因子有效性检验----------以市值因子为为例

一、数据筛选

1.1 筛选期间：2015年1月1日至2023年10月30日

1.2 筛选范围：全部A

1.3 筛选规则：剔除选股日的ST/PT，暂停上市、退市等股票；剔除上市不满6个月的股票；剔除选股日由于停牌等原因而无法买入的股

1.4 剔除北交所的股票（存在短暂的无行业分类情况）

1.5 数据已经处理好，直接读取即可票。

In [3]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd  #Version: 2.0.0
import numpy as np  #Version: 1.26.2
import seaborn as sns
from datetime import *
from  math  import *
import statsmodels.api as sm
from collections import OrderedDict #保持Key的顺序
import matplotlib.pyplot as plt
import numpy.linalg as la   #用来做线性代数运算
from scipy import stats, optimize #Version: 1.9.1
import statsmodels.api as sm #Version: 0.13.5


data=pd.read_excel("C:\\Users\\WSD跑出来数据.xlsx",index_col=(0,1)) #获取数

data.index.names=['date','codes']  #设置索引名
data.head()
```

Out[3]:

		industry_citic	CAP	ln_CAP	next_ret
date	**codes**				
20150130	**002414.SZ**	国防军工	994398.3049	13.809893	11.0229
	600302.SH	机械	233556.6177	12.361180	18.8148
	300290.SZ	计算机	150846.1772	11.924016	3.0978
	600168.SH	电力及公用事业	727598.9167	13.497505	14.7772
	002389.SZ	电子	330746.1640	12.709106	8.4559

二、异常值处理

1、异常值判断方法常见的主要有均值标准差法、绝对中位数法MAD和boxplot箱线图法，后两者相对最前者受异常值的影响较小，其处理效果更稳健。

2、一般绝对值中位数法MAD在实际处理中用的较多，进行偏度调整后的boxplot法减弱了分布偏度的影响，在异常处理时也是个不错的选择。

3 在对异常值进行处理时，需根据因子的具体情况来决定是直接删除异常值还是将异常值设置为上下限的数值，一般选用后者处理方式。

方法1：均值标准差法

根据3σ 准则，认为在均值正负三倍标准差之外的值为异常值。但值得注意的是，样本均值和样本标准差并非稳健统计量，其计算本身易受极值影响，这就会导致在分布图上明显异常的点，其值可能仍在均值正负三倍标准差之内。而且很多因子的分布存在厚尾现象，并不满足正态分布假设。

In [4]:
```
def get_feature_names(data):  #该函数用于获取数据集中需测试的因子名
    columns = data.columns.tolist()
    fea_names = [i for i in columns if i not in ["industry_citic",'CAP',"
    return fea_names
```

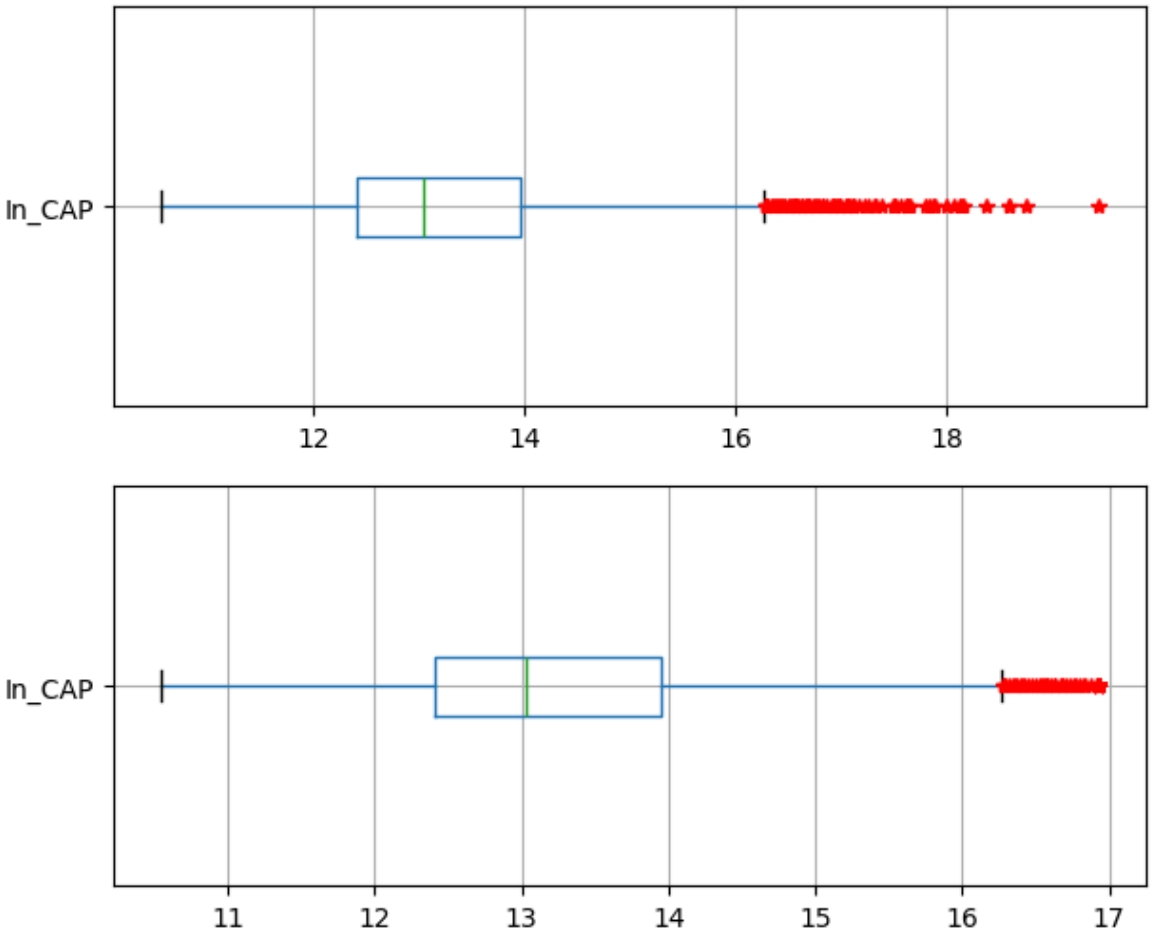
In [5]:
```
def extreme_process_3sigma(data,num=3):#data为输入的数据集，如果数值超过num个
    data_=data.copy()  # 为不破坏原始数据，先对其进行拷贝
    feature_names = get_feature_names(data_)    #获取数据集中需处理的因子名
    mean=data_[feature_names].mean(axis=0)  #获取均值
    std=data_[feature_names].std(axis=0)  #获取标准差
    data_.loc[:,feature_names]=data_.loc[:,feature_names].clip(lower=mean
    return data_
```

In [6]:
```
data1 = data.groupby(level=0).apply(extreme_process_3sigma)  #对数据分组进
```

通过画箱线图可视化异常处理结果：发现与原始数据相比，数据极端值有所减少。

In [7]:
```
sns.set_style="whitegrid"
fig = plt.figure(figsize=(7,6))
plt.subplot(2,1,1)
data.loc[20210531][['In_CAP']].boxplot(sym='r*',vert=False)  #绘制最后一个
plt.subplot(2,1,2)
data1.loc[20210531][['In_CAP']].boxplot(sym='r*',vert=False)  #绘制最后一个
```

Out[7]:
```
<Axes: >
```



方法2：绝对中位数法MAD

绝对中位数法是一种更为稳健的异常值处理方法，是对均值标准差法的改进，用不受极端值影响的样本中位数和绝对中位值MAD来划分异常值的范围，其计算公式如下所示

$$MAD = median(|f_i - Mdeian_f|), Mdeian_f$$为因子值的

将大于$Mdeian_f + 3 * 1.4826 * MAD$的值或小于$Mdeian_f - 3 * 1.4826 * MAD$的值定义为异常值

中位数绝对偏差会乘上一个常数比例因子（通常使用1.4826），使得在正态分布下，中位数绝对偏差与标准偏差具有相同的尺度。。

In [8]:
```
def extreme_process_MAD(data,num=3):#data为输入的数据集，如果数值超过num个判断
    data_=data.copy()  # 为不破坏原始数据，先对其进行拷贝
    feature_names = get_feature_names(data_)   #获取数据集中需测试的因子名
