                        lsew.append(0)
                        lsvw.append(0)
                    else:
                        dfeq.iloc[i, a * m + b] = ret.iloc[tempindex, i + 1].mean()
                        total = pd.concat([ret.iloc[tempindex, i + 1], size.iloc[tempindex, i + 1]], axis=1)
                        total.columns = [’ret’, ’size’]
                        total[’vw’] = total.apply(lambda x: x[’size’] * x[’ret’], axis=1)
                        dfvw.iloc[i, a * m + b] = total[’vw’].sum() / total[’size’].sum()
                        lsew.append(ret.iloc[tempindex, i + 1].mean())
                        lsvw.append(total[’vw’].sum() / total[’size’].sum())
    return dfeq, dfvw


RETEQ, RETVQ = [], []
for i in range(len(k)):
    k[i] = k[i].iloc[:, :264]
    k[i].columns = ret.columns
    reteq, retvw = gendata(BM, k[i], ret, size)
    RETEQ.append(reteq)
    RETVQ.append(retvw)

#删除 BM 因子本身（不与自身分组）
del RETEQ[27]
del RETVQ[27]
del factor[27]

for l in range(5):
    longew=pd.concat([i.iloc[:, l*5+4] for i in RETEQ], axis=1)
    shortew = pd.concat([i.iloc[:, l * 5 ] for i in RETEQ], axis=1)
    longvw = pd.concat([i.iloc[:, l * 5 + 4] for i in RETVQ], axis=1)
    shortvw = pd.concat([i.iloc[:, l * 5] for i in RETVQ], axis=1)
    longew.columns = factor
    longvw.columns = factor
    shortew.columns = factor
    shortvw.columns = factor
    lsew=longew-shortew
    lsvw=longvw-shortvw
    resultew = pd.DataFrame(columns=factor,
                            index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',
'ff30ret', 't'])
    resultvw = pd.DataFrame(columns=factor,
                            index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',
'ff30ret', 't'])
    for j in range(len(factor)):
        faceq, facvw = lsew.iloc[:, j], lsvw.iloc[:, j]
        reteq0 = faceq.dropna()
        resultew.iloc[0, j] = reteq0.mean()
        ttest = nwttest_1samp(reteq0, 0)
        resultew.iloc[1, j] = ttest.statistic
        retvw0 = facvw.dropna()
        resultvw.iloc[0, j] = retvw0.mean()
        ttest = nwttest_1samp(retvw0, 0)
        resultvw.iloc[1, j] = ttest.statistic
        ##excess return
        exeq = faceq - rf.RF / 100
        exvw = facvw - rf.RF / 100
        exeq0 = exeq.dropna()
        resultew.iloc[2, j] = exeq0.mean()
        # ttest=stats.ttest_1samp(exeq0,0)
        ttest = nwttest_1samp(exeq0, 0)
        resultew.iloc[3, j] = ttest.statistic
        exvw0 = exvw.dropna()
        resultvw.iloc[2, j] = exvw0.mean()
        # ttest=stats.ttest_1samp(exvw0,0)
        ttest = nwttest_1samp(exvw0, 0)
        resultvw.iloc[3, j] = ttest.statistic
        ##CAPM
        ff3 = pd.read_csv(’..\DataBase\\ff3.csv’)
        capm = ff3.iloc[:, 1]
        x1 = sm.add_constant(capm)
        X1 = pd.concat([exeq[:-2], x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw[:-2], x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[4, j] = regeq.params[0]
        resultew.iloc[5, j] = regeq.tvalues[0]
        resultvw.iloc[4, j] = regvw.params[0]
        resultvw.iloc[5, j] = regvw.tvalues[0]
        x1 = sm.add_constant(ff3.iloc[:, 1:])
        X1 = pd.concat([exeq[:-2], x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw[:-2], x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[6, j] = regeq.params[0]
        resultew.iloc[7, j] = regeq.tvalues[0]
        resultvw.iloc[6, j] = regvw.params[0]
        resultvw.iloc[7, j] = regvw.tvalues[0]
        ff30 = pd.read_csv(’..\DataBase\\ff30.csv’)
        x1 = sm.add_constant(ff30.iloc[:, 1:])
        X1 = pd.concat([exeq, x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw, x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[8, j] = regeq.params[0]
        resultew.iloc[9, j] = regeq.tvalues[0]
        resultvw.iloc[8, j] = regvw.params[0]
        resultvw.iloc[9, j] = regvw.tvalues[0]
    resultew.to_csv('longshort55BMEW' + str(l) + '.csv')
    resultvw.to_csv('longshort55BMVW' + str(l) + '.csv')

