**中国股市单因子资产定价模型的实证检验**

**及A股市场反转效应检验**

摘要：本研究旨在通过实证检验探讨中国股市中单因子资产定价模型（CAPM）的有效性，并分析A股市场是否存在惯性效应或反转效应。研究选取了八支股票作为样本，利用2000-2022年的日收益和月收益数据进行分析。通过单资产时间序列检验和多资产时间序列检验，结果表明大多数股票符合CAPM模型的预期，即资产的预期收益与其市场风险（β系数）成正比。此外，通过构造基于不同时间窗口的累积收益率投资组合，研究发现中国A股市场表现出明显的反转效应，即过去表现不佳的股票在未来表现出较高的正收益，而过去表现良好的股票则相反。本研究为投资者提供了关于市场行为的理论依据，并为资产定价模型在中国股市的应用提供了实证支持。

1 文献综述

资产定价模型（CAPM）自Sharpe和Lintner提出以来，一直是金融领域研究的核心课题之一。CAPM模型基于一系列假设，包括市场是有效的、投资者是风险规避的、资产收益呈正态分布等，从而推导出资产的预期收益与其系统性风险（市场风险）之间的关系。然而，CAPM模型的有效性在实证研究中受到了广泛的检验和讨论。 在国内外的众多研究中，CAPM模型的实证检验结果并不一致。一些研究支持CAPM模型的有效性，认为资产的预期收益确实与其市场风险成正比（Fama & MacBeth, 1973）。然而，也有研究指出CAPM模型无法完全解释资产收益的异质性，特别是考虑到市场微观结构、行为金融学等因素后（Shi, Jiang & Zhou, 2015）。 此外，市场是否存在惯性效应或反转效应也是金融研究中的热点问题。惯性效应假设过去表现良好的股票将继续表现良好，而反转效应则认为过去表现良好的股票将表现不佳。在中国市场，由于其特有的市场结构和投资者行为，关于惯性效应和反转效应的研究结果也呈现出多样性。一些研究发现中国市场存在显著的反转效应（靳云汇 & 刘霖, 2001），而另一些研究则表明市场表现出惯性效应（贾权 & 陈章武, 2003）。 本研究在现有文献的基础上，通过实证分析中国A股市场的数据，旨在为CAPM模型在中国股市的应用提供新的证据，并探讨市场行为的惯性效应或反转效应。通过对单因子资产定价模型的实证检验和市场反转效应的分析，期望为投资者和市场参与者提供有价值的见解。

2 模型和方法

2.1 CAPM模型的单资产时间序列检验方法

在对这一问题的实证检验中，使用的股票的样本为股票代码为000400、000600、000800、000900、600000、600100、600200、600600的八支股票，使用的指数数据为上证指数数据。实验中所用数据为选定股票样本从2000年至2022年的日度数据。数据全部来自于RESSET数据库，数据处理计算程序使用Python编写。

CAPM理论模型：在均值-方差理论的框架下，Sharpe & Lintner 导出了存在无风险资产的资产定价均衡模型即CAPM模型（式2-1）

(2-1)

为了对CAPM模型进行单资产时间序列检验，进一步将公式改写为（式2-2）

(2-2)

其中，ri = Ri-Rf 称为资产i超额收益

如果CAPM成立，那么资产收益可分解为两个不相关的部分βimrm和εi，前者是系统风险，后者是个体风险，并且式2-3成立

(2-3)

实验中使用最小二乘回归法OLS对CAPM框架的回归方程（式2-4）进行回归

(2-4)

根据以上理论，本实验中对单个资产CAPM检验步骤为：

1. 用O LS估计市场模型，得到 αi 的估计值
2. 计算 αi = 0 的 t 检验统计量
3. 确定显著性水平，比较分位数或计算p值，作出统计推断

实证过程中使用到的Python代码如下：

计算市场收益率Rm，在本实验中使用的是上证指数。

index\_data['return'] = np.log(index\_data['close']) - np.log(index\_data['close'].shift(periods=1))

计算股票收益率Ri

stock\_data['return']=np.log(stock\_data['close'])np.log(stock\_data600000['close'].shift(periods=1))

获取无风险利率数据，将市场收益率和股票收益率于无风险利率相减就可以获得超额收益率的值

rf\_ret = merge\_data['rfreturn'].values

x = sm.add\_constant(ind\_ret-rf\_ret)

y = stk\_ret - rf\_ret

model = sm.OLS(y, x)

results = model.fit()

print(results.summary())

对数据进行OLS回归，并打印回归结果供后续分析

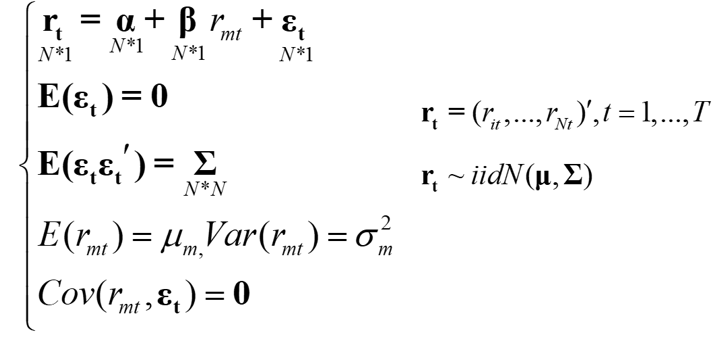
2.2 CAPM模型做多资产时间序列检验方法

在2.1部分中介绍了CAPM模型原理和进行单资产时间序列检验的方法，若要考虑N个资产对CAPM进行多资产时间序列检验，那么联合检验的结果需满足式2-5

(2-5)

即需要联合检验所有资产的α值是否为0。

将单资产的回归方程写成矩阵形式如图2-1



**图2-1 单资产回归方程的矩阵形式**

在独立同分布的假设下，用极大似然估计与最小二乘估计一样。因此，本实验用最小二乘估计每个资产的参数，然后再进行联合检验。

以下为在实验中使用的具体方法和代码：

在实验中我们设置无限制模型和限制模型进行假设检验，评估CAPM模型的有效性。无限制模型不基于任何特定的假设，它允许估计的回归模型包含一个截距项在CAPM的背景下，这个截距项代表了资产的特定风险或超额收益，即资产收益超过CAPM模型预测收益的部分。无限制模型的设定允许我们检验市场中是否存在系统性风险之外的其他因素影响资产收益。如果无限制模型的截距项显著不为零，这可能表明存在其他未被市场风险解释的风险因素。

限制模型是基于CAPM理论的假设建立的，它不包含截距项，即假设所有资产的超额收益为零。这是因为CAPM模型预测，资产的预期收益应该完全由其市场风险（即β系数）决定，而与特定风险无关。在限制模型中，我们检验CAPM模型是否能够准确预测资产的预期收益，即市场风险溢价是否能够完全解释资产收益。