    median=data_[feature_names].median(axis=0)  #获取中位数
    MAD=abs(data_[feature_names].sub(median,axis=1)).median(axis=0) #按列
    data_.loc[:,feature_names]=data_.loc[:,feature_names].clip(lower=medi
    return data_
```

In [9]:
```
data2 = data.groupby(level=0).apply(extreme_process_MAD)  #对数据分组进行异
data2.index=data2.index.droplevel(level=0)
```

In [10]:
```
data2
```

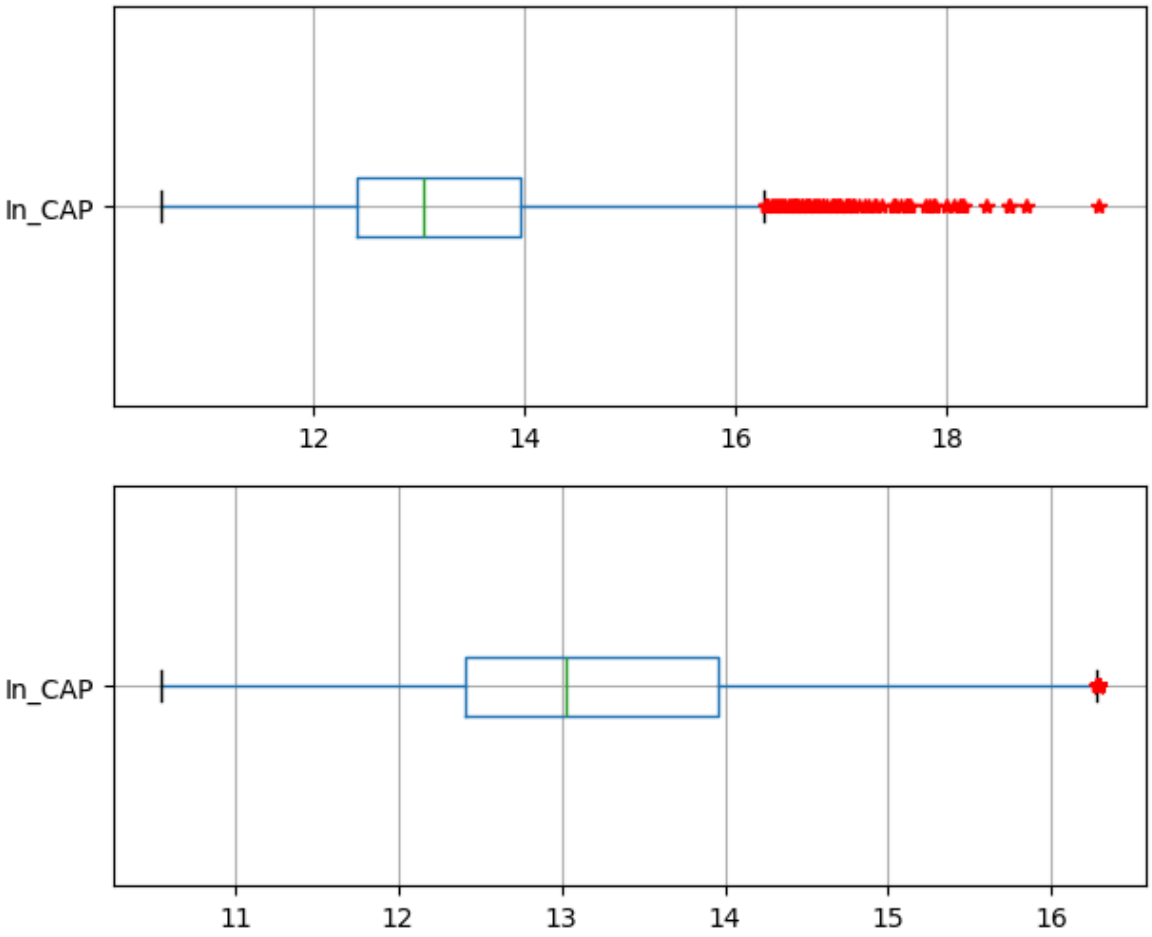
Out[10]:

date	codes	industry_citic	CAP	ln_CAP	next_ret
20150130	002414.SZ	国防军工	9.943983e+05	13.809893	11.0229
	600302.SH	机械	2.335566e+05	12.361180	18.8148
	300290.SZ	计算机	1.508462e+05	11.924016	3.0978
	600168.SH	电力及公用事业	7.275989e+05	13.497505	14.7772
	002389.SZ	电子	3.307462e+05	12.709106	8.4559
...	...	...	...	...	...
20230928	603299.SH	基础化工	7.146703e+05	13.479577	-5.1724
	603508.SH	电子	6.115641e+05	13.323775	-2.7431
	002531.SZ	电力设备及新能源	2.306586e+06	14.651279	2.0139
	002753.SZ	基础化工	1.731125e+05	12.061697	8.2749
	002774.SZ	机械	2.324711e+05	12.356521	-2.3030

348212 rows × 4 columns

In [11]:
```
fig = plt.figure(figsize=(7,6))
plt.subplot(2,1,1)
data.loc[20210531][['In_CAP']].boxplot(sym='r*',vert=False)  #绘制最后一个月
plt.subplot(2,1,2)
data2.loc[20210531][['In_CAP']].boxplot(sym='r*',vert=False)  #绘制最后一个
```

Out[11]:
```
<Axes: >
```



方法3：boxplot法

boxplot法是由Turkey(1977年)提出的一种经验处理方法。假设Q1和Q3分别为数据从小到大排列的25%和75%分位数，记IQR=Q1–Q3, 把$(-\infty, Q1-3*IQR)\bigcup(Q3+3*IQR,+\infty)$区间里的数据标识为异常点。由于分位数是稳健统计量，Boxplot 方法对极值不敏感。但值得注意的是，若样本数据严重正偏，且右尾分布明显偏厚时，Boxplot 方法会把过多的数据划分为异常数据，因此Hubert& Vandervieren （2007）对原有Boxplot 方法进行了偏度调整。首先样本偏度定义采用了Brys(2004)提出的MedCouple方法：

$$md = median(x_i, i = 1, 2, 3, \ldots, n)$$

$$mc = median(\frac{(x_i - md) - (md - x_j)}{x_i - x_j}, x_i >= md, x_j <= md)$$

然后再给出经过偏度调整Boxplot方法的上下限：

$$L = \begin{cases} Q1 - 1.5 * exp(-3.5 * mc) * IQR, mc \geqslant 0 \\ Q1 - 1.5 * exp(-4 * mc) * IQR, mc < 0 \end{cases}$$

$$U = \begin{cases} Q3 + 1.5 * exp(4 * mc) * IQR, mc \geqslant 0 \\ Q3 + 1.5 * exp(3.5 * mc) * IQR, mc < 0 \end{cases}$$

区间$(-\infty, L) \bigcup (U, +\infty)$中的数据被定义为异常数据。

和原始Boxplot方法相比，当样本数据分布右偏时，此方法会提升正常数据区间上限的数值；样本数据左偏时，则会降低正常数据区间下限的数值。

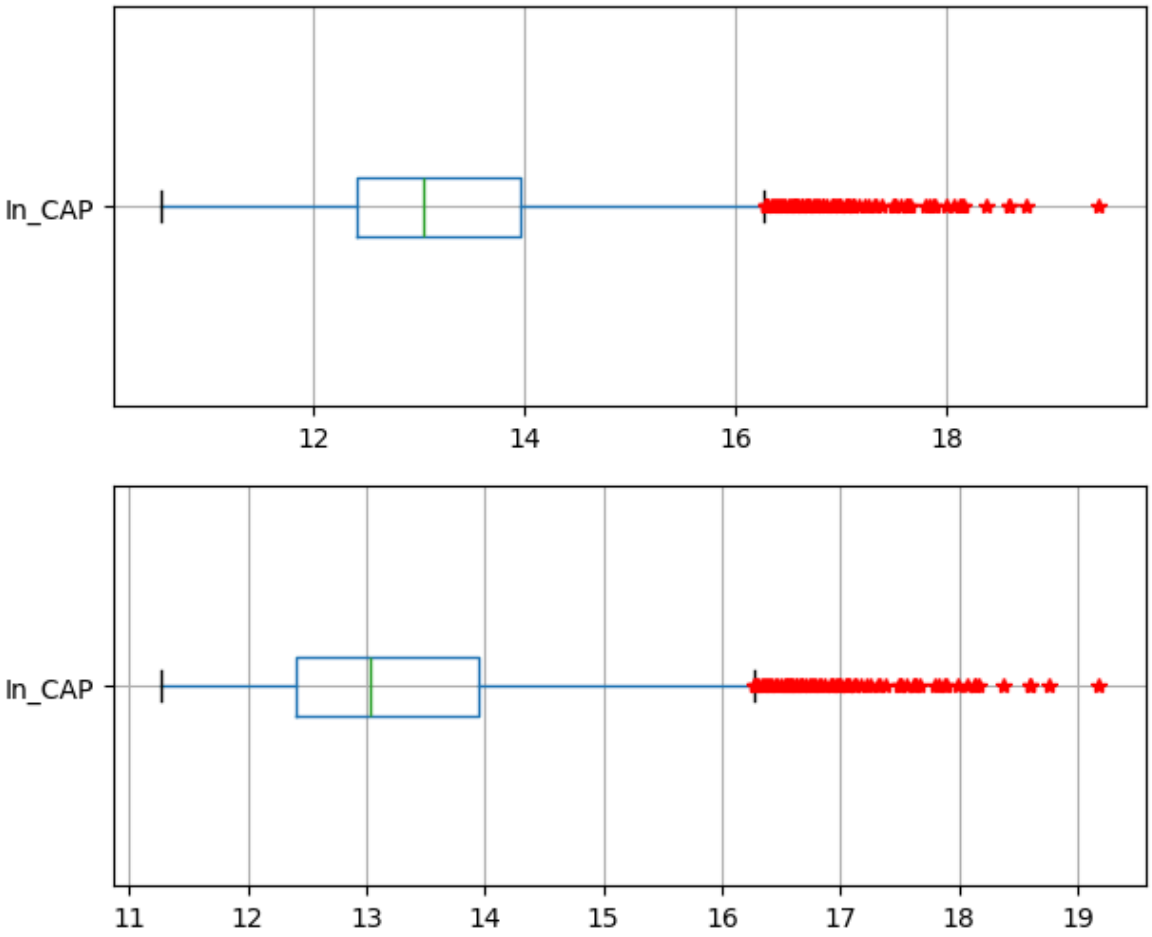
（注：mc值可用statsmodels包中的函数sm.stats.stattools.medcouple(x)直接计算。）

In [12]:
```
def extreme_process_adjboxplot(data):
    data_=data.copy()  # 为不破坏原始数据，先对其进行拷贝
    feature_names = get_feature_names(data_)    #获取数据集中需测试的因子名
    for name in feature_names:
        factor=data_[name]
        mc=sm.stats.stattools.medcouple(factor)
        factor=factor.sort_values()
        Q1=factor.iloc[int(0.25*len(factor))]  #计算下分位数
        Q3=factor.iloc[int(0.75*len(factor))]  #计算上分为数
        IQR=Q3-Q1
        #计算上下限
        if mc>=0:
            L= Q1-1.5*np.exp(-3.5*mc)*IQR
            U = Q3+1.5*np.exp(4*mc)*IQR
        else:
            L = Q1-1.5*np.exp(-4*mc)*IQR
            U = Q3+1.5*np.exp(3.5*mc)*IQR

