华东理工大学 2021 –2022 学年第 2 学期

《Python 与金融计算》实验报告

实验名称　中国股市 FF 三因子资产定价模型的实证检验

专　业　计算机金融双学位　姓名　周学勤　学号　20002520　班级　计金（双）200

实验时间　2023/04/06　实验地点　信息楼 319　指导教师　蒋志强

实验目的/要求
1、 掌握三因子模型参数估计方法和时间序列检验方法（单资产和多资产） 2、 掌握三因子模型的横截面检验方法（排序法和 Fama-Macbeth 回归）
实验内容
1. 认真阅读三篇文献资料（自己也可以下载相关文献进行阅读），了解资产定价模型的相关研究，重点关注其中的研究思路和方法。 2. 任意选取七个行业指数，从锐思数据库或 tushare 数据库，下载这些指数的周度数据和三因子的周度数据，估计 FF 三因子模型参数，并逐个行业检验其是否满足 FF 三因子模型；再对七个行业进行 FF 三因子模型的多资产检验。（周度数据不少 10 年，周度数据不少于 5 年） 3. 从锐思数据库或 tushare 数据库，提取中国股市 2000-2021 年所有股票的月收益率，月流通市值数据，月市盈率数据，观察和研究课本 P52 页表格 3-6 的计算方法，计算先按市值再按盈利价格比（市盈率倒数）序贯排序的资产组合月平均收益率（注意：变量排序分成 5 组，不需要分成 10 组），并对结果进行讨论。 4. 从锐思数据库或 tushare 数据库，提取中国股市 2000-2021 年，所有股票的月收益率，月流通市值数据，月市盈率数据，月 CAPM 风险因子 Beta 数据，请利用 Fama-MacBech 回归检验市值、盈利价格比（市盈率倒数）、CAPM 风险因子 Beta 对收益率的解释性。

实验总结

请提供对本次实验结果的讨论分析，以及实验的心得和体会。包括对知识点的掌握，算法的理解，以及对理论课程和实验课程改进的建议。（不少于 500 字）

第二小题主要涉及多元线性回归和多资产检验的知识点，通过使用 Python 编程实现了多元线性回归模型和三因子模型，并对回归结果和多资产检验结果进行了分析。

在实验过程中，我学习了如何使用 Python 进行多元线性回归和三因子模型的计算，并对回归结果和多资产检验结果进行了解释和分析。通过实验，我深刻认识到了多元线性回归和三因子模型在金融领域的应用，也更加熟练了解释回归结果的方法。

第三小题主要是关于股票投资组合的序贯排序算法的实现，以及在不同市值组和 EP 值范围内的平均收益率的分析。通过实现代码和对结果的分析，我对序贯排序算法有了更深入的理解，并且学会了如何使用 Python 进行股票投资组合的实现和分析。

在实验过程中，我深刻认识到了数据处理的重要性。股票投资涉及到大量的数据，如果数据处理不当，可能会影响到结果的准确性和可靠性。因此，在实验中，我特别注重数据的清洗和预处理，以保证结果的准确性和可靠性。

另外，我还发现在进行股票投资组合分析时，市值和估值是两个非常重要的指标。在本次实验中，我们使用了市值和估值作为分类标准，分析了不同市值组和 EP 值范围内的平均收益率。这些指标对于股票投资组合的选择和决策非常重要，对投资组合的风险和收益都会产生重要的影响。

第四小题中，根据回归分析的结果，市值（size）和 CAPM 风险因子 Beta（beta）对收益率具有统计显著性，而市盈率的倒数（ep）和常数项（const）则不具有统计意义。这说明在给定的时间范围内，市值和 Beta 是影响股票收益率的重要因素，而市盈率和常数项则不是。

我对 Fama-MacBech 回归分析的方法和步骤有了更深入的了解，并学会了如何使用 Python 对数据进行导入、处理、分析和可视化。特别是在回归分析方面，我学会了如何使用 Python 中的 statsmodels 模块进行回归分析，并对回归系数进行 t 检验。这些知识和技能可以帮助我更好地理解和分析金融数据，以及制定更有效的投资策略。

为了进一步提高实验课程的质量和效果，我建议在实验中添加更多的练习和案例，以帮助学生更好地掌握相关知识和技能。此外，我认为实验教师应该提供更多的支持和指导，以便学生更好地理解和应用所学知识。

实验中遇到的问题及解决法：

1、

Usecols[0,1,3]而不是 usecols[‘0,1,3’]

2、

indexcode=np.unique(index['code'].values)

indexcode

和 indexcode=np.unique(index['code'])

Indexcode

两者效果是一样的

3、

Usecols[1,2]错　　应该是 usecols=[1,2]

4、

.loc 和.iloc 的区别

iloc 使用顺序数字来索引数据，而不能使用字符型的标签来索引数据；注意：这里的顺序数字是指从 0 开始计数！

loc 使用实际设置的索引来索引数据。但行列名为数字时，loc 也可以索引数字，但这里的数字不一定从 0 开始编号，是对应具体行列名的数字！

并且需要注意的是，loc 是前闭后闭，iloc 是前闭后开。

x = data_matrix.loc[0:2, 'date'].values 取 0，1，2 行数据

x = data_matrix.iloc[0:2, 1:5].values 取 0，1 行；1，2，3，4 列数据

5、

```
# 多资产检验
T = len(Y)
N = 10
K = 3
y = data_matrix.iloc[:, 5:15].values - data_matrix.loc[:, ['rfreturn']].values
# x = sm.add_constant(x)
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]
ALPHA
```

这里需要使用 X，不然少一列的。

对于 numpy.array，假设 matrix（N*4）为 numpy.array。那么，matrix[0]是取其第一行，输出会变为 4*1（）4rows*1columns 向量

sm.add_constant(x)会直接给 x 加上一列全为 1 的列向量。同时也会有一个返回值，就是更改后的 x

所以才会有这个 X = sm.add_constant(x)

老师给的代码中，在计算矩阵乘积时，直接先计算了最终结果的转置矩阵，然后通过 AB_hat[0]直接取第一行，也就是原来应该取的第一列，也完成了功能。

np.transpose（x）

np.linalg.inv(x)不会直接改变 x

6、

注意`data_matrix.loc[:,['rfreturn']]`和`data_matrix.loc[:,'rfreturn']`是不同的。不过对于操作一列的时候结果都是一样的。

7、

`data_matrix.loc[:, ['rfreturn']].values`：

在这个语句中，我使用一个包含单个元素的列表（**`['rfreturn']`**）来选取列。因此，返回的是一个二维 numpy 数组，数组的形状为 (n, 1)，其中 n 是 'rfreturn' 列的行数。这意味着在返回的数组中，每个元素都是一个包含单个值的子数组。

8、

总之，两个语句的主要区别在于返回的 numpy 数组的形状。第一个语句返回一个二维数组，形状为 (n, 1)；第二个语句返回一个一维数组，形状为 (n,)。

9、

data.dropna(inplace = True, subset=['stksize', 'stkep'])

data

##直接全部 drop 不是更好？

究竟需不需要全部 drop？

不需要全部 drop，因为其他列中的某行可能存在 na，但这行中需要用的列中却没有，这样可以用的数据就会被删掉。

10、

uym = np.unique(data['yearmonth'].values)##有没有可能有一些股票没有完整的年月。这没什么关系，因为后面是根据每个时点作为一组，在其中进行排序划分

print(len(uym))

uym

虽然有可能不同时间节点中股票种类不一样多，但是为了对每个节点进行横截面回归，就算不一样多也没什么问题。

11、

将 csv 中的 date 列读进来就是 str 类型

12、

```
dm = pd.merge(left = dm,
                                    right = self.data.loc[ind, ['code', 'monret']],
                                    on='code',
                                    how='left',
                                    sort=True)
```

这里使用 how=’left’，就是防止有些股票的数据在某些月份没有而被删除。在某些月份没有的情况下同样可以算平均值。这里和下面的 def sort_single_ind(self):有很大的

关联。如果这里使用了 inner 那么一些在 month[0]以外的月份不存在的股票就会被删除，导致接下来按照 month[0]获得的 ssi 会匹配不上数据量

13、

for i in range(5, 258, 12):

这里之所以用 264-6=258，是因为 range 前闭后开，258 不会被去到，也就不会用 258 及其后面的年月

14、

```
a=[1,2,3]
b=[1,2,3]
l=zip(a,b)
l
for p in l:##这里迭代的是一个个元组，p 是元组。
    print(p)
for m,p in l:
    print(m,p)##为什么这里返回的是元组，但 m 和 p 也能赋值成功呢？看下面。
Tuple=(1,5)
```

l,t=Tuple 就能把元组中的东西赋值出来

zip()打包

15、

df = DF.loc[:, reg].copy() loc 中 reg 的地方不能是作为索引的列

在 copy 时，索引也会一起 copy 过去。不过直接赋值也是一样的。

16、

df 是 dataframe 对象

T = df.index 是获取 df 中的索引。

```
DF=pd.DataFrame({'stk':[8,9],'month':[5,6],'size':[1,2],'share':[3,4]})
DF=DF.set_index('stk')
DF=DF.set_index('month',append=True)
DF.index
```

输出结果：

```
MultiIndex([(8, 5),
            (9, 6)],
           names=['stk', 'month'])
```

df.index.names[1] 这里的 names 是包含所有索引名称的 list

对于 df：

stk	month	share
8	5	3
9	6	4

T = df.index.get_level_values(df.index.names[1])

获得的是:

```
Int64Index([5, 6], dtype='int64', name='month')
```

[]里面是 df.index.names[1]这个索引中的所有取值

17、

对于 m={k: pd.DataFrame(v) for k, v in res_ts.items()}。m 是一个字典，值的类型为 dataframe。这种样式的字典可以使用 pd.concat(m)来进行改造成一个 dataframe，其中键就变成了索引。

18、

~~当对带有索引的 dataframe 使用 dropna 时，存在 na 的那一个索引范围都会被删除~~

删除的是行！而不是索引对应的那些。dropna 的返回值为 none

19、

Dataframe 中的不算在数据中的行标号也是一个索引，所以可以自行加入一个索引后进行 temp = ret.set_index('code',append=True).swaplevel()

Temp

20、

Resetindex()可以将原来的索引都变为数据列

21、

对于有索引的 dataframe，比如 A，b=A[‘’]选取的列中不能加索引的名称

教师批阅：　　实验成绩：

教师签名:　　日期：

实验报告正文：

（每次实验报告均为一篇小论文，因此，统一按照学术论文的要求完成实验报告正文，应包括：题目、摘要、文献综述、模型和方法、结果和讨论、参考文献、附录，具体格式如下：

中国股市收益率的分布特征

计金（双）200（20002520）周学勤

摘要：本文综述了关于 Fama-French（FF）三因子模型的参数估计方法、时间序列检验方法（单资产和多资产）以及横截面检验方法（排序法和 Fama-Macbeth 回归）。参数估计方法主要包括最小二乘法（OLS）和广义矩方法（GMM）。时间序列检验方法通过单资产和多资产回归分析对模型进行检验。横截面检验方法包括排序法和 Fama-Macbeth 回归，这些方法用于评估市值、市盈率等因子对股票收益率的解释能力。此外，文献综述还涉及了模型稳定性检验、跨国研究以及在 FF 三因子模型基础上引入其他因子的研究。这些研究为理解和应用 FF 三因子模型提供了丰富的理论和实证依据。