for u in range(25):
    DATAEQ = pd.concat([i.iloc[:, u] for i in RETEQ], axis=1)
    DATAVQ = pd.concat([i.iloc[:, u] for i in RETVQ], axis=1)
    DATAEQ.columns = factor
    DATAVQ.columns = factor
    resultew = pd.DataFrame(columns=factor,
                            index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',
'ff30ret', 't'])
    resultvw = pd.DataFrame(columns=factor,
                            index=['ret', 't', 'exret', 't', 'capmret', 't', 'ff3ret', 't',
'ff30ret', 't'])
    for j in range(len(factor)):
        faceq, facvw = DATAEQ.iloc[:, j], DATAVQ.iloc[:, j]
        reteq0 = faceq.dropna()
        resultew.iloc[0, j] = reteq0.mean()
        ttest = nwttest_1samp(reteq0, 0)
        resultew.iloc[1, j] = ttest.statistic
        retvw0 = facvw.dropna()
        resultvw.iloc[0, j] = retvw0.mean()
        ttest = nwttest_1samp(retvw0, 0)
        resultvw.iloc[1, j] = ttest.statistic
        ##excess return
        exeq = faceq - rf.RF / 100
        exvw = facvw - rf.RF / 100
        exeq0 = exeq.dropna()
        resultew.iloc[2, j] = exeq0.mean()
        # ttest=stats.ttest_1samp(exeq0,0)
        ttest = nwttest_1samp(exeq0, 0)
        resultew.iloc[3, j] = ttest.statistic
        exvw0 = exvw.dropna()
        resultvw.iloc[2, j] = exvw0.mean()
        # ttest=stats.ttest_1samp(exvw0,0)
        ttest = nwttest_1samp(exvw0, 0)
        resultvw.iloc[3, j] = ttest.statistic
        ##CAPM
        ff3 = pd.read_csv('..\DataBase\\ff3.csv')
        capm = ff3.iloc[:, 1]
        x1 = sm.add_constant(capm)
        X1 = pd.concat([exeq[:-2], x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw[:-2], x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[4, j] = regeq.params[0]
        resultew.iloc[5, j] = regeq.tvalues[0]
        resultvw.iloc[4, j] = regvw.params[0]
        resultvw.iloc[5, j] = regvw.tvalues[0]
        x1 = sm.add_constant(ff3.iloc[:, 1:])
        X1 = pd.concat([exeq[:-2], x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw[:-2], x1], axis=1)
        X2 = X2.dropna()
        regeq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvw = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[6, j] = regeq.params[0]
        resultew.iloc[7, j] = regeq.tvalues[0]
        resultvw.iloc[6, j] = regvw.params[0]
        resultvw.iloc[7, j] = regvw.tvalues[0]
        ff30 = pd.read_csv('..\DataBase\\ff30.csv')
        x1 = sm.add_constant(ff30.iloc[:, 1:])
        X1 = pd.concat([exeq, x1], axis=1)
        X1 = X1.dropna()
        X2 = pd.concat([exvw, x1], axis=1)
        X2 = X2.dropna()
        regeqq = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        regvwq = sm.OLS(np.asarray(X2.iloc[:, 0]), np.asarray(X2.iloc[:, 1:])).fit()
        resultew.iloc[8, j] = regeqq.params[0]
        resultew.iloc[9, j] = regeqq.tvalues[0]
        resultvw.iloc[8, j] = regvwq.params[0]
        resultvw.iloc[9, j] = regvwq.tvalues[0]
    resultew.to_csv('EW55BM' + reteq.columns[u] + '.csv')
    resultvw.to_csv('VW55BM' + reteq.columns[u] + '.csv')


#******************************6.96 项因子 fama macbeth 回归检验结果
****************************************#
import pandas as pd
import numpy as np
import glob,os
```