        data_[name]=factor.clip(lower=L,upper=U)  #利用clip()函数，将因子取值

    return data_
```

In [13]:
```
data3 = data.groupby(level=0).apply(extreme_process_adjboxplot)
```

In [14]:
```
fig = plt.figure(figsize=(7,6))
plt.subplot(2,1,1)
data.loc[20210531][['In_CAP']].boxplot(sym='r*',vert=False)  #绘制最后一个月
plt.subplot(2,1,2)
data3.loc[20210531][['In_CAP']].boxplot(sym='r*',vert=False)  #绘制最后一个
```

Out[14]: <Axes: >



从箱线图可以看出，三种异常值处理方法中绝对中位数法的效果最好。

用绝对中位数法做后续的计算

三、缺失值处理

缺失值处理方式包括直接剔除和数值插补。当缺失数据较少时，其信息遗漏较少，可直接将其删除；当缺失数据较多时，直接删除会造成信息丢失或遗漏，需选择合适的值进行缺失值插补。可用均值、行业均值、行业中位数等填充，也可运用回归、K近邻等算法求得缺失值的估计值进行填充。

In [15]:
```
#计算缺失值个数
def null_count(data):
    data_=data.copy()  # 为不破坏原始数据，先对其进行拷贝
    feature_names = get_feature_names(data_)    #获取数据集中需测试的因子名
    null_count={}
    for name in feature_names:
        null_count[name]=len(data_[data_[name].isnull()])
        #null_count.append(len(data_[data_[name].isnull()]))
    print(null_count)
    return null_count
```

In [16]:
```
data2.groupby(level=0).apply(null_count)
```

{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}

```
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
{'In_CAP': 0}
```

Out[16]:

```
date
20150130    {'In_CAP': 0}
20150227    {'In_CAP': 0}
20150331    {'In_CAP': 0}
20150430    {'In_CAP': 0}
20150529    {'In_CAP': 0}
                ...
20230531    {'In_CAP': 0}
20230630    {'In_CAP': 0}
20230731    {'In_CAP': 0}
20230831    {'In_CAP': 0}
20230928    {'In_CAP': 0}
Length: 105, dtype: object
```

In [17]:
```
def dropna(data):
    data=data.dropna()
    return data
print(len(data2))
data2 = data2.dropna()
print(len(data2))
#我们此次数据无缺失值
```

```
348212
348212
```

调仓日缺失值值的处理方式

In [18]:
```
#若某一调仓日的因子值有缺失，则使用上一个调仓日的因子值进行替代，若仍有缺失，则使用这一
def unll_process(datas):
    datas_=datas.copy()
    ##step1:先用上一个调仓日的因子值进行缺失值填充
    def null_process1(data):

        if  True in data.isnull().any().values:#判断数据是否存在缺失值
            data.fillna(method='ffill',limit=1,inplace=True)  #先用上一个调
        return data

    datas1=datas_.swaplevel(0,1)  #先将数据的索引互换，令股票为一级索引，时间为二
    datas1=datas1.sort_index(axis=0)
    datas2=datas1.groupby(level=0).apply(null_process1)  #对重新索引后的数据