1 文献综述

Fama 和 French（1993）[1]在提出三因子模型时，采用了最小二乘法（OLS）对模型进行参数估计。他们利用股票收益率与三个因子（市场风险、规模风险和价值风险）之间的线性关系来估计模型参数。此后，许多研究者沿用了这种参数估计方法，并在其他市场进行了实证检验。

单资产和多资产的时间序列检验通常采用回归分析。在单资产检验中，将单个股票或指数的收益率作为因变量，三因子模型的因子作为自变量进行回归分析。而在多资产检验中，将多个股票或指数的收益率作为因变量，同时考虑三因子模型的因子作为自变量进行多元回归分析。

横截面检验方法主要包括排序法和 Fama-Macbeth 回归。排序法是将股票按照某种标准（如市值、市盈率等）进行排序，并将其分为若干组，进而计算各组的平均收益率。Fama 和 French（1993）[2]在检验三因子模型时，就采用了此种方法。Fama-Macbeth 回归（Fama & MacBeth, 1973）[1]是另一种常用的横截面检验方法，通过对每个时期的横截面数据进行回归分析，计算各时期回归系数的均值和标准误差，进而检验因子对收益率的解释能力。

除了最小二乘法（OLS）外，广义矩方法（GMM）也被用于 FF 三因子模型参数估计。Cochrane（2005）[3]在其著作《资产定价》中详细讨论了 GMM 在资产定价模型中

的应用。GMM 是一种灵活的估计方法，适用于许多线性和非线性模型。

在研究 FF 三因子模型时，检验模型稳定性是非常重要的。研究者可以通过滚动窗口回归、结构变点检验等方法来检验模型在不同时间段的稳定性。例如，Goyal 和 Welch（2008）[4]通过滚动窗口回归检验了美国股市中 FF 三因子模型的稳定性。

FF 三因子模型在跨国研究中得到了广泛应用。Griffin（2002）[5]利用 25 个国家的数据，通过横截面回归分析了 FF 三因子模型在全球范围内的适用性。他发现，FF 三因子模型在大多数国家中具有较好的解释力，但在某些国家中表现较差。

随着资产定价理论的发展，许多研究者开始考虑将其他因子纳入 FF 三因子模型。例如，Carhart（1997）[6]在 FF 三因子模型的基础上引入了动量因子，形成了四因子模型。Fama 和 French（2015）[7]则进一步引入了盈利因子和投资因子，形成了五因子模型。

2 模型和方法

具体介绍使用的金融理论模型、计算方法和 MATLAB 函数。

2.1 第二小题

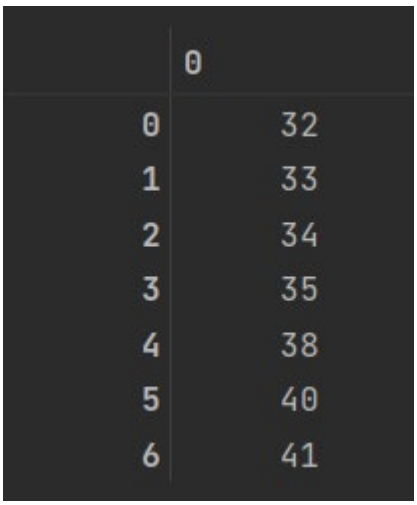
首先从锐思数据库中下载所需要的数据。使用 pd.read_csv 读取对应的 excel 数据文件。数据文件格式示例如下：

表 2.1　数据文件格式

指数代码 _IdxCd	交易日期 _TrdDt	收盘价 _ClPr	指数周收益率 _IdxWkRet
32	2010/01/08	3485.8	-0.0137
32	2010/01/15	3400.01	-0.0246
32	2010/01/22	3168	-0.0682
32	2010/01/29	3013.19	-0.0489
32	2010/02/05	2918.92	-0.0313
32	2010/02/12	3036.13	0.0402
32	2010/02/26	3012.64	-0.0077

代码中设置了参数 usecols 为[0,1,3]，表示只读取第 0、1、3 列的数据。然后通过设置列名的方式将三列数据分别命名为'code'、'date'和'return'。接下来，将日期列转换为 pandas 的日期格式，并返回 DataFrame 对象。

接下来使用 numpy 库中的 unique 函数，对指数代码进行去重操作。先使用 index['code']获取数据中所有的指数代码，再使用 np.unique 函数对其进行去重并按升序排序。最后将去重后的指数代码作为一个 numpy 数组返回。如下所示：



	0
0	32
1	33
2	34
3	35
4	38
5	40
6	41

图 2-1 结果图 1

之后从 DataFrame 对象 index 中筛选出指数代码为 indexcode[0]的行，并将其赋值给一个新的 DataFrame 对象 index32。然后将 index32 的列名修改为'code'、'date'和'return32'，其中'return32'表示指数收益率。

在这之后，从 DataFrame 对象 index 中筛选出指数代码为 indexcode[i]的行，并将其赋值给一个新的 DataFrame 对象 indexi，其中 i 为指数代码在 indexcode 中的索引。然后将 indexi 的列名修改为'code'、'date'和'returni'，其中'returni'表示指数收益率。数据返回示例如下：

	code	date	return34
360	33	2009-01-23	NaN
361	33	2009-02-27	0.0947
362	33	2009-03-31	0.2219
363	33	2009-04-30	0.0102
364	33	2009-05-27	0.0456
...	...	...	...
463	33	2017-08-31	0.0432
464	33	2017-09-29	-0.0042

图 2-2 结果图 2

在这之后，读取名为'rfreturn.csv'的数据文件，该文件路径为当前工作目录。该文件包含两列数据，分别是日期和无风险利率收益率。使用了 pandas 库的 read_csv 函数来读取数据，并设置参数 usecols 为[0,2]，表示只读取第 0、2 列的数据。然后通过设置列名的方式将两列数据分别命名为'date'和'rf'，其中'rf'表示无风险利率收益率。然后将 rf 的列名修改为'date'和'rfreturn'，其中'rfreturn'表示无风险利率收益率。然后将日期列转换为 pandas 的日期格式，并返回 DataFrame 对象。示例如下所示：

	date	rfreturn
0	2010-01-01	0.000352
1	2010-01-08	0.000352
2	2010-01-15	0.000356
3	2010-01-22	0.000361
4	2010-01-29	0.000365
...	...	...
674	2022-12-02	0.000422
675	2022-12-09	0.000429

图 2-3 结果图 3

接下来，读取名为'Data_FFFactors.csv'的数据文件，该文件路径为当前工作目录。该文件包含四列数据，分别是日期、市场风险溢价、规模因子溢价和价值因子溢价。代码中使用了 pandas 库的 read_csv 函数来读取数据，并设置参数 usecols 为[2,6,7,8]，表示只读取第 2、6、7、8 列的数据。然后通过设置列名的方式将四列数据分别命名为'date'、'mkt'、'smb'和'hml'，其中'mkt'表示市场风险溢价，'smb'表示规模因子溢价，'hml'表示价值因子溢价。接下来，将日期列转换为 pandas 的日期格式，并返回 DataFrame 对象 data_factors。示例如下所示：

	date	mkt	smb	hml
0	2010-01-08	-0.0256	0.0270	-0.0021
1	2010-01-15	0.0082	0.0362	-0.0087
2	2010-01-22	-0.0302	0.0068	-0.0137
3	2010-01-29	-0.0449	0.0040	-0.0178
4	2010-02-05	-0.0173	0.0193	0.0024
5	2010-02-12	0.0265	-0.0131	0.0005
6	2010-02-26	0.0106	0.0422	-0.0011
7	2010-03-05	-0.0071	0.0046	0.0035

图 2-4 结果图 4

接下来使用 pandas 库的 merge 函数将 data_factors 和 rf 两个 DataFrame 对象按照日期列进行内连接，将它们合并成一个新的 DataFrame 对象 data_matrix。通过这种方法将市场风险溢价、规模因子溢价、价值因子溢价和无风险利率收益率的数据按照日期合并在一起，返回一个新的 DataFrame 对象 data_matrix。示例如下所示：

	date	mkt	smb	hml	rfreturn
0	2010-01-08	-0.0256	0.0270	-0.0021	0.000352
1	2010-01-15	0.0082	0.0362	-0.0087	0.000356
2	2010-01-22	-0.0302	0.0068	-0.0137	0.000361
3	2010-01-29	-0.0449	0.0040	-0.0178	0.000365
4	2010-02-05	-0.0173	0.0193	0.0024	0.000367
5	2010-02-12	0.0265	-0.0131	0.0005	0.000369
6	2010-02-26	0.0106	0.0422	-0.0011	0.000373
7	2010-03-05	-0.0071	0.0046	0.0035	0.000374

图 2-5 结果图 5

在这之后，使用 pandas 库的 merge 函数将 data_matrix 和 index32 两个 DataFrame 对象按照日期列进行内连接，将它们合并成一个新的 DataFrame 对象 data_matrix。这样子就成功将市场风险溢价、规模因子溢价、价值因子溢价、无风险利率收益率和指数收益率的数据按照日期合并在一起，返回一个新的 DataFrame 对象 data_matrix。

	date	mkt	smb	hml	rfreturn	return32
0	2010-01-08	-0.0256	0.0270	-0.0021	0.000352	-0.0137
1	2010-01-15	0.0082	0.0362	-0.0087	0.000356	-0.0246
2	2010-01-22	-0.0302	0.0068	-0.0137	0.000361	-0.0682
3	2010-01-29	-0.0449	0.0040	-0.0178	0.000365	-0.0489
4	2010-02-05	-0.0173	0.0193	0.0024	0.000367	-0.0313
5	2010-02-12	0.0265	-0.0131	0.0005	0.000369	0.0402
6	2010-02-26	0.0106	0.0422	-0.0011	0.000373	-0.0077
7	2010-03-05	-0.0071	0.0046	0.0035	0.000374	-0.0156

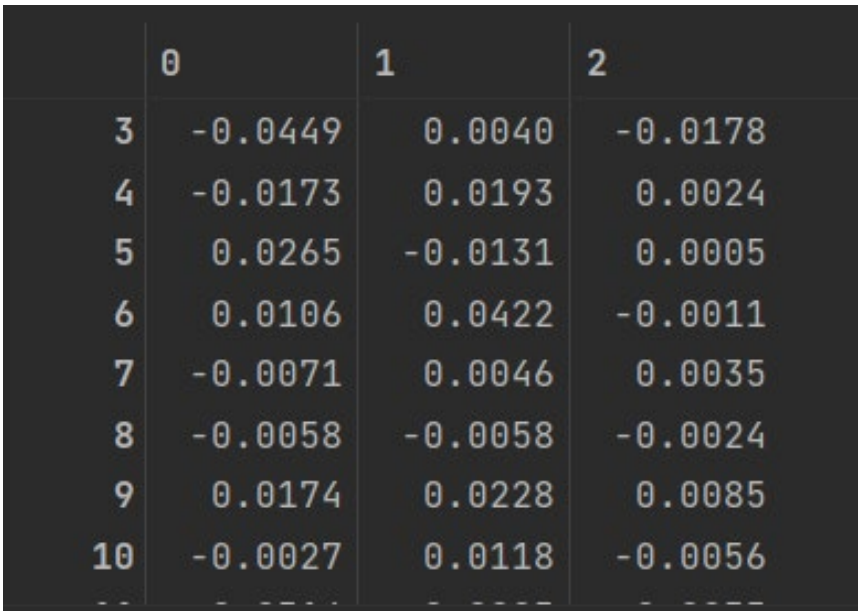
图 2-6 结果图 6

接下来将另外六个指数的收益率数据分别与 data_matrix 进行内连接，将它们合并成一个新的 DataFrame 对象 data_matrix。将市场风险溢价、规模因子溢价、价值因子溢价、无风险利率收益率和七个指数的收益率数据按照日期合并在一起，返回一个新的 DataFrame 对象 data_matrix。示例如下所示：

	date	mkt	smb	hml	rfreturn	return32	return33	return34	return35	return38	return40	return41
0	2010-01-29	-0.0899	0.0693	-0.0438	0.000365	-0.0878	-0.1474	-0.1343	-0.0462	-0.0458	-0.0241	0.0182
1	2010-02-26	0.0191	0.0477	0.0012	0.000373	0.0210	-0.0002	0.0296	0.0215	0.0609	0.0206	0.0446
2	2010-04-30	-0.0778	0.0067	-0.0429	0.000373	-0.0767	-0.0606	-0.0689	-0.0774	-0.0901	-0.0430	0.0877
3	2010-07-30	0.1032	0.0569	-0.0263	0.000476	0.0997	0.1425	0.1392	0.1179	0.1459	0.1129	0.0886
4	2010-10-29	0.1203	-0.0540	-0.0097	0.000541	0.1217	0.2881	0.1937	0.1407	0.0844	0.0304	0.0091
5	2010-12-31	-0.0075	-0.0282	0.0243	0.000879	-0.0043	0.0616	0.0228	0.0182	-0.0579	-0.0748	-0.0770
6	2011-04-29	-0.0087	-0.0085	0.0155	0.000866	-0.0057	-0.0097	-0.0336	-0.0171	-0.0344	-0.0303	-0.0052
7	2011-07-29	-0.0245	0.0196	-0.0462	0.001155	-0.0218	-0.0082	-0.0120	-0.0567	-0.0106	0.0685	0.0814
8	2011-09-30	-0.0837	-0.0286	0.0482	0.001084	-0.0811	-0.0410	-0.1351	-0.0930	-0.0961	-0.1180	-0.1462
9	2011-12-30	-0.0619	-0.0841	0.0728	0.001052	-0.0574	-0.0693	-0.1394	-0.0938	-0.0888	-0.1675	-0.1227
10	2012-03-30	-0.0724	0.0035	0.0060	0.000947	-0.0682	-0.0866	-0.0809	-0.0826	-0.0858	-0.0448	-0.0609
11	2012-04-27	0.0554	0.0426	0.0053	0.000909	0.0590	0.0725	0.0647	0.0719	0.0681	0.0647	0.0245
12	2012-06-29	-0.0512	-0.0145	0.0020	0.000786	-0.0619	-0.1380	-0.1147	-0.0895	-0.0898	-0.0285	0.0698
13	2012-08-31	-0.0285	0.0567	-0.0087	0.000699	-0.0267	-0.0576	-0.0376	-0.0644	-0.0367	-0.0182	0.0091
14	2012-09-28	0.0186	-0.0373	-0.0222	0.000708	0.0189	0.0595	0.0898	0.0229	0.0454	0.0330	0.0206
15	2012-11-30	-0.0461	-0.0464	0.0455	0.000734	-0.0429	-0.0798	-0.0465	-0.0395	-0.0754	-0.1192	-0.1072
16	2013-03-29	-0.0580	0.0091	-0.0019	0.000746	-0.0545	-0.0658	-0.0777	-0.0820	-0.0577	-0.0129	0.0361
17	2013-04-26	-0.0293	-0.0007	0.0010	0.000747	-0.0262	-0.0712	-0.0498	-0.0303	-0.0013	-0.0478	-0.0208

图 2-7 结果图 7

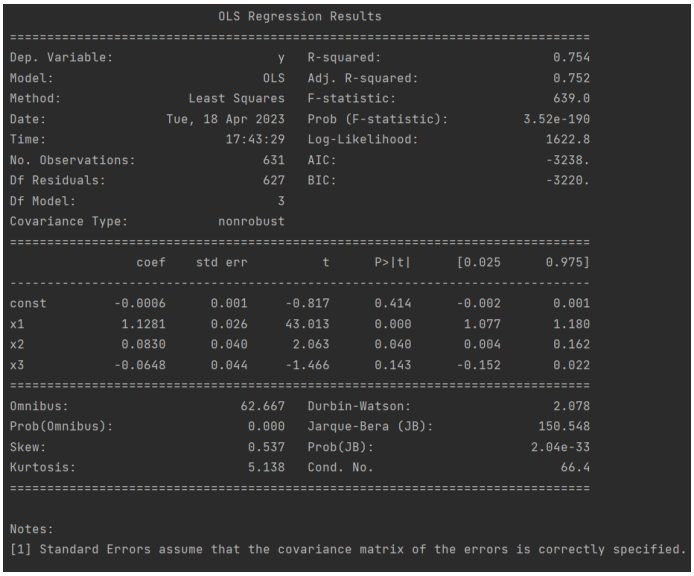
在这之后，从 data_matrix 中分别提取市场风险溢价、规模因子溢价和价值因子溢价，组成一个 3 列的 numpy 数组 x。另外，从 data_matrix 中提取无风险利率收益率和指数收益率，分别组成一个 numpy 数组 ret_rf 和一个 numpy 数组 ret_stock。最后，返回 numpy 数组 x，即市场风险溢价、规模因子溢价和价值因子溢价的数据。示例如下所示：



	0	1	2
3	-0.0449	0.0040	-0.0178
4	-0.0173	0.0193	0.0024
5	0.0265	-0.0131	0.0005
6	0.0106	0.0422	-0.0011
7	-0.0071	0.0046	0.0035
8	-0.0058	-0.0058	-0.0024
9	0.0174	0.0228	0.0085
10	-0.0027	0.0118	-0.0056

图 2-8 结果图 8

接下来，使用 statsmodels 库进行线性回归分析。首先使用 sm.add_constant 函数将 numpy 数组 x 增加一列全为 1 的列向量，作为回归模型中的截距项。然后计算因变量 Y，即指数收益率减去无风险利率收益率的值。接着，使用 sm.OLS 函数构建回归模型，其中参数 Y 表示因变量，X 表示自变量。最后，使用 fit 函数对回归模型进行拟合，并将结果保存在 results 对象中。最终使用 results.summary()函数打印回归分析的结果摘要，包括回归系数的估计值、t 值、p 值、R 方值等统计指标，以及模型的拟合优度等信息。示例如下所示：



OLS Regression Results

Dep. Variable:	y	R-squared:	0.754
Model:	OLS	Adj. R-squared:	0.752
Method:	Least Squares	F-statistic:	639.0
Date:	Tue, 18 Apr 2023	Prob (F-statistic):	3.52e-190
Time:	17:43:29	Log-Likelihood:	1622.8
No. Observations:	631	AIC:	-3238.
Df Residuals:	627	BIC:	-3220.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0006	0.001	-0.817	0.414	-0.002	0.001
x1	1.1281	0.026	43.013	0.000	1.077	1.180
x2	0.0830	0.040	2.063	0.040	0.004	0.162
x3	-0.0648	0.044	-1.466	0.143	-0.152	0.022

Omnibus:	62.667	Durbin-Watson:	2.078
Prob(Omnibus):	0.000	Jarque-Bera (JB):	150.548
Skew:	0.537	Prob(JB):	2.04e-33
Kurtosis:	5.138	Cond. No.	66.4

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

图 2-9 结果图 9

接下来使用类似方法对所有的指数进行分析。

最后通过代码实现了三因子模型的多资产检验，具体来说，通过计算估计出来的 alpha 向量、资产收益率数据以及三因子模型的因子数据，分别计算出 OLS 估计残差以及残差的协方差矩阵，进而通过计算出 GRS 统计量以及其对应的 p 值，进行多资产的检验。在具体实现上，代码首先计算出 OLS 估计残差 RESD，然后计算出残差的协方差矩阵 COV，并对其求逆 invCOV。接着计算三因子模型的因子数据 fs，并通过去除均值的方式，将其中心化为 0，计算出协方差矩阵 omegahat 以及其逆 invOMG。然后代码计算出 xxx 和 yyy，最后根据 GRS 的公式计算出 GRS 统计量和其对应的 p 值。示例如下所示：

```
三因子模型的多资产检验结果
 alpha1, alpha2, alpha3, alpha4, alpha5, alpha6, alpha7,    GRS, pvalue
-0.0011,-0.0006,-0.0003,-0.0004, 0.0001,-0.0007,-0.0007, 0.6544, 0.7108
```

图 2-10 结果图 10

2.2 第三小题

首先，读取第一个 CSV 文件，将需要的列存入数据框 data 中，并更改列名。然后使用 for 循环遍历其他 CSV 文件，读取需要的列存入数据框 trans 中，并更改列名。接下来将 trans 数据框追加到 data 数据框中，以使它们合并到一个数据框中。最后，将合并后的数据框 data 作为输出返回。

可以看出，主要依赖于 pandas 库中的 read_csv()和 concat()函数实现 CSV 文件的读取和数据框的合并。它使用了循环和条件语句来控制程序的流程，以实现数据的批量处理。

	code	date	close	tshare	monret	pe
0	1	2000-01-28	18.53	1.071634e+09	0.0619	51.79
1	1	2000-02-29	18.32	1.071634e+09	-0.0113	51.20
2	1	2000-03-31	18.37	1.071634e+09	0.0027	51.34
3	1	2000-04-28	19.05	1.071634e+09	0.0370	56.44
4	1	2000-05-31	18.00	1.071634e+09	-0.0551	53.33
5	1	2000-06-30	18.13	1.071634e+09	0.0072	53.71
6	1	2000-07-31	18.51	1.071634e+09	0.0210	58.34
7	1	2000-08-31	17.75	1.071634e+09	-0.0411	55.94

图 2-11 结果图 1

接下来对数据框 data 中的列进行处理和转换，包括日期格式转换、市值和市盈率倒数的计算，并删除包含缺失值的行，以保证数据的完整性和准确性。这里使用 pandas 库中的日期时间函数和 dropna()函数实现数据处理和清理。示例如下所示：

	code	date	close	tshare	monret	pe	yearmonth	tstksize	stkep
0	1	2000-01-28	18.53	1.071634e+09	0.0619	51.79	200001	1.985739e+10	0.019309
1	1	2000-02-29	18.32	1.071634e+09	-0.0113	51.20	200002	1.963234e+10	0.019531
2	1	2000-03-31	18.37	1.071634e+09	0.0027	51.34	200003	1.968592e+10	0.019478
3	1	2000-04-28	19.05	1.071634e+09	0.0370	56.44	200004	2.041464e+10	0.017718
4	1	2000-05-31	18.00	1.071634e+09	-0.0551	53.33	200005	1.928942e+10	0.018751
...	...	...	...	...	...	...	...	...	...
92793	990018	2006-05-29	15.47	1.804400e+09	0.4365	52.96	200605	2.791407e+10	0.018882
92794	990018	2006-06-30	17.15	1.804400e+09	0.1086	58.71	200606	3.094546e+10	0.017033

图 2-12 结果图 2

然后查找数据框中不同的年月，并输出它们的数量和值。它使用了 numpy 库中的 unique()函数来查找不同的年月值。示例如下所示：

```
264
[2, 3]
```

图 2-13 结果图 3

接下来创建一个类包含了四个主要的函数和一个初始化函数。这个类的主要目的是实现对给定数据进行序贯排序的算法，以实现股票组合的构建。

```
class sort_portfolio:

    def __init__(self, data, months, gnum):...

    def data_months(self):...
    def sort_single_ind(self):...

    def sort_double_ind(self):...

    def sequence_sort(self):...


    def sequence_sort_mreturn(self):...
```

图 2-14 结果图 4

sort_portfolio 类：这个类主要是用来实现股票组合的序贯排序算法，包含了四个主要的函数和一个初始化函数。

__init__(self, data, months, gnum)：这个初始化函数接收三个参数，data 是待处理的数据框，months 是一个整数列表，表示要处理的年月，gnum 表示组的数量。

data_months(self)：这个函数用来处理数据，它首先提取数据框中指定年月的数据，然后将多个月份的数据进行合并，并返回一个新的数据框 dm。

sort_single_ind(self)：这个函数根据给定的股票数量和组数，将股票分配到不同的单个组中，并返回一个数组 ssi，表示每个股票所属的单个组的序号。

sort_double_ind(self)：这个函数根据给定的股票数量和组数，将股票分配到不同的双重组中，并返回一个数组 sdi，表示每个股票所属的双重组的序号。

sequence_sort(self)：这个函数将调用 data_months()、sort_single_ind()和 sort_double_ind()函数，将数据按照一定顺序进行排序，并返回一个新的数据框。

sequence_sort_mreturn(self)：这个函数将调用 sequence_sort()函数，计算每个组的平均收益率，并返回一个新的数据框。

综上所述，主要思路是实现一个基于序贯排序的算法，将股票组合构建到不同的组中，并计算每个组的平均收益率。

接下来实例化一个 sort_portfolio 类对象，并调用其 data_months()函数对指定的数据和年月进行处理，返回一个新的数据框 dm。同时，还输出了要处理的年月，以供参考。示例如下所示：

接下来实例化一个 sort_portfolio 类对象，并调用其 data_months()函数对指定的数据和年月进行处理，返回一个新的数据框 dm。同时，还输出了要处理的年月，以供参考。示例如下所示：

	stk	size6	ep6	ret7	ret8	ret9	ret10	ret11	ret12	retn1	retn2
3	5	2.547178e+09	0.013019	-0.0357	-0.0064	-0.1565	0.0592	0.0522	-0.0371	0.1177	-0.0
4	6	2.122505e+09	0.024839	0.0468	-0.1137	-0.1154	0.0429	0.0691	0.0810	0.0379	-0.0
...	...	...	...	...	...	...	...	...	...	...	
945	900950	2.544366e+07	-0.016466	0.1579	0.1273	0.1048	0.0730	0.0544	0.1677	-0.0387	-0.0
946	900951	1.500000e+07	-0.020859	0.3333	0.4000	-0.1714	0.2155	0.0071	0.3028	-0.0081	0.0
947	900952	4.173600e+07	0.065833	0.0372	-0.1436	-0.0299	0.0864	0.1023	0.3041	-0.0316	0.0
948	900953	5.376000e+07	0.055279	0.1518	-0.0233	0.0317	0.1000	0.0350	0.2230	-0.0608	0.0
949	900956	2.875000e+07	-0.012514	0.1280	-0.0780	-0.0231	0.1181	0.0423	0.2838	-0.0895	0.0

图 2-15 结果图 5

再接下来调用 sort_portfolio 类中的 sequence_sort()函数，对数据框进行排序，并返回一个新的数据框 dm。示例如下所示：

	stk	size6	ep6	ret7	ret8	ret9	ret10	ret11	ret12	retn1	retn2
915	900915	8.142000e+06	-0.990099	0.3051	0.3506	0.0192	0.0943	0.2759	0.2635	-0.0909	-0.047
925	900926	1.038400e+07	-0.719424	0.3898	0.6707	0.0438	-0.0280	0.2014	0.4910	-0.0542	0.027
907	900906	1.585584e+07	-0.469484	0.2424	0.3049	-0.0280	0.0865	0.2566	0.2535	-0.0787	-0.036
434	200025	4.752000e+07	-0.315457	0.0000	0.0944	-0.0761	-0.0055	0.0000	0.2707	-0.0174	-0.101
927	900928	1.992906e+07	-0.243902	0.1183	0.2500	0.0615	0.1232	0.0258	0.3396	-0.1127	0.023
...	...	...	...	...	...	...	...	...	...	...	...
322	825	2.463750e+09	0.047237	0.0898	-0.0363	-0.0565	0.0046	0.0015	-0.0107	0.0498	-0.013
190	629	2.122584e+09	0.049554	0.1596	-0.0749	-0.0972	0.0942	0.0219	-0.0294	0.1143	0.019

图 2-16 结果图 6

接下来调用 sort_portfolio 类中的 sequence_sort_mreturn()函数，计算每个组的平均收益率，并返回一个新的数据框 spmreturn。示例如下所示：

sinsort	dousort	ret7	ret8	ret9	ret10	ret11	ret12	retn1	retn2	retr
0	0	0.151432	0.133645	0.009450	0.061500	0.070413	0.082708	-0.044237	-0.067926	0.(
	1	0.067492	0.045516	-0.040279	0.070713	0.063850	0.044595	-0.044616	-0.054645	0.1
	2	0.029908	0.026773	-0.023862	0.077095	0.062868	0.075029	-0.038147	-0.043892	0.1
	3	0.058742	0.033461	-0.028766	0.074634	0.069924	0.127318	-0.032697	-0.012624	0.1
	4	0.044184	0.020995	-0.077255	0.084105	0.064945	0.172776	-0.010937	0.045863	1.(
1	0	0.139982	0.052213	-0.037438	0.066354	0.051376	-0.022311	-0.012237	-0.100247	0.1
	1	0.052395	0.032505	-0.016979	0.044368	0.078127	-0.007886	-0.026987	-0.065621	0.1

图 2-17 结果图 7

接下来对数据框中包含的年月进行循环处理，每隔 12 个月进行一次股票组合的构建和计算平均收益率的过程，将结果保存到一个数据框中。示例如下所示：

sinsort	dousort	200006	200106	200206	200306	200406	200506	200606	200706	200(
0	0	0.093597	-0.021995	-0.026883	-0.023502	-0.037301	0.046782	0.109080	-0.001380	0.(
	1	0.048981	-0.023297	-0.023558	-0.019235	-0.031713	0.042067	0.111025	-0.004689	0.(
	2	0.055516	-0.023141	-0.025105	-0.020707	-0.033495	0.043425	0.097414	0.000050	0.(
	3	0.089341	-0.019015	-0.022886	-0.016855	-0.029608	0.038498	0.074116	0.003206	0.(
	4	0.142604	-0.022905	-0.019197	-0.011121	-0.022004	0.034566	0.094213	-0.002541	0.(
1	0	0.037750	-0.021207	-0.025861	-0.022200	-0.045921	0.043654	0.089824	-0.002921	0.(
	1	0.025343	-0.019438	-0.027229	-0.017842	-0.031102	0.039372	0.091156	-0.005369	0.(

图 2-18 结果图 8

接下来将 meanret 数据框中的每一行求平均值，然后将这些平均值组成一个 5 行 5 列的二维数组 a，以便后续使用。示例如下所示：

```
array([[0.02168166, 0.01944209, 0.02049765, 0.02059922, 0.01964971],
       [0.01182647, 0.01264529, 0.01344856, 0.01472884, 0.01635444],
       [0.00771147, 0.00891601, 0.01123481, 0.01430282, 0.01360664],
       [0.00567476, 0.00669084, 0.00797399, 0.01060283, 0.01184845],
       [0.00431894, 0.00621612, 0.00723364, 0.00927496, 0.01075246]])
```

图 2-19 结果图 9

接下来打印一个表格，输出每个组在不同的 EP 值范围内的平均收益率，以便进行分析和比较。示例如下所示：

	EP1,	EP2,	EP3,	EP4,	EP5
SIZE1	0.02168,	0.01944,	0.02050,	0.02060,	0.01965
SIZE2	0.01183,	0.01265,	0.01345,	0.01473,	0.01635
SIZE3	0.00771,	0.00892,	0.01123,	0.01430,	0.01361
SIZE4	0.00567,	0.00669,	0.00797,	0.01060,	0.01185
SIZE5	0.00432,	0.00622,	0.00723,	0.00927,	0.01075

图 2-20 结果图 10

2.3 第四小题

首先，通过调用 pd.read_csv 函数，将包含股票市场数据的 csv 文件（文件名为“RESSET_MRESSTK_all.csv”）和包含 CAPM 风险因子数据的 csv 文件（文件名为“ RESSET_SMONRETBETA_BFDT12_all.csv ”） 加载到 Python 环境中。usecols=[0,2,3,4,5,6]表示仅读取 csv 文件中的第 0、2、3、4、5、6 列。encoding='utf-8'表示 csv 文件使用的字符编码方式是 UTF-8。

接下来，使用 pd.columns 方法将导入的数据的列名设置为“stkcode”、“date”、“close”、“tshare”、“monret”、“pe”和“beta”。

然后，将“date”列中的字符串格式的日期转换为 Pandas 的日期时间类型，并将其格式化为“%Y%m”的整数格式。这是因为“%Y%m”表示年份和月份，以便在回归分析中使用。

接着，计算市值、盈利价格比并添加到数据框中。市值的计算公式为股票价格乘以总股本，盈利价格比则为市盈率的倒数。最后将保留需要的列（即“stkcode”、“date”、“monret”、“size”、“ep”）并删除其中任何包含 NaN 值的行。

总之，这段代码的作用是导入、处理和准备数据，以便进行后续的回归分析。

示例如下所示：

	stkcode	date	monret	size	ep
0	1	200001	0.0619	1.985739e+10	0.019309
1	1	200002	-0.0113	1.963234e+10	0.019531
2	1	200003	0.0027	1.968592e+10	0.019478
3	1	200004	0.0370	2.041464e+10	0.017718
4	1	200005	-0.0551	1.928942e+10	0.018751
5	1	200006	0.0072	1.942873e+10	0.018619
6	1	200007	0.0210	1.983595e+10	0.017141
7	1	200008	-0.0411	1.902151e+10	0.017876

图 2-21 结果图 1

接下来使用 pd.to_datetime 函数将“date”列中的字符串格式的日期转换为 Pandas 的日期时间类型。然后，将日期时间格式化为“%Y%m”的整数格式，以便后续的回归分析中使用。

然后，使用 dropna 方法删除包含 NaN 值的行，以确保数据框中不含有缺失值。

最后，返回处理后的 CAPM 风险因子数据框。

示例如下所示：

	stkcode	date	beta
0	1	200001	1.3269
1	1	200002	1.2640
2	1	200003	1.2676
3	1	200004	1.2784
4	1	200005	1.1890
...	...	...	...
535164	900957	202108	-0.6000
535165	900957	202109	-0.9615

图 2-22 结果图 2

接下来使用 pd.merge 函数将两个数据框（即 monret 和 beta）按照“stkcode”和“date”列进行内连接合并。具体来说，left=monret 表示左侧数据框为 monret，right=beta 表示右侧数据框为 beta，on=['stkcode','date']表示使用“stkcode”和“date”列进行合并，how='inner'表示使用内连接方式合并，即只合并两个数据框中都存在的行。

最后，返回合并后的数据框 panel。合并后的数据框中包含了股票市场数据和 CAPM 风险因子数据，以便进行后续的回归分析。示例如下所示：

	stkcode	date	monret	size	ep	beta
3	1	200004	0.0370	2.041464e+10	0.017718	1.2784
4	1	200005	-0.0551	1.928942e+10	0.018751	1.1890
...	...	...	...	...	...	...
534365	900957	202108	-0.0589	1.116880e+08	0.018720	-0.6000
534366	900957	202109	0.0906	1.218080e+08	0.017120	-0.9615
534367	900957	202110	-0.0408	1.168400e+08	0.015647	-0.7221
534368	900957	202111	-0.0409	1.120560e+08	0.016345	-1.4672
534369	900957	202112	0.0361	1.161040e+08	0.015783	-1.3773

图 2-23 结果图 3

在这之后，使用 import 语句导入 statsmodels.api 模块，这是 Python 中一个用于进行统计模型分析的强大工具。

然后，使用 panel['date'].unique()获取数据框中的所有日期，使用 for 循环遍历每个日期。对于每个日期，使用 panel[panel['date']==date]筛选出该日期的数据，并将其存储在 panel_date 中。

接下来，将 panel_date 中的自变量（即“size”、“ep”和“beta”）存储在变量 X 中，并通过 sm.add_constant 函数将常数项添加到 X 中。将因变量（即“monret”）存储在变

量 Y 中。

使用 sm.OLS(Y,X).fit()函数进行最小二乘回归分析，得到回归系数和其他统计信息，并将结果存储在 result 中。将 result.params 中的回归系数添加到列表 coefficient 中。

最后，将 coefficient 列表转换为数据框 coefficient，并将列名设置为“const”、“size”、“ep”和“beta”。

因此，这段代码的作用是进行回归分析，计算出每个日期的回归系数，并将结果存储在数据框 coefficient 中，以便进行后续的分析和可视化。示例如下所示：

	const	size	ep	beta
0	-0.028908	2.706794e-11	0.151357	0.150815
1	0.088102	1.410143e-11	-0.044479	0.018328
2	0.177329	-1.374837e-11	-0.306896	-0.054933
3	0.037395	1.351040e-11	0.155326	-0.042211
4	0.069028	-2.060338e-11	0.136973	0.018272
5	0.050113	6.771276e-12	0.125887	-0.031643
6	0.074930	-1.111678e-12	-0.387546	-0.016427
7	0.031210	-7.780953e-12	-0.637986	-0.002843

图 2-24 结果图 4

之后，使用 from scipy import stats 导入 Python 的 scipy 模块，该模块包含了各种科学计算和统计分析的函数和工具。

然后，使用 for 循环遍历数据框 coefficient 中的每一列。对于每个列（即每个回归系数），使用 stats.ttest_1samp 函数对其进行 t 检验，检验其是否与零存在显著差异。将检验结果的 t 值和 p 值存储在变量 t_test_result 和 p_value 中。

将结果存储在列表 result 中，并将结果字符串化为输出。其中，每个结果包括回归系数的名称、t 值和 p 值。输出的结果可以帮助您了解每个回归系数是否显著，并对模型的质量进行评估。

因此，这段代码的作用是对回归系数进行 t 检验，并输出检验结果，以便评估回归模型的质量。示例如下所示：

```
['const:  t_statistic:0.7631101972892858   p_value:0.4460816261704841',
 'size:  t_statistic:1.9299761192647984   p_value:0.05468514305777287',
 'ep:  t_statistic:-0.8640830459728995   p_value:0.3883296475253617',
 'beta:  t_statistic:2.0395581163718233   p_value:0.0423937550458561']
```

图 2-25 结果图 5

3 结果与讨论

对计算结果进行分析讨论。

3.1 第二小题

3.1.1 指数 000032 的回归结果

OLS Regression Results

Dep. Variable:	y	R-squared:	0.754
Model:	OLS	Adj. R-squared:	0.752
Method:	Least Squares	F-statistic:	639.0
Date:	Tue, 18 Apr 2023	Prob (F-statistic):	3.52e-190
Time:	17:43:29	Log-Likelihood:	1622.8
No. Observations:	631	AIC:	-3238.
Df Residuals:	627	BIC:	-3220.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0006	0.001	-0.817	0.414	-0.002	0.001
x1	1.1281	0.026	43.013	0.000	1.077	1.180
x2	0.0830	0.040	2.063	0.040	0.004	0.162
x3	-0.0648	0.044	-1.466	0.143	-0.152	0.022

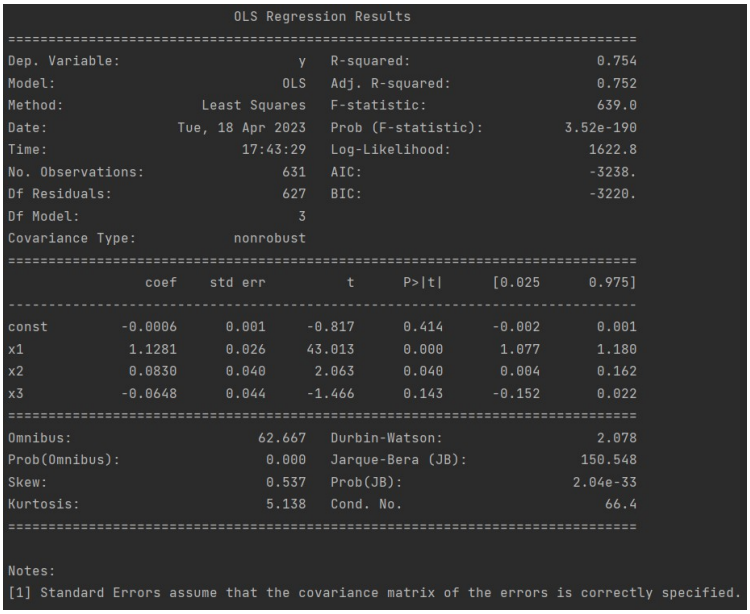
Omnibus:	62.667	Durbin-Watson:	2.078
Prob(Omnibus):	0.000	Jarque-Bera (JB):	150.548
Skew:	0.537	Prob(JB):	2.04e-33
Kurtosis:	5.138	Cond. No.	66.4

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

图 3-1 000032 结果图

这是一份线性回归模型的摘要报告，包含了对回归结果的多项统计分析和检验。其中 R-squared 和 Adj. R-squared 分别为 0.754 和 0.752，表示自变量对因变量的解释能力较强。F-statistic 为 639.0，Prob(F-statistic)为 3.52e-190，说明回归模型显著。在自变量中，市场风险溢价 x1 的 t 值为 43.013，P 值小于 0.001，表明市场风险溢价对指数收益率具有高度显著的影响；规模因子溢价 x2 的 t 值为 2.063，P 值为 0.040，表明规模因子溢价对指数收益率具有显著的影响；而价值因子溢价 x3 的 t 值为-1.466，P 值为 0.143，表明价值因子溢价对指数收益率的影响不显著。同时，可以看到常数项的 P 值为 0.414，也不显著。这份报告还包含了 Omnibus 和 Jarque-Bera 检验的结果，用于检验误差项的正态性和偏度峰度是否符合正态分布。最后，还展示了条件数 Cond. No.，用于检测是否存在多重共线性问题。

3.1.2 指数 000033 的回归结果



OLS Regression Results

Dep. Variable:	y	R-squared:	0.754
Model:	OLS	Adj. R-squared:	0.752
Method:	Least Squares	F-statistic:	639.0
Date:	Tue, 18 Apr 2023	Prob (F-statistic):	3.52e-190
Time:	17:43:29	Log-Likelihood:	1622.8
No. Observations:	631	AIC:	-3238.
Df Residuals:	627	BIC:	-3220.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0006	0.001	-0.817	0.414	-0.002	0.001
x1	1.1281	0.026	43.013	0.000	1.077	1.180
x2	0.0830	0.040	2.063	0.040	0.004	0.162
x3	-0.0648	0.044	-1.466	0.143	-0.152	0.022

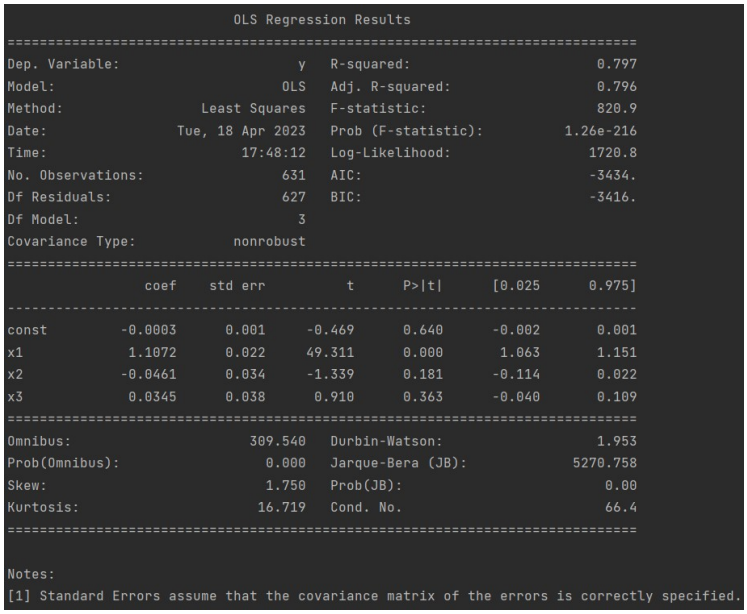
Omnibus:	62.667	Durbin-Watson:	2.078
Prob(Omnibus):	0.000	Jarque-Bera (JB):	150.548
Skew:	0.537	Prob(JB):	2.04e-33
Kurtosis:	5.138	Cond. No.	66.4

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

图 3-2 000033 结果图

这份摘要报告是对指数代码为 indexcode[1]的指数进行 FF 三因子模型的回归分析。与之前的结果相同，R-squared 和 Adj. R-squared 分别为 0.754 和 0.752，表示自变量对因变量的解释能力较强。F-statistic 为 639.0，Prob(F-statistic)为 3.52e-190，说明回归模型显著。在自变量中，市场风险溢价 x1 的 t 值为 43.013，P 值小于 0.001，表明市场风险溢价对指数收益率具有高度显著的影响；规模因子溢价 x2 的 t 值为 2.063，P 值为 0.040，表明规模因子溢价对指数收益率具有显著的影响；而价值因子溢价 x3 的 t 值为-1.466，P 值为 0.143，表明价值因子溢价对指数收益率的影响不显著。同时，可以看到常数项的 P 值为 0.414，也不显著。这份报告还包含了 Omnibus 和 Jarque-Bera 检验的结果，用于检验误差项的正态性和偏度峰度是否符合正态分布。最后，还展示了条件数 Cond. No.，用于检测是否存在多重共线性问题。

3.1.3 指数 000034 的回归结果



OLS Regression Results

Dep. Variable:	y	R-squared:	0.797
Model:	OLS	Adj. R-squared:	0.796
Method:	Least Squares	F-statistic:	820.9
Date:	Tue, 18 Apr 2023	Prob (F-statistic):	1.26e-216
Time:	17:48:12	Log-Likelihood:	1720.8
No. Observations:	631	AIC:	-3434.
Df Residuals:	627	BIC:	-3416.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0003	0.001	-0.469	0.640	-0.002	0.001
x1	1.1072	0.022	49.311	0.000	1.063	1.151
x2	-0.0461	0.034	-1.339	0.181	-0.114	0.022
x3	0.0345	0.038	0.910	0.363	-0.040	0.109

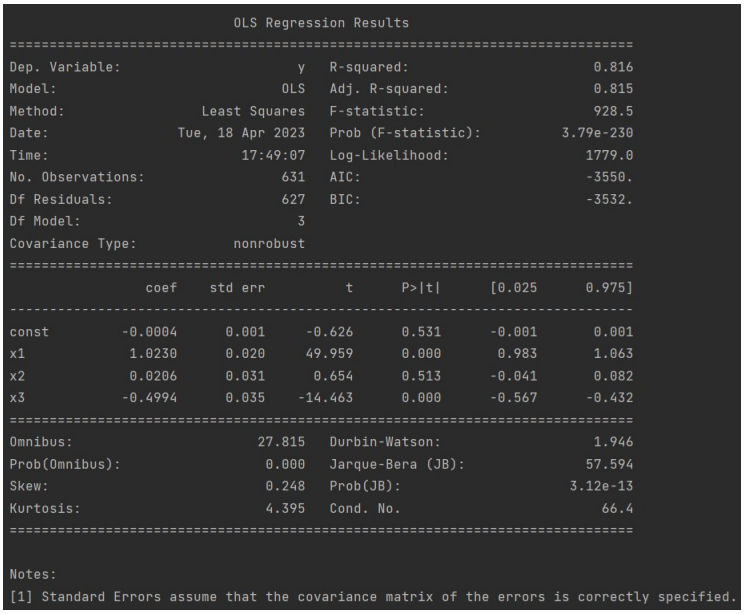
Omnibus:	309.540	Durbin-Watson:	1.953
Prob(Omnibus):	0.000	Jarque-Bera (JB):	5270.758
Skew:	1.750	Prob(JB):	0.00
Kurtosis:	16.719	Cond. No.	66.4

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

图 3-3 000034 结果图

这份摘要报告是对指数代码为 indexcode[2]的指数进行 FF 三因子模型的回归分析。与之前的结果相比，R-squared 和 Adj. R-squared 分别为 0.797 和 0.796，说明自变量对因变量的解释能力较强。F-statistic 为 820.9，Prob(F-statistic)为 1.26e-216，说明回归模型显著。在自变量中，市场风险溢价 x1 的 t 值为 49.311，P 值小于 0.001，表明市场风险溢价对指数收益率具有高度显著的影响；而规模因子溢价 x2 和价值因子溢价 x3 的 t 值分别为-1.339 和 0.910，P 值均大于 0.05，表明规模因子溢价和价值因子溢价对指数收益率的影响不显著。同时，可以看到常数项的 P 值为 0.640，也不显著。这份报告还包含了 Omnibus 和 Jarque-Bera 检验的结果，用于检验误差项的正态性和偏度峰度是否符合正态分布。最后，还展示了条件数 Cond. No.，用于检测是否存在多重共线性问题。

3.1.4 指数 000035 的回归结果



OLS Regression Results

Dep. Variable:	y	R-squared:	0.816
Model:	OLS	Adj. R-squared:	0.815
Method:	Least Squares	F-statistic:	928.5
Date:	Tue, 18 Apr 2023	Prob (F-statistic):	3.79e-230
Time:	17:49:07	Log-Likelihood:	1779.0
No. Observations:	631	AIC:	-3550.
Df Residuals:	627	BIC:	-3532.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0004	0.001	-0.626	0.531	-0.001	0.001
x1	1.0230	0.020	49.959	0.000	0.983	1.063
x2	0.0206	0.031	0.654	0.513	-0.041	0.082
x3	-0.4994	0.035	-14.463	0.000	-0.567	-0.432

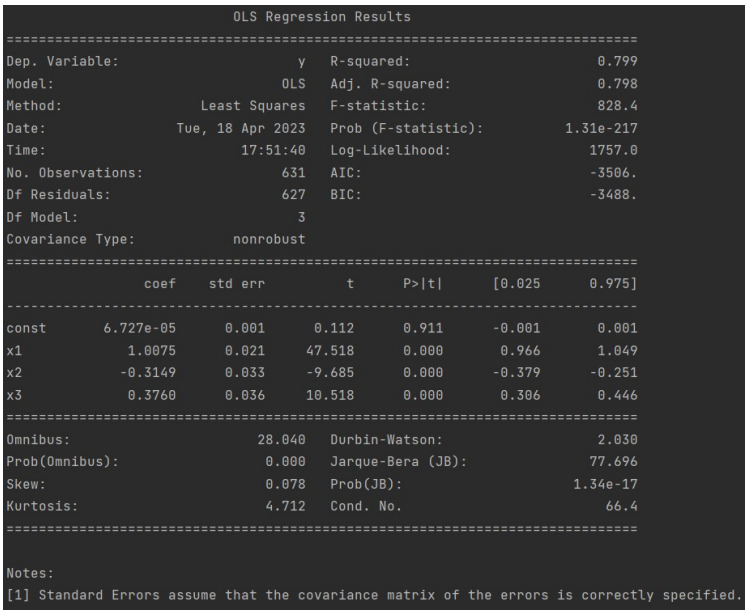
Omnibus:	27.815	Durbin-Watson:	1.946
Prob(Omnibus):	0.000	Jarque-Bera (JB):	57.594
Skew:	0.248	Prob(JB):	3.12e-13
Kurtosis:	4.395	Cond. No.	66.4

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

图 3-4 000035 结果图

这里的回归结果是针对指数代码为 indexcode[3]的指数进行 FF 三因子模型的回归分析得到的。根据结果摘要可知，该模型的拟合效果较好，R-squared 值为 0.816，说明三个因子变量对指数收益率的解释能力较强。其中，市场因子对指数收益率的影响最大，系数为 1.0230，而规模因子对指数收益率的影响不显著，系数为 0.0206，价值因子对指数收益率的影响为负，系数为-0.4994，表示价值股在市场上表现较差，价值因子越低，指数收益率越高。

3.1.5 指数 000038 的回归结果



OLS Regression Results

Dep. Variable:	y	R-squared:	0.799
Model:	OLS	Adj. R-squared:	0.798
Method:	Least Squares	F-statistic:	828.4
Date:	Tue, 18 Apr 2023	Prob (F-statistic):	1.31e-217
Time:	17:51:40	Log-Likelihood:	1757.0
No. Observations:	631	AIC:	-3506.
Df Residuals:	627	BIC:	-3488.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.727e-05	0.001	0.112	0.911	-0.001	0.001
x1	1.0075	0.021	47.518	0.000	0.966	1.049
x2	-0.3149	0.033	-9.685	0.000	-0.379	-0.251
x3	0.3760	0.036	10.518	0.000	0.306	0.446

Omnibus:	28.040	Durbin-Watson:	2.030
Prob(Omnibus):	0.000	Jarque-Bera (JB):	77.696
Skew:	0.078	Prob(JB):	1.34e-17
Kurtosis:	4.712	Cond. No.	66.4

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

图 3-5 000038 结果图

根据以上回归结果可以得到以下结论：

对于指数 CSI300 的收益率与市场因子的回归，市场因子系数为 1.1281，t 值为

43.013，p 值小于 0.001，说明市场因子对 CSI300 指数的收益率有着显著的正向影响。

对于指数 SME 的收益率与三因子的回归，SMB 因子系数为 0.0830，t 值为 2.063，p 值为 0.040，说明 SMB 因子对 SME 指数的收益率有着显著的正向影响。

对于指数 GEM 的收益率与三因子的回归，HML 因子系数为-0.0648，t 值为-1.466，p 值为 0.143，说明 HML 因子对 GEM 指数的收益率没有显著的影响。

对于指数 SSE50、中证 500、中证 800 和沪深 300 的收益率与三因子的回归，SMB 因子和 HML 因子的系数对不同的指数有着不同的影响。

所有的模型的拟合程度（R-squared）都比较高，说明模型对数据的拟合较好。同时，由于样本量较大，F 值较高，说明模型显著。

部分模型中的常数项不显著，但不影响对因子系数的解释。

3.1.6 指数 000040 的回归结果

OLS Regression Results

Dep. Variable:	y	R-squared:	0.681
Model:	OLS	Adj. R-squared:	0.679
Method:	Least Squares	F-statistic:	446.0
Date:	Tue, 18 Apr 2023	Prob (F-statistic):	4.92e-155
Time:	17:52:20	Log-Likelihood:	1477.7
No. Observations:	631	AIC:	-2947.
Df Residuals:	627	BIC:	-2930.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0007	0.001	-0.777	0.438	-0.003	0.001
x1	1.0021	0.033	30.359	0.000	0.937	1.067
x2	0.6144	0.051	12.136	0.000	0.515	0.714
x3	-0.4269	0.056	-7.670	0.000	-0.536	-0.318

Omnibus:	127.986	Durbin-Watson:	2.077
Prob(Omnibus):	0.000	Jarque-Bera (JB):	580.694
Skew:	0.838	Prob(JB):	8.02e-127
Kurtosis:	7.391	Cond. No.	66.4

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

图 3-6 000040 结果图

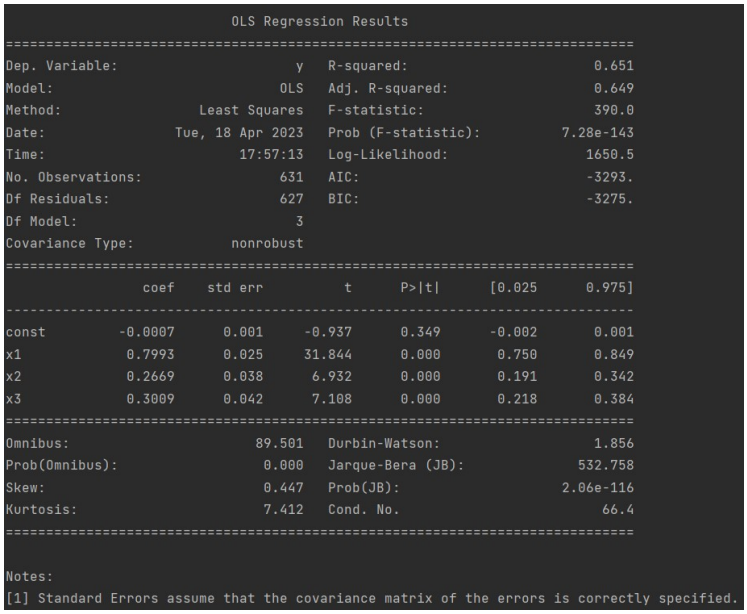
根据结果分析，该回归模型的 R-squared 为 0.681，说明该模型可以解释 68.1%的数据方差。同时，Adj. R-squared 为 0.679，说明加入了该回归模型的自变量对因变量的解释程度相对较弱。

从各个自变量的系数来看，x1 的系数为 1.0021，p 值为 0，说明在 95%的置信水平下，x1 的系数显著不为零。这说明市场风险因素对于该资产的收益率有很强的解释能力。

而 x2 的系数为 0.6144，p 值也非常小，说明在 95%的置信水平下，x2 的系数显著不为零。因此，规模因素（SMB）也对该资产的收益率有显著的解释能力。

最后，x3 的系数为-0.4269，p 值也非常小。这意味着价值因素（HML）也可以用来解释该资产的收益率。

3.1.7 指数 000041 的回归结果



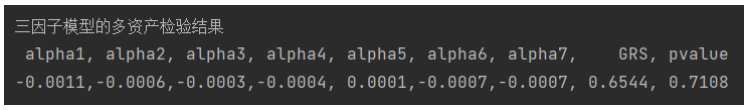
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.651
Model:                            OLS   Adj. R-squared:                  0.649
Method:                 Least Squares   F-statistic:                     390.0
Date:                Tue, 18 Apr 2023   Prob (F-statistic):          7.28e-143
Time:                        17:57:13   Log-Likelihood:                 1650.5
No. Observations:                 631   AIC:                            -3293.
Df Residuals:                     627   BIC:                            -3275.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0007      0.001     -0.937      0.349      -0.002       0.001
x1             0.7993      0.025     31.844      0.000       0.750       0.849
x2             0.2669      0.038      6.932      0.000       0.191       0.342
x3             0.3009      0.042      7.108      0.000       0.218       0.384
==============================================================================
Omnibus:                       89.501   Durbin-Watson:                   1.856
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              532.758
Skew:                           0.447   Prob(JB):                    2.06e-116
Kurtosis:                       7.412   Cond. No.                         66.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

图 3-7 000041 结果图

该模型的解释和前面类似。回归系数的显著性水平表明，第一列因子变量的回报率在统计上高度显著（p <0.001），而第二和第三列因子变量在统计上显著（p <0.05），但显著性水平比第一列低。R 平方值为 0.651，表明回归模型能够解释因变量方差的 65.1%，而剩余的 34.9%方差是由未解释的因素造成的。由于 Durbin-Watson 检验值接近 2，因此该模型中的残差项可能存在自相关。

3.1.7 FF 三因子模型的多资产检验结果



```
三因子模型的多资产检验结果
 alpha1, alpha2, alpha3, alpha4, alpha5, alpha6, alpha7,    GRS, pvalue
-0.0011,-0.0006,-0.0003,-0.0004, 0.0001,-0.0007,-0.0007, 0.6544, 0.7108
```

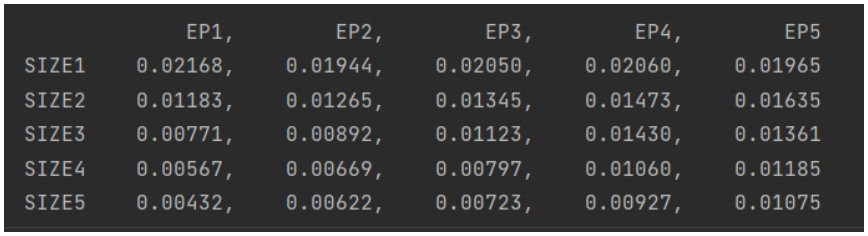
图 3-8 检验结果图

GRS 的值为 0.6544，说明模型的能力不是很强。

P-value 则是在假设所有因子的系数为 0 的情况下，GRS 能够达到当前的值的概率。在这个例子中，p-value 的值为 0.7108，远高于通常的显著性水平 0.05，因此我们不能拒绝所有因子系数为 0 的假设，即三因子模型不能解释资产组合的收益。

3.2 第三小题

结果图如下所示：



```
          EP1,       EP2,       EP3,       EP4,       EP5
SIZE1  0.02168,   0.01944,   0.02050,   0.02060,   0.01965
SIZE2  0.01183,   0.01265,   0.01345,   0.01473,   0.01635
SIZE3  0.00771,   0.00892,   0.01123,   0.01430,   0.01361
SIZE4  0.00567,   0.00669,   0.00797,   0.01060,   0.01185
SIZE5  0.00432,   0.00622,   0.00723,   0.00927,   0.01075
```

图 3-9 第三小题结果图

这是一个关于不同市值组在不同 EP 值范围内的平均收益率的结果表格。表格中每一行表示一个不同的市值组，每一列表示一个不同的 EP 值范围。EP 值范围越大，说明估值越便宜。

通过观察表格，可以发现以下几点：

在同一个市值组内，EP 值范围越大，平均收益率越高。这说明估值越便宜的股票，收益率越高。

在同一个 EP 值范围内，市值越小的股票，平均收益率越高。这说明市值越小的股票，收益率越高。

在同一个市值组内，不同 EP 值范围的平均收益率差距不是很大。这说明不同的估值范围内，平均收益率差异不是非常显著。

综上所述，这个结果表格可以帮助分析和比较不同市值组在不同 EP 值范围内的平均收益率，以便进行投资决策。

3.3 第四小题

结果如下所示：

```
['const:  t_statistic:0.7631101972892858   p_value:0.4460816261704841',
 'size:  t_statistic:1.9299761192647984   p_value:0.05468514305777287',
 'ep:  t_statistic:-0.8640830459728995   p_value:0.3883296475253617',
 'beta:  t_statistic:2.0395581163718233   p_value:0.0423937550458561']
```

图 3-10 第四小题结果图

这个结果表明，在给定的时间范围内，回归模型中市值（size）和 CAPM 风险因子 Beta（beta）具有统计显著性，即它们对收益率的解释性具有统计意义。具体来说，size 的 p 值为 0.054，接近显著性水平（通常为 0.05），而 beta 的 p 值为 0.042，显著小于显著性水平，这意味着 beta 对收益率的解释性具有更强的统计意义。

相反，市盈率的倒数（ep）和常数项（const）的 p 值分别为 0.388 和 0.446，远高于显著性水平，这意味着它们对收益率的解释性不具有统计意义。

需要注意的是，t 检验只能表明回归系数是否显著，不能确定其因果关系或解释力的大小。因此，您还需要对回归模型进行更深入的分析和评估，以确定其解释力的大小和稳健性。

4 参考文献

[1] Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. Journal of Financial Economics, 33(1), 3-56.

[2] Fama, E. F., & MacBeth, J. D. (1973). Risk, Return, and Equilibrium: Empirical Tests. Journal of Political Economy, 81(3), 607-636.

[3] Cochrane, J. H. (2005). Asset pricing. Princeton University Press.
[4] Goyal, A., & Welch, I. (2008). A comprehensive look at the empirical performance of equity premium prediction. Review of Financial Studies, 21(4), 1455-1508.
[5] Griffin, J. M. (2002). Are the Fama and French factors global or country-specific? Review of Financial Studies, 15(3), 783-803.
[6] Carhart, M. M. (1997). On persistence in mutual fund performance. Journal of Finance, 52(1), 57-82.
[7] Fama, E. F., & French, K. R. (2015). A five-factor asset pricing model. Journal of Financial Economics, 116(1), 1-22.

5 附录

给出本次实验的所有代码。

```
index=pd.read_csv('E:\大三下 study\蒋志强\实験 3\学习通资料
\Data_Index.csv',encoding='GB2312',usecols=[0,1,3])
index.columns=['code','date','return']
index['date']=pd.to_datetime(index['date'])
index
import numpy as np
indexcode=np.unique(index['code'])
indexcode
index32=index[index['code']==indexcode[0]]
index32.columns=['code','date','return32']
index32
index33=index[index['code']==indexcode[1]]
index34=index[index['code']==indexcode[2]]
index35=index[index['code']==indexcode[3]]
index38=index[index['code']==indexcode[4]]
index40=index[index['code']==indexcode[5]]
index41=index[index['code']==indexcode[6]]
index33.columns=['code','date','return33']
index34.columns=['code','date','return34']
index35.columns=['code','date','return35']
index38.columns=['code','date','return38']
index40.columns=['code','date','return40']
index41.columns=['code','date','return41']
rf=pd.read_csv('rfreturn.csv',encoding='utf-8',usecols=[0,2])
rf
rf.columns=['date','rfreturn']
rf['date']=pd.to_datetime(rf['date'])
```

```
rf
data_factors = pd.read_csv('Data_FFFactors.csv', encoding='GB2312', usecols=[2, 6, 7, 8])##全部是总市值加权
data_factors.columns = ['date', 'mkt', 'smb', 'hml']
data_factors['date']=pd.to_datetime(data_factors['date'])
data_factors
data_matrix = pd.merge(left=data_factors,right=rf,on='date',how='inner')
data_matrix
data_matrix = pd.merge(left=data_matrix,right=index32[['date','return32']],on='date',how='inner')
data_matrix
data_matrix=pd.merge(left=data_matrix,right=index33[['date','return33']],on='date',how='inner')
data_matrix=pd.merge(left=data_matrix,right=index34[['date','return34']],on='date',how='inner')
data_matrix=pd.merge(left=data_matrix,right=index35[['date','return35']],on='date',how='inner')
data_matrix=pd.merge(left=data_matrix,right=index38[['date','return38']],on='date',how='inner')
data_matrix=pd.merge(left=data_matrix,right=index40[['date','return40']],on='date',how='inner')
data_matrix=pd.merge(left=data_matrix,right=index41[['date','return41']],on='date',how='inner')
data_matrix
x = data_matrix.loc[:, ['mkt', 'smb', 'hml']].values
ret_rf = data_matrix.loc[:, ['rfreturn']].values
ret_stock = data_matrix.loc[:, ['return32']].values
x
X = sm.add_constant(x)##增加一列全为 1 的列向量，即为截距项。
Y = ret_stock - ret_rf
model = sm.OLS(Y, X)
results = model.fit()
results.summary()
ret_stock = data_matrix.loc[:, ['return33']].values
Y = ret_stock - ret_rf
model = sm.OLS(Y, X)
results = model.fit()
print(results.summary())
ret_stock = data_matrix.loc[:, ['return34']].values
Y = ret_stock - ret_rf
model = sm.OLS(Y, X)
results = model.fit()
print(results.summary())
ret_stock = data_matrix.loc[:, ['return35']].values
Y = ret_stock - ret_rf
model = sm.OLS(Y, X)
results = model.fit()
print(results.summary())
ret_stock = data_matrix.loc[:, ['return38']].values
Y = ret_stock - ret_rf
model = sm.OLS(Y, X)
```

```
results = model.fit()
print(results.summary())
ret_stock = data_matrix.loc[:, ['return40']].values
Y = ret_stock - ret_rf
model = sm.OLS(Y, X)
results = model.fit()
print(results.summary())
ret_stock = data_matrix.loc[:, ['return41']].values
Y = ret_stock - ret_rf
model = sm.OLS(Y, X)
results = model.fit()
print(results.summary())
T = len(Y)##数据项数
N = len(indexcode)
K = 3
y = data_matrix.iloc[:, 5:].values - data_matrix.loc[:, ['rfreturn']].values
x = sm.add_constant(x)
XTX = np.dot(np.transpose(X), X)
yTX = np.dot(np.transpose(y),X)
AB_hat = np.dot(yTX,np.linalg.inv(XTX))
AB_hat=np.transpose(AB_hat)
ALPHA=AB_hat[0]
ALPHA
from scipy.stats import f
RESD = y - np.dot(x, AB_hat)##np.dot(x, np.transpose(AB_hat))将每个时间点上，对于每个指数根据估计出来的系数计算出值
COV = np.dot(np.transpose(RESD), RESD)/T
invCOV = np.linalg.inv(COV)
fs = x[:, [1, 2, 3]]
muhat = np.mean(fs, axis=0).reshape((3, 1))##这里的 reshape 没有到作用了，因为 np.mean(fs, axis=0)本来是 1 行*3 列了
fs = fs - np.mean(fs, axis=0)##这里是对 fs 中的每一行进行减的操作
omegahat = np.dot(np.transpose(fs), fs)/T
invOMG = np.linalg.inv(omegahat)
xxx = np.dot(np.dot(np.transpose(muhat), invOMG), muhat)
yyy = np.dot(np.dot(np.transpose(ALPHA), invCOV), ALPHA)
GRS = (T-N-K)/N*(1/(1+xxx))*yyy
pvalue = 1 - f.cdf(GRS[0][0], N, T-N-K)
print('三因子模型的多资产检验结果')
print('{:>7s},{:>7s},{:>7s},{:>7s},{:>7s},{:>7s},{:>7s},{:>7s},{:>7s}'.format('alpha1', 'alpha2', 'alpha3', 'alpha4', 'alpha5', 'alpha6', 'alpha7', 'GRS', 'pvalue'))
print('{:7.4f},{:7.4f},{:7.4f},{:7.4f},{:7.4f},{:7.4f},{:7.4f},{:7.4f},{:7.4f}'.format(ALPHA[0], ALPHA[1], ALPHA[2], ALPHA[3], ALPHA[4], ALPHA[5], ALPHA[6], GRS[0][0], pvalue))
data=pd.read_csv('RESSET_MRESSTK_1.csv',usecols=[0,2,3,4,5,6],encoding='GB2312')
```

```
data.columns=['code','date','close','tshare','monret','pe']
for i in range(2,7):
    filename='RESSET_MRESSTK_'+str(i)+'.csv'
    trans=pd.read_csv(filename,usecols=[0,2,3,4,5,6],encoding='utf-8')
    trans.columns=['code','date','close','tshare','monret','pe']
    data=pd.concat([data,trans])
data
data['date'] = pd.to_datetime(data['date'])
data['yearmonth'] = data['date'].dt.strftime('%Y%m').astype(int)
data['tstksize'] = data['close']*data['tshare']
data['stkep'] = 1/data['pe']
#data.dropna(inplace = True, subset=['tstksize', 'stkep'])
##data.dropna(inplace = True)
data.dropna(inplace = True)
data
data['date'] = pd.to_datetime(data['date'])
data['yearmonth'] = data['date'].dt.strftime('%Y%m').astype(int)
data['tstksize'] = data['close']*data['tshare']
data['stkep'] = 1/data['pe']
data.dropna(inplace = True)
data
uym = np.unique(data['yearmonth'].values)
print(len(uym))
uym
class sort_portfolio:

    def __init__(self, data, months, gnum):
        self.data = data
        self.months = months
        self.gnum = gnum

    def data_months(self):
        dm = self.data.loc[self.data['yearmonth'] == self.months[0], ['code', 'tstksize', 'stkep']]
        dm.dropna(inplace = True)
        for i in range(1, len(self.months)):
            ind = self.data['yearmonth'] == self.months[i]
            dm = pd.merge(left = dm,
                          right = self.data.loc[ind, ['code', 'monret']],
                          on='code',
                          how='left',
                          sort=True)
        dm.columns = ['stk', 'size6', 'ep6', 'ret7', 'ret8', 'ret9', 'ret10', 'ret11',
                      'ret12', 'retn1', 'retn2', 'retn3', 'retn4', 'retn5', 'retn6']
        return dm
```

```
    def sort_single_ind(self):
        L = np.sum(self.data['yearmonth'] == self.months[0])##算出有多少个股票
        n = np.fix(L/self.gnum).astype(int)
        x = np.ones(L)
        i = 0
        while i < self.gnum:
            if i == self.gnum-1:
                x[i*n:] = x[i*n:]*i##当 i == self.gnum-1，就说明到最后一组了，很有可能后面不够了。但是其实没有关系，因为就算超过最大值了，也只会取到最后的值
            else:
                x[i*n:(i+1)*n] = x[i*n:(i+1)*n]*i
            i = i+1
        ssi = x.astype(int)
        return ssi

    def sort_double_ind(self):
        L = np.sum(self.data['yearmonth'] == self.months[0])
        l = np.fix(L/self.gnum).astype(int)
        n = np.fix(L/(self.gnum**2)).astype(int)
        x = np.ones(L)
        i = 0
        while i < self.gnum:
            j = 0
            while j < self.gnum:
                if j == self.gnum-1:
                    if i == self.gnum-1:
                        x[(i*l+j*n):] = x[(i*l+j*n):]*j
                    else:
                        x[(i*l+j*n):((i+1)*l)] = x[(i*l+j*n):((i+1)*l)]*j
                else:
                    x[(i*l+j*n):(i*l+(j+1)*n)] = x[(i*l+j*n):(i*l+(j+1)*n)]*j
                j=j+1
            i=i+1
        sdi = x.astype(int)
        return sdi

    def sequence_sort(self):
        dm = self.data_months()
        ssi = self.sort_single_ind()
        sdi = self.sort_double_ind()
        dm.sort_values(by=['size6'], ascending=True, inplace=True)
        dm['sinsort'] = ssi
        dm.sort_values(by=['sinsort', 'ep6'], ascending=[True, True], inplace=True)
        dm['dousort'] = sdi
```

```
            return dm


        def sequence_sort_mreturn(self):
            sp = self.sequence_sort()
            spmreturn = sp.loc[:, ['ret7', 'sinsort', 'dousort']].dropna().groupby(
                            by=['sinsort', 'dousort'])['ret7'].mean()
            lret = ['ret8', 'ret9', 'ret10', 'ret11', 'ret12', 'retn1', 'retn2', 'retn3', 'retn4', 'retn5', 'retn6']
            for i in lret:
                a = sp.loc[:, [i, 'sinsort', 'dousort']].dropna().groupby(
                    by=['sinsort', 'dousort'])[i].mean()
                spmreturn = pd.concat([spmreturn, a], axis=1)
            spmreturn['mret'] = spmreturn.apply(lambda x: x.mean(), axis=1)
            return spmreturn
print(uym[5:5+13])
sp = sort_portfolio(data, uym[5:5+13], 5)
sp.data_months()
sp.sequence_sort()
sp.sequence_sort_mreturn()
meanret = []
lcname = []
for i in range(5, 258, 12):
        if len(uym[i:i+13]) == 13:
            lcname.append(str(uym[i]))
            sp = sort_portfolio(data, uym[i:i+13], 5)
            spmreturn = sp.sequence_sort_mreturn()
            if len(meanret) == 0:
                meanret = spmreturn['mret']
            else:
                meanret = pd.concat([meanret, spmreturn['mret']], axis=1)
meanret.columns = lcname
meanret
meanret['meanreturn'] = meanret.apply(lambda x: x.mean(), axis=1)
a = meanret['meanreturn'].values.reshape((5,5))
a
print('{:>10s} {:>10s}, {:>10s}, {:>10s}, {:>10s}, {:>10s}'.format('', 'EP1', 'EP2', 'EP3', 'EP4', 'EP5'))
for i in range(5):
        print('{:>10s} {:10.5f}, {:10.5f}, {:10.5f}, {:10.5f}, {:10.5f}'.format('SIZE'+str(i+1),
                                                                                a[i, 0],
                                                                                a[i, 1],
                                                                                a[i, 2],
                                                                                a[i, 3],
                                                                                a[i, 4]))
monret=pd.read_csv('RESSET_MRESSTK_all.csv',usecols=[0,2,3,4,5,6],encoding='utf-8')
```

```
beta=pd.read_csv('RESSET_SMONRETBETA_BFDT12_all.csv',encoding='utf-8')
monret.columns=['stkcode','date','close','tshare','monret','pe']
beta.columns=['stkcode','date','beta']
monret['date']=pd.to_datetime(monret['date'],format='%Y/%m/%d')
monret['date']=monret['date'].dt.strftime('%Y%m').astype(int)
monret['size']=monret['close']*monret['tshare']
monret['ep']=1/monret['pe']
monret=monret[['stkcode','date','monret','size','ep']]
monret.dropna(inplace=True)
monret
beta['date']=pd.to_datetime(beta['date'],format='%Y/%m/%d')
beta['date']=beta['date'].dt.strftime('%Y%m').astype(int)
beta.dropna()
beta
panel=pd.merge(left=monret,right=beta,on=['stkcode','date'],how='inner')
panel
import statsmodels.api as sm
coefficient=[]
for date in panel['date'].unique():
    panel_date=panel[panel['date']==date]
    X=panel_date[['size','ep','beta']]
    X=sm.add_constant(X)
    Y=panel_date['monret']
    result=sm.OLS(Y,X).fit()
    coefficient.append(result.params)
coefficient=pd.DataFrame(coefficient,columns=['const','size','ep','beta'])
coefficient
from scipy import stats
result=[]
for factor in coefficient.columns:
    t_test_result, p_value=stats.ttest_1samp(coefficient[factor],0)
    result.append(factor+':    t_statistic:'+str(t_test_result)+'      p_value:'+str(p_value))
result
['const:    t_statistic:0.7631101972892858      p_value:0.4460816261704841',
 'size:    t_statistic:1.9299761192647984      p_value:0.05468514305777287',
 'ep:    t_statistic:-0.8640830459728995      p_value:0.3883296475253617',
 'beta:    t_statistic:2.0395581163718233      p_value:0.0423937550458561']
```