1. 无限制模型估计

x = np.ones((T, 2)) *# 创建设计矩阵，包含截距和市场收益率*

x[:, 1] = ret\_ind y = ret\_stocks *# 股票收益率*

*# 计算X的转置乘以X和X的转置乘以y*

xTx = np.dot(np.transpose(x), x)

xTy = np.dot(np.transpose(x), y)

*# 使用最小二乘法估计参数*

AB\_hat = np.dot(np.linalg.inv(xTx), xTy)

ALPHA = AB\_hat[0] *# 截距项，即超额收益*

BETA = AB\_hat[1] *# 斜率项，即风险系数*

1. 限制模型估计

xr = np.ones((T, 1)) *# 创建只包含截距的设计矩阵*

xr[:, 0] = ret\_ind yr = ret\_stocks *# 股票收益率*

*# 计算Xr的转置乘以Xr和Xr的转置乘以yr*

xrTxr = np.dot(np.transpose(xr), xr)

xrTyr = np.dot(np.transpose(xr), yr)

*# 使用最小二乘法估计参数*

ABr\_hat = np.dot(np.linalg.inv(xrTxr), xrTyr)

在实证中使用了三种统计检验方法来评估CAPM模型的有效性：Wald检验、似然比（LR）检验和拉格朗日乘子（LM）检验。

1. Wald检验是一种基于估计参数的检验方法，在实验中用于检验CAPM模型中的超额收益（ALPHA）是否显著不为零。实验代码中，SWchi2 是Wald检验的卡方统计量，SWF 是Wald检验的F统计量，pvalue\_Wchi2 和 pvalue\_WF 分别是卡方分布和F分布下的p值，用于判断超额收益是否显著不为零。

2. 似然比检验是一种比较无限制模型和限制模型的似然函数值的方法，用于检验两个模型之间的差异是否显著。实验代码中，SLRchi2 是似然比检验的卡方统计，pvalue\_SLRchi2 是卡方分布下的p值，用于判断无限制模型和限制模型之间的差异是否显著。

3. 拉格朗日乘子检验是一种用于检验模型中是否存在遗漏变量的检验方法。在本实验中，LM检验用于检验CAPM模型是否完全由市场风险因子解释，即是否存在未被市场风险因子捕捉到的风险。实验代码中，score 是得分向量，用于计算LM统计量。Minfo 是信息矩阵，用于调整得分向量的协方差。SLMchi2 是拉格朗日乘子检验的卡方统计量。pvalue\_SLMchi2 是卡方分布下的p值，用于判断CAPM模型是否存在遗漏变量。

使用三种统计检验方法得出的结果将在3.2部分对CAPM模型的多资产时间序列检验问题中进行讨论。

2.3 A股市市场惯性效应/反转效应的实证检验方法

在对这一问题的实证检验中，使用的股票的样本为865支2001年之前已经上市且2022年还未退市的A股股票，实验中所用数据为选定股票样本从2001年至2022年的月度数据，共计227232条有效数据。数据全部来自于RESSET数据库，数据处理计算程序全部使用Python编写。

2.3.1实证研究思路

惯性效应是指过去表现好的股票（称为赢者）会继续表现好，过去表现差的股票（称为输者）会继续表现差。与惯性效应正好相反，反转效应是指过去表现好的股票接下来会表现差，过去表现差的股票接下来会表现好。在该实证分析中所用的数据是所有在上交所交易股票的月度收益率。

在每一个月，首先计算过去1个月、3个月、6个月和12个月的累积收益率。然后，在每个月的月末，根据过去1个月、3个月、6个月和12个月的收益率，分别将所有股票根据过去的收益率由低到高排序，构造5个等权重投资组合。接下来持有这5个组合1个月、3个月、6个月或12个月。也就是说，实验中调整组合的频率是每月1次，构造组合利用的信息分别是过去1个月、3个月、6个月或12个月的股票收益率，组合的持有期分别是1个月、3个月、6个月或12个月。

2.3.1数据分组

在该问题的实证研究中，采用横截面检验的检验方法。根据实验思路，在每个月月末需要根据股票过去N（1、3、6、12）个月的累积收益率对股票进行分组。分组组数为5组，分组依据所有股票过去N个月的累积收益率进行排序，累积收益率较低的股票被分配到第一组，累积收益率较高的股票被分配到第五组。也就是说，第一组内的股票是过去表现较差的股票，第五组内的股票是过去表现较好的股票。每组内部的股票数量是相等的，合理精确的分组是后续分析的基础。

接下来说明如何利用Python对股票数据进行分组：

for date, group in all\_returns.groupby('date'):

group['portfolio'] = pd.qcut(group['cumulative\_return\_' + str(sample\_month)], q=5, labels=False) + 1

这两行代码的功能是在每一个月的月末根据股票前sample\_month个月的累积收益率将所有股票划分为了5个组，’portfolio’是每支股票的组别号。all\_returns中存放了每一支股票在各个时间点上前N（1、3、6、12）个月的累积收益率，列名分别为cumulative\_return\_N，为后续计算提供数据支撑。group代表了按照日期（date列）分组后的数据, 包含了同一日期下的所有股票数据。使用pd.qcut()函数根据股票的累积收益率将所有股票分成5个等分位数的组，并将组号记录在group['portfolio']中。

2.3.2投资组合M个月的累积收益率计算

按照实验思路，在获得五个股票投资组合后，需要跟踪投资组合收益率序列，计算每个投资组合M（1、3、6、12）个月的累积收益率。

下面说明使用Python计算累积收益率的方法：

M\_values = [1, 3, 6, 12] # 持有期M

for M in M\_values:

future\_data = all\_returns[all\_returns['date'] > date]

future\_data['date'] = pd.to\_datetime(future\_data['date'])

future\_data = future\_data[future\_data['date'] <= pd.to\_datetime(date) + pd.DateOffset(months=M)]

在每一持有期M下，筛选出日期大于当前日期同时小于当前日期+M的数据，即找到未来M个月的数据。

future\_returns = future\_data.groupby('stock\_code')['return'].sum()

group['future\_return\_' + str(M)] = group['stock\_code'].map(future\_returns)