    ##step2:若仍存在缺失值，则用该天的股票池因子的平均值代替
    def null_process2(data):
        if  True in data.isnull().any().values: #判断数据是否存在缺失值
            col=[i for i in data.columns.tolist() if i not in ['INDUSTRY_
            mean=data[col].mean(axis=0)  #按列求均值
            dict_=OrderedDict((key,value) for key,value in zip(mean.index
            data.fillna(dict_,inplace=True)  #先用上一个调仓日的因子值进行缺失
        return data

    datas2=datas2.swaplevel(0,1)  # 进行第一步缺失值处理后，将索引次序换回，为第
    datas2=datas2.sort_index(axis=0)
    datas3=datas2.groupby(level=0).apply(null_process2) #对剩余的缺失值进行均

    ##step3:直接删除行业缺失值
    datas4=datas3.dropna()  #对剩余的行业缺失值直接删除
    return datas4
```

四、标准化处理

对数据进行标准化处理，能消除量纲的影响，使数据在一个固定范围内取值。标准化处理的方法主要采用：Z 值标准化（Z-Score）。

In [19]:
```
def data_scale_Z_Score(data):
    data_=data.copy()  # 为不破坏原始数据，先对其进行拷贝
    feature_names = get_feature_names(data)
    data_.loc[:,feature_names] = (data_.loc[:,feature_names] - data_.loc[
    return data_
```

In [20]:
```
data2_std=data2.groupby(level=0).apply(data_scale_Z_Score)
data2_std.index=data2_std.index.droplevel(level=0)
data2_std
```

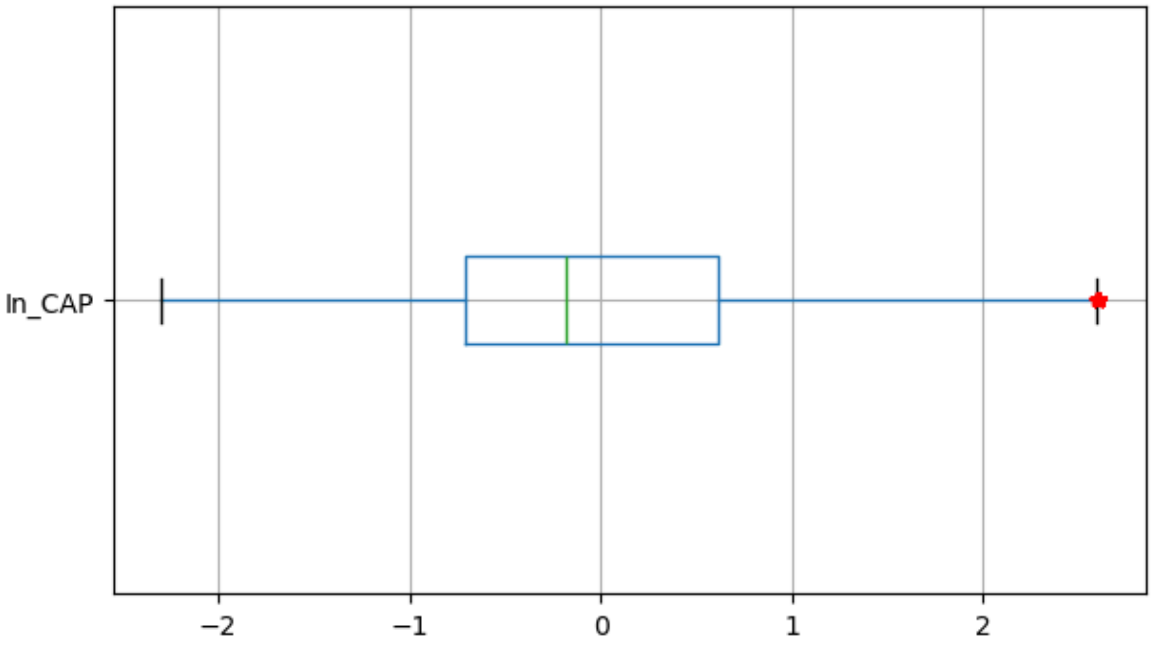
Out[20]:

date	codes	industry_citic	CAP	ln_CAP	next_ret
20150130	002414.SZ	国防军工	9.943983e+05	0.629491	11.0229
	600302.SH	机械	2.335566e+05	-0.872728	18.8148
	300290.SZ	计算机	1.508462e+05	-1.326038	3.0978
	600168.SH	电力及公用事业	7.275989e+05	0.305566	14.7772
	002389.SZ	电子	3.307462e+05	-0.511951	8.4559
...	...	...	...	...	...
20230928	603299.SH	基础化工	7.146703e+05	0.269787	-5.1724
	603508.SH	电子	6.115641e+05	0.128339	-2.7431
	002531.SZ	电力设备及新能源	2.306586e+06	1.333545	2.0139
	002753.SZ	基础化工	1.731125e+05	-1.017468	8.2749
	002774.SZ	机械	2.324711e+05	-0.749805	-2.3030

348212 rows × 4 columns

In [21]:
```
fig = plt.figure(figsize=(7,4))
data2_std.loc[20210531][['ln_CAP']].boxplot(sym='r*',vert=False)
#这里没有其他的变量，如果有的话，会发现不同因子之间取值范围会明显的缩小，量纲去除
```

Out[21]: <Axes: >



五、市值中性化处理

市值中性化处理与标准化处理类似，只需将Z值标准化（Z-Score）处理过程中的均值用通过以个股在A股市值中所占比重为权重进行加权平均值代替即可。

市值因子本身不需要市值中性化

也可以通过回归把行业市值一起去中性化

In [22]:
```
#市值中性化处理
def data_scale_CAP(data):
    feature_names = get_feature_names(data)
    data_=data.copy()
    cap_weight = data_["CAP"]/ data_["CAP"].sum()
    for name in feature_names:
        if name!='In_CAP'  :  #对于非规模因子
            avg=(data_[name]*cap_weight).sum()
            data_[name]=(data_[name]-avg)/data_[name].std()
        else:
            data_[name] = data_[name]
    return data_