```python
import statsmodels.api as sm
from sklearn.preprocessing import scale
from factorNWttest import nwttest_1samp
ret=pd.read_csv('final_return.csv')
size=pd.read_csv('..\DataBase\\factor\\01size.csv')
path = r'..\DataBase\\factor'
file = glob.glob(os.path.join(path, "*.csv"))
rf=pd.read_csv('..\DataBase\\RF.csv')
k = []  # 每个因子一个表
for i in range(96):
    k.append(pd.read_csv(file[i]))
factor = []#因子名称
for i in range(len(file)):
    factor.append(file[i][29:-4])
factor.insert(0, 'constant')
xishu=pd.DataFrame(index=ret.columns[1:],columns=factor)
for i in range(ret.shape[1]-1):
    final=pd.concat([j.iloc[:,i+1] for j in k],axis=1)
    final.columns=factor[1:]
    final = sm.add_constant(final)
    final['ret']=ret.iloc[:,i+1]
    X1=final.dropna()
    if len(X1)==0:
        xishu.iloc[i,:]=np.nan
    else:
        X1.iloc[:, 1:-1] = scale(X1.iloc[:, 1:-1], axis=0)
        reg = sm.OLS(np.asarray(X1.iloc[:, 0]), np.asarray(X1.iloc[:, 1:])).fit()
        xishu.iloc[i,:]=reg.params
FM=pd.DataFrame(columns=xishu.columns,index=['mean','t-stats'])
gen=xishu.dropna()
for i in range(xishu.shape[1]):
    fa = gen.iloc[:, i]
    FM.iloc[0, i] = fa.mean()#截面系数时序均值
    ttest = nwttest_1samp(fa, 0)#t检验-因子系数是否异于0
    FM.iloc[1, i] = ttest.statistic

FM.to_csv('FM.csv')
```

g) DFN.py

```python
# -*- coding:utf-8 -*-
'''
@description:
深度前馈网络库函数，依赖mxnet和gpu，在有GPU的环境下运行。
'''
from mxnet import gluon,nd,gpu,autograd
from mxnet.gluon import loss as gloss,nn,data as gdata
import mxnet as mx
import numpy as np

class DFN(nn.Block):
    def __init__(self, outputdim, batch_size=10, epoch=10, depth=1, neuralset=[],
                 activiatemethod='relu', ctx=gpu(), trainmethod='sgd', lr=0.5, **kwargs):
        '''
        主函数，用于深度前馈网络训练
        :param outputdim: 输出维度
        :param batch_size: 小批量大小
        :param epoch: 训练论述
        :param depth: 神经网络深度
        :param neuralset: 各层神经元数，list形式 eg. [96,50,10]
        :param activiatemethod: 激活函数
        :param ctx: 训练用设备 默认gpu
        :param trainmethod: 训练器
        :param lr: 学习率
        :param kwargs: 其他参数
        '''
        super(DFN,self).__init__(**kwargs)
        mx.random.seed(521,ctx=gpu())
        mx.random.seed(521)
        self.ctx = ctx
        self.outputdim = outputdim
        self.net = nn.Sequential()
        self.neuralset = neuralset
        self.activiatemethod = activiatemethod
        self.depth = depth
        self.trainmethod = trainmethod
        self.lr = lr
        if neuralset:
            pass
        else:
            neuralset = [256]*depth
        for i in range(depth):
            self.net.add(nn.Dense(neuralset[i], activation=activiatemethod))
        self.net.add(nn.Dense(outputdim))
        self.net.initialize(ctx=self.ctx)
        self.Trainer = gluon.Trainer(self.net.collect_params(), trainmethod,
{’learning_rate’: lr})
        self.loss = gloss.L2Loss()
        self.batch_size = batch_size
        self.epoch = epoch
        print(’network initializing finished’)