按股票代码分组，计算未来M个月的累积收益率，并将结果存放在group['future\_return\_' + str(M)]中

通过多次循环，就可以获得每个月的数据，通过使用pd.concat()函数，将每个月的数据拼接汇总到monthly\_returns中，monthly\_returns中的数据包括'date', 'future\_return\_1', 'future\_return\_3', 'future\_return\_6', 'future\_return\_12','portfolio'，分别为日期（月份），未来1、3、6、12月的累积收益率，股票所属的组别号。

average\_returns = monthly\_returns.groupby(['date', 'portfolio']).mean()

grouped\_returns = average\_returns.groupby('portfolio').sum()

num\_months = len(monthly\_returns.groupby('date'))

final\_returns = grouped\_returns / num\_months

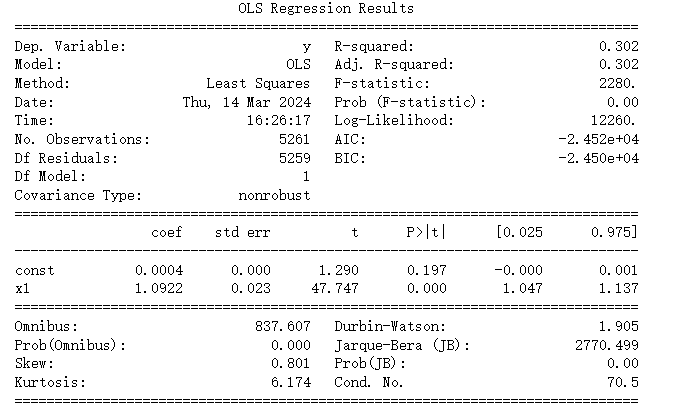
通过以上代码可以计算得出每个分组的264个月（2001年1月至2022年12月）的平均累积收益率。

获得这一数据后，接下来就可以对这一数据进行分析并判断中国A股市场存在惯性效应还是反转效应。

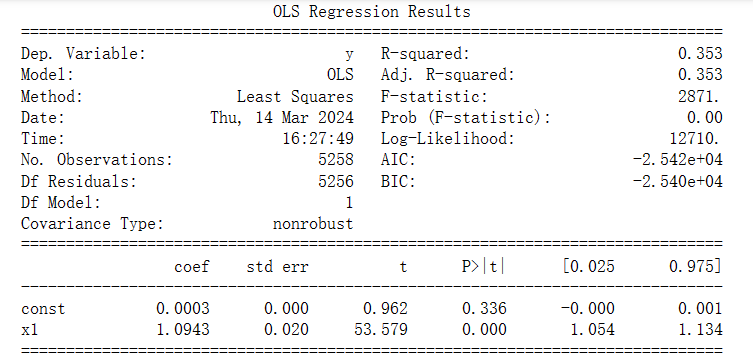
3 结果与讨论

3.1CAPM模型的单资产时间序列检验的结果与讨论

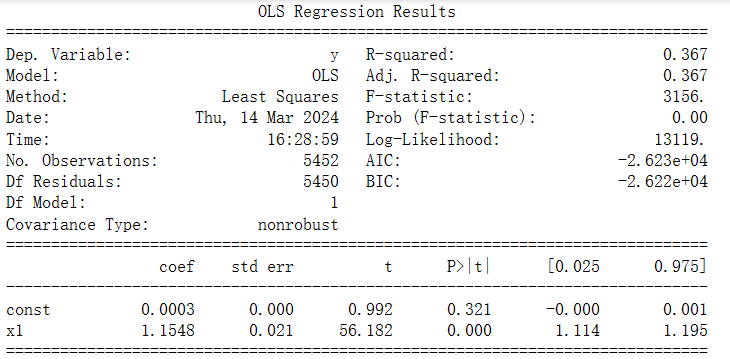
在CAPM模型的单资产时间序列检验中，分别对股票代码为000400、000600、000800、000900、600000、600100、600200、600600的八支股票数据进行OLS回归，回归结果见下图。



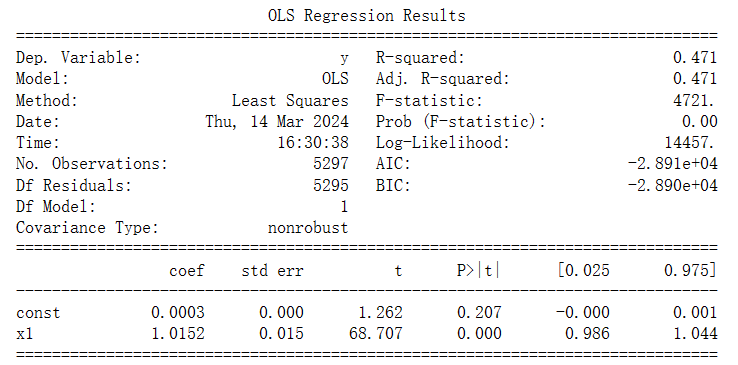
**图3-1 000400股票的回归结果**



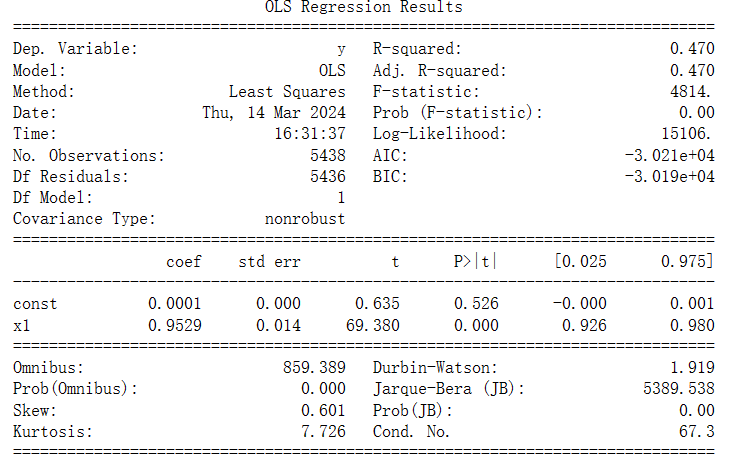
**图3-2 000600股票的回归结果**



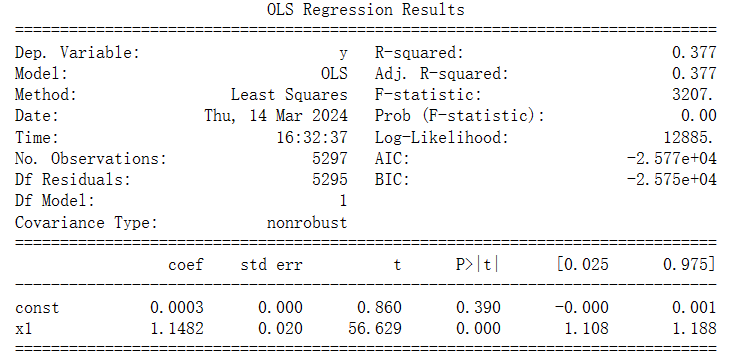
**图3-3 000800股票的回归结果**



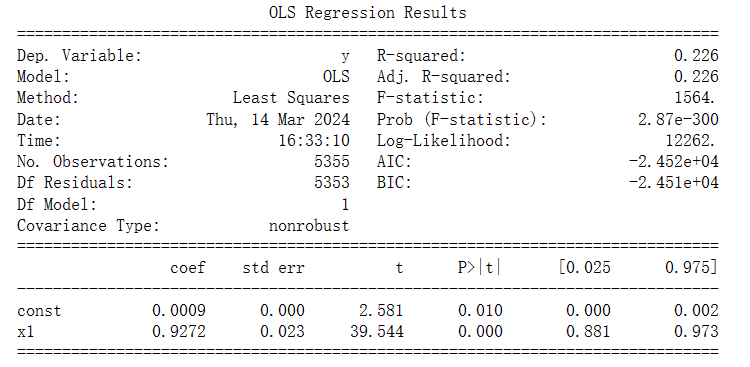
**图3-4 000900股票的回归结果**



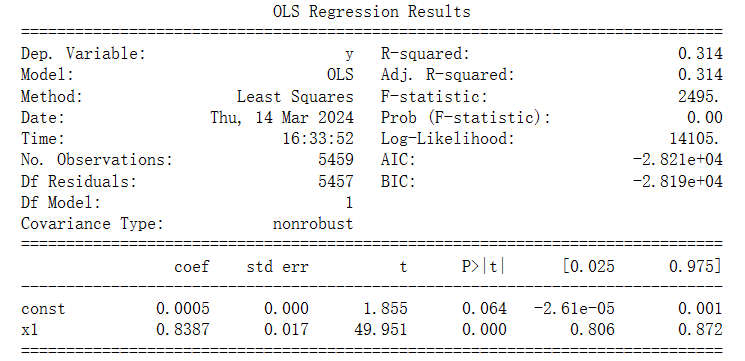
**图3-5 600000股票的回归结果**



**图3-6 600100股票的回归结果**



**图3-7 600200股票的回归结果**



**图3-8 600600股票的回归结果**

以上分别为8支股票的OLS回归结果，其中const就是式2-4中的截距项alpha，如果回归结果显示const为0，那么该股票就是符合CAPM模型的。

以000400股票为例，其进行OLS回归后，得到const的P值为0.197，在5%的显著性水平下不显著，不能拒绝原假设***H*0: *αi* = 0，**说明回归的截距项为0，即000400股票的数据是符合CAPM模型的。

类似的在000600、000800、000900、600000、600100、600600的股票回归数据中，其P值也都明显大于5%的显著性水平，即这6支股票也都符合CAPM模型。

但在8支股票中也有一支股票600200的回归数据中P值为0.01在5%的显著性水平下显著，即可以拒绝原假设***H*0: *αi* = 0，**接受备择假设***H*1: *αi* ≠ 0**，式2-4中的截距项alpha不等于0，说明股票600200不符合CAPM模型。

综上，大多数股票的回归结果显示其满足CAPM模型，只有一支股票不满足CAPM模型，说明中国A股中大多数股票是符合CAPM模型的，因此单资产检验支持CAPM模型。

3.2CAPM模型多资产时间序列检验

在CAPM模型多资产时间序列检验中，我们使用到了Wald检验、似然比（LR）检验和拉格朗日乘子（LM）检验。

在Wald检验中，计算得到的结果中pvalue\_Wchi2的值为0.3761，pvalue\_WF的值为0.4052。pvalue\_Wchi2 和 pvalue\_WF 分别是卡方分布和F分布下的p值，用于判断超额收益是否显著不为零。在Wald检验结果中pvalue\_Wchi2和pvalue\_WF两个p值都相对较高，远高于显著性水平（5%）。这说明，在Wald检验中，不能拒绝原假设，即不存在超额收益alpha。

在似然比（LR）检验中，计算得到的结果pvalue\_SLRchi2的值为0.3880，pvalue\_SLRchi2值是用来评估两个模型之间的差异是否统计显著。在似然比检验结果中，pvalue\_SLRchi2的值为0.3880。这个p值相对较高，远高于显著性水平（5%）。所以没有足够的证据拒绝原假设，即限制模型和无限制模型之间没有显著差异。这个检验结果表明，CAPM模型在统计上能够合理地解释资产收益，没有必要添加额外的风险因子来提高模型的解释能力。

在拉格朗日乘子（LM）检验中，计算得到的结果pvalue\_SLMchi2的值为0.3999。LM检验是一种用于检测模型中是否存在遗漏变量或未观测到的异方差性的方法。在本实验背景下，LM检验可以帮助我们判断市场风险因子是否充分捕捉了资产收益的所有系统性风险。在LM检验结果中，pvalue\_SLMchi2的值为0.3999。这个p值高于显著性水平（5%），所以没有足够的证据拒绝原假设，即CAPM模型已经包含了所有重要的系统性风险因子，没有必要添加额外的解释变量来提高模型的解释能力。

通过以上检验结果我们可以得出结论，在联合检验中我们不能拒绝原假设，因此，检验股票不存在超额回报。综上，多资产联合检验支持CAPM模型。

3.3判断中国A股市场存在惯性效应还是反转效应

根据2.3中提出的计算模型计算得出的数据如下：

**表3.3.1 基于过去1个月的累积收益率构造的五分位投资组合的平均收益率**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **持有期**  **组号** | **1个月** | **3个月** | **6个月** | **12个月** |
| **Q1** | **0.5108%** | **0.7057%** | **0.2011%** | **-1.1530%** |
| **Q2** | **0.5155%** | **0.7621%** | **0.6741%** | **0.6685%** |
| **Q3** | **0.3478%** | **0.5928%** | **0.6659%** | **0.8610%** |
| **Q4** | **-0.1974%** | **-0.5672%** | **-0.8033%** | **-0.8529%** |
| **Q5** | **-1.5967%** | **-3.0430%** | **-4.2549%** | **-5.5096%** |
| **Q5-Q1** | **-2.1075%** | **-3.7487%** | **-4.4560%** | **-4.3566%** |

**表3.3.2 基于过去3个月的累积收益率构造的五分位投资组合的平均收益率**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **持有期**  **组号** | **1个月** | **3个月** | **6个月** | **12个月** |
| **Q1** | **0.6952%** | **1.0281%** | **0.3385%** | **-0.6747%** |
| **Q2** | **0.4620%** | **0.8905%** | **0.8590%** | **0.9637%** |
| **Q3** | **0.2972%** | **0.6957%** | **0.7645%** | **1.3490%** |
| **Q4** | **-0.2377%** | **-0.7059%** | **-0.6738%** | **-0.9070%** |
| **Q5** | **-1.6333%** | **-3.4351%** | **-4.7843%** | **-6.6876%** |
| **Q5-Q1** | **-2.3258%** | **-4.4632%** | **-5.1228%** | **-6.0129%** |

**表3.3.3 基于过去6个月的累积收益率构造的五分位投资组合的平均收益率**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **持有期**  **组号** | **1个月** | **3个月** | **6个月** | **12个月** |
| **Q1** | **0.5782%** | **0.6410%** | **0.0673%** | **-0.1890%** |
| **Q2** | **0.5476%** | **0.9136%** | **0.9571%** | **1.4434%** |
| **Q3** | **0.2595%** | **0.6703%** | **0.8551%** | **1.2709%** |
| **Q4** | **-0.3062%** | **-0.5032%** | **-0.5846%** | **-0.9635%** |
| **Q5** | **-1.4911%** | **-3.2414%** | **-4.7800%** | **-7.5138%** |
| **Q5-Q1** | **-2.0693%** | **-3.8824%** | **-4.8473%** | **-7.3248%** |

**表3.3.4 基于过去12个月的累积收益率构造的五分位投资组合的平均收益率**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **持有期**  **组号** | **1个月** | **3个月** | **6个月** | **12个月** |
| **Q1** | **0.4080%** | **0.6067 %** | **0.5932%** | **-1.0216%** |
| **Q2** | **0.4642%** | **0.8953%** | **1.1971%** | **2.1287%** |
| **Q3** | **0.2415%** | **0.6190%** | **0.8138%** | **1.0691%** |
| **Q4** | **-0.2395%** | **-0.5798%** | **-1.0937%** | **-1.6824%** |
| **Q5** | **-1.2879%** | **-3.0645%** | **-4.9996%** | **-8.4910%** |
| **Q5-Q1** | **-1.6959%** | **-3.6712%** | **-5.5928%** | **-7.4694%** |

表3.3.1-3.3.4报告了以2001年1月至2022年12月作为投资组合构造期的每一分位投资组合的平均收益率。

根据表3.3.1的结果显示，当股票依过去1个月的收益率排序时，过去的输者（组Q1）的未来表现胜过了过去的赢者（组Q5）。过去输者的一个月持有期收益率为0.5108%，然而过去赢者的一个月持有期收益率只有-1.5967%，它们的差是-2.1075%。其他3个月、6个月和12个月持有期收益率也具有同样的模式。过去输者与过去赢者的平均收益率之差分别等于-3.7487%、-4.4560%和-4.3566%。以上结果显示中国A股市场存在反转效应，即过去的输者将表现好而过去的赢者将表现差。

表3.3.2、3.3.3、3.3.4依次报告了依过去3、6、12个月的收益率排序时的结果，它们都具有与表3.3.1中分析出的类似的反转效应特征。

4 参考文献

1. 靳云汇，刘霖. 中国股票市场CAPM的实证研究[J]. 金融研究，2001，(7)
2. 贾权，陈章武. 中国股市有效性的实证分析[J]. 金融研究，2003，(7)：86-92.
3. Fama, E. F., & MacBeth, J. D. Risk, Return, and Equilibrium: Empirical Tests[J]. The Journal of Political Economy, 1973, 81 (3): 607-636.
4. Shi, H.-L., Jiang, Z.-Q., & Zhou, W.-X. Profitability of Contrarian Strategies in the Chinese Stock Market[J]. PLoS ONE, 2015, 10 (9): e0137892.

5 附录

本次实验中使用的所有代码：

1. CAPM模型的单资产时间序列检验代码

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

# 读取上证指数的数据

index\_data = pd.read\_excel('./stock\_data/日度数据/000001\_index/000001RESSET\_IDXQTTN\_1.xls', encoding='GB2312', usecols=[1, 5])

#修改列的名称

index\_data.columns = ['date', 'close']

#去除数据中的nan值

index\_data.dropna(inplace=True)

#计算上证指数每日的对数收益率

index\_data['return'] = np.log(index\_data['close']) - np.log(index\_data['close'].shift(periods=1))

#去除收益率为NAN的值

index\_data.dropna(inplace=True)

#读取股票的数据

stock\_data600000 = pd.read\_excel('./stock\_data/日度数据/600000/600000stock\_data\_2000\_2022\_combined.xls', encoding='GB2312', usecols=[1, 5, 10])

#修改列名 日期，收盘价，无风险收益率

stock\_data600000.columns = ['date', 'close', 'rfreturn']

# 去除股票数据中的NAN值

stock\_data600000.dropna(inplace=True)

# 计算股票的收益率

stock\_data600000['return'] = np.log(stock\_data600000['close']) - np.log(stock\_data600000['close'].shift(periods=1))

# 去除NAN值

stock\_data600000.dropna(inplace=True)

# 收益率应该在-0.1-0.1之间，去除掉异常数据

ind = (stock\_data600000['return'] >= -0.1) & (stock\_data600000['return'] <= 0.1)

stock\_data600000 = stock\_data600000.loc[ind, :]

# 拼接股票数据和上证指数数据

merge\_data = pd.merge(left=stock\_data600000[['date', 'return', 'rfreturn']],

right=index\_data[['date', 'return']],

on='date',

how='inner')

merge\_data.columns = ['date', 'return\_stk600000', 'rfreturn', 'return\_ind']

stk\_ret = merge\_data['return\_stk600000'].values

rf\_ret = merge\_data['rfreturn'].values

ind\_ret = merge\_data['return\_ind'].values

# 股票超额收益率和市场超额收益率散点图

plt.plot(ind\_ret - rf\_ret, stk\_ret - rf\_ret, 'o', ms=5, mfc='w', lw=2)

plt.xlabel(r'$r\_m-r\_f$', fontsize=20)

plt.ylabel(r'$r\_i-r\_f$', fontsize=20)

x = sm.add\_constant(ind\_ret-rf\_ret)

y = stk\_ret - rf\_ret

model = sm.OLS(y, x)

results = model.fit()

print(results.summary()) #打印回归结果

# 定义一个函数，快速进行数据处理和OLS回归分析

def DataOLS(stockData,indexData):

merge\_data = pd.merge(left=stockData[['date', 'return', 'rfreturn']],

right=indexData[['date', 'return']],

on='date',

how='inner')

merge\_data.columns = ['date', 'return\_stk', 'rfreturn', 'return\_ind']

stk\_ret = merge\_data['return\_stk'].values

rf\_ret = merge\_data['rfreturn'].values

ind\_ret = merge\_data['return\_ind'].values

x = sm.add\_constant(ind\_ret-rf\_ret)

y = stk\_ret - rf\_ret

model = sm.OLS(y, x)

results = model.fit()

print(results.summary())

# 定义一个股票数据清洗函数

def stockDataClean(stockData):

#修改列名 日期，收盘价，无风险收益率

stockData.columns = ['date', 'close', 'rfreturn']

# 去除股票数据中的NAN值

stockData.dropna(inplace=True)

# 计算股票的收益率

stockData['return'] = np.log(stockData['close']) - np.log(stockData['close'].shift(periods=1))

# 去除NAN值

stockData.dropna(inplace=True)

# 收益率应该在-0.1-0.1之间，去除掉异常数据

ind = (stockData['return'] >= -0.1) & (stockData['return'] <= 0.1)

stockData = stockData.loc[ind, :]

return stockData

# 利用封装的函数进行OLS回归得到结果

# 000400股票数据

stock\_000400 = pd.read\_excel('./stock\_data/日度数据/000400/000400stock\_data\_2000\_2022\_combined.xls', encoding='GB2312', usecols=[1, 5, 10])

stock\_000400\_pro = stockDataClean(stock\_000400)

DataOLS(stock\_000400\_pro,index\_data)

# 000600股票数据

stock\_000600 = pd.read\_excel('./stock\_data/日度数据/000600/000600stock\_data\_2000\_2022\_combined.xls', encoding='GB2312', usecols=[1, 5, 10])

stock\_000600\_pro = stockDataClean(stock\_000600)

DataOLS(stock\_000600\_pro,index\_data)

# 000800股票数据

stock\_000800 = pd.read\_excel('./stock\_data/日度数据/000800/000800stock\_data\_2000\_2022\_combined.xls', encoding='GB2312', usecols=[1, 5, 10])

stock\_000800\_pro = stockDataClean(stock\_000800)

DataOLS(stock\_000800\_pro,index\_data)

# 000900股票数据

stock\_000900 = pd.read\_excel('./stock\_data/日度数据/000900/000900stock\_data\_2000\_2022\_combined.xls', encoding='GB2312', usecols=[1, 5, 10])

stock\_000900\_pro = stockDataClean(stock\_000900)

DataOLS(stock\_000900\_pro,index\_data)

# 600000股票数据

stock\_600000 = pd.read\_excel('./stock\_data/日度数据/600000/600000stock\_data\_2000\_2022\_combined.xls', encoding='GB2312', usecols=[1, 5, 10])

stock\_600000\_pro = stockDataClean(stock\_600000)

DataOLS(stock\_600000\_pro,index\_data)

# 600100股票数据

stock\_600100 = pd.read\_excel('./stock\_data/日度数据/600100/600100stock\_data\_2000\_2022\_combined.xls', encoding='GB2312', usecols=[1, 5, 10])

stock\_600100\_pro = stockDataClean(stock\_600100)

DataOLS(stock\_600100\_pro,index\_data)

# 600200股票数据

stock\_600200 = pd.read\_excel('./stock\_data/日度数据/600200/600200stock\_data\_2000\_2022\_combined.xls', encoding='GB2312', usecols=[1, 5, 10])

stock\_600200\_pro = stockDataClean(stock\_600200)

DataOLS(stock\_600200\_pro,index\_data)

# 600600股票数据

stock\_600600 = pd.read\_excel('./stock\_data/日度数据/600600/600600stock\_data\_2000\_2022\_combined.xls', encoding='GB2312', usecols=[1, 5, 10])

stock\_600600\_pro = stockDataClean(stock\_600600)

DataOLS(stock\_600600\_pro,index\_data)

2.CAPM模型做多资产时间序列检验代码

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

from scipy.stats import chi2, f

import warnings

warnings.filterwarnings("ignore")

stock\_data000400 = pd.read\_excel('./stock\_data/月度数据/000400/000400RESSET\_MRESSTK\_1.xls',encoding='GB2312',usecols=[1,2])

stock\_data000600 = pd.read\_excel('./stock\_data/月度数据/000600/000600RESSET\_MRESSTK\_1.xls',encoding='GB2312',usecols=[1,2])

stock\_data000800 = pd.read\_excel('./stock\_data/月度数据/000800/000800RESSET\_MRESSTK\_1.xls',encoding='GB2312',usecols=[1,2])

stock\_data000900 = pd.read\_excel('./stock\_data/月度数据/000900/000900RESSET\_MRESSTK\_1.xls',encoding='GB2312',usecols=[1,2])

stock\_data600000 = pd.read\_excel('./stock\_data/月度数据/600000/600000RESSET\_MRESSTK\_1.xls',encoding='GB2312',usecols=[1,2])

stock\_data600100 = pd.read\_excel('./stock\_data/月度数据/600100/600100RESSET\_MRESSTK\_1.xls',encoding='GB2312',usecols=[1,2])

stock\_data600200 = pd.read\_excel('./stock\_data/月度数据/600200/600200RESSET\_MRESSTK\_1.xls',encoding='GB2312',usecols=[1,2])

stock\_data600600 = pd.read\_excel('./stock\_data/月度数据/600600/600600RESSET\_MRESSTK\_1.xls',encoding='GB2312',usecols=[1,2])

index\_data = pd.read\_excel('./stock\_data/日度数据/000001\_index/000001RESSET\_IDXQTTN\_1.xls',encoding='GB2312',usecols=[1,5]) # 读取的是日度上证指数

rf\_data = pd.read\_excel('./stock\_data/月度数据/600000/600000RESSET\_MRESSTK\_1.xls',encoding='GB2312',usecols=[1,3]) #月无风险收益率

# 获取指数的月度数据

def monthIndex(stock\_data,index\_data):

stock\_data.columns = ['date', 'stock\_close']

index\_data.columns = ['date', 'index\_close']

mergeData = pd.merge(left=stock\_data,

right=index\_data[['date', 'index\_close']],

on='date',

how='inner',

sort=True)

filtered\_data = mergeData[['date', 'index\_close']]

filtered\_data['return'] = np.log(filtered\_data['index\_close']) - np.log(filtered\_data['index\_close'].shift(periods=1))

filtered\_data['date'] = pd.to\_datetime(filtered\_data['date'])

filtered\_data['date'] = filtered\_data['date'].dt.strftime('%Y-%m')

return filtered\_data

index\_data\_cal = monthIndex(stock\_data600000,index\_data)

# 股票数据处理函数

def stockDataProcess(data):

data.columns = ['date', 'close']

data['date'] = pd.to\_datetime(data['date'])

data['date'] = data['date'].dt.strftime('%Y-%m')

data['return'] = np.log(data['close']) - np.log(data['close'].shift(periods=1))

return data

# 无风险收益率数据处理

def rfDataProcess(rfData):

rfData.columns = ['date', 'rfreturn']

rfData['date'] = pd.to\_datetime(rfData['date'])

rfData['date'] = rfData['date'].dt.strftime('%Y-%m')

return rfData

# 处理所有的股票数据

stock\_data000400 = stockDataProcess(stock\_data000400)

stock\_data000600 = stockDataProcess(stock\_data000600)

stock\_data000800 = stockDataProcess(stock\_data000800)

stock\_data000900 = stockDataProcess(stock\_data000900)

stock\_data600000 = stockDataProcess(stock\_data600000)

stock\_data600100 = stockDataProcess(stock\_data600100)

stock\_data600200 = stockDataProcess(stock\_data600200)

stock\_data600600 = stockDataProcess(stock\_data600600)

#将所有的数据拼接

data\_matrix = pd.merge(left=index\_data\_cal[['date', 'return']],

right=stock\_data000400[['date', 'return']],

on='date',

how='inner',

sort=True) # sort=True参数确保合并后的DataFrame按照日期排序

data\_matrix = pd.merge(left=data\_matrix,

right=stock\_data000600[['date', 'return']],

on='date',

how='inner',

sort=True)

data\_matrix = pd.merge(left=data\_matrix,

right=stock\_data000800[['date', 'return']],

on='date',

how='inner',

sort=True)

data\_matrix = pd.merge(left=data\_matrix,

right=stock\_data000900[['date', 'return']],

on='date',

how='inner',

sort=True)

# 拼接stock\_data600000的数据

data\_matrix = pd.merge(left=data\_matrix,

right=stock\_data600000[['date', 'return']],

on='date',

how='inner',

sort=True)

# 2

data\_matrix = pd.merge(left=data\_matrix,

right=stock\_data600100[['date', 'return']],

on='date',

how='inner',

sort=True)

# 3

data\_matrix = pd.merge(left=data\_matrix,

right=stock\_data600200[['date', 'return']],

on='date',

how='inner',

sort=True)

data\_matrix = pd.merge(left=data\_matrix,

right=stock\_data600600[['date', 'return']],

on='date',

how='inner',

sort=True)

# 处理月无风险收益

rf\_data = rfDataProcess(rf\_data)

# 将月无风险收益拼接

data\_matrix = pd.merge(left=data\_matrix,

right=rf\_data[['date', 'rfreturn']],

on='date',

how='inner',

sort=True)

data\_matrix.columns = ['date', 'index', 'stk000400', 'stk000600', 'stk000800', 'stk000900', 'stk600000', 'stk600100', 'stk600200', 'stk600600','rf']

# 对全部拼接完成的数据进行处理

data\_matrix.dropna(inplace=True) #去除掉na值

# 计算超额收益率

data\_matrix['index'] = data\_matrix['index'] - data\_matrix['rf']

data\_matrix['stk000400'] = data\_matrix['stk000400'] - data\_matrix['rf']

data\_matrix['stk000600'] = data\_matrix['stk000600'] - data\_matrix['rf']

data\_matrix['stk000800'] = data\_matrix['stk000800'] - data\_matrix['rf']

data\_matrix['stk000900'] = data\_matrix['stk000900'] - data\_matrix['rf']

data\_matrix['stk600000'] = data\_matrix['stk600000'] - data\_matrix['rf']

data\_matrix['stk600100'] = data\_matrix['stk600100'] - data\_matrix['rf']

data\_matrix['stk600200'] = data\_matrix['stk600200'] - data\_matrix['rf']

data\_matrix['stk600600'] = data\_matrix['stk600600'] - data\_matrix['rf']

ret\_ind = data\_matrix['index'].values

T = len(ret\_ind) #计算了变量 ret\_ind 的长度

N = 8 #变量 N，表示有多少支股票

mu\_market = np.mean(ret\_ind) #计算了 ret\_ind 中所有收益率数据的平均值，代表了市场收益率的平均值。

sigma\_market = np.sum((ret\_ind-mu\_market)\*\*2)/T #计算了市场收益率的方差

ret\_stocks = data\_matrix[['stk000400', 'stk000600', 'stk000800', 'stk000900', 'stk600000', 'stk600100','stk600200','stk600600']].values

ret\_stocks

# 无限制模型

x = np.ones((T, 2))

x[:, 1] = ret\_ind

y = ret\_stocks

xTx = np.dot(np.transpose(x), x)

xTy = np.dot(np.transpose(x), y)

AB\_hat = np.dot(np.linalg.inv(xTx), xTy)

ALPHA = AB\_hat[0]

print(ALPHA)

BETA = AB\_hat[1]

RESD = y - np.dot(x, AB\_hat)

COV = np.dot(np.transpose(RESD), RESD)/T

invCOV = np.linalg.inv(COV)

# 限制模型

xr = np.ones((T, 1))

xr[:, 0] = ret\_ind

yr = ret\_stocks

xrTxr = np.dot(np.transpose(xr), xr)

xrTyr = np.dot(np.transpose(xr), yr)

ABr\_hat = np.dot(np.linalg.inv(xrTxr), xrTyr)

RESDr = yr - np.dot(xr, ABr\_hat)

COVr = np.dot(np.transpose(RESDr), RESDr)/T

invCOVr = np.linalg.inv(COVr)

# Wald检验

trans\_ALPHA = np.ones((len(ALPHA), 1))

trans\_ALPHA[:, 0] = ALPHA

SWchi2 = T\*(1/(1+mu\_market\*\*2/sigma\_market))\*np.dot(np.dot(ALPHA, invCOV), trans\_ALPHA)

SWF = (T-N-1)/N\*(1/(1+mu\_market\*\*2/sigma\_market))\*np.dot(np.dot(ALPHA, invCOV), trans\_ALPHA)

pvalue\_Wchi2 = 1 - chi2.cdf(SWchi2[0], N)

pvalue\_WF = 1 - f.cdf(SWF[0], N, T-N-1)

print(pvalue\_Wchi2)

print(pvalue\_WF)

# 似然比检验

SLRchi2 = T\*(np.log(np.linalg.det(COVr)) - np.log(np.linalg.det(COV)))

pvalue\_SLRchi2 = 1 - chi2.cdf(SLRchi2, N)

print(pvalue\_SLRchi2)

# 拉格朗日乘子检验

a = np.zeros((8, 1))

a[:, 0] = np.sum(RESDr, axis=0)

salpha = np.dot(invCOVr, a)

b = np.dot(ret\_ind, RESDr)

sbeta = np.zeros((8,1))

sbeta[:, 0] = np.dot(invCOVr, b)

score = np.concatenate((salpha, sbeta), axis=0)

print(score)

a = np.concatenate((invCOVr\*T, invCOVr\*np.sum(ret\_ind)), axis=1)

b = np.concatenate((invCOVr\*np.sum(ret\_ind), invCOVr\*np.sum(ret\_ind\*\*2)), axis=1)

Minfo = np.concatenate((a, b), axis=0)

SLMchi2 = np.dot(np.dot(np.transpose(score), np.linalg.inv(Minfo)), score)

pvalue\_SLMchi2 = 1-chi2.cdf(SLMchi2[0][0], N)

print(pvalue\_SLMchi2)

3.中国A股市场存在惯性效应/反转效应实验代码

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** statsmodels.api **as** sm

**from** scipy.stats **import** chi2, f

**import** warnings

warnings**.**filterwarnings("ignore")

stock\_data1 **=** pd**.**read\_excel('./stockData\_2001\_2022/stockData\_2001\_2022/RESSET\_MRESSTK\_1.xls',

usecols**=**[0, 1, 2])

stock\_data2 **=** pd**.**read\_excel('./stockData\_2001\_2022/stockData\_2001\_2022/RESSET\_MRESSTK\_2.xls',

usecols**=**[0, 1, 2])

stock\_data3 **=** pd**.**read\_excel('./stockData\_2001\_2022/stockData\_2001\_2022/RESSET\_MRESSTK\_3.xls',

usecols**=**[0, 1, 2])

stock\_data4 **=** pd**.**read\_excel('./stockData\_2001\_2022/stockData\_2001\_2022/RESSET\_MRESSTK\_4.xls',

usecols**=**[0, 1, 2])

stock\_data5 **=** pd**.**read\_excel('./stockData\_2001\_2022/stockData\_2001\_2022/RESSET\_MRESSTK\_5.xls',

usecols**=**[0, 1, 2])

*# 将所有股票数据放入stock\_data中*

stock\_data **=** stock\_data1**.**append(stock\_data2, ignore\_index**=True**)

stock\_data **=** stock\_data**.**append(stock\_data3, ignore\_index**=True**)

stock\_data **=** stock\_data**.**append(stock\_data4, ignore\_index**=True**)

stock\_data **=** stock\_data**.**append(stock\_data5, ignore\_index**=True**)

stock\_data**.**columns **=** ['stock\_code','date','close']

stock\_data['date'] **=** pd**.**to\_datetime(stock\_data['date'])

stock\_data['date'] **=** stock\_data['date']**.**dt**.**strftime('%Y-%m')

*# 提取各个股票的代码*

stk\_codes **=** stock\_data['stock\_code']**.**unique()

*# 创建一个空的数据框存储所有股票的收益率数据*

all\_returns **=** pd**.**DataFrame()

*# 循环处理每支股票的数据*

**for** code **in** stk\_codes:

*# 提取每支股票的数据*

stock\_data\_code **=** stock\_data[stock\_data['stock\_code'] **==** code]**.**copy()

*# 按日期排序*

stock\_data\_code**.**sort\_values(by**=**'date', inplace**=True**)

*# 计算收益率*

stock\_data\_code['return'] **=** np**.**log(stock\_data\_code['close']) **-** np**.**log(stock\_data\_code['close']**.**shift(periods**=**1))

*# 将每支股票的收益率数据存储到all\_returns数据框中*

all\_returns **=** pd**.**concat([all\_returns, stock\_data\_code[['stock\_code', 'date', 'return']]], ignore\_index**=True**)

*# 去除收益率中的NAN值*

all\_returns**.**dropna(inplace**=True**)

*# 先根据过去N个月，计算累计收益率*

N\_values **=** [1, 3, 6, 12] *# 过去N个月的累积收益率*

*# 计算每个月的累积收益率并构建投资组合*

**for** N **in** N\_values:

*# 计算每个月的累积收益率*

all\_returns['cumulative\_return\_' **+** str(N)] **=** all\_returns**.**groupby('stock\_code')['return']**.**rolling(window**=**N, min\_periods**=**N)**.**sum()**.**reset\_index(0, drop**=True**)

*# 分装一个函数，传入的参数为sample\_month = n #根据过去n个月的累计收益进行分组*

**def** exp\_test(sample\_month,all\_returns):

**if**(sample\_month **!=** 1):

all\_returns **=** all\_returns**.**dropna()

M\_values **=** [1, 3, 6, 12] *# 持有期M*

*# 创建一个空的数据框存储所有月份的结果---根据过去1个月表现的横截面组*

results\_One\_month **=** pd**.**DataFrame()

*# 循环处理每个月的数据*

**for** date, group **in** all\_returns**.**groupby('date'):

*# 根据过去一个月的累积收益率排序并分组*

*#根据每个月的累积收益率将股票分成5组，并为每支股票赋予一个组合编号，存储在名为'portfolio'的新列中*

group['portfolio'] **=** pd**.**qcut(group['cumulative\_return\_' **+** str(sample\_month)], q**=**5, labels**=False**) **+** 1

*# 计算未来1、3、6、12个月的累积收益率*

**for** M **in** M\_values:

*# 找到未来M个月的数据*

future\_data **=** all\_returns[all\_returns['date'] **>** date]

future\_data['date'] **=** pd**.**to\_datetime(future\_data['date'])

future\_data **=** future\_data[future\_data['date'] **<=** pd**.**to\_datetime(date) **+** pd**.**DateOffset(months**=**M)] *#筛选出未来数据中日期小于或等于未来M个月的日期的所有数据，这样就得到了未来M个月的数据。*

*# 按股票代码分组，计算未来M个月的累积收益率*

future\_returns **=** future\_data**.**groupby('stock\_code')['return']**.**sum()

group['future\_return\_' **+** str(M)] **=** group['stock\_code']**.**map(future\_returns)

*# 将当月结果添加到总结果中*

results\_One\_month **=** pd**.**concat([results\_One\_month, group], ignore\_index**=True**)

results\_One\_month**.**dropna(inplace**=True**)

*# 假设你已经有了每个月的数据，存储在名为'monthly\_returns'的数据框中，包括每个月的五个分组的1、3、6、12个月的累积收益率*

monthly\_returns **=** results\_One\_month**.**loc[:, ['date', 'future\_return\_1', 'future\_return\_3', 'future\_return\_6', 'future\_return\_12','portfolio']]

*# 计算每个月每个分组的未来1、3、6、12个月的平均累积收益率*

average\_returns **=** monthly\_returns**.**groupby(['date', 'portfolio'])**.**mean()

*# 对结果按照分组进行汇总*

grouped\_returns **=** average\_returns**.**groupby('portfolio')**.**sum()

*# 计算每个分组的平均累积收益率*

num\_months **=** len(monthly\_returns**.**groupby('date')) *#一共有多少个月*

final\_returns **=** grouped\_returns **/** num\_months *#每个分组的最终n个月的平均累积收益率*

**return** final\_returns

# 调用函数获取计算结果

final\_returns\_1 **=** exp\_test(1,all\_returns)

final\_returns\_1

final\_returns\_3 **=** exp\_test(3,all\_returns)

final\_returns\_3

final\_returns\_6 **=** exp\_test(6,all\_returns)

final\_returns\_6

final\_returns\_12 **=** exp\_test(12,all\_returns)

final\_returns\_12