#可以细分是否是市值因子，市值因子不用过滤市值中性
#if feature_names!='ln_CAP11':  #对于非规模因子
#    data_scale_neutral
#else:  #对于规模因子只剔除行业因素
#    data_scale_neutral
```

In [23]:
```
data2_cap=data2_std.groupby(level=0).apply(data_scale_CAP)
data2_cap.index=data2_cap.index.droplevel(level=0)
```

In [24]:
```
data2_cap
#发现市值中性化通过添加条件，并不会对市值因子去除，但是会对其他因子市值中性化
```

Out[24]:

date	codes	industry_citic	CAP	ln_CAP	next_ret
20150130	002414.SZ	国防军工	9.943983e+05	0.629491	11.0229
	600302.SH	机械	2.335566e+05	-0.872728	18.8148
	300290.SZ	计算机	1.508462e+05	-1.326038	3.0978
	600168.SH	电力及公用事业	7.275989e+05	0.305566	14.7772
	002389.SZ	电子	3.307462e+05	-0.511951	8.4559
...	...	...	...	...	...
20230928	603299.SH	基础化工	7.146703e+05	0.269787	-5.1724
	603508.SH	电子	6.115641e+05	0.128339	-2.7431
	002531.SZ	电力设备及新能源	2.306586e+06	1.333545	2.0139
	002753.SZ	基础化工	1.731125e+05	-1.017468	8.2749
	002774.SZ	机械	2.324711e+05	-0.749805	-2.3030

348212 rows × 4 columns

六、行业中性化处理

行业中性处理一般采用行业虚拟变量回归的方法,以因子值为因变量，一级行业虚拟变量为自变量，进行回归，并取回归的残差作为因子值。

此处行业划分采用中信一级行业进行划分。

In [25]:
```
#行业中性化处理
def data_scale_neutral(data):
    feature_names = get_feature_names(data)
    data_=data.copy()
    data_.dropna(inplace=True)
    industrys=data_['industry_citic']  #获取所属中信一级行业代码
    data_med = pd.get_dummies(data_,columns=['industry_citic'],drop_first
    n = len(industrys.unique())    #确定产生虚拟变量个数
    X = np.array(data_med.iloc[:,-n:])  #行业虚拟变量作为为自变量
    for name in feature_names:
        y = np.array(data_[name])
        if la.matrix_rank(X.T.dot(X)) == n: #当矩阵满秩时，估计回归参数
            beta_ols = la.inv(X.T.dot(X)).dot(X.T).dot(y)
            residual = y - X.dot(beta_ols)      #计算残差，并将其作为剔除行业
        else:
            residual = y   #如果逆不存在的话 则 用原值
        data_[name]=residual
    return data_
```

In [26]:
```
data2_neu=data2_cap.groupby(level=0).apply(data_scale_neutral)
data2_neu.index=data2_neu.index.droplevel(level=0)
data2_neu.head()
```

Out[26]:

date	codes	industry_citic	CAP	ln_CAP	next_ret
20150130	002414.SZ	国防军工	994398.3049	-0.031229	11.0229
	600302.SH	机械	233556.6177	-0.440836	18.8148
	300290.SZ	计算机	150846.1772	-1.225197	3.0978
	600168.SH	电力及公用事业	727598.9167	0.021039	14.7772
	002389.SZ	电子	330746.1640	-0.262239	8.4559

注：在实际数据处理过程中，也可以将行业虚拟变量和市值一起与因子值进行回归，即可同时完成行业中性化和市值中性化的处理。

适用于对非市值因子进行中性化处理时候使用

In [27]:
```
#行业中性化处理,但是在这里不适合规模因子，因为市值因子本身不用市值中性化处理，可以加代码
def data_scale_neutral1(data):
    feature_names = get_feature_names(data)
    data_=data.copy()
    data_.dropna(inplace=True)
    industrys=data_['industry_citic']  #获取所属中信一级行业代码
    data_med = pd.get_dummies(industrys,columns=['industry_citic'],drop_f
    data_med['cap']=(data_['CAP']- data_['CAP'].mean())/data_['CAP'].std(
    n = len(data_med.columns)   # 确定产生虚拟变量个数，再加上1个市值指标，即为总
    X = np.array(data_med)  #行业虚拟变量作为为自变量
    for name in feature_names:
        #print(name)
        if name!='In_CAP':  #对于非规模因子
            y = np.array(data_[name])
            if la.matrix_rank((X.T.dot(X))) == n: #当矩阵满秩时，估计回归参数
                beta_ols = la.inv(X.T.dot(X)).dot(X.T).dot(y)
                residual = y - X.dot(beta_ols)      #计算残差，并将其作为剔除
            else:
                residual = y   #如果逆不存在的话 则 用原值
            data_[name]=residual
        else:
            #print(11111111111111)
            data_=data_.groupby(level=0).apply(data_scale_neutral)
    return data_
```

In [28]:
```
#两种方法下计算市值因子的中性化得到的结果是一致的。
#如果有其他的因子，可能会有一些误差，回归的自变量参数不同。差异不大
data2_std.groupby(level=0).apply(data_scale_neutral1)
```

Out[28]:

date	date	date	codes	industry_citic	CAP	In
20150130	20150130	20150130	002414.SZ	国防军工	9.943983e+05	-0.0
			600302.SH	机械	2.335566e+05	-0.44
			300290.SZ	计算机	1.508462e+05	-1.2
			600168.SH	电力及公用事业	7.275989e+05	0.0
			002389.SZ	电子	3.307462e+05	-0.26
...	...	...	...	...	...	
20230928	20230928	20230928	603299.SH	基础化工	7.146703e+05	0.4
			603508.SH	电子	6.115641e+05	0.1
			002531.SZ	电力设备及新能源	2.306586e+06	1.23
			002753.SZ	基础化工	1.731125e+05	-0.8
			002774.SZ	机械	2.324711e+05	-0.3

348212 rows × 4 columns

In [29]:
```
data2_neu.sort_index(ascending=True)
```

Out[29]:

date	codes	industry_citic	CAP	ln_CAP	next_ret
20150130	000001.SZ	银行	1.370254e+07	0.067143	0.4307
	000002.SZ	房地产	1.273556e+07	2.411838	-2.8201
	000004.SZ	医药	1.455239e+05	-1.575279	7.8963
	000006.SZ	房地产	8.702831e+05	0.373167	1.0769
	000007.SZ	有色金属	2.976058e+05	-0.916202	-8.8520
...	...	...	...	...	...
20230928	688799.SH	医药	2.317138e+05	-0.852383	17.5459
	688800.SH	电子	4.170830e+05	-0.212214	2.6067
	688819.SH	电力设备及新能源	4.561865e+05	-0.231828	-6.3989
	688981.SH	电子	1.009501e+07	2.611839	13.9785
	689009.SH	机械	1.785209e+06	1.529189	-5.2541

348212 rows × 4 columns

七、利用IC序列检验因子有效性因子稳定性。

检验指标如下：

IC大于0的比例:用于判断因子作用方向及作用的稳定性；

IC显著性P值小于0.05的比例：用于判断因子作用的显著情况及显著是否稳定；

IC绝对值大于0.03的比例：一般认为IC值大于0.03的因子是有效的，该指标就是用来判断因子有效性是否稳定；

IC序列均值、IC序列标准差：用于反映因子平均有效情况及有效是否稳定；用于计算ICIR；

ICIR值：用于反映因子稳定性。

In [30]:
```
def ic_test(test_data,method='rank'):     #IC用np.corrcoef 函数或者stats.pe
    feature_names=get_feature_names(test_data)
    dict_ic={}
    for name in feature_names:
        if method=='rank':
            ic_test=test_data.groupby(level=0).apply(lambda frame: stats.
        elif method=='normal':
            ic_test=test_data.groupby(level=0).apply(lambda frame: stats.
        ic=ic_test.map(lambda i:i[0])
        p_value=ic_test.map(lambda i:i[1])
        dict_ic[name]=pd.DataFrame({'ic':ic,"p_value": p_value})
    df_ic=pd.concat(dict_ic.values(),keys=dict_ic.keys())

    ic_mean = df_ic.groupby(level=0).apply(lambda frame: frame['ic'].mean
    ic_std =  df_ic.groupby(level=0).apply(lambda frame: frame['ic'].std(
    ic_ir = ic_mean/ic_std
    ic_prosive= df_ic.groupby(level=0).apply(lambda frame:  len([i for i
    ic_pvalue= df_ic.groupby(level=0).apply(lambda frame:  len([i for i i
    ic_abs= df_ic.groupby(level=0).apply(lambda frame:  len([i for i in n


    df_ic_stats = pd.DataFrame({
            'IC大于0的比例':ic_prosive,
            'IC显著性P值小于0.05的比例':ic_pvalue,
            'IC绝对值大于0.03的比例':ic_abs,
            'IC序列均值': ic_mean,
            'IC序列标准差': ic_std,
            'IC_IR值': ic_ir }).T