    def retrain(self,Xnd,Ynd,data_iter,k):
        knew = 0
        self.net.initialize(ctx=self.ctx, force_reinit=True)
        self.Trainer = gluon.Trainer(self.net.collect_params(), self.trainmethod,
{’learning_rate’: self.lr/2})
        print(’reinitialize finished and retrain begin’)
        for i in range(k - 1):
            # with mx.autograd.record():
            #     l = self.loss(self.net(Xnd), Ynd).mean()
            # l.backward()
            # self.Trainer.step(4400)
            for xtrain, ytrain in data_iter:
                ytrain = nd.array(ytrain, ctx=self.ctx)
                with mx.autograd.record():
                    l = self.loss(self.net(xtrain), ytrain).mean()
                l.backward()
                self.Trainer.step(1)
                print(’#’)
            knew += 1
            lossout = self.loss(self.net(Xnd), Ynd)
            print(’epoch ’ + str(i) + ’: loss is %f’ % lossout.mean().asnumpy())
            if lossout.mean().asnumpy()>1:
                self.retrain(Xnd, Ynd, data_iter, knew)
                break
            elif np.isnan(lossout.mean().asnumpy()):
                self.retrain(Xnd, Ynd, data_iter, knew)
                break


        # print(lossout)


    def fit(self,X,Y):
        Xnd = nd.array(X,ctx = self.ctx)
        Ynd = nd.array(Y,ctx = self.ctx)
        dataset = gdata.ArrayDataset(Xnd, Ynd)
        # 随机读取小批量
        data_iter = gdata.DataLoader(dataset, self.batch_size, shuffle=False)
        k = 1
        for i in range(self.epoch):
            # with mx.autograd.record():
            #     l = self.loss(self.net(Xnd),Ynd).mean()
            # l.backward()
            # self.Trainer.step(4400)
            for xtrain, ytrain in data_iter:
                ytrain = nd.array(ytrain,ctx=self.ctx)
                with mx.autograd.record():
                    l = self.loss(self.net(xtrain), ytrain).mean()
                l.backward()
                self.Trainer.step(1)
                print(’#’, )
            k += 1
            lossout = self.loss(self.net(Xnd), Ynd)
            print(’epoch ’ + str(i) + ’: loss is %f’ % lossout.mean().asnumpy())
            if lossout.mean().asnumpy()>1:
                retrainNN = True
                break
            elif np.isnan(lossout.mean().asnumpy()):
                retrainNN = True
                break
            else:
                retrainNN = False
        if retrainNN:
            self.retrain(Xnd,Ynd,data_iter,k)
        print(’all finished’)

    def predict(self,Xtest):
        Xtestnd = nd.array(Xtest,ctx = self.ctx)
        return list(map(lambda x:x[0],nd.array(self.net(Xtestnd),ctx=mx.cpu()).asnumpy()))
```

h) RNNMODEL.py

```
# -*- coding:utf-8 -*-
'''
@description:
  用于构建循环神经网络
'''

import mxnet as mx
from mxnet import nd,gluon,gpu
from mxnet.gluon import rnn,nn,loss as gloss,data as gdata
import numpy as np

class BaseRnn(nn.Block):
    def __init__(self,num_feature,num_hidden,num_layers,ntype=’RNN’,bidirectional =
False,dropout=0,**kwargs):
        '''
        基本循环神经网络单元
        :param num_feature: 特征个数
        :param num_hidden: 隐层神经元个数
        :param num_layers: 隐层数目
        :param ntype: 神经网络类型，默认 RNN
        :param bidirectional: 是否双向，默认否
        :param dropout: 丢弃率，即神经元多大概率不激活
        :param kwargs: 其他参数
        '''
        super(BaseRnn,self).__init__(**kwargs)
        self.num_hidden = num_hidden
        if ntype == ’RNN’:
            with self.name_scope():
                self.rnn =
rnn.RNN(num_hidden,num_layers,input_size=num_feature,bidirectional=bidirectional,
layout=’TNC’,dropout=dropout)
                self.decoder = nn.Dense(1, in_units=num_hidden)
        elif ntype == ’LSTM’:
            with self.name_scope():
                self.rnn =
rnn.LSTM(num_hidden,num_layers,input_size=num_feature,bidirectional=bidirectional,
layout=’TNC’,dropout=dropout)
                self.decoder = nn.Dense(1, in_units=num_hidden)
        elif ntype == ’GRU’:
            with self.name_scope():
                self.rnn =
rnn.GRU(num_hidden,num_layers,input_size=num_feature,bidirectional=bidirectional,
layout='TNC',dropout=dropout)
                self.decoder = nn.Dense(1, in_units=num_hidden)

    def forward(self, inputs, hidden):
        output, hidden = self.rnn(inputs, hidden)
        decoded = self.decoder(output.reshape((-1, self.num_hidden)))
        return decoded, hidden

    def begin_state(self, *args, **kwargs):
        return self.rnn.begin_state(*args, **kwargs)


class lstmmodule(object):
    def
__init__(self,num_feature,num_hidden,num_layers,epoch,batch_size,ntype='LSTM',bidirectional
= False,dropout=0,trainmethod = 'adam',\
                 lr=0.01,loss = gloss.L1Loss(),ctx =
gpu(0),datashuffle=False,initialfunc=mx.init.Xavier(),prin=False,**kwargs):
        '''
        用于循环神经网络的训练和预测,  主函数
        :param num_feature: 特征个数
        :param num_hidden: 隐层神经元数目
        :param num_layers: 隐层数
        :param epoch: 训练轮数
        :param batch_size: 小批量大小, 在这里是股票实体数目多少
        :param ntype: 神经网络类型, 默认 LSTM
        :param bidirectional: 是否双向神经网络, 默认否
        :param dropout: 丢弃率, 默认不丢弃
        :param trainmethod: 训练器, 默认 adam
        :param lr: 学习率
        :param loss: 损失函数
        :param ctx: 训练设备, 默认 gpu
        :param datashuffle: 是否随机打乱数据, 默认否
        :param initialfunc: 初始化方法, 默认 Xaiver
        :param prin: 是否输出训练指示, 默认否
        :param kwargs: 其他参数
        '''
        super(lstmmodule,self).__init__(**kwargs)
        mx.random.seed(521,ctx=gpu())
        mx.random.seed(521)
        self.ctx = ctx
        self.net = BaseRnn(num_feature,num_hidden,num_layers,ntype,bidirectional,dropout)
        self.trainmethod = trainmethod
        self.lr = lr
        self.intialfunc = initialfunc
        self.net.collect_params().initialize(self.intialfunc, ctx=self.ctx)
        self.Trainer = gluon.Trainer(self.net.collect_params(), trainmethod,
{’learning_rate’: lr})
        self.loss = loss
        self.datashuffle = datashuffle
        self.epoch = epoch
        self.batch_size = batch_size
        self.prin = prin

    def fit(self,X,Y):
        Xnd = nd.array(X,ctx = self.ctx)
        Ynd = nd.array(Y,ctx = self.ctx)
        for i in range(self.epoch):
            state = self.net.begin_state(batch_size=self.batch_size, ctx=self.ctx)
            for s in state:
                s.detach()
            with mx.autograd.record():
                (output, state) = self.net(Xnd, state)
                l = self.loss(output.reshape(len(output)//self.batch_size,self.batch_size),
Ynd).mean()
            l.backward()
            self.Trainer.step(1)
            if self.prin:
                print(’#’)
                if i == self.epoch -1:
                    print(’the last epoch loss is ’, l.asnumpy())
        if self.prin:
            print(’all finished’)

    def predict(self,Xtest):
        state = self.net.begin_state(batch_size=self.batch_size, ctx=self.ctx)
        Xtestnd = nd.array(Xtest,ctx = self.ctx)
        (Y,statenew) = self.net(Xtestnd,state)
        return nd.array(Y,ctx=mx.cpu()).asnumpy()
```

i) Ensembleall.py

```
# -*- coding:utf-8 -*-
"""
@description:
    集成所有函数模型模块
"""
import numpy as np
class Ensemblelr(object):
    def __init__(self,classifymodellist,newmodellist,modelname):
        '''

        :param classifymodellist: 非 RNN/LSTM 模型
        :param newmodellist: RNN/LSTM 模型
        :param modelname: 所有模型名称,
        注：PLS 放到传统模型倒数第三位
        '''
        self.cmodelist = classifymodellist
        self.nml = newmodellist
        self.modelname = modelname
        self.cmodelnum = len(classifymodellist)
        self.nmodelnum = len(newmodellist)
        print(’Ensembleall model initializing’)

    def fit(self,TX,TY,NX,NY):
        for i in range(self.cmodelnum):
            self.cmodelist[i].fit(TX, TY)
            print(self.modelname[i]+’ finished’)
        for i in range(self.nmodelnum):
            self.nml[i].fit(NX, NY)
            print(self.modelname[self.cmodelnum + i])
        print(’fit finished’)

    def predict(self, TXtest, NXtest,indlist,length):
        predictlist = []
        for i in range(self.cmodelnum):
            predictlist.append(self.cmodelist[i].predict(TXtest))
        predictlist[-3] = list(map(lambda x: x[0], predictlist[-3]))  # 为了 PLS 的输出结果变
成数值
        for i in range(self.nmodelnum):
            KP = self.nml[i].predict(NXtest)
            predictlist.append(np.array([KP[m][0] for m in range(3571 * (length - 1), 3571 *
length)])[~indlist])
        return list(np.mean(predictlist, axis=0))
```

j) transecfee.py

```python
# -*- coding:utf-8 -*-
'''
@description
  用于输出交易费用之后的结果
'''
import numpy as np
import pandas as pd
import os
import gc
from NWttest import nwttest_1samp
import statsmodels.formula.api as smf

factorlist = os.listdir('../Database/returnseries/12')
rf=pd.read_csv('../Database/RF.csv')#无风险 rf
rf12=rf.iloc[12:-1,:]
def trasecfee(feerate):
    '''
    :param feerate: 交易费率
    :return:不输出只打印结果
    '''
    for i in range(len(factorlist)):
        temp = pd.read_csv('../Database/returnseries/12/' + factorlist[i])
        long_short = np.array(temp['long-short'])
        afterLS = list(long_short - feerate*2)
        length = 12
        name = factorlist[i]
        T_value = []
        Mean = []
        p_value = []
        sharpratio = []
        Std = []
        TO = [afterLS]
        for l in TO:
            t_test = nwttest_1samp(l, 0, L=1)
            mean = np.average(l) * 12- rf12.mean().tolist()[0] * 12 / 100
            STD = np.std(l) * np.sqrt(12)
            sharp = (mean) / STD
            T_value.append(t_test.statistic)
            p_value.append(t_test.pvalue)
            Mean.append(mean)
            Std.append(STD)
            sharpratio.append(sharp)
        print(name, 'long-short')
        print('mean', Mean[0] / 12)
        print('t-statistic', '('+str(round(T_value[0],4))+')')
        ff3 = pd.read_csv('../Database/ff3.csv')
        ff5 = pd.read_csv('../Database/ff5.csv')
        A = pd.DataFrame(afterLS, columns=['long-short'])
        M = pd.concat([A], axis=1)
        alpha3 = []
        t3 = []
        t5 = []
        alpha5 = []
        for i in range(1):
            X1 = ff3.iloc[length:, 1:]
            X2 = ff5.iloc[length:, 1:]
            Y = M.iloc[:-2, i]
            Y.index = X1.index
            Y = Y - rf12.RF[:-1] / 100
            used1 = {'X': X1, 'Y': Y}
            reg = smf.ols(formula='Y~1+X', data=used1).fit(cov_type='HAC',
cov_kwds={'maxlags': 1})
            t3.append(reg.tvalues[0])
            alpha3.append(reg.params[0] * 12)
            used2 = {'X': X2, 'Y': Y}
            reg = smf.ols(formula='Y~1+X', data=used2).fit(cov_type='HAC',
cov_kwds={'maxlags': 1})
            t5.append(reg.tvalues[0])
            alpha5.append(reg.params[0] * 12)
        print('alpha-FF3', alpha3[0]/12)
        print('t-statistic', '('+str(round(t3[0],4))+')')
        print('alpha-FF5', alpha5[0]/12,)
        print('t-statistic','('+str(round(t5[0],4))+')')
        print('sharpe', sharpratio[0])
        gc.collect()
        print('*'*30)#分隔开不同收益序列


def showtrasecfee(transectionfee):
    '''
    :param transectionfee: 交易费用比例
    :return: 无返回直接打印交易费用调整后的结果
    '''
    trasecfee(transectionfee)# 方便调用
```

k) selectFactor.py

```python
# -*- coding:utf-8 -*-
'''
@description
用于筛选特征
'''
import pandas as pd
import numpy as np
from sklearn.preprocessing import scale
from sklearn.linear_model import LinearRegression
import gc


def dropimportant(length,CLF,name, factor, timeseries, base =  0.028813732000000005,inpath =
’output’):
    '''
    主函数 1, 用于非循环神经网络特征筛选
    :param length: 滑动窗口长度
    :param CLF: 模型
    :param name: 筛选用模型名称
    :param timeseries: 数据
    :param factor: 因子名称
    :param base: 多空对应月度收益
    :param inpath:输出路径
    :return:月度收益差
    '''
    meanlist = []
    for j in range(96):
        print(j)
        print(factor[j])
        Long_Short = []
        Long = []
        Short = []
        for i in range(len(timeseries) - (length+1)):

            FINALm = pd.concat(timeseries[i:i + (length+1)], axis=0)
            FINALm = FINALm.fillna(0)
            FINAL_X= FINALm.iloc[:, :-2].copy()
            FINAL_X.drop(columns=factor[j],inplace=True)
            FINAL_x = scale(FINAL_X)
            final = pd.concat(timeseries[i:i + length], axis=0)
            x_train = FINAL_x[:len(final)]
            x_test = FINAL_x[len(final):]
            y_train = final.iloc[:, -1]
            test = timeseries[i + length]
            y_test = test.iloc[:, -1]
            # 基准-linear
            clf = CLF
            clf.fit(x_train, y_train)
            PREDICTION = clf.predict(x_test)
            # 构建投资组合
            prediction = pd.DataFrame(PREDICTION)
            r_predict = pd.DataFrame(PREDICTION, columns=['predict'])
            r_ture = pd.DataFrame(y_test)
            r_ture.columns = ['ture']
            r_ture.index = r_predict.index
            FINAL = pd.concat([r_predict, r_ture], axis=1)
            FINAL_sort = FINAL.sort_values(by='predict', axis=0)
            r_final = np.array(FINAL_sort['ture'])
            m = int(len(r_final) * 0.1) + 1
            r_final = r_final.tolist()
            long = r_final[-m:]
            short = r_final[:m]
            r_end = (np.sum(long) - np.sum(short)) / m
            Long_Short.append(r_end)
            Long.append(np.average(long))
            Short.append(np.average(short))
            gc.collect()
        A = pd.DataFrame(Long_Short, columns=['long-short'])
        B = pd.DataFrame(Long, columns=['long'])
        C = pd.DataFrame(Short, columns=['short'])
        M = pd.concat([A, B, C], axis=1)
        meanlist.append(np.average(Long_Short))
        if (i%10) == 0:
            print(i)
    gc.collect()
    u = base - np.array(meanlist)
    u = pd.DataFrame(u)
    u.to_csv('../' + inpath + '/'+name+'returngap.csv')
    return u

def dropimportant2(length,CLF,name, factor, timeseries,base = 0.028813732000000005,inpath =
'output',a=0,b=96):
    '''
    主函数 2 用于循环神经网络特征筛选
    :param length: 滑动窗口大小
    :param CLF: 筛选用模型
    :param name: 模型名称
    :param factor: 因子名称
    :param timeseries: 数据
    :param base: 月度收益
    :param inpath: 输出位置
    :param a: 起始因子
    :param b: 终止因子
    :return: 返回收益差
    ’’’
    meanlist = []
    for j in range(a,b):
        print(j)
        print(factor[j])
        Long_Short = []
        Long = []
        Short = []
        for i in range(len(timeseries) - length):
            FINALm = pd.concat(timeseries[i:(i + length + 1)], axis=0)
            FINALm[~FINALm[’ret’].isin([’null’])] =
FINALm[~FINALm[’ret’].isin([’null’])].fillna(0)
            FINAL_X = FINALm.iloc[:, :-2]
            FINAL_X.drop(columns=factor[j], inplace=True)
            FINAL_x = FINAL_X
            FINAL_x[~FINALm[’ret’].isin([’null’])] =
scale(FINAL_X[~FINALm[’ret’].isin([’null’])])
            FINAL_x[FINALm[’ret’].isin([’null’])] = 0
            FINALm[FINALm[’ret’].isin([’null’])] = 0
            x_train = [FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in
range(length)]
            y_train = [FINALm.iloc[j * 3571:(j + 1) * 3571, -1].values for j in
range(length)]
            x_test = [FINAL_x.iloc[j * 3571:(j + 1) * 3571, :].values for j in range(1,
length + 1)]
            y_test = list(FINALm.iloc[(length) * 3571:(length + 1) * 3571, -1].values)
            # 基准-linear
            clf = CLF
            clf.fit(x_train, y_train)
            PREDICTION = clf.predict(x_test)
            PREDICTION = [PREDICTION[m][0] for m in range(3571 * (length - 1), 3571 *
length)]
            # 构建投资组合
            r_predict = pd.DataFrame(PREDICTION, columns=[’predict’])
            r_ture = pd.DataFrame(y_test)
            r_ture.columns = [’ture’]
            r_ture.index = r_predict.index
            FINAL = pd.concat([r_predict, r_ture], axis=1)
            FINAL = FINAL[~timeseries[i + length]['ret'].isin(['null'])]
            FINAL_sort = FINAL.sort_values(by='predict', axis=0)
            r_final = np.array(FINAL_sort['ture'])
            m = int(len(r_final) * 0.1) + 1
            r_final = r_final.tolist()
            long = r_final[-m:]
            short = r_final[:m]
            r_end = (np.sum(long) - np.sum(short)) / m
            Long_Short.append(r_end)
            Long.append(np.average(long))
            Short.append(np.average(short))
            gc.collect()
        A = pd.DataFrame(Long_Short, columns=['long-short'])
        B = pd.DataFrame(Long, columns=['long'])
        C = pd.DataFrame(Short, columns=['short'])
        M = pd.concat([A, B, C], axis=1)
        meanlist.append(np.average(Long_Short))
        if (i%10) == 0:
            print(i)
    gc.collect()
    u = base - np.array(meanlist)
    u = pd.DataFrame(u)
    u.to_csv('../' + inpath + '/' + name + 'returngap.csv')
    return u

def FCselect(factor, timeseries):
    '''
    用于FC筛选因子
    :param factor:因子名称
    :param timeseries:数据
    :return:
    '''
    length = 12
    base = 0.0228
    name = 'FC'
    inpath = 'output'
    meanlist = []
    for j in range(96):
        print(j)
        print(factor[j])
        Long_Short = []
        Long = []
        Short = []
        for i in range(len(timeseries) - (length + 1)):
            FINALm = pd.concat(timeseries[i:i + (length + 1)], axis=0)
            FINALm = FINALm.fillna(0)
            FINAL_X = FINALm.iloc[:, :-2].copy()
            FINAL_X.drop(columns=factor[j], inplace=True)
            FINAL_x = scale(FINAL_X)
            final = pd.concat(timeseries[i:i + length], axis=0)
            x_train = FINAL_x[:len(final)]
            x_test = FINAL_x[len(final):]
            y_train = final.iloc[:, -1]
            test = timeseries[i + length]
            y_test = test.iloc[:, -1]
            clf = LinearRegression()
            k = []
            for ax in range(95):
                x = x_train[:, ax].reshape(-1, 1)
                clf.fit(x, y_train)
                k.append(clf.coef_[0])
            PREDICTION = []
            for ap in range(len(x_test)):
                test0 = np.array(x_test[ap])
                y = 0
                for ass in range(95):
                    y = y + test0[ass] * k[ass]
                PREDICTION.append(y)
            y_test = test.iloc[:, -1]
            prediction = pd.DataFrame(PREDICTION)
            r_predict = pd.DataFrame(PREDICTION, columns=[’predict’])
            r_ture = pd.DataFrame(y_test)
            r_ture.columns = [’ture’]
            r_ture.index = r_predict.index
            FINAL = pd.concat([r_predict, r_ture], axis=1)
            FINAL_sort = FINAL.sort_values(by=’predict’, axis=0)
            r_final = np.array(FINAL_sort[’ture’])
            m = int(len(r_final) * 0.1) + 1
            r_final = r_final.tolist()
            long = r_final[-m:]
            short = r_final[:m]
            r_end = (np.sum(long) - np.sum(short)) / m
            Long_Short.append(r_end)
            Long.append(np.average(long))
            Short.append(np.average(short))
            gc.collect()
        meanlist.append(np.average(Long_Short))
        if (i % 10) == 0:
            print(i)
    gc.collect()

    u = base - np.array(meanlist)
    u = pd.DataFrame(u)
    u.to_csv(’../’ + inpath + ’/’ + name +’returngap.csv’
```