    return df_ic_stats, df_ic

ic_sum, df_ic=ic_test(data2_neu)
```

In [31]:
```
ic_sum
```

Out[31]:

	ln_CAP
IC大于0的比例	0.457143
IC显著性P值小于0.05的比例	0.857143
IC绝对值大于0.03的比例	0.866667
IC序列均值	-0.020634
IC序列标准差	0.164159
IC_IR值	-0.125692

In [32]:
```
df_ic
```

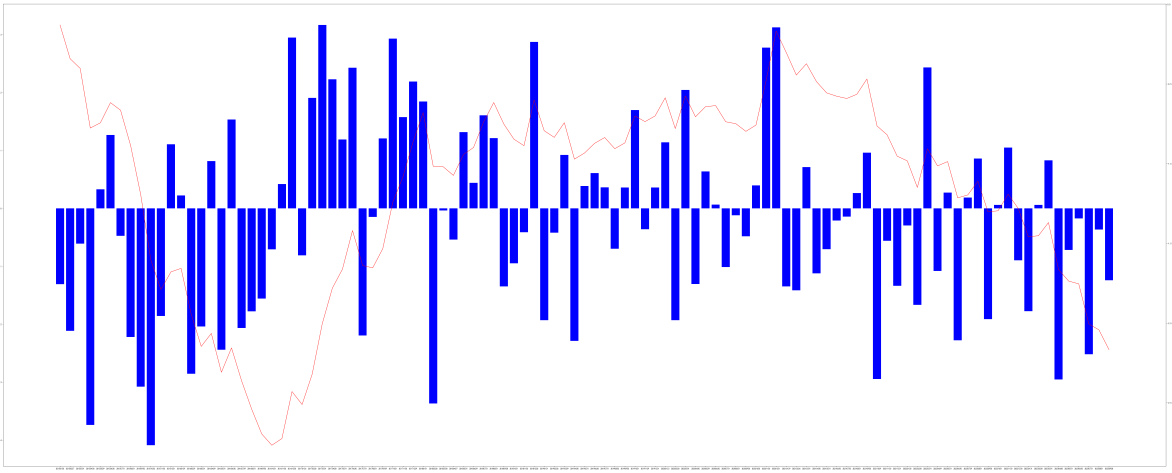
Out[32]:

		ic	p_value
	date		
ln_CAP	**20150130**	-0.130787	1.053123e-09
	20150227	-0.211359	1.292054e-22
	20150331	-0.060989	6.132881e-03
	20150430	-0.373826	2.176082e-65
	20150529	0.032786	1.547665e-01
	...	...	...
	20230531	-0.071816	1.343951e-06
	20230630	-0.017524	2.314352e-01
	20230731	-0.251695	2.591901e-69
	20230831	-0.036632	1.163575e-02
	20230928	-0.124123	7.907297e-18

105 rows × 2 columns

In [33]:
```
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.sans-serif'] = ['SimHei']
#想要某个因子的IC序列或者RankIC序列
IC_list = df_ic.loc["In_CAP"]#修改EP_TTM即可
#想要某个因子的累计IC序列或者累计RankIC序列
Sum_IC = IC_list.cumsum()

#画图，累计IC和IC的走势图
def IC_plot(feature_names,df_ic):#feature_names
    IC_list = df_ic.loc[feature_names,"ic"]
    Sum_IC = IC_list.cumsum()
    index = df_ic.loc[feature_names].index

    fig = plt.figure(figsize = (100,40))
    ax = plt.axes()
    xtick = np.arange(0,IC_list.shape[0])##
    xticklabel = index
    plt.bar(np.arange(IC_list.shape[0]),IC_list,color = "blue")
```

```
    ax1 = plt.twinx()
    ax1.plot(np.arange(IC_list.shape[0]),Sum_IC ,color = 'red')
    ax.set_xticks(xtick)
    ax.set_xticklabels(xticklabel)

IC_plot("In_CAP",df_ic)
```



In [34]:
```
Sum_IC
```

Out[34]:

date	ic	p_value
20150130	-0.130787	1.053123e-09
20150227	-0.342147	1.053123e-09
20150331	-0.403135	6.132882e-03
20150430	-0.776961	6.132882e-03
20150529	-0.744175	1.608994e-01
...	...	...
20230531	-1.736547	5.616950e+00
20230630	-1.754071	5.848385e+00
20230731	-2.005766	5.848385e+00
20230831	-2.042398	5.860021e+00
20230928	-2.166522	5.860021e+00

105 rows × 2 columns

In [35]:
```
IC_list
```

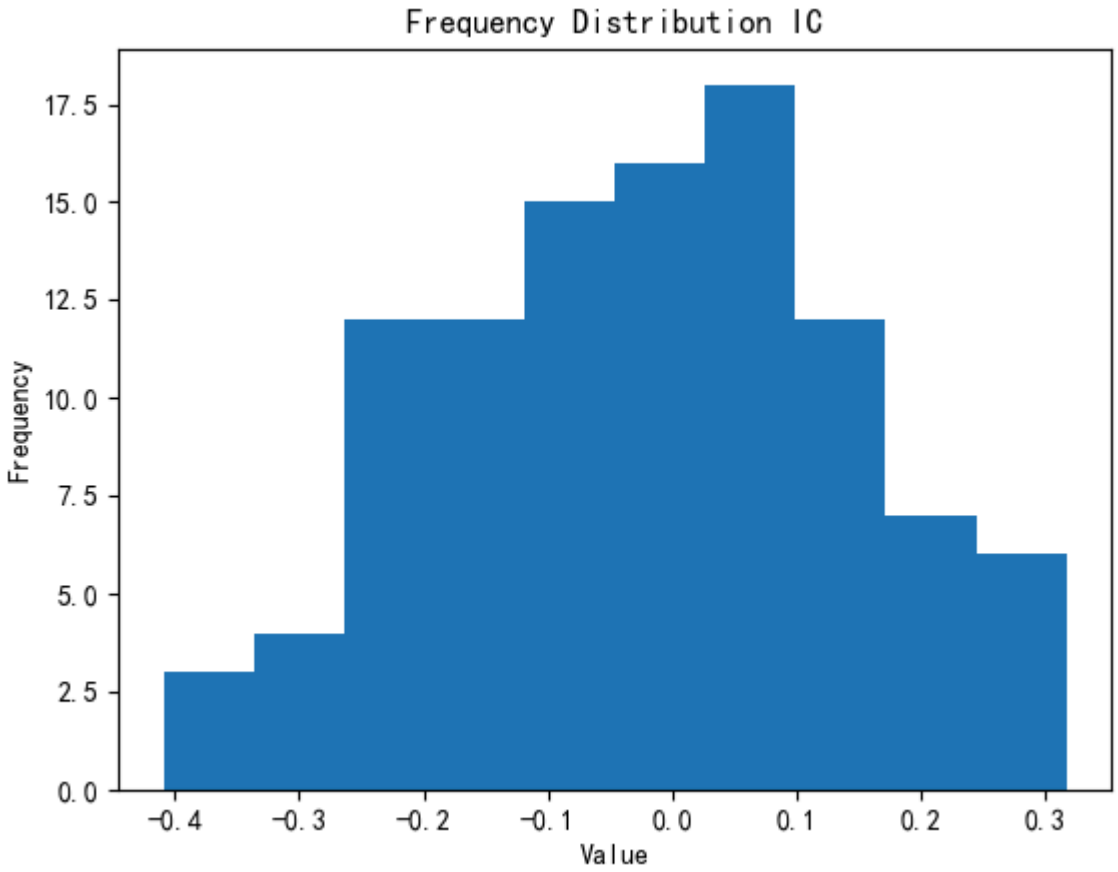
Out[35]:

date	ic	p_value
20150130	-0.130787	1.053123e-09
20150227	-0.211359	1.292054e-22
20150331	-0.060989	6.132881e-03
20150430	-0.373826	2.176082e-65
20150529	0.032786	1.547665e-01
...	...	...
20230531	-0.071816	1.343951e-06
20230630	-0.017524	2.314352e-01
20230731	-0.251695	2.591901e-69
20230831	-0.036632	1.163575e-02
20230928	-0.124123	7.907297e-18

105 rows × 2 columns

In [36]:

```
#分组画图，展示所有的图，曲线图
# 'IC大于0的比例':ic_prosive,
#              'IC显著性P值小于0.05的比例':ic_pvalue,
#              'IC绝对值大于0.03的比例':ic_abs,
#              'IC序列均值': ic_mean,
#              'IC序列标准差': ic_std,
#              'IC_IR值': ic_ir
def result_plot(result,result_number):
    fig,axes=plt.subplots(result_number,1)
    plt.subplots_adjust(wspace=0.5,hspace=0.5)
    plt.xticks(range(len(result.columns)),list(result.columns))
    for i in range(len(result)):
        result.iloc[i].plot(ax=axes[i],marker='v',title=result.iloc[i].na
    plt.subplots_adjust(wspace=0,hspace=0.2)
    plt.show()


# 绘制IC的的频率分布直方图
plt.hist(df_ic.loc["In_CAP","ic"], bins=10)
# 添加标题和标签
plt.title("Frequency Distribution IC")
plt.xlabel("Value")
plt.ylabel("Frequency")
# 显示图形
plt.show()
```



第八、利用回归法检验因子有效性

对于一般的OLS估计，当误差满足独立同分布与正态时，OLS 是有效无偏估计，但当误差服从非正态分布时，OLS 就很容易给异常值赋予较高的权重，从而导致模型结果失真

而股票收益率常常存在异常值，同时不同股票之间的收益率波动性也不尽相同，使用 OLS 时，异常值会对回归结果和 回归测试的显著性检验失真，所以常用加权最小二乘回归或稳健回归

性的影响。

1.加权最小二乘回归WLS: WLS是对OLS的一种改进，它以A股市值的平方根作为最下化残差平方和时求和权重来估计回归系数。

2.稳健回归RLM： RLM 中常用的M-estimator方法是采用迭代加权最小二乘估计回归系数，根据回归残差的大小确定各点的权重wi，以达到稳健的目的。为减少“异常值”作用，RLM 可以对不同的样本点赋予不同的权重，即对残差小的点给予较大的权重，而对残差较大的点给予较小的权重，根据残差大小确定权重，并据此建立加权的最小二乘估计，反复迭代以改进权重系数， 直至权重系数的改变小于一定的允许误差内。RLM 通过迭代的赋权回归来有效的减小异常值对参数估计结果有效性和稳定性的影响。

计算回归法检验的判断指标

因子收益均值：用于衡量因子收益的平均水平，且当平均值大于0时，该因子整体上会带来正收益；

因子收益大于0的比例：用于衡量因子带来的正负收益情况；

t值绝对值得均值：用于衡量因子平均水平上有效性的显著情况；

t值绝对值大于等于2的比例：用于判断因子的显著性是否稳定；

因子收益显著性P值小于0.05的比例：用于判断因子的显著情况和显著性稳定情况。

In [37]:
```
#用之前中性化之前的数据来做，在这里一步过程中，回归求因子收益时候，需要排除掉市值和行业
def regression_test(datas,method,dates):
    params_=OrderedDict()
    tvalues_=OrderedDict()
    pvalues_=OrderedDict()
    for date in dates:

        params=OrderedDict()
        tvalues=OrderedDict()
        pvalues=OrderedDict()
        data=datas.loc[[date]]

        feature_names=get_feature_names(data)

        for name in feature_names:  #对每个因子进行有效性检验feature_name中删
            industry=data['industry_citic']
            x0=np.array(list(industry))
            dummy=sm.categorical(x0,drop=True)  #得到中信一级行业虚拟变量
            x1=np.array(list(data[name]))

            #可以细分是否是市值因子，市值因子不用过滤市值中性
            if name!='In_CAP':  #对于非规模因子
                x=np.column_stack((x1, dummy,data['CAP']))   #合并回归所需自
                y=list(data['next_ret']) #将收益率作为因变量

                if method=='WLS':   #加权最小二乘回归
                    l = sm.WLS(y, x, weights=data['CAP'])
                    l_result = l.fit()
                    #print(l_result.params[0])

                elif method=='RLM' :  #稳健回归法
                    l= sm.RLM(y,x, M=sm.robust.norms.HuberT())
                    l_result=l.fit()

                params[name]=l_result.params[0]  #申万一级行业虚拟变量共27个,
                tvalues[name]=l_result.tvalues[0]  #回归系数t值
                pvalues[name]=l_result.pvalues[0]  #回归系数显著性检验P值

            else:  #对于规模因子只剔除行业因素
                x=np.column_stack((x1, dummy))   #合并回归所需自变量
                y=list(data['next_ret']) #将收益率作为因变量

                if method=='WLS':   #加权最小二乘回归
                    l = sm.WLS(y, x, weights=data['CAP'])
                    l_result = l.fit()
                    #print(l_result.params[0])

                elif method=='RLM' :  #稳健回归法
                    l= sm.RLM(y,x, M=sm.robust.norms.HuberT())
                    l_result=l.fit()
```

```
                params[name]=l_result.params[0]  #申万一级行业虚拟变量共27个,
                tvalues[name]=l_result.tvalues[0]  #回归系数t值
                pvalues[name]=l_result.pvalues[0]  #回归系数显著性检验P值

        params_[date]= params
        tvalues_[date]= tvalues
        pvalues_[date]=pvalues

    params_=pd.DataFrame(params_).T  #字典转为DataFrame，key做列名，值也为字典
    t_test= pd.DataFrame(tvalues_).T
    p_value=pd.DataFrame(pvalues_).T

    return params_, t_test,p_value



dates=sorted(list(set(pd.MultiIndex.to_frame(data2_std.index)['date'].val
WLS, t_test,p_value=regression_test(data2_std,'WLS',dates[:-1])  #data2_s
WLS.head(5)   #估计的回归系数，即求得的因子收益值
t_test.head(5)
```

Out[37]:

	In_CAP
20150130	-8.196429
20150227	-15.195761
20150331	4.297545
20150430	-23.473925
20150529	2.970583

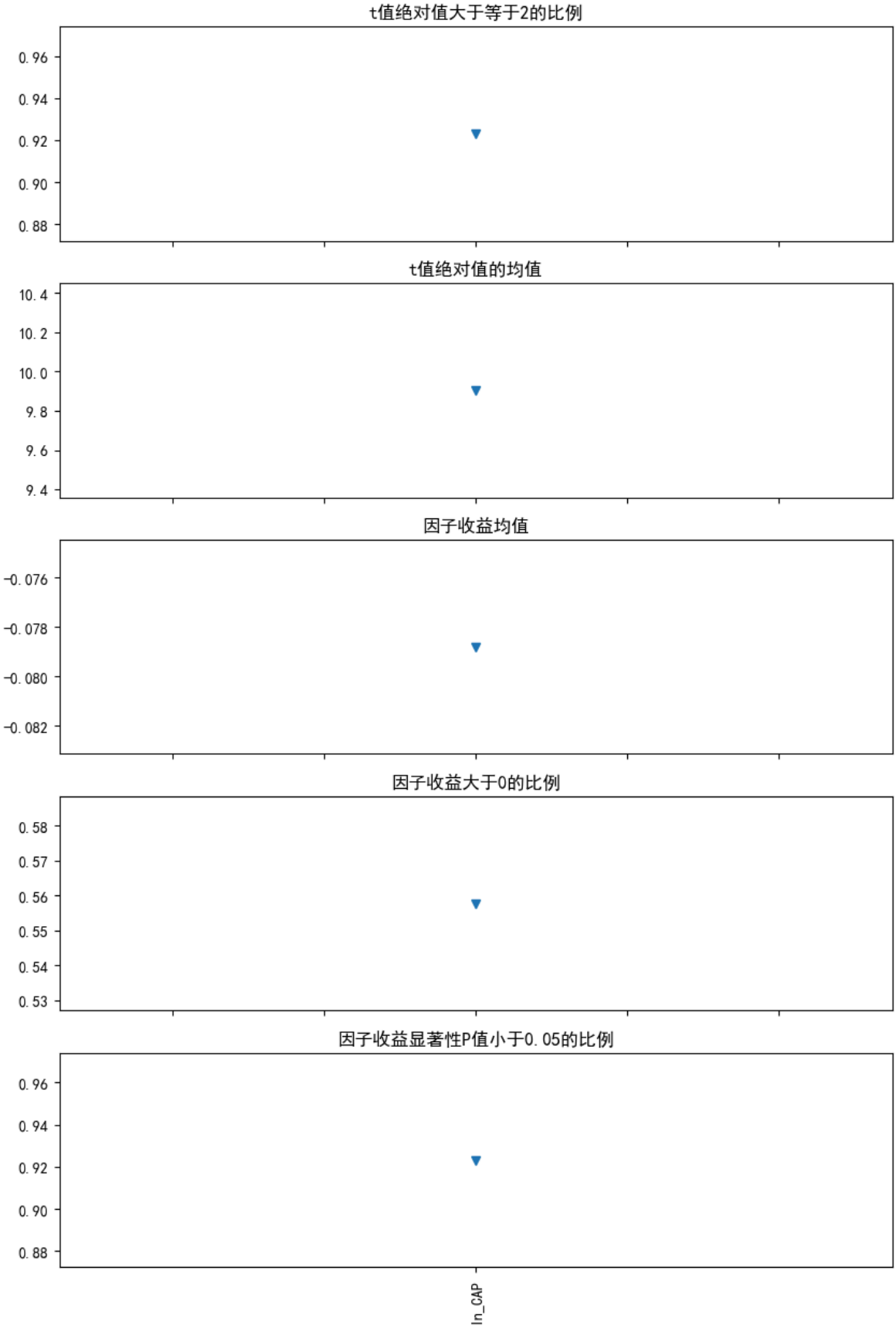
In [38]:

```
def test_result(t_test,reg_params,p_value):
    factors=list(t_test.columns)
    factors_=OrderedDict()
    for factor in factors:
        dic_df=OrderedDict()

        params=reg_params[factor]
        dic_df['因子收益均值']=params.mean()
        param=[i for i in params.values if i>0]
        dic_df['因子收益大于0的比例']=len(param)/len(params)

        tvalues=t_test[factor]
        dic_df['t值绝对值的均值']=tvalues.abs().sum()/len(tvalues)
        t=[i for i in np.abs(tvalues.values) if i>=2]
        dic_df['t值绝对值大于等于2的比例']=len(t)/len(tvalues)

        p_values=p_value[factor]
        p=[i for i in p_values.values if i<0.05]
        dic_df['因子收益显著性P值小于0.05的比例']=len(p)/len(p_values)

        factors_[factor]=dic_df

    result=pd.DataFrame(factors_)
    return result
```

```
#加权最小二乘回归WLS，稳健回归回归算法一样的
results=test_result(t_test,WLS,p_value)
```

In [39]:

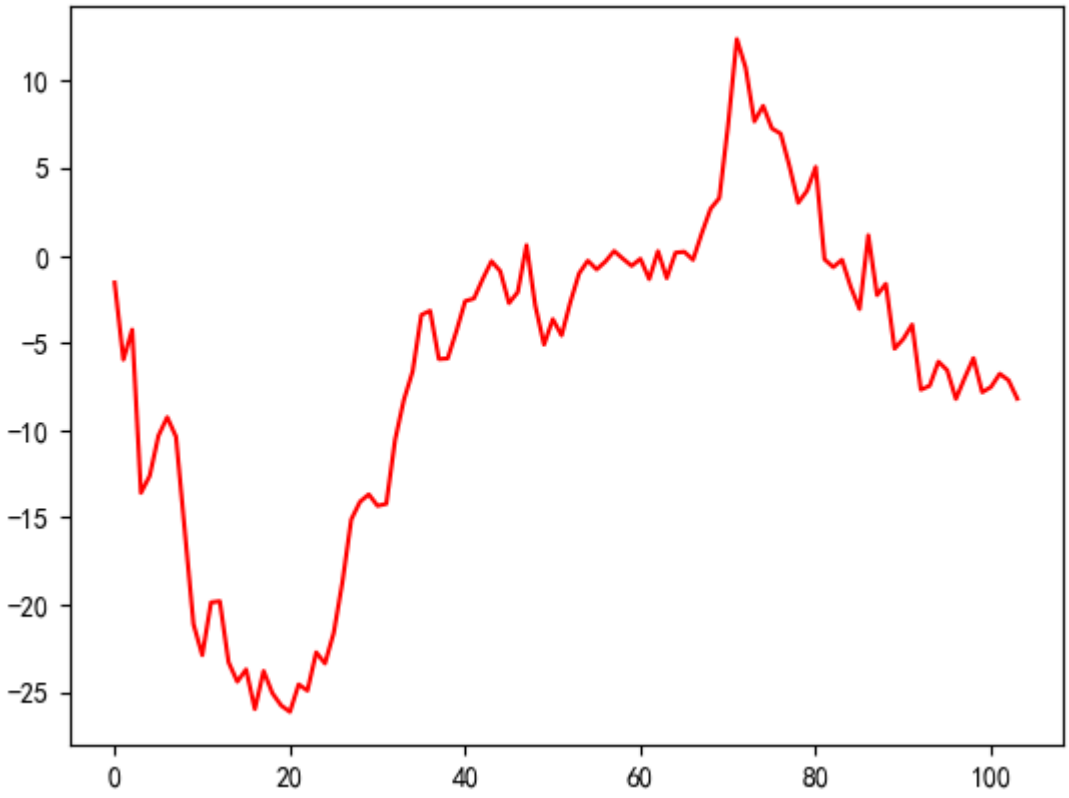
```
#分组画图(分别画所有因子上面计算指标的折线图)
def result_plot(result,result_number):
    fig,axes=plt.subplots(result_number,1)
    plt.subplots_adjust(wspace=0.5,hspace=0.5)
    plt.xticks(range(len(result.columns)),list(result.columns))
    for i in range(len(result)):
        result.iloc[i].plot(ax=axes[i],marker='v',title=result.iloc[i].na
    plt.subplots_adjust(wspace=0,hspace=0.2)
    plt.show()
result_plot(results,result_number=5)
```



In [40]:
```
# 因子累计收益率
# 因子累计收益率由因子收益累计求和得到，用于反映随着时间的推移，因子带来的总收益情况。计
#可以预期一个因子回归结果和IC的差异不会太大，既用WLS算出来的因子累计收益加总和IC分析块
WLS.head()
WLS_cum=WLS.cumsum()  #将每期的因子收益累加（对列累计求和）
import matplotlib.pyplot as plt
```

```
plt.plot(np.arange(WLS_cum.shape[0]),WLS_cum ,color = 'red')
```

Out[40]: [<matplotlib.lines.Line2D at 0x231eff7e700>]



In [ ]: