实验二

1. 文献阅读

认真阅读 3 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. 股票数据实验

尝试找出以 0 或者 6 开头，最后两位数字和自己学号最后两位数字一样的股票代码，并从中任意选取 8 只股票下载 2000-2022 年的日收益和月收益数据，进行如下实验：

- 利用股票日收益数据对 CAPM 模型做单资产时间序列检验。
- 利用股票月收益数据对 CAPM 模型做多资产时间序列检验。

3. 市场效应实证检验

请利用 2001-2022 年中国 A 股月度数据实证检验中国 A 股市市场是具有惯性效应，还是反转效应？（数据请从锐思数据库或者 Tushare 下载）

研究思路（参考课本《金融计量学》P48）

在每个月，计算过去 N 个月的累积收益率，再根据此累积收益率由低到高排序，构造 5 个等权重投资组合，并计算持有这些投资组合 M 个月的累积收益率，追踪投资组合收益率序列，判断中国 A 股市场存在惯性效应还是反转效应。

- **参数设置**：
  - N = 1, 3, 6, 12
  - M = 1, 3, 6, 12

数据选择

我们先用如下脚本来选择尾号为 54 的股票代码

```python
import requests

def search_stock_by_tail(tail="54"):
    # 定义股票前缀及对应市场标识符
    market_config = [
        {"prefix": "600", "market": "sh"},  # 沪市主板
        {"prefix": "601", "market": "sh"},
        {"prefix": "603", "market": "sh"},
        {"prefix": "605", "market": "sh"},
        {"prefix": "688", "market": "sh"},  # 科创板
        {"prefix": "000", "market": "sz"},  # 深市主板
        {"prefix": "001", "market": "sz"},
        {"prefix": "002", "market": "sz"},  # 中小板
        {"prefix": "003", "market": "sz"},
    ]

    results = []
    for config in market_config:
        prefix = config["prefix"]
        market = config["market"]

        # 生成中间位0-9循环的代码 (如600054, 600154, ..., 600954)
        for middle in range(10):
            code = f"{prefix}{middle}{tail}"  # 直接拼接6位代码
            url = f"http://qt.gtimg.cn/q=s_{market}{code}" # 腾讯财经

            try:
                response = requests.get(url, timeout=5)
                response.encoding = "gbk"
                data = response.text

                if "~" in data:
                    name = data.split("~")[1]
                    results.append({"code": code, "name": name})
            except:
                continue

    return results

stocks = search_stock_by_tail(tail="54")
for stock in stocks:
    print(f"代码: {stock['code']} | 名称: {stock['name']}")
```

```
代码: 600054 | 名称: 黄山旅游
代码: 600354 | 名称: 敦煌种业
代码: 600654 | 名称: 中安科
代码: 600754 | 名称: 锦江酒店
代码: 600854 | 名称: 春兰股份
代码: 000554 | 名称: 泰山石油
代码: 002054 | 名称: 德美化工
代码: 002154 | 名称: 报 喜 鸟
代码: 002254 | 名称: 泰和新材
代码: 002354 | 名称: 天娱数科
代码: 002454 | 名称: 松芝股份
代码: 002554 | 名称: 惠博普
代码: 002654 | 名称: 万润科技
```

由于我们只要取 8 只作为实验数据，所以选择如下：

```
代码: 600054 | 名称: 黄山旅游
代码: 600354 | 名称: 敦煌种业
代码: 600654 | 名称: 中安科
代码: 600754 | 名称: 锦江酒店
代码: 600854 | 名称: 春兰股份
代码: 000554 | 名称: 泰山石油
代码: 002054 | 名称: 德美化工
代码: 002154 | 名称: 报 喜 鸟
```

数据爬取

我们利用 Tushare 的 api，对如上的股票信息进行爬取，并存储数据。

由于我没有下载月数据的权限，所以我们这里直接计算代替。

```python
import tushare as ts
import pandas as pd
import os
import time

def create_directory(path):
    """创建目录，如果目录不存在"""
    if not os.path.exists(path):
        os.makedirs(path)
        print(f"创建目录: {path}")

def format_stock_code(code):
    """根据股票代码格式化为tushare所需的格式"""
    code = str(code)
    if code.startswith('6'):
        return f"{code}.SH"  # 上海证券交易所
    elif code.startswith('0') or code.startswith('3'):
        return f"{code}.SZ"  # 深圳证券交易所
    return code

def get_stock_daily_data(pro, ts_code, start_date, end_date):
    """获取股票日线数据"""
    try:
        df = pro.daily(ts_code=ts_code, start_date=start_date, end_date=end_date)
        # 按照日期升序排序
        df = df.sort_values('trade_date', ascending=True)
        return df
    except Exception as e:
        print(f"获取股票 {ts_code} 日线数据失败: {e}")
        return None

def calculate_daily_returns(df):
    """计算日收益率"""
    if df is None or df.empty:
        return None

    # 确保数据按日期排序
    df = df.sort_values('trade_date', ascending=True)

    # 计算日收益率 (当日收盘价/前日收盘价 - 1)
    df['daily_return'] = df['close'].pct_change()

    # 第一行的收益率为NaN，设置为0
    df.loc[df.index[0], 'daily_return'] = 0

    return df

def calculate_monthly_returns(daily_df):
    """从日线数据计算月度收益率"""
    if daily_df is None or daily_df.empty:
        return None

    # 将交易日期字符串转换为日期对象
    daily_df['date'] = pd.to_datetime(daily_df['trade_date'], format='%Y%m%d')

    # 提取年月信息
    daily_df['year_month'] = daily_df['date'].dt.strftime('%Y%m')

    # 获取每个月的第一个和最后一个交易日的收盘价
    monthly_data = []
    for year_month, group in daily_df.groupby('year_month'):
        group = group.sort_values('date')
        first_day = group.iloc[0]
        last_day = group.iloc[-1]
```

```
            monthly_data.append({
                'ts_code': first_day['ts_code'],
                'trade_date': year_month + '01',  # 使用月份的第一天作为标识
                'year_month': year_month,
                'open': first_day['open'],
                'close': last_day['close'],
                'high': group['high'].max(),
                'low': group['low'].min(),
                'vol': group['vol'].sum(),
                'amount': group['amount'].sum(),
                'first_day': first_day['trade_date'],
                'last_day': last_day['trade_date']
            })

        # 创建月度DataFrame
        monthly_df = pd.DataFrame(monthly_data)
        monthly_df = monthly_df.sort_values('year_month')

        # 计算月度收益率
        monthly_df['monthly_return'] = monthly_df['close'].pct_change()
        monthly_df.loc[monthly_df.index[0], 'monthly_return'] = 0

        return monthly_df

    def get_and_store_stock_data():
        # Tushare token
        ts.set_token('903410572598901f5f2b8dc42698829afb5fcd12c0cdaa5e11f5c92b')
        pro = ts.pro_api()

        # 设置日期范围
        start_date = '20000101'
        end_date = '20221231'

        # 股票代码列表
        stock_list = [
            {'code': '600054', 'name': '黄山旅游'},
            {'code': '600354', 'name': '敦煌种业'},
            {'code': '600654', 'name': '中安科'},
            {'code': '600754', 'name': '锦江酒店'},
            {'code': '600854', 'name': '春兰股份'},
            {'code': '000554', 'name': '泰山石油'},
            {'code': '002054', 'name': '德美化工'},
            {'code': '002154', 'name': '报喜鸟'}
        ]

        # 对每个股票代码进行处理
        for stock in stock_list:
            code = stock['code']
            name = stock['name']
            ts_code = format_stock_code(code)

            print(f"\n开始处理股票: {code} | {name} ({ts_code})")

            # 创建存储目录
            daily_dir = f"./assets/data/stock/{code}/daily"
            monthly_dir = f"./assets/data/stock/{code}/monthly"
            create_directory(daily_dir)
            create_directory(monthly_dir)

            # 获取日线数据
            print(f"获取 {code} 日线数据...")
            daily_df = get_stock_daily_data(pro, ts_code, start_date, end_date)

            if daily_df is not None and not daily_df.empty:
                # 计算日收益率
```

```
            daily_df_with_returns = calculate_daily_returns(daily_df)
            # 保存到CSV
            daily_file = f"{daily_dir}/{code}_daily_returns.csv"
            daily_df_with_returns.to_csv(daily_file, index=False)
            print(f"日线数据已保存到: {daily_file}, 共 {len(daily_df_with_returns)} 条记录")

            # 根据日线数据计算月线数据
            print(f"计算 {code} 月线数据...")
            monthly_df = calculate_monthly_returns(daily_df)
            if monthly_df is not None and not monthly_df.empty:
                # 保存到CSV
                monthly_file = f"{monthly_dir}/{code}_monthly_returns.csv"
                monthly_df.to_csv(monthly_file, index=False)
                print(f"月线数据已保存到: {monthly_file}, 共 {len(monthly_df)} 条记录")
            else:
                print(f"无法计算 {code} 月线数据")
        else:
            print(f"无法获取 {code} 日线数据")

        # 避免频繁请求被限流
        time.sleep(1)

get_and_store_stock_data()
```

```
开始处理股票: 600054 | 黄山旅游 (600054.SH)
获取 600054 日线数据...
日线数据已保存到: ./assets/data/stock/600054/daily/600054_daily_returns.csv, 共 5507 条记录
计算 600054 月线数据...
月线数据已保存到: ./assets/data/stock/600054/monthly/600054_monthly_returns.csv, 共 276 条记录

开始处理股票: 600354 | 敦煌种业 (600354.SH)
获取 600354 日线数据...
日线数据已保存到: ./assets/data/stock/600354/daily/600354_daily_returns.csv, 共 4468 条记录
计算 600354 月线数据...
月线数据已保存到: ./assets/data/stock/600354/monthly/600354_monthly_returns.csv, 共 227 条记录

开始处理股票: 600654 | 中安科 (600654.SH)
获取 600654 日线数据...
日线数据已保存到: ./assets/data/stock/600654/daily/600654_daily_returns.csv, 共 5007 条记录
计算 600654 月线数据...
月线数据已保存到: ./assets/data/stock/600654/monthly/600654_monthly_returns.csv, 共 258 条记录

开始处理股票: 600754 | 锦江酒店 (600754.SH)
获取 600754 日线数据...
日线数据已保存到: ./assets/data/stock/600754/daily/600754_daily_returns.csv, 共 5360 条记录
计算 600754 月线数据...
月线数据已保存到: ./assets/data/stock/600754/monthly/600754_monthly_returns.csv, 共 273 条记录

开始处理股票: 600854 | 春兰股份 (600854.SH)
获取 600854 日线数据...
日线数据已保存到: ./assets/data/stock/600854/daily/600854_daily_returns.csv, 共 5131 条记录
计算 600854 月线数据...
月线数据已保存到: ./assets/data/stock/600854/monthly/600854_monthly_returns.csv, 共 258 条记录

开始处理股票: 000554 | 泰山石油 (000554.SZ)
获取 000554 日线数据...
日线数据已保存到: ./assets/data/stock/000554/daily/000554_daily_returns.csv, 共 5489 条记录
计算 000554 月线数据...
月线数据已保存到: ./assets/data/stock/000554/monthly/000554_monthly_returns.csv, 共 275 条记录

开始处理股票: 002054 | 德美化工 (002054.SZ)
获取 002054 日线数据...
日线数据已保存到: ./assets/data/stock/002054/daily/002054_daily_returns.csv, 共 3961 条记录
计算 002054 月线数据...
月线数据已保存到: ./assets/data/stock/002054/monthly/002054_monthly_returns.csv, 共 198 条记录

开始处理股票: 002154 | 报喜鸟 (002154.SZ)
获取 002154 日线数据...
日线数据已保存到: ./assets/data/stock/002154/daily/002154_daily_returns.csv, 共 3711 条记录
计算 002154 月线数据...
月线数据已保存到: ./assets/data/stock/002154/monthly/002154_monthly_returns.csv, 共 185 条记录
```

现在数据已经保存在了 `./assets/data/stock/{stock_code}/{monthly or daily}/xxx.csv` 下，接下来我们就可以进行计算了

利用股票日收益数据对 CAPM 模型做单资产时间序列检验

我们写一个函数，8 只股票都能复用这个代码

```python
# 加载股票数据
def load_stock_data(stock_code, base_dir="./assets/data/stock"):
    daily_path = os.path.join(base_dir, stock_code, "daily", f"{stock_code}_daily_returns.csv")
    df = pd.read_csv(daily_path)

    # 将交易日期转换为datetime格式
    df['trade_date'] = pd.to_datetime(df['trade_date'], format='%Y%m%d')
    df = df.sort_values('trade_date')  # 按日期排序

    # 去除空值
    df = df.dropna()

    # 删除daily_return大于0.1或小于-0.1的行
    df = df[(df['daily_return'] >= -0.1) & (df['daily_return'] <= 0.1)]

    return df

test_df = load_stock_data('000554')
test_df
```

	ts_code	trade_date	open	high	low	close	pre_close	change	pct_chg	vol	amount	daily_ret
0	000554.SZ	2000-01-04	4.35	4.40	4.21	4.36	4.31	0.05	1.1600	20419.00	8881.6099	0.000000
1	000554.SZ	2000-01-05	4.39	4.58	4.37	4.49	4.36	0.13	2.9800	53491.00	24043.2010	0.029817
2	000554.SZ	2000-01-06	4.51	4.90	4.42	4.90	4.49	0.41	9.1300	78348.00	37304.5762	0.091314
3	000554.SZ	2000-01-07	4.91	5.02	4.70	4.95	4.90	0.05	1.0200	73676.00	35790.6146	0.010204
4	000554.SZ	2000-01-10	4.95	5.00	4.75	4.80	4.95	-0.15	-3.0300	43372.00	21044.1573	-0.030303
...	...	...	...	...	...	...	...	...	...	...	...	...
5484	000554.SZ	2022-12-26	5.46	5.65	5.45	5.63	5.43	0.20	3.6832	180948.34	100956.0160	0.036832
5485	000554.SZ	2022-12-27	5.67	5.82	5.62	5.69	5.63	0.06	1.0657	156155.62	89200.0680	0.010657
5486	000554.SZ	2022-12-28	5.70	5.70	5.47	5.58	5.69	-0.11	-1.9332	118117.25	65754.3970	-0.019332
5487	000554.SZ	2022-12-29	5.57	5.77	5.46	5.51	5.58	-0.07	-1.2545	288520.45	161581.1960	-0.012545
5488	000554.SZ	2022-12-30	5.54	5.55	5.34	5.53	5.51	0.02	0.3630	181718.19	99209.6140	0.003630

5425 rows × 12 columns

```python
# 加载市场指数，数据来源于学校锐思数据库
def load_market_index(index_path):
    df = pd.read_excel(index_path)

    column_mapping = {
        '交易日期_TrdDt': 'trade_date',
        '收盘价(元/点)_ClPr': 'close',
        '涨跌幅(%)_ChgPct': 'daily_return_pct',
        '指数代码_IdxCd': 'index_code',
        '指数名称_IdxNm': 'index_name'
    }

    # 重命名存在的列
    for old_name, new_name in column_mapping.items():
        if old_name in df.columns:
            df = df.rename(columns={old_name: new_name})

    # 日期格式转换
    date_sample = str(df['trade_date'].iloc[0])
    if '/' in date_sample or '-' in date_sample:
        # YYYY/MM/DD or YYYY-MM-DD
        df['trade_date'] = pd.to_datetime(df['trade_date'], format='mixed')
    else:
        # YYYYMMDD
        df['trade_date'] = pd.to_datetime(df['trade_date'], format='%Y%m%d')

    # 按日期排序
    df = df.sort_values('trade_date')

    # 涨跌幅已经以百分比形式提供，需要转换为小数
    if 'daily_return_pct' in df.columns:
        df['daily_return'] = df['daily_return_pct'] / 100
    else:
        # 如果没有涨跌幅列，则计算日收益率
        df['daily_return'] = df['close'].pct_change()

    # 移除缺失值
    df = df.dropna(subset=['daily_return'])

    # 删除daily_return大于0.1或小于-0.1的行
    df = df[(df['daily_return'] >= -0.1) & (df['daily_return'] <= 0.1)]

    return df

# 更新文件路径
test_idx = load_market_index('./assets/data/market/szzs_daily.xls')
test_idx
```

	index_code	index_name	trade_date	close	daily_return_pct	daily_return
0	1	上证指数	2000-01-04	1406.3710	2.911721	0.029117
1	1	上证指数	2000-01-05	1409.6820	0.235429	0.002354
2	1	上证指数	2000-01-06	1463.9420	3.849095	0.038491
3	1	上证指数	2000-01-07	1516.6040	3.597274	0.035973
4	1	上证指数	2000-01-10	1545.1120	1.879726	0.018797
...	...	...	...	...	...	...
5569	1	上证指数	2022-12-26	3065.5626	0.646653	0.006467
5570	1	上证指数	2022-12-27	3095.5678	0.978783	0.009788

	index_code	index_name	trade_date	close	daily_return_pct	daily_return
5571	1	上证指数	2022-12-28	3087.3997	-0.263864	-0.002639
5572	1	上证指数	2022-12-29	3073.7016	-0.443678	-0.004437
5573	1	上证指数	2022-12-30	3089.2579	0.506110	0.005061

5574 rows × 6 columns

```
# 无风险利率，数据来源于学校锐思数据库
def calculate_risk_free_rate_daily(treasury_bill_path="./assets/data/rf_daily.xls"):
    df = pd.read_excel(treasury_bill_path)

    # 重命名列
    column_renames = {'日期_Date': 'date', '日无风险收益率_DRFRet': 'rf_rate'}

    df.rename(columns=column_renames, inplace=True)

    df['date'] = pd.to_datetime(df['date'], errors='coerce')
    df.dropna()
    return df

calculate_risk_free_rate_daily()
```

	date	rf_rate
0	2000-01-01	0.000054
1	2000-01-02	0.000054
2	2000-01-03	0.000054
3	2000-01-04	0.000054
4	2000-01-05	0.000054
...	...	...
8396	2022-12-27	0.000066
8397	2022-12-28	0.000066
8398	2022-12-29	0.000066
8399	2022-12-30	0.000066
8400	2022-12-31	0.000066

8401 rows × 2 columns

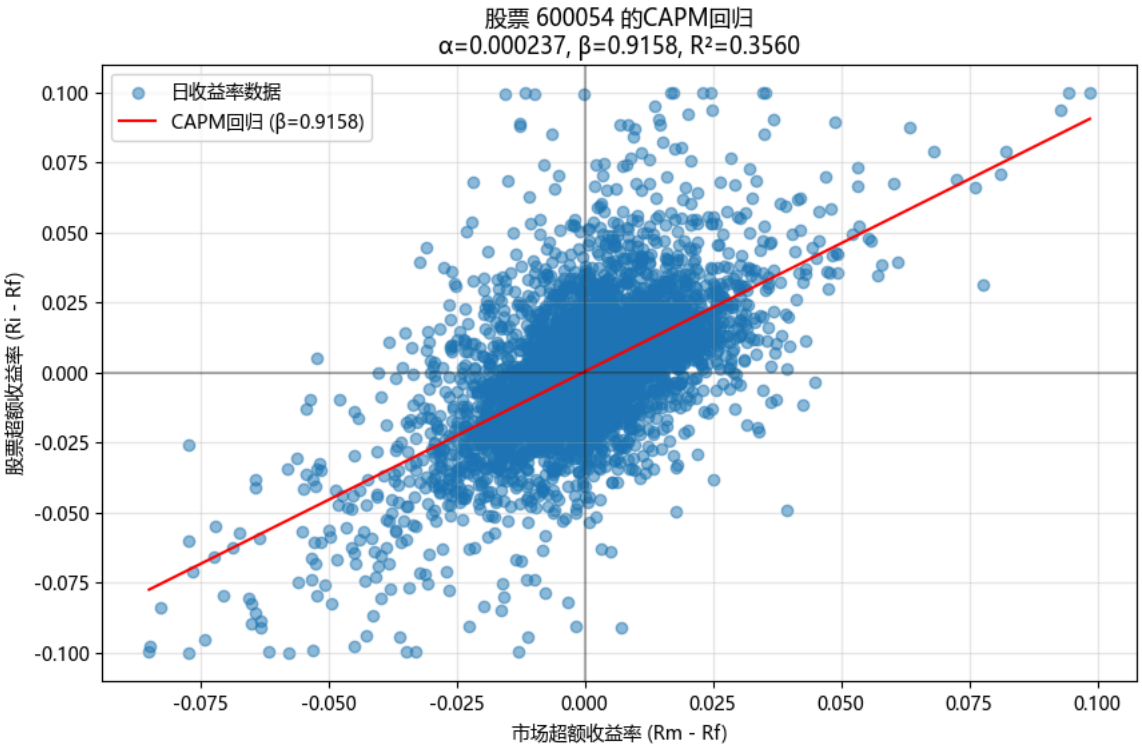
```python
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_white
from scipy import stats
from statsmodels.stats.diagnostic import acorr_ljungbox

# 单个CAPM检验
def test_capm_time_series(stock_code, market_index_df=None, risk_free_rate=None,
                          base_dir="./assets/data/stock", start_date=None, end_date=None):
    """
    stock_code (str): 股票代码
    market_index_df (pandas.DataFrame): 市场指数数据
    risk_free_rate (float): 无风险利率，如果为None则计算
    base_dir (str): 数据存储的基础目录
    start_date (str): 开始日期，格式为'YYYY-MM-DD'
    end_date (str): 结束日期，格式为'YYYY-MM-DD'
    market_index_path (str): 市场指数数据的路径，如果market_index_df为None时使用
    """
    # 加载股票数据
    stock_df = load_stock_data(stock_code, base_dir)

    # 根据日期筛选数据
    if start_date:
        start_date = pd.to_datetime(start_date)
        stock_df = stock_df[stock_df['trade_date'] >= start_date]
    if end_date:
        end_date = pd.to_datetime(end_date)
        stock_df = stock_df[stock_df['trade_date'] <= end_date]

    # 加载市场指数（沪深300）
    index_path = "./assets/data/market/szzs_daily.xls"
    market_index_df = load_market_index(index_path)

    # 将市场指数数据与股票数据合并
    merged_df = pd.merge(stock_df, market_index_df[['trade_date', 'daily_return']],
                    on='trade_date', suffixes=('', '_market'))


    risk_free_rate_df = calculate_risk_free_rate_daily()

    # 和前面的数据日期格式统一
    risk_free_rate_df['date'] = pd.to_datetime(risk_free_rate_df['date'])
    risk_free_rate_df = risk_free_rate_df.rename(columns={'date': 'trade_date'})

    # 合并无风险利率数据
    merged_df = pd.merge(merged_df, risk_free_rate_df, on='trade_date', how='left')

    # 对缺失值进行处理
    if merged_df['rf_rate'].isna().any():
        mean_rf = risk_free_rate_df['rf_rate'].mean()
        merged_df['rf_rate'] = merged_df['rf_rate'].fillna(mean_rf)

    # 超额收益率
    merged_df['excess_return_stock'] = merged_df['daily_return'] - merged_df['rf_rate']
    merged_df['excess_return_market'] = merged_df['daily_return_market'] - merged_df['rf_rate']

    # CAPM回归：Ri - Rf = αi + βi(Rm - Rf) + εi
    X = sm.add_constant(merged_df['excess_return_market'])
    Y = merged_df['excess_return_stock']

    # OLS回归
    model = sm.OLS(Y, X).fit()

    # 回归结果
    alpha = model.params[0]
    beta = model.params[1]
```

```
alpha_pvalue = model.pvalues[0]
beta_pvalue = model.pvalues[1]
r_squared = model.rsquared
adj_r_squared = model.rsquared_adj

# 模型检验
# 1. 异方差性检验 (White's test)
white_test = het_white(model.resid, X)

# 2. 残差的正态性检验
jb_test = stats.jarque_bera(model.resid)

# 3. 残差的自相关性检验
lb_test = acorr_ljungbox(model.resid, lags=[10])

results = {
    "stock_code": stock_code,
    "alpha": alpha,
    "beta": beta,
    "alpha_pvalue": alpha_pvalue,
    "beta_pvalue": beta_pvalue,
    "r_squared": r_squared,
    "adj_r_squared": adj_r_squared,
    "white_test": {
        "statistic": white_test[0],
        "pvalue": white_test[1],
        "reject_homoscedasticity": white_test[1] < 0.05
    },
    "jarque_bera_test": {
        "statistic": jb_test[0],
        "pvalue": jb_test[1],
        "reject_normality": jb_test[1] < 0.05
    },
    "ljung_box_test": {
        "statistic": lb_test.iloc[0, 0],
        "pvalue": lb_test.iloc[0, 1],
        "reject_no_autocorrelation": lb_test.iloc[0, 1] < 0.05
    },
    "regression_summary": model.summary().as_text(),
    "model": model
}

# 解释CAPM检验结果
results["capm_valid"] = alpha_pvalue > 0.05  # 显著程度**，0.05
results["interpretation"] = []

if alpha_pvalue <= 0.05:
    if alpha > 0:
        results["interpretation"].append(f"α={alpha:.6f}显著为正(p={alpha_pvalue:.4f})，表明该股票相对市场提供了超额收益")
    else:
        results["interpretation"].append(f"α={alpha:.6f}显著为负(p={alpha_pvalue:.4f})，表明该股票表现不及市场")
else:
    results["interpretation"].append(f"α={alpha:.6f}不显著(p={alpha_pvalue:.4f})，符合CAPM模型的预期")

if beta_pvalue <= 0.05:
    if beta > 1:
        results["interpretation"].append(f"β={beta:.6f}显著大于1(p={beta_pvalue:.4f})，表明该股票比市场更具波动性")
    elif beta < 1 and beta > 0:
        results["interpretation"].append(f"β={beta:.6f}显著在0和1之间(p={beta_pvalue:.4f})，表明该股票波动性低于市场")
    elif beta < 0:
        results["interpretation"].append(f"β={beta:.6f}显著为负(p={beta_pvalue:.4f})，表明该股票与市场呈负相关")
    else:
        results["interpretation"].append(f"β={beta:.6f}接近1(p={beta_pvalue:.4f})，表明该股票与市场波动性相似")
else:
    results["interpretation"].append(f"β={beta:.6f}不显著(p={beta_pvalue:.4f})，表明该股票风险与市场风险无显著关系")
```

```
    results["interpretation"].append(f"R²={r_squared:.4f}，表明市场因子解释了{r_squared*100:.2f}%的股票收益率变动")

    if white_test[1] < 0.05:
        results["interpretation"].append("存在异方差性，可能影响标准误差的估计")

    if jb_test[1] < 0.05:
        results["interpretation"].append("残差不服从正态分布，可能影响统计推断")

    if lb_test.iloc[0, 1] < 0.05:
        results["interpretation"].append("残差存在自相关性，可能导致OLS估计量效率降低")

    # 检验CAPM模型的假设：零β形式
    # 我们使用二次项检验非线性关系
    X_quadratic = X.copy()
    X_quadratic['squared_market'] = X_quadratic['excess_return_market']**2
    model_quadratic = sm.OLS(Y, X_quadratic).fit()

    # 判断二次项是否显著
    squared_term_pvalue = model_quadratic.pvalues[-1]
    results["non_linearity"] = {
        "squared_term_coef": model_quadratic.params[-1],
        "squared_term_pvalue": squared_term_pvalue,
        "is_significant": squared_term_pvalue < 0.05,
        "r_squared": model_quadratic.rsquared
    }

    if squared_term_pvalue < 0.05:
        results["interpretation"].append("二次项显著，表明市场风险与收益可能存在非线性关系，CAPM线性假设不成立")
    else:
        results["interpretation"].append("二次项不显著，支持CAPM的线性假设")

    return results

single_stock_code = '600054'
single_result = test_capm_time_series(single_stock_code, start_date='2000-01-01', end_date='2022-12-31')
single_result
```

{'stock_code': '600054',
 'alpha': 0.00023717286465442988,
 'beta': 0.9158261811029287,
 'alpha_pvalue': 0.3371390940746921,
 'beta_pvalue': 0.0,
 'r_squared': 0.3559919959080059,
 'adj_r_squared': 0.35587471162388795,
 'white_test': {'statistic': 41.91157128206748,
  'pvalue': 7.92534053777749e-10,
  'reject_homoscedasticity': True},
 'jarque_bera_test': {'statistic': 3805.6127504038227,
  'pvalue': 0.0,
  'reject_normality': True},
 'ljung_box_test': {'statistic': 21.28863495826197,
  'pvalue': 0.019168416645017124,
  'reject_no_autocorrelation': True},
 'regression_summary': '   OLS Regression Results   \n==================
 'model': <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x199937ca990>,
 'capm_valid': True,
 'interpretation': ['α=0.000237不显著(p=0.3371)，符合CAPM模型的预期',
  'β=0.915826显著在0和1之间(p=0.0000)，表明该股票波动性低于市场',
  'R²=0.3560，表明市场因子解释了35.60%的股票收益率变动',
  '存在异方差性，可能影响标准误差的估计',
  '残差不服从正态分布，可能影响统计推断',
  '残差存在自相关性，可能导致OLS估计量效率降低',
  '二次项显著，表明市场风险与收益可能存在非线性关系，CAPM线性假设不成立'],
 'non_linearity': {'squared_term_coef': -1.39677574662689,
  'squared_term_pvalue': 0.001200435974210505,
  'is_significant': True,
  'r_squared': 0.3572214308631625}}

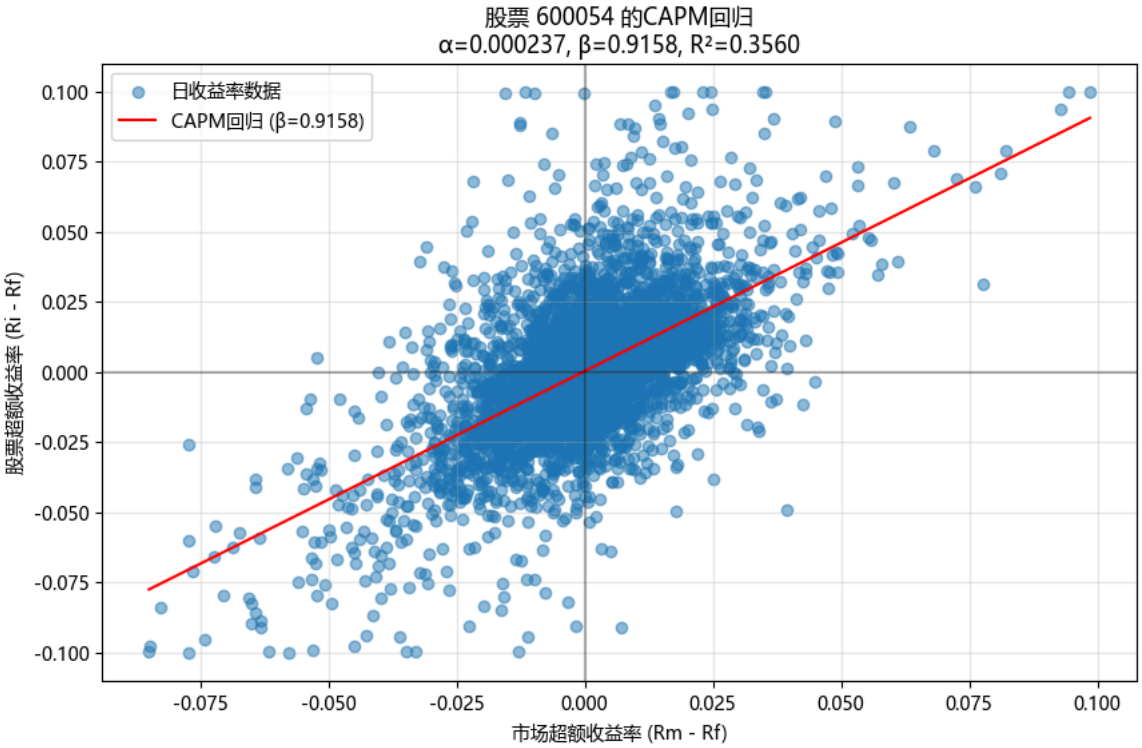
```python
import matplotlib.pyplot as plt
import numpy as np

plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

# 绘制回归图
def plot_capm_regression(results):
    # 提取模型和数据
    model = results["model"]
    X = model.model.exog  # 包含常数项和市场超额收益率
    Y = model.model.endog  # 股票超额收益率

    # 创建绘图
    plt.figure(figsize=(10, 6))

    # 散点图
    plt.scatter(X[:, 1], Y, alpha=0.5, label='日收益率数据')

    # 回归线
    x_range = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x_range_with_const = sm.add_constant(x_range)
    y_pred = model.predict(x_range_with_const)
    plt.plot(x_range, y_pred, 'r-', label=f'CAPM回归 (β={results["beta"]:.4f})')

    # 添加信息
    plt.axhline(y=0, color='k', linestyle='-', alpha=0.3)
    plt.axvline(x=0, color='k', linestyle='-', alpha=0.3)

    # 标题和标签
    plt.title(f'股票 {results["stock_code"]} 的CAPM回归\nα={results["alpha"]:.6f}, β={results["beta"]:.4f}, R²={results["r_squared"]:.4f}')
    plt.xlabel('市场超额收益率 (Rm - Rf)')
    plt.ylabel('股票超额收益率 (Ri - Rf)')
    plt.grid(True, alpha=0.3)
    plt.legend()

    plt.show()

plot_capm_regression(single_result)
```



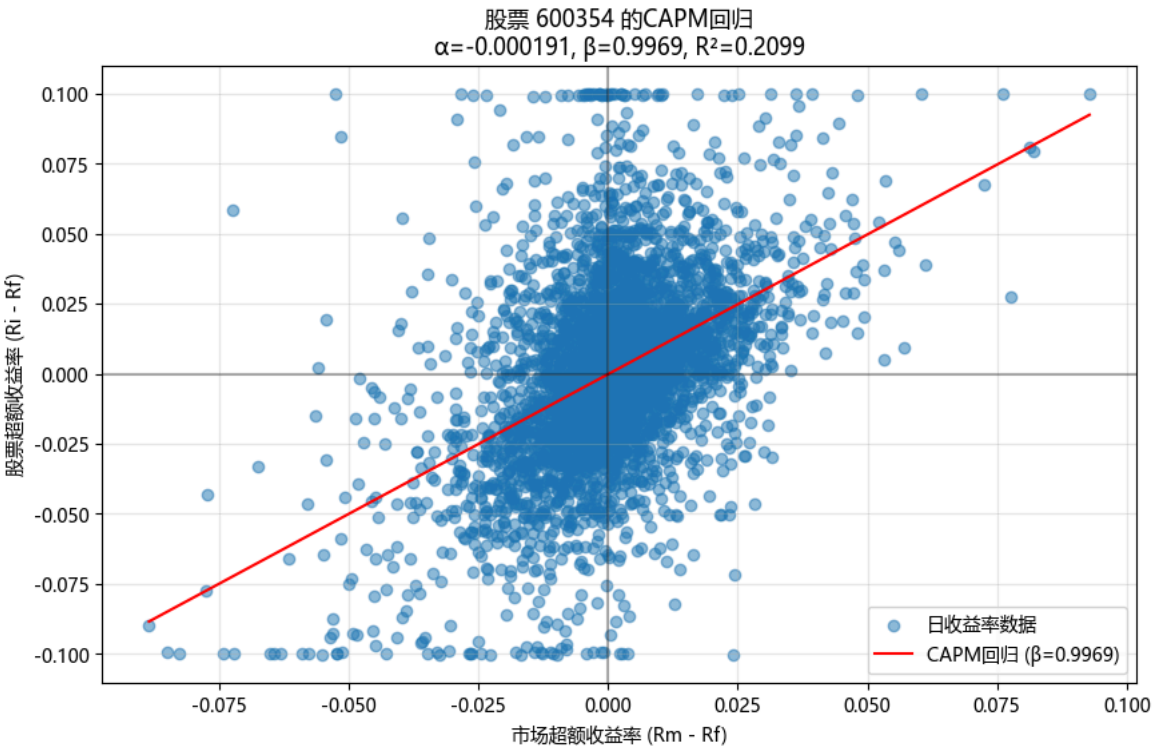

```python
# 多个一起检验
def test_all_stocks(stock_codes, market_index_path=None, risk_free_rate=None,
                    base_dir="./assets/data/stock", start_date=None, end_date=None):
    # 只要加载一次市场指数，可以复用
    market_index_df = None

    # 对每个股票进行检验
    results = {}
    for code in stock_codes:
        print(f"正在检验股票 {code}...")
        stock_result = test_capm_time_series(
            code, market_index_df, risk_free_rate,
            base_dir, start_date, end_date
        )
        results[code] = stock_result

        # 绘制图片
        plot_capm_regression(stock_result)

    return results
```

```python
# 汇总结果
def summarize_results(results):
    summary = []

    for code, result in results.items():
        if "error" in result:
            row = {
                "股票代码": code,
                "错误": result["error"],
                "Alpha": None,
                "Beta": None,
                "R²": None,
                "CAPM是否成立": None
            }
        else:
            row = {
                "股票代码": code,
                "Alpha": result["alpha"],
                "Alpha P值": result["alpha_pvalue"],
                "Beta": result["beta"],
                "Beta P值": result["beta_pvalue"],
                "R²": result["r_squared"],
                "CAPM是否成立": result["capm_valid"],
                "异方差性": result["white_test"]["reject_homoscedasticity"],
                "残差非正态": result["jarque_bera_test"]["reject_normality"],
                "残差自相关": result["ljung_box_test"]["reject_no_autocorrelation"],
                "非线性关系": result["non_linearity"]["is_significant"]
            }
        summary.append(row)

    df = pd.DataFrame(summary)

    return df


# 股票代码
stock_codes = ['600054', '600354', '600654', '600754', '600854', '000554', '002054', '002154']

# 测试时间范围
start_date = '2000-01-01'
end_date = '2022-12-31'

# 执行CAPM检验
all_results = test_all_stocks(
    stock_codes,
    start_date=start_date,
    end_date=end_date
)

# 汇总结果
summary_df = summarize_results(all_results)
summary_df
```

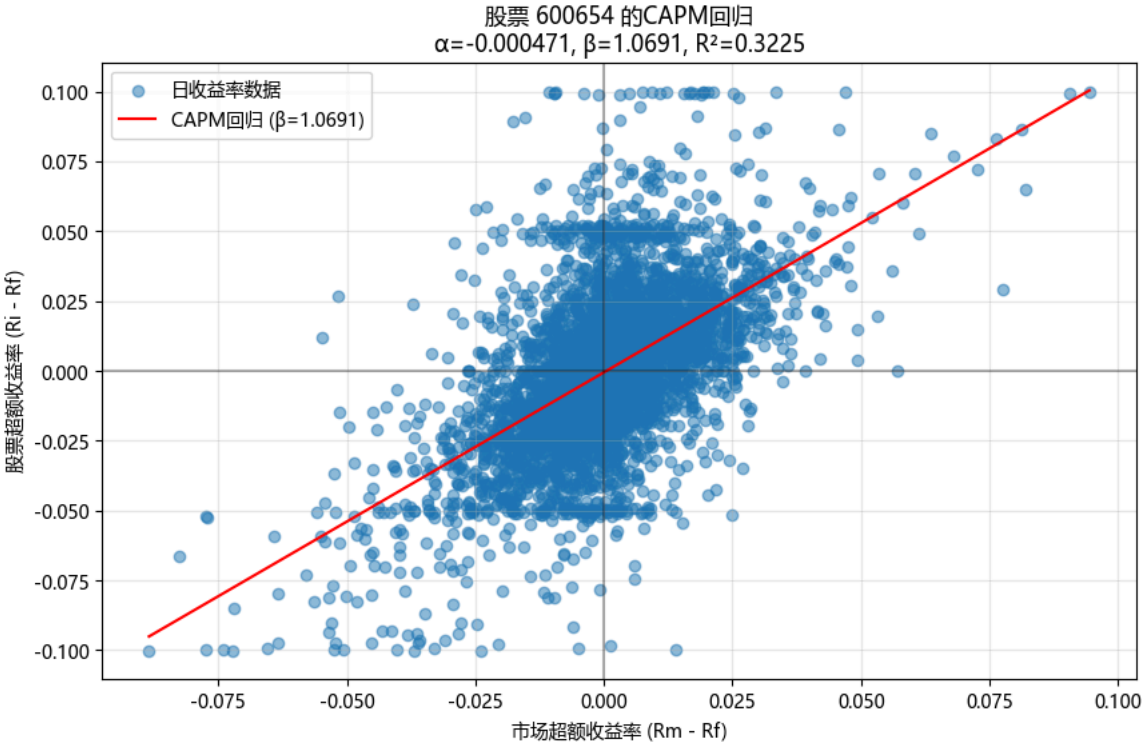
正在检验股票 600054...



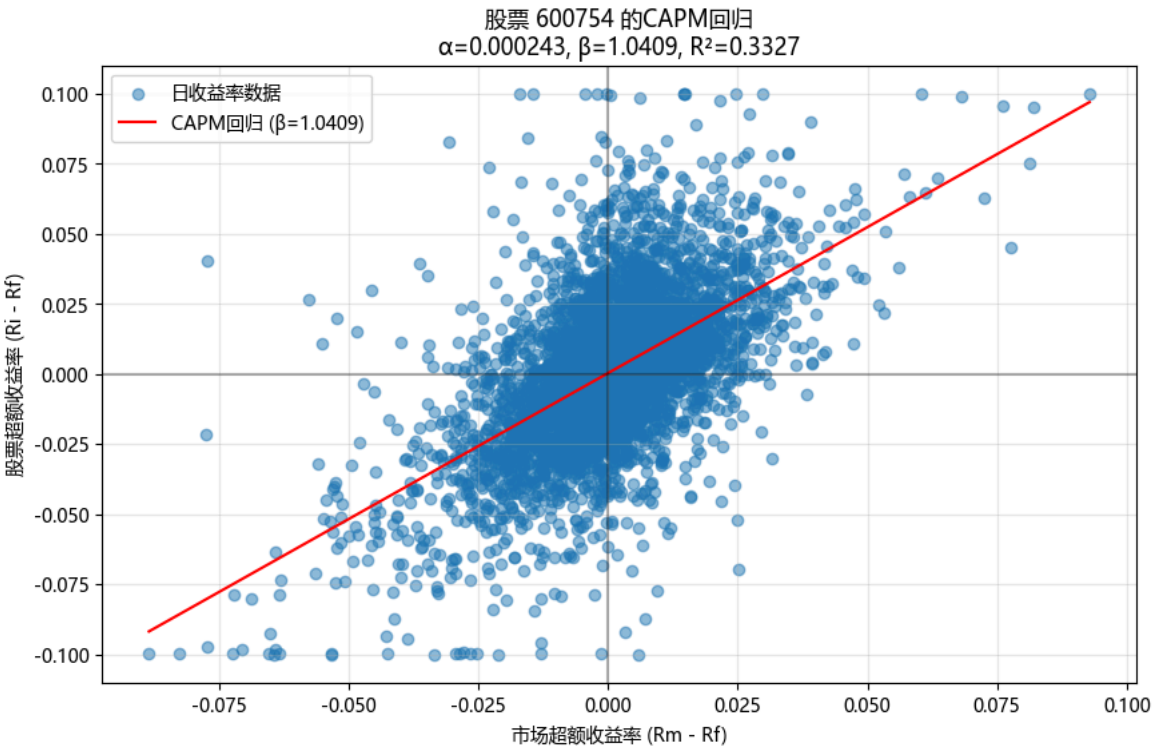
正在检验股票 600354...



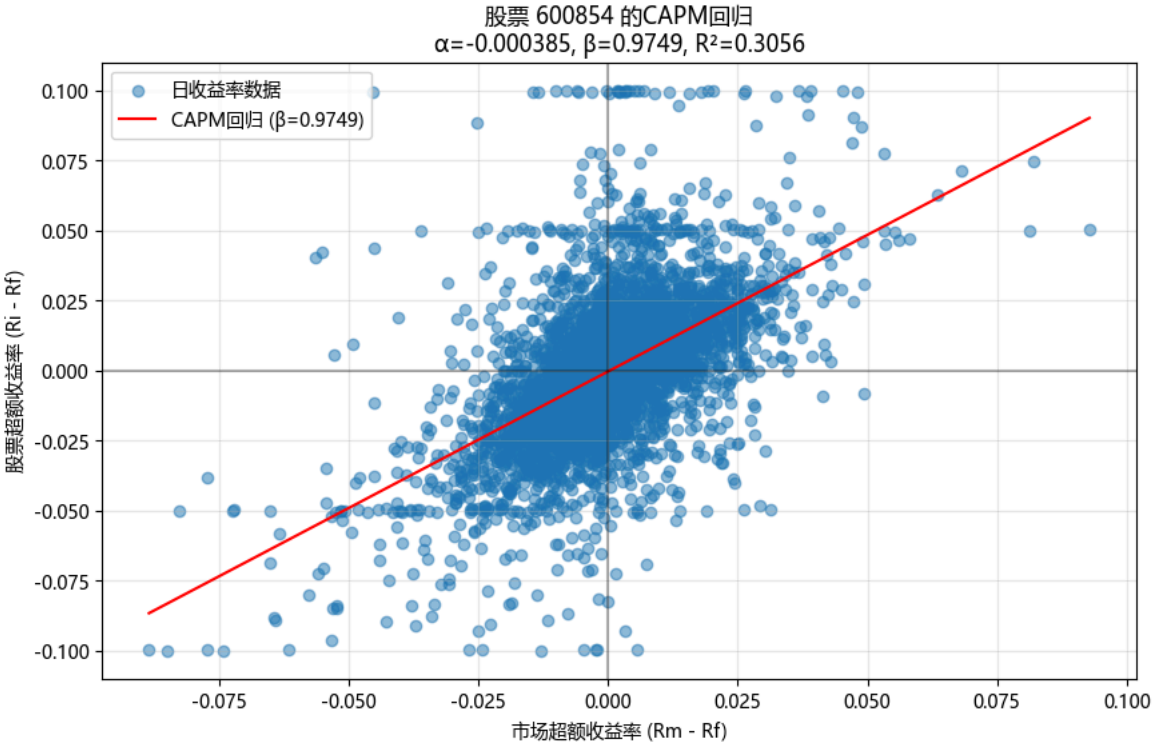
正在检验股票 600654...



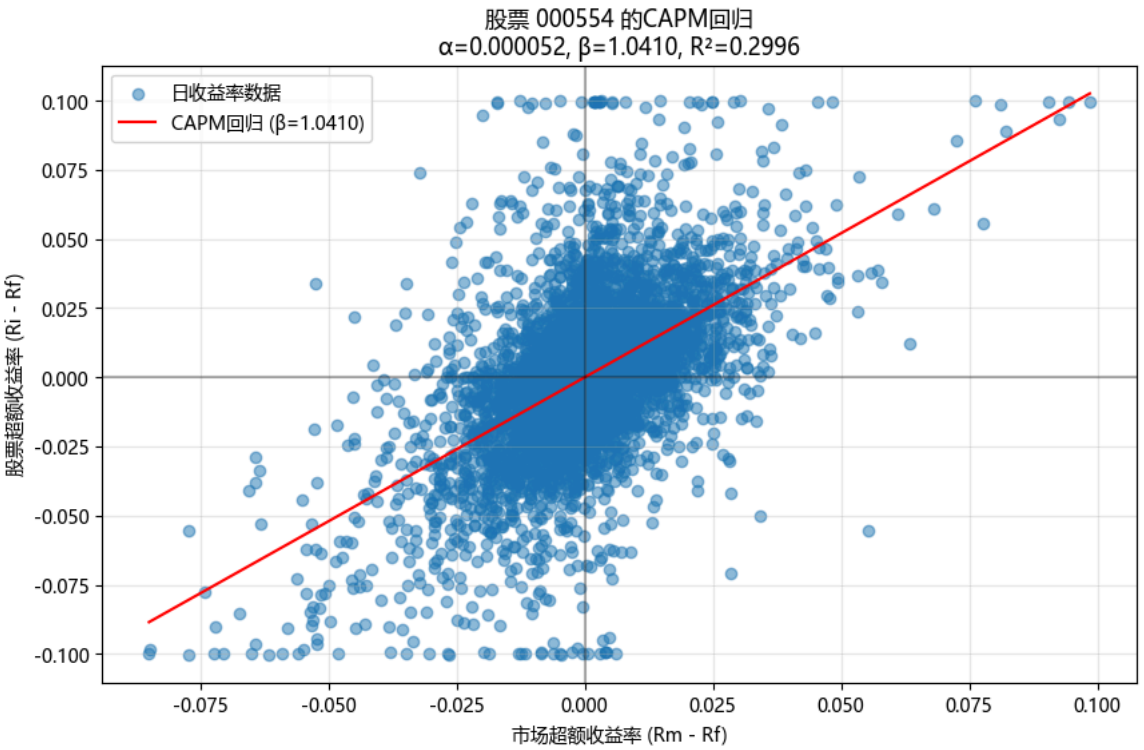
正在检验股票 600754...



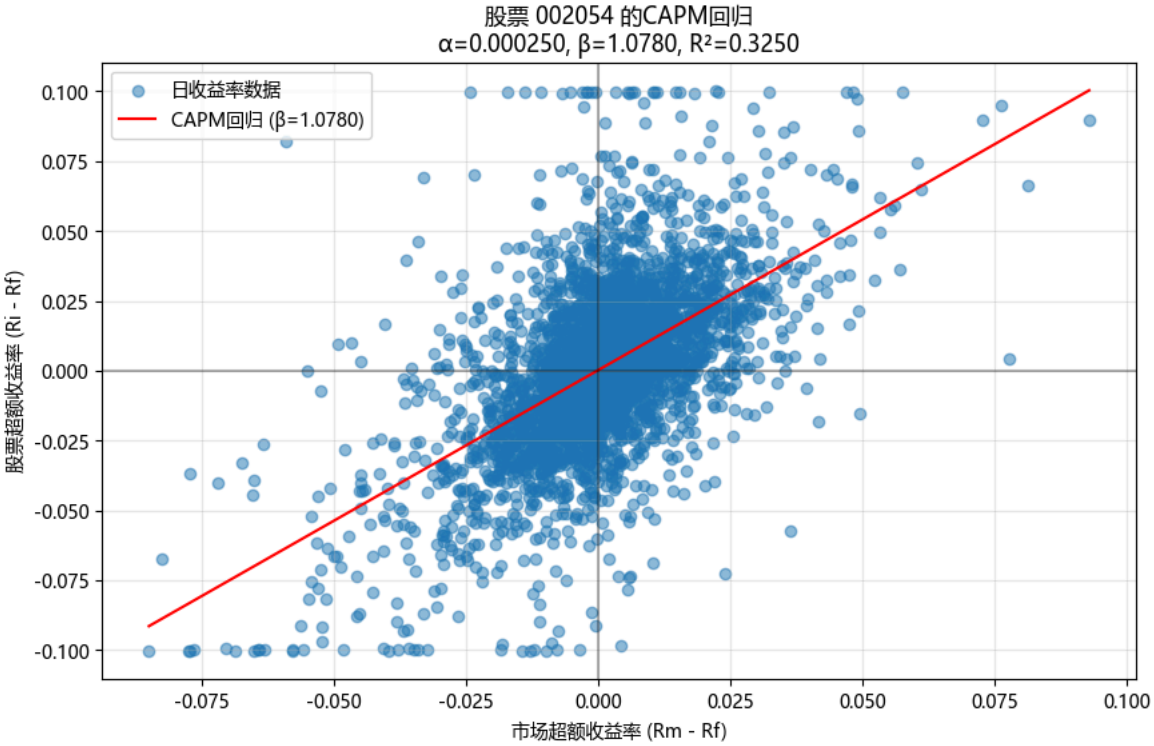
正在检验股票 600854...



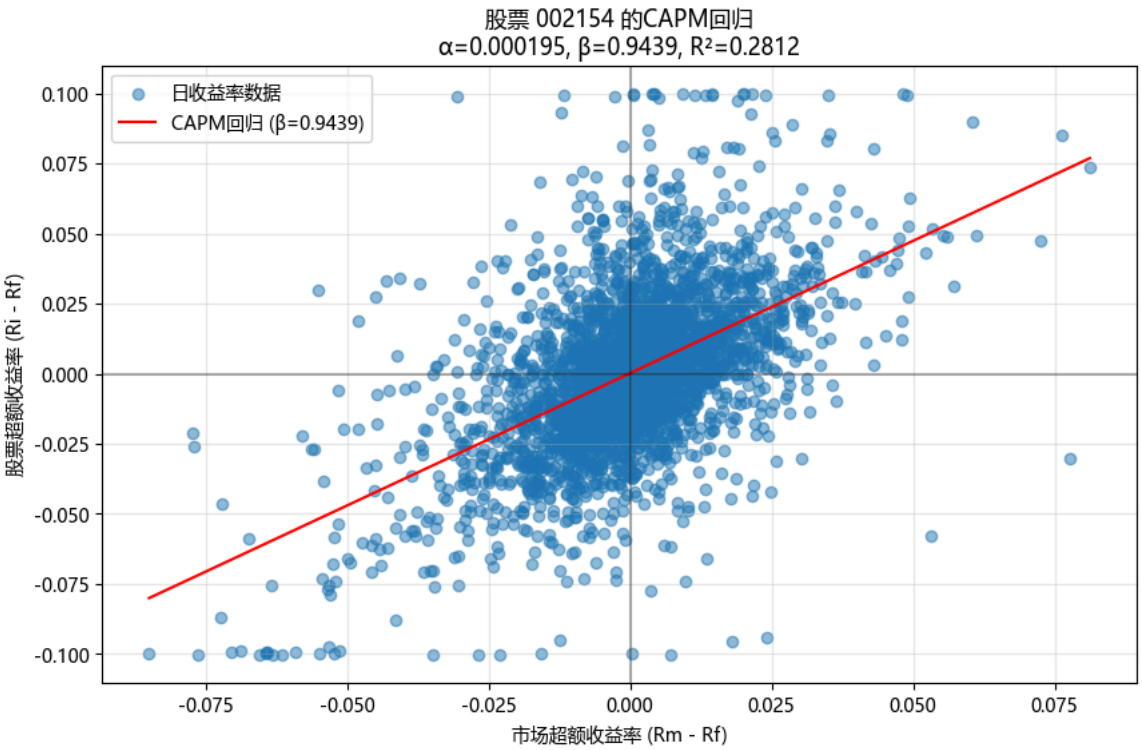
正在检验股票 `000554...`



正在检验股票 `002054...`



正在检验股票 `002154...`



	股票代码	Alpha	Alpha P值	Beta	Beta P值	R²	CAPM是否成立	异方差性	残差非正态	残差自相关	非
0	600054	0.000237	0.337139	0.915826	0.000000e+00	0.355992	True	True	True	True	T
1	600354	-0.000191	0.658322	0.996877	6.773955e-227	0.209900	True	True	True	True	F
2	600654	-0.000471	0.141689	1.069081	0.000000e+00	0.322469	True	True	True	True	T

	股票代码	Alpha	Alpha P值	Beta	Beta P值	R^2	CAPM是否成立	异方差性	残差非正态	残差自相关	非
3	600754	0.000243	0.401964	1.040942	0.000000e+00	0.332724	True	True	True	True	F
4	600854	-0.000385	0.173864	0.974885	0.000000e+00	0.305619	True	True	True	True	T
5	000554	0.000052	0.869651	1.040979	0.000000e+00	0.299610	True	True	True	True	T
6	002054	0.000250	0.506489	1.078045	0.000000e+00	0.325020	True	True	True	True	T
7	002154	0.000195	0.595806	0.943878	3.966296e-266	0.281159	True	True	True	True	T

结果解读：

1. **Alpha 值**：
   - 所有股票的 Alpha 值接近零，且 p 值均大于 0.05，表明没有显著的超额收益
2. **Beta 值**：
   - 所有股票 Beta 值接近 1（0.94-1.08 之间），且 p 值极小，表明与市场走势高度相关
   - Beta>1 的股票（如 600654、002054 等）比市场略微波动性更大
3. **R^2 值**：
   - R^2 值在 0.21-0.36 之间，表明市场因素仅能解释这些股票 21-36%的收益率变动
   - 其他非市场因素对这些股票影响较大
4. **模型检验**：
   - 尽管 CAPM 形式上成立（Alpha 不显著），但几乎所有股票都存在异方差性、残差非正态和自相关问题
   - 大部分股票还存在非线性关系问题（6 只为 True，2 只为 False）

结论：

这些股票与大盘走势基本同步，没有显著异常回报。但统计检验问题表明，简单 CAPM 模型无法完全解释这些股票的收益模式，可能需要考虑更复杂的多因素模型。

利用股票月收益数据对 CAPM 模型做多资产时间序列检验

```
# 加载月度数据
def load_stock_monthly_data(stock_code, base_dir="./assets/data/stock"):
    monthly_path = os.path.join(base_dir, stock_code, "monthly", f"{stock_code}_monthly_returns.csv")

    # 读取月收益率数据
    df = pd.read_csv(monthly_path)

    # 将交易日期转换为datetime格式
    df['trade_date'] = pd.to_datetime(df['trade_date'], format='%Y%m%d')
    df = df.sort_values('trade_date')  # 按日期排序

    df = df.dropna()

    # 删除monthly_return大于0.5或小于-0.5的行
    df = df[(df['monthly_return'] >= -0.) & (df['monthly_return'] <= 0.5)]

    return df

test_df = load_stock_monthly_data('000554')
test_df
```

	ts_code	trade_date	year_month	open	close	high	low	vol	amount	first_day	last_day	m
0	000554.SZ	2000-01-01	200001	4.35	4.42	5.02	4.20	495956.00	2.275593e+05	20000104	20000128	0.(

	ts_code	trade_date	year_month	open	close	high	low	vol	amount	first_day	last_day	m
1	000554.SZ	2000-02-01	200002	4.43	4.87	5.28	4.43	730117.00	3.524337e+05	20000214	20000229	0.
2	000554.SZ	2000-03-01	200003	4.87	5.48	5.58	4.68	1323940.00	6.720840e+05	20000301	20000331	0.
3	000554.SZ	2000-04-01	200004	5.50	6.08	6.44	5.20	1748156.00	1.036279e+06	20000403	20000428	0.
4	000554.SZ	2000-05-01	200005	6.10	8.07	8.19	6.10	1686216.00	1.207961e+06	20000508	20000531	0.
...	...	...	...	...	...	...	...	...	...	...	...	...
264	000554.SZ	2022-02-01	202202	5.65	6.19	6.53	5.53	5400110.53	3.236588e+06	20220207	20220228	0.
267	000554.SZ	2022-05-01	202205	4.60	5.89	6.42	4.54	3728455.87	2.052676e+06	20220505	20220531	0.
269	000554.SZ	2022-07-01	202207	5.26	5.47	5.61	4.97	2418218.71	1.268595e+06	20220701	20220729	0.
270	000554.SZ	2022-08-01	202208	5.45	5.53	6.39	5.23	4624893.02	2.663975e+06	20220801	20220831	0.
273	000554.SZ	2022-11-01	202211	4.91	5.74	5.82	4.87	2812866.98	1.526250e+06	20221101	20221130	0.

140 rows × 12 columns

```python
# 月度指数我们采用手动计算的方式，因为只给了日度数据
def load_market_monthly_index(index_path="./assets/data/market/szzs_daily.xls"):
    df = pd.read_excel(index_path)

    column_mapping = {
        '指数代码_IdxCd': 'index_code',
        '指数名称_IdxNm': 'index_name',
        '交易日期_TrdDt': 'trade_date',
        '收盘价(元/点)_ClPr': 'close',
        '涨跌幅(%)_ChgPct': 'daily_return_pct'
    }

    # 重命名存在的列
    for old_name, new_name in column_mapping.items():
        if old_name in df.columns:
            df = df.rename(columns={old_name: new_name})

    # 检查日期格式，处理各种可能的格式
    date_sample = str(df['trade_date'].iloc[0])
    if '/' in date_sample or '-' in date_sample:
        df['trade_date'] = pd.to_datetime(df['trade_date'])
    else:
        df['trade_date'] = pd.to_datetime(df['trade_date'], format='%Y%m%d')

    # 按日期排序
    df = df.sort_values('trade_date')

    # 使用已有的涨跌幅数据，转换为小数
    if 'daily_return_pct' in df.columns:
        df['daily_return'] = df['daily_return_pct'] / 100
    else:
        # 如果没有涨跌幅列，则计算日收益率
        df['daily_return'] = df['close'].pct_change()

    df = df[(df['daily_return'] >= -0.1) & (df['daily_return'] <= 0.1)]

    # 提取年月信息用于分组
    df['year_month'] = df['trade_date'].dt.strftime('%Y-%m')

    # 计算月度收益率，每月最后一天收盘价相对上月最后一天收盘价的变化
    monthly_df = df.groupby('year_month').agg({
        'trade_date': 'last',     # 取每月最后一个交易日
        'close': 'last',          # 取月末收盘价
        'index_code': 'first',    # 保留指数代码
        'index_name': 'first'     # 保留指数名称
    }).reset_index()

    monthly_df['monthly_return'] = monthly_df['close'].pct_change()

    # 创建最终输出数据框
    result_df = monthly_df[['trade_date', 'index_code', 'index_name', 'close', 'monthly_return']].copy()

    result_df = result_df.dropna(subset=['monthly_return'])

    return result_df

load_market_monthly_index()
```

	trade_date	index_code	index_name	close	monthly_return
1	2000-02-29	1	上证指数	1714.5780	0.116991
2	2000-03-31	1	上证指数	1800.2250	0.049952

	trade_date	index_code	index_name	close	monthly_return
3	2000-04-28	1	上证指数	1836.3210	0.020051
4	2000-05-31	1	上证指数	1894.5540	0.031712
5	2000-06-30	1	上证指数	1928.1060	0.017710
...	...	...	...	...	...
271	2022-08-31	1	上证指数	3202.1378	-0.015708
272	2022-09-30	1	上证指数	3024.3905	-0.055509
273	2022-10-31	1	上证指数	2893.4829	-0.043284
274	2022-11-30	1	上证指数	3151.3353	0.089115
275	2022-12-30	1	上证指数	3089.2579	-0.019699

275 rows × 5 columns

```
# 月度无风险收益率
def calculate_monthly_risk_free_rate(treasury_bill_path="./assets/data/rf_monthly.xls"):
    # 读取Excel文件
    df = pd.read_excel(treasury_bill_path)

    # 统一列名
    column_renames = {'日期_Date': 'date', '月无风险收益率_MonRFRet': 'rf_rate'}
    df.rename(columns=column_renames, inplace=True)


    # 格式化日期列
    df['date'] = pd.to_datetime(df['date'], errors='coerce')
    return df

calculate_monthly_risk_free_rate()
```

	date	rf_rate
0	2000-01-01	0.001650
1	2000-02-01	0.001650
2	2000-03-01	0.001650
3	2000-04-01	0.001650
4	2000-05-01	0.001650
...	...	...
271	2022-08-01	0.001366
272	2022-09-01	0.001342
273	2022-10-01	0.001413
274	2022-11-01	0.001676
275	2022-12-01	0.001931

276 rows × 2 columns

```python
def prepare_panel_data(stock_codes, base_dir, start_date=None, end_date=None, fill_method='ffill'):
    """
    加载所有数据

    参数:
    stock_codes (list): 股票代码列表
    base_dir (str): 数据存储的基础目录
    start_date (str): 开始日期，格式为'YYYY-MM-DD'
    end_date (str): 结束日期，格式为'YYYY-MM-DD'
    fill_method (str): 缺失值填充方法，'ffill'表示前向填充，'mean'表示均值填充，None表示不填充

    返回:
    tuple: (基础面板数据, 股票数据字典, 有效股票代码列表)
    """
    # 日期筛选条件
    if start_date:
        start_date = pd.to_datetime(start_date)
    if end_date:
        end_date = pd.to_datetime(end_date)

    # 1. 收集所有可能出现的年月
    all_months = set()

    # 2. 加载市场数据
    market_df = load_market_monthly_index()
    market_df['year_month'] = market_df['trade_date'].dt.strftime('%Y-%m')

    if start_date:
        market_df = market_df[market_df['trade_date'] >= start_date]
    if end_date:
        market_df = market_df[market_df['trade_date'] <= end_date]

    all_months.update(market_df['year_month'])

    # 3. 加载无风险利率数据
    risk_free_df = calculate_monthly_risk_free_rate()
    if 'date' in risk_free_df.columns:
        risk_free_df['date'] = pd.to_datetime(risk_free_df['date'])
        risk_free_df = risk_free_df.rename(columns={'date': 'trade_date'})

    risk_free_df['year_month'] = risk_free_df['trade_date'].dt.strftime('%Y-%m')


    # 4. 加载每只股票数据
    stock_data_dict = {}

    for code in stock_codes:
        stock_df = load_stock_monthly_data(code, base_dir)

        # 处理缺失值
        stock_df['trade_date'] = pd.to_datetime(stock_df['trade_date'])
        stock_df['year_month'] = stock_df['trade_date'].dt.strftime('%Y-%m')

        if start_date:
            stock_df = stock_df[stock_df['trade_date'] >= start_date]
        if end_date:
            stock_df = stock_df[stock_df['trade_date'] <= end_date]

        # 收集年月
        all_months.update(stock_df['year_month'])

        # 保存股票数据
        stock_data_dict[code] = stock_df

    # 5. 创建基础面板数据框
```

```
all_months = sorted(list(all_months))
panel_data = pd.DataFrame({'year_month': all_months})

# 6. 合并市场收益率
market_data = market_df[['year_month', 'monthly_return']].rename(columns={'monthly_return': 'market_return'})
panel_data = pd.merge(panel_data, market_data, on='year_month', how='left')

# 7. 合并无风险利率
panel_data = pd.merge(panel_data, risk_free_df[['year_month', 'rf_rate']], on='year_month', how='left')

# 8. 填充缺失的市场数据和无风险利率
if fill_method:
    # 填充市场收益率
    if panel_data['market_return'].isna().any():
        if fill_method == 'ffill':
            panel_data['market_return'] = panel_data['market_return'].fillna(method='ffill')
            panel_data['market_return'] = panel_data['market_return'].fillna(method='bfill')
        elif fill_method == 'mean':
            mean_market = panel_data['market_return'].mean()
            panel_data['market_return'] = panel_data['market_return'].fillna(mean_market)

    # 填充无风险利率
    if panel_data['rf_rate'].isna().any():
        if fill_method == 'ffill':
            panel_data['rf_rate'] = panel_data['rf_rate'].fillna(method='ffill')
            panel_data['rf_rate'] = panel_data['rf_rate'].fillna(method='bfill')
        elif fill_method == 'mean':
            mean_rf = panel_data['rf_rate'].mean()
            panel_data['rf_rate'] = panel_data['rf_rate'].fillna(mean_rf)

# 检查利率列是否完整
if panel_data['market_return'].isna().any() or panel_data['rf_rate'].isna().any():
    print("警告: 市场收益率或无风险利率存在缺失值")
    # 删除有缺失的行
    panel_data = panel_data.dropna(subset=['market_return', 'rf_rate'])
    print(f"删除缺失值后的数据形状: {panel_data.shape}")

# 识别有效的股票代码
valid_stocks = list(stock_data_dict.keys())

return panel_data, stock_data_dict, valid_stocks
```

```python
from scipy.stats import chi2

# CAPM联合检验
def perform_joint_tests(panel_data, stock_data_dict, valid_stocks, fill_method='ffill'):
    # 使用所有可能的年月，使用外连接，交集会导致数据集非常少
    all_stock_months = set()

    for code in valid_stocks:
        stock_df = stock_data_dict[code]
        months = set(stock_df['year_month'])
        all_stock_months.update(months)

    # 创建包含所有月份的面板
    common_panel = panel_data[panel_data['year_month'].isin(all_stock_months)].copy()

    # 为每只股票添加收益率，使用左连接
    for code in valid_stocks:
        stock_df = stock_data_dict[code]
        stock_returns = stock_df[['year_month', 'monthly_return']].copy()
        stock_returns = stock_returns.rename(columns={'monthly_return': f'return_{code}'})
        common_panel = pd.merge(common_panel, stock_returns, on='year_month', how='left')

    # 填充缺失的股票收益率数据
    for code in valid_stocks:
        if common_panel[f'return_{code}'].isna().any():
            if fill_method == 'ffill':
                common_panel[f'return_{code}'] = common_panel[f'return_{code}'].fillna(method='ffill')
                common_panel[f'return_{code}'] = common_panel[f'return_{code}'].fillna(method='bfill')
            elif fill_method == 'mean':
                mean_return = common_panel[f'return_{code}'].mean(skipna=True)
                common_panel[f'return_{code}'] = common_panel[f'return_{code}'].fillna(mean_return)
            else:
                # 如果没有指定填充方法，使用全局均值
                mean_return = stock_data_dict[code]['monthly_return'].mean()
                common_panel[f'return_{code}'] = common_panel[f'return_{code}'].fillna(mean_return)

        # 计算超额收益率
        common_panel[f'excess_return_{code}'] = common_panel[f'return_{code}'] - common_panel['rf_rate']

    # 计算市场超额收益率
    common_panel['excess_market_return'] = common_panel['market_return'] - common_panel['rf_rate']

    # 删除市场和无风险利率缺失的行
    common_panel = common_panel.dropna(subset=['market_return', 'rf_rate'])

    # 残差矩阵
    N = len(valid_stocks)
    resid_matrix = np.zeros((len(common_panel), N))
    alpha_hat_common = np.zeros(N)
    beta_hat_common = np.zeros(N)

    # 对每只股票进行回归，保存残差
    for i, code in enumerate(valid_stocks):
        y = common_panel[f'excess_return_{code}'].values
        X = sm.add_constant(common_panel['excess_market_return'].values)

        model = sm.OLS(y, X).fit()

        alpha_hat_common[i] = model.params[0]
        beta_hat_common[i] = model.params[1]
        resid_matrix[:, i] = model.resid

    # 计算残差协方差矩阵
    sigma_hat = np.dot(resid_matrix.T, resid_matrix) / len(common_panel)
```

```python
# Wald检验
mu_market = np.mean(common_panel['excess_market_return'])
sigma_market = np.var(common_panel['excess_market_return'])

wald_stat = (len(common_panel) - N - 1) / N * (1 + mu_market**2 / sigma_market) * \
            np.dot(np.dot(alpha_hat_common, np.linalg.inv(sigma_hat)), alpha_hat_common)

wald_pvalue = 1 - chi2.cdf(wald_stat, N)

# 似然比检验
log_L_unrestricted = -len(common_panel)/2 * (N * np.log(2*np.pi) + np.log(np.linalg.det(sigma_hat)) + N)

# 约束模型（α=0）
restricted_resid_matrix = np.zeros((len(common_panel), N))
for i, code in enumerate(valid_stocks):
    y = common_panel[f'excess_return_{code}'].values
    X = common_panel['excess_market_return'].values.reshape(-1, 1)

    beta_restricted = np.dot(np.linalg.inv(np.dot(X.T, X)), np.dot(X.T, y))
    y_pred = np.dot(X, beta_restricted)
    restricted_resid_matrix[:, i] = y - y_pred

sigma_hat_restricted = np.dot(restricted_resid_matrix.T, restricted_resid_matrix) / len(common_panel)

log_L_restricted = -len(common_panel)/2 * (N * np.log(2*np.pi) + np.log(np.linalg.det(sigma_hat_restricted)) + N)

lr_stat = -2 * (log_L_restricted - log_L_unrestricted)

lr_pvalue = 1 - chi2.cdf(lr_stat, N)

# 拉格朗日乘子检验
T = len(common_panel)  # 样本量
lm_stat = 0

# 为每个股票执行辅助回归
for i, code in enumerate(valid_stocks):
    # 获取约束模型(Alpha=0)的残差
    y_resid = restricted_resid_matrix[:, i]

    # 创建包含常数项的设计矩阵
    X_aux = sm.add_constant(common_panel['excess_market_return'].values)

    # 将残差对常数和市场超额收益率进行回归
    aux_model = sm.OLS(y_resid, X_aux).fit()

    # 计算辅助回归的R^2
    r_squared = aux_model.rsquared

    # 计算每个资产的LM统计量并累加
    lm_stat += T * r_squared

# 计算LM p值
lm_pvalue = 1 - chi2.cdf(lm_stat, N)

# 联合检验结果
joint_tests = {
    'wald_test': {
        'statistic': float(wald_stat),
        'pvalue': float(wald_pvalue),
        'reject_null': float(wald_pvalue) < 0.05
    },
    'lr_test': {
        'statistic': float(lr_stat),
        'pvalue': float(lr_pvalue),
        'reject_null': float(lr_pvalue) < 0.05
```

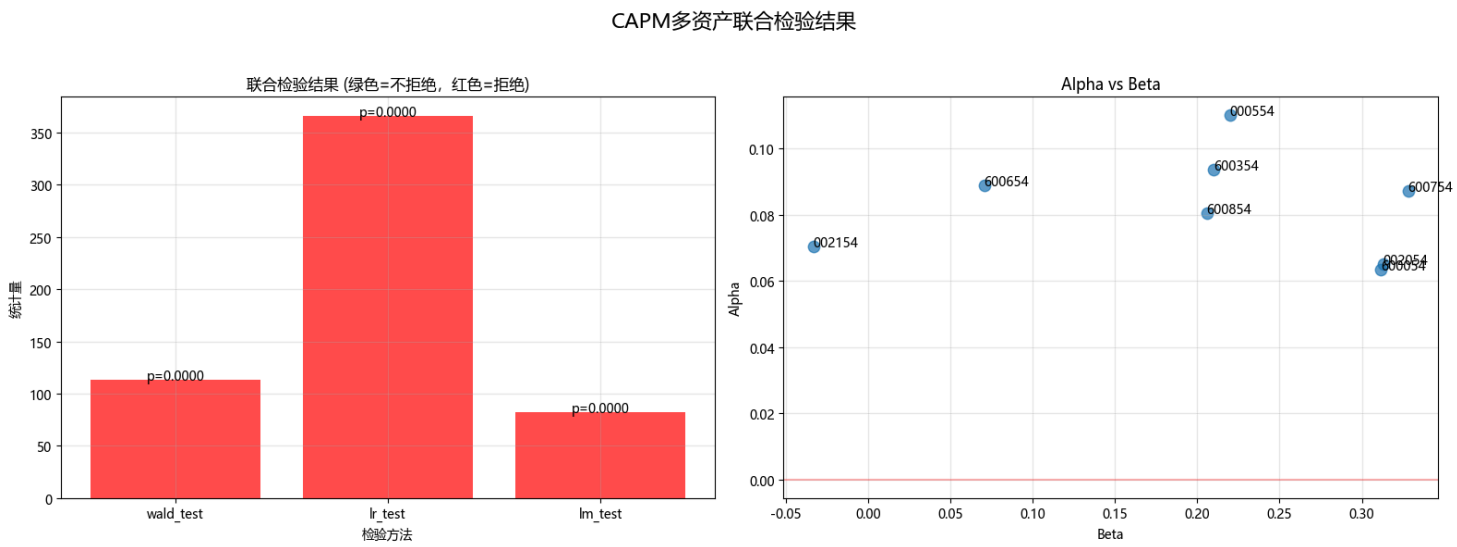
```
        },
        'lm_test': {
            'statistic': float(lm_stat),
            'pvalue': float(lm_pvalue),
            'reject_null': float(lm_pvalue) < 0.05
        },
        'common_sample_size': len(common_panel),
        'alpha_hat': alpha_hat_common.tolist(),
        'beta_hat': beta_hat_common.tolist()
    }

    return joint_tests


# 画图
def plot_joint_tests_results(joint_tests, valid_stocks):
    fig, axes = plt.subplots(1, 2, figsize=(16, 6))

    # 1. 联合检验结果
    test_names = []
    stats = []
    pvalues = []

    for test_name, test_info in joint_tests.items():
        if test_name not in ['error', 'common_sample_size', 'alpha_hat', 'beta_hat'] and isinstance(test_info, dict) and 'statistic' in test_info
            test_names.append(test_name)
            stats.append(test_info['statistic'])
            pvalues.append(test_info['pvalue'])

    if test_names:
        bar_colors = ['green' if p > 0.05 else 'red' for p in pvalues]
        bars = axes[0].bar(test_names, stats, alpha=0.7, color=bar_colors)

        for i, p in enumerate(pvalues):
            axes[0].text(i, stats[i] + 0.1, f'p={p:.4f}', ha='center')

        axes[0].set_xlabel('检验方法')
        axes[0].set_ylabel('统计量')
        axes[0].set_title('联合检验结果 (绿色=不拒绝，红色=拒绝)')
        axes[0].grid(True, alpha=0.3)

    # 2. Alpha值散点图
    if 'alpha_hat' in joint_tests and 'beta_hat' in joint_tests:
        alphas = joint_tests['alpha_hat']
        betas = joint_tests['beta_hat']

        axes[1].scatter(betas, alphas, alpha=0.7, s=80)
        for i, code in enumerate(valid_stocks):
            axes[1].annotate(code, (betas[i], alphas[i]), fontsize=10)

        axes[1].axhline(y=0, color='r', linestyle='-', alpha=0.3)
        axes[1].set_xlabel('Beta')
        axes[1].set_ylabel('Alpha')
        axes[1].set_title('Alpha vs Beta')
        axes[1].grid(True, alpha=0.3)

    # 整体标题
    plt.suptitle('CAPM多资产联合检验结果', fontsize=16)
    plt.tight_layout(rect=[0, 0, 1, 0.95])

    return fig
```

```python
# 解释检验结果
def interpret_joint_tests(joint_tests, valid_stocks):
    interpretations = []

    if isinstance(joint_tests, dict) and 'error' not in joint_tests:
        # 检查是否所有测试都不拒绝原假设
        all_tests_agree = True
        any_test_rejects = False

        for test_name, test_results in joint_tests.items():
            # 跳过非测试结果的键
            if test_name in ['error', 'common_sample_size', 'alpha_hat', 'beta_hat']:
                continue

            # 确保test_results是字典而不是整数或其他类型
            if isinstance(test_results, dict) and 'reject_null' in test_results:
                if test_results['reject_null']:
                    any_test_rejects = True
                    all_tests_agree = False
            else:
                all_tests_agree = False

        if all_tests_agree and not any_test_rejects:
            interpretations.append("所有联合检验（Wald、LR、LM）均不拒绝原假设，表明CAPM模型在联合检验下成立。")
            interpretations.append("所有股票的Alpha值联合为零，符合CAPM模型的预期。")
            capm_valid = True
        elif any_test_rejects:
            rejection_tests = []
            for test_name, test_results in joint_tests.items():
                # 确保test_results是字典并且包含reject_null键
                if test_name not in ['error', 'common_sample_size', 'alpha_hat', 'beta_hat'] and isinstance(test_results, dict) and 'reject_null'
                    rejection_tests.append(test_name)

            if rejection_tests:
                interpretations.append(f"以下检验拒绝原假设：{', '.join(rejection_tests)}，表明CAPM模型在联合检验下不成立。")
                interpretations.append("至少有部分股票存在非零Alpha值，不符合CAPM模型的预期。")
                capm_valid = False
            else:
                interpretations.append("联合检验结果不一致，部分检验支持CAPM模型，部分检验不支持。")
                capm_valid = None
        else:
            interpretations.append("联合检验未给出明确结论。")
            capm_valid = None
    else:
        interpretations.append("未能进行联合检验。")
        capm_valid = None

    return interpretations, capm_valid
```

```python
def test_capm_joint(stock_codes, base_dir="./assets/data/stock", start_date=None, end_date=None, fill_method='ffill'):
    # 1. 准备面板数据
    panel_data, stock_data_dict, valid_stocks = prepare_panel_data(
        stock_codes=stock_codes,
        base_dir=base_dir,
        start_date=start_date,
        end_date=end_date,
        fill_method=fill_method
    )

    # 2. 直接进行联合检验
    joint_tests = perform_joint_tests(
        panel_data=panel_data,
        stock_data_dict=stock_data_dict,
        valid_stocks=valid_stocks,
        fill_method=fill_method
    )

    # 3. 解释结果
    interpretations, capm_valid = interpret_joint_tests(
        joint_tests=joint_tests,
        valid_stocks=valid_stocks
    )

    # 4. 返回结果
    return {
        'stock_codes': valid_stocks,
        'joint_tests': joint_tests,
        'interpretations': interpretations,
        'capm_valid': capm_valid
    }
```

```python
# 完整流程
def run_capm_joint_analysis(stock_codes, start_date=None, end_date=None, base_dir="./assets/data/stock"):
    print("\n====== CAPM联合检验分析 ======")
    print(f"分析股票: {', '.join(stock_codes)}")
    print(f"日期范围: {start_date or '所有'} 到 {end_date or '所有'}")
    print("================================\n")

    # 1. 进行CAPM联合检验
    results = test_capm_joint(
        stock_codes=stock_codes,
        base_dir=base_dir,
        start_date=start_date,
        end_date=end_date,
        fill_method='ffill'
    )

    # 检查是否有错误
    if 'error' in results:
        print(f"\n错误: {results['error']}")
        return results

    # 2. 打印结果
    print("\n=== 联合检验结果 ===")
    if 'joint_tests' in results and isinstance(results['joint_tests'], dict) and 'error' not in results['joint_tests']:
        for test_name, test_info in results['joint_tests'].items():
            if test_name not in ['error', 'common_sample_size', 'alpha_hat', 'beta_hat'] and isinstance(test_info, dict) and 'statistic' in test_
                print(f"{test_name}: 统计量 = {test_info['statistic']:.4f}, P值 = {test_info['pvalue']:.4f}, 拒绝原假设: {test_info['reject_null']
    else:
        print("无法进行联合检验")

    print("\n=== 解释 ===")
    for interp in results['interpretations']:
        print(f"- {interp}")

    capm_valid = results.get('capm_valid')
    if capm_valid is not None:
        print(f"\nCAPM模型是否成立: {'是' if capm_valid else '否'}")
    else:
        print("\nCAPM模型是否成立: 无法确定")

    # 3. 绘制图形
    fig = plot_joint_tests_results(
        joint_tests=results['joint_tests'],
        valid_stocks=results['stock_codes']
    )

    # 4. 返回结果
    return {
        'results': results,
        'figure': fig
    }


stock_codes = ['600054', '600354', '600654', '600754', '600854', '000554', '002054', '002154']
start_date = '2000-01-01'
end_date = '2022-12-31'

# 运行分析
analysis_results = run_capm_joint_analysis(
    stock_codes=stock_codes,
    start_date=start_date,
    end_date=end_date
)
```

```
====== CAPM联合检验分析 ======
分析股票: 600054, 600354, 600654, 600754, 600854, 000554, 002054, 002154
日期范围: 2000-01-01 到 2022-12-31
================================


=== 联合检验结果 ===
wald_test: 统计量 = 113.4335, P值 = 0.0000, 拒绝原假设: True
lr_test: 统计量 = 366.0917, P值 = 0.0000, 拒绝原假设: True
lm_test: 统计量 = 82.4018, P值 = 0.0000, 拒绝原假设: True

=== 解释 ===
- 以下检验拒绝原假设：wald_test, lr_test, lm_test，表明CAPM模型在联合检验下不成立。
- 至少有部分股票存在非零Alpha值，不符合CAPM模型的预期。

CAPM模型是否成立: 否
```



CAPM 联合检验分析简要总结

三种检验方法（Wald、LR 和 LM）一致拒绝"所有股票 Alpha 值共同为零"的原假设：

- Wald 检验：统计量=113.4335，P 值=0.0000
- LR 检验：统计量=366.0917，P 值=0.0000
- LM 检验：统计量=82.4018，P 值=0.0000

主要结论：

1. CAPM 模型在解释所分析的 8 只中国股票（600054、600354 等）收益率方面不成立
2. 所有股票均显示出正的 Alpha 值，表明它们产生了 CAPM 模型无法解释的超额收益
3. 这暗示市场风险溢价（单一因素）不足以解释这些股票的表现，应考虑采用多因素模型

单资产 vs 多资产检验的差异

1. **检验角度不同**：
   - **单资产检验**：单独检验每只股票的 Alpha 是否显著不为零，着重于个体显著性
   - **多资产联合检验**：检验所有股票的 Alpha 是否共同为零，考虑了股票间的相关性
2. **统计效力不同**：
   - 多资产联合检验通常具有更高的统计效力，能发现单资产检验可能忽略的效应
   - 当样本中有多只股票共同表现出微弱但一致的 Alpha 时，单独看可能不显著，但联合起来可能非常显著

从表格中可以看到：

- 大多数股票的 Alpha 值虽然不为零，但 P 值较大（如 600054 的 Alpha P 值为 0.337139），表明在单资产检验中这些 Alpha 不显著
- 这就是为什么单资产检验显示"CAPM 是否成立: True"

- 然而，当使用 Wald、LR 和 LM 联合检验时，考虑到所有股票 Alpha 的整体模式及其相关性，结果显示这些 Alpha 共同显著不为零

为什么会出现这种情况

1. **相关性影响**：股票收益之间可能存在显著相关性，联合检验考虑了这种相关性
2. **样本量影响**：联合检验使用更多数据点，提高了统计效力
3. **小效应累积**：多只股票的小效应累积起来可能导致显著的整体效应

总结：单资产检验和联合检验结果不一致是正常的现象，反映了两种方法的不同统计特性。联合检验结果通常更可靠，因为它考虑了资产间的相关性并具有更高的统计效力。这也说明中国股票市场上的 CAPM 模型局限性可能比单独看每只股票时显得更为明显。

市场效应实证检验

这里获取月线数据需要 ts 至少 2000 积分的权限，建议淘宝解决

```
import pandas as pd
import numpy as np
import tushare as ts
import matplotlib.pyplot as plt
from scipy import stats

ts.set_token('2207b01a244311f7742512463d9e1588954d174b86d695f8a9f1cc4b')
pro = ts.pro_api()
```

```python
def get_monthly_data():
    # 获取股票列表
    stocks = pro.stock_basic(exchange='', list_status='L')
    stock_list = stocks['ts_code'].tolist()

    # 获取交易日历
    trade_cal = pro.trade_cal(exchange='SSE', start_date='20010101', end_date='20221231')
    monthly_dates = trade_cal[trade_cal['is_open']==1]['cal_date'].tolist()

    # 按月获取最后一个交易日
    monthly_end_dates = []
    current_month = monthly_dates[0][:6]  # 年月YYYYMM
    for date in monthly_dates:
        if date[:6] != current_month:
            monthly_end_dates.append(monthly_dates[monthly_dates.index(date)-1])
            current_month = date[:6]
    monthly_end_dates.append(monthly_dates[-1])  # 添加最后一个月

    # 创建列表存储每只股票的数据框
    stock_dfs = []

    for stock in stock_list:
        try:
            df = pro.monthly(ts_code=stock, start_date='20010101', end_date='20221231')
            if not df.empty:
                # 将该股票的收盘价数据转为DataFrame并添加到列表
                stock_df = df.set_index('trade_date')[['close']]
                stock_df.columns = [stock]  # 将列名设为股票代码
                stock_dfs.append(stock_df)
        except:
            continue

    # 使用pd.concat一次性连接所有股票数据
    if stock_dfs:
        monthly_prices = pd.concat(stock_dfs, axis=1)
    else:
        monthly_prices = pd.DataFrame()

    return monthly_prices

# 获取月度收盘价数据
monthly_prices = get_monthly_data()
monthly_prices
```

	000001.SZ	000002.SZ	000004.SZ	000006.SZ	000007.SZ	000008.SZ	000009.SZ	000010.SZ	000011.SZ	0000
trade_date										
20221230	13.16	18.20	9.68	6.30	7.78	2.30	12.09	3.75	11.59	6.71
20221130	13.03	18.65	10.20	5.84	7.54	2.43	13.10	3.34	11.28	7.25
20221031	10.34	13.52	8.74	3.66	8.00	2.25	11.56	3.04	9.20	6.88
20220930	11.84	17.83	8.44	4.13	8.06	2.20	11.22	3.15	9.19	6.94
20220831	12.75	16.63	9.34	4.21	8.04	2.41	14.76	3.53	10.03	6.52
...	...	...	...	...	...	...	...	...	...	...
20010330	16.22	15.13	28.58	14.00	12.99	25.60	7.94	15.88	11.62	20.32
20010228	14.09	13.77	25.00	13.09	10.56	24.78	7.18	13.71	9.59	22.18
20010119	14.98	14.93	28.00	13.71	12.89	24.54	7.92	13.74	11.56	23.42
20100831	NaN	8.42	11.31	8.18	7.85	12.33	11.20	NaN	8.96	15.48

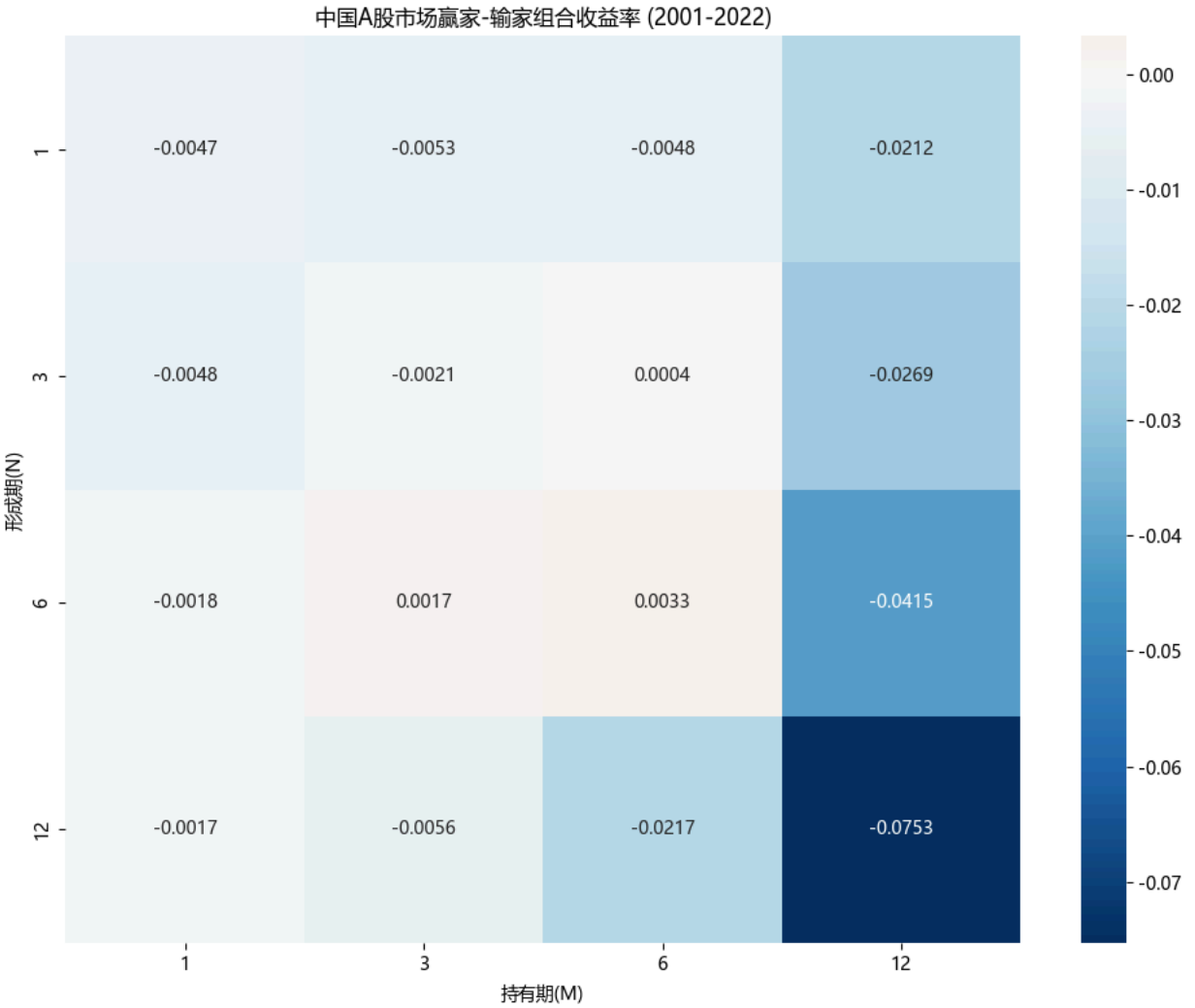
	000001.SZ	000002.SZ	000004.SZ	000006.SZ	000007.SZ	000008.SZ	000009.SZ	000010.SZ	000011.SZ	0000
trade_date										
20100730	NaN	8.29	9.63	8.93	7.58	11.07	9.96	NaN	9.64	11.92

264 rows × 4965 columns

```
# 计算月度收益率
monthly_returns = monthly_prices.pct_change()

# 处理缺失值
monthly_returns = monthly_returns.dropna(how='all')  # 删除全是缺失值的行
monthly_returns = monthly_returns.fillna(0)  # 将剩余缺失值填充为0
```

```python
def momentum_reversal_test(returns_data, formation_periods=[1, 3, 6, 12], holding_periods=[1, 3, 6, 12]):
    """
    实施排序法检验惯性/反转效应

    参数:
    returns_data: DataFrame, 股票月度收益率
    formation_periods: list, 形成期长度(月)
    holding_periods: list, 持有期长度(月)

    返回:
    results: 包含不同参数组合下的检验结果
    """
    results = {}

    for N in formation_periods:  # 形成期
        for M in holding_periods:  # 持有期
            key = f'N{N}_M{M}'
            results[key] = {'winner_returns': [], 'loser_returns': [], 'WML_returns': []}

            # 遍历每个时间点
            for t in range(N, len(returns_data) - M):
                # 1. 计算形成期的累积收益率
                if N == 1:
                    past_returns = returns_data.iloc[t-1]
                else:
                    # 计算过去N个月的累积收益率
                    past_returns = (1 + returns_data.iloc[t-N:t]).prod() - 1

                # 剔除缺失值
                past_returns = past_returns.dropna()

                # 2. 根据过去收益率排序，形成五分位组合
                past_returns_sorted = past_returns.sort_values()
                num_stocks = len(past_returns_sorted)
                quintile_size = num_stocks // 5

                if quintile_size == 0:
                    continue  # 股票数量不足，跳过

                # 选取最低分位(输家)和最高分位(赢家)股票
                losers = past_returns_sorted.iloc[:quintile_size].index
                winners = past_returns_sorted.iloc[-quintile_size:].index

                # 3. 计算持有期收益率
                if M == 1:
                    future_returns = returns_data.iloc[t+1]
                else:
                    # 计算未来M个月的累积收益率
                    future_returns = (1 + returns_data.iloc[t+1:t+M+1]).prod() - 1

                # 计算组合收益率
                loser_return = future_returns[losers].mean()
                winner_return = future_returns[winners].mean()
                wml_return = winner_return - loser_return  # 赢家减输家

                # 存储结果
                results[key]['loser_returns'].append(loser_return)
                results[key]['winner_returns'].append(winner_return)
                results[key]['WML_returns'].append(wml_return)

            # 计算平均收益率
            results[key]['avg_loser_return'] = np.mean(results[key]['loser_returns'])
            results[key]['avg_winner_return'] = np.mean(results[key]['winner_returns'])
            results[key]['avg_WML_return'] = np.mean(results[key]['WML_returns'])
```

```
            # t检验
            t_stat, p_value = stats.ttest_1samp(results[key]['WML_returns'], 0)
            results[key]['t_stat'] = t_stat
            results[key]['p_value'] = p_value
            results[key]['significant'] = p_value < 0.05

    return results

# 执行检验
test_results = momentum_reversal_test(monthly_returns)


import seaborn as sns

plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

def analyze_results(results):
    """分析并展示检验结果"""
    # 创建结果汇总表格
    summary = pd.DataFrame(columns=['形成期(N)', '持有期(M)', '赢家收益', '输家收益',
                                    'WML收益', 't统计量', 'p值', '显著性', '效应类型'])

    for key, result in results.items():
        N, M = key.split('_')
        N = int(N[1:])
        M = int(M[1:])

        wml = result['avg_WML_return']
        effect_type = '惯性效应' if wml > 0 else '反转效应'
        significance = '显著' if result['significant'] else '不显著'

        # 创建新行数据
        new_row = pd.DataFrame({
            '形成期(N)': [N],
            '持有期(M)': [M],
            '赢家收益': [result['avg_winner_return']],
            '输家收益': [result['avg_loser_return']],
            'WML收益': [wml],
            't统计量': [result['t_stat']],
            'p值': [result['p_value']],
            '显著性': [significance],
            '效应类型': [effect_type]
        })

        summary = pd.concat([summary, new_row], ignore_index=True)

    # 绘制热力图
    pivot_table = summary.pivot_table(index='形成期(N)', columns='持有期(M)', values='WML收益')
    plt.figure(figsize=(10, 8))
    plt.title('中国A股市场赢家-输家组合收益率 (2001-2022)')
    sns.heatmap(pivot_table, annot=True, cmap='RdBu_r', center=0, fmt='.4f')
    plt.tight_layout()
    plt.show()

    return summary

# 分析结果
summary_results = analyze_results(test_results)
summary_results
```



	形成期(N)	持有期(M)	赢家收益	输家收益	WML收益	t统计量	p值	显著性	效应类型
0	1	1	0.008200	0.012855	-0.004655	-1.037014	0.300693	不显著	反转效应
1	1	3	0.032584	0.037887	-0.005303	-0.609694	0.542601	不显著	反转效应
2	1	6	0.066952	0.071729	-0.004778	-0.327545	0.743524	不显著	反转效应
3	1	12	0.133461	0.154653	-0.021191	-0.968177	0.333896	不显著	反转效应
4	3	1	0.008274	0.013041	-0.004767	-1.114531	0.266089	不显著	反转效应
5	3	3	0.031173	0.033320	-0.002147	-0.240034	0.810496	不显著	反转效应
6	3	6	0.065700	0.065308	0.000393	0.025972	0.979300	不显著	惯性效应
7	3	12	0.123873	0.150744	-0.026871	-1.102330	0.271391	不显著	反转效应
8	6	1	0.009915	0.011738	-0.001823	-0.440298	0.660094	不显著	反转效应
9	6	3	0.035017	0.033328	0.001688	0.200206	0.841480	不显著	惯性效应
10	6	6	0.066661	0.063384	0.003277	0.209808	0.833989	不显著	惯性效应
11	6	12	0.114907	0.156421	-0.041513	-1.670993	0.096005	不显著	反转效应
12	12	1	0.010363	0.012095	-0.001731	-0.413866	0.679329	不显著	反转效应
13	12	3	0.031178	0.036776	-0.005598	-0.634156	0.526566	不显著	反转效应
14	12	6	0.055756	0.077472	-0.021717	-1.342870	0.180562	不显著	反转效应
15	12	12	0.104642	0.179908	-0.075266	-2.912684	0.003924	显著	反转效应

主要发现

1. **反转效应占主导**：在 16 种形成期-持有期组合中，有 12 种组合呈现反转效应（即 WML 收益为负），仅 4 种组合呈现惯性效应。
2. **统计显著性**：在所有组合中，仅有一种组合显示出统计显著的效应：
   - 形成期 12 个月/持有期 12 个月的组合显示出显著的反转效应(p 值=0.003924)
3. **形成期与持有期关系**：
   - 短期形成期(1 个月)下，所有持有期均呈现反转效应
   - 中期形成期(3 个月和 6 个月)下，短期和长期持有显示反转效应，而中期持有(3-6 个月)则出现弱惯性效应
   - 长期形成期(12 个月)下，所有持有期均呈现反转效应，且长期持有时效应最强
4. **反转效应强度**：
   - 长期形成期(12 个月)配合长期持有期(12 个月)产生最强的反转效应，WML 收益率为-7.53%，且统计显著

结果解读

这些发现与文献中关于中国 A 股市场的特征高度一致：

1. **中国 A 股市场反转效应主导**：结果支持了已有研究的结论 - 中国 A 股市场不同于成熟市场，更倾向于表现出反转效应而非惯性效应。
2. **长期反转效应最显著**：长期形成(12 个月)和长期持有(12 个月)组合显示出最显著的反转效应，这符合文献中提到的"长期反转"特征。
3. **投资者行为解释**：这可能反映了中国散户主导的市场结构 - 投资者过度追逐短期热点导致价格在短期内过度反应，随后出现回调。
4. **短期惯性与中期反转并存**：在部分中期形成期(3 和 6 个月)下，中期持有(3-6 个月)时出现弱惯性效应，也与文献中描述的 A 股"短期反转-中期动量-长期反转"的混合现象部分吻合。

投资启示

1. **反转策略潜力**：在中国 A 股市场，尤其是基于 12 个月形成期的反转策略(买入过去表现差的股票，卖出过去表现好的股票)可能有效。
2. **长期持有的重要性**：对于反转效应，长期持有期(12 个月)通常产生更显著的效果。
3. **市场时机判断**：需要注意市场情绪和宏观环境变化对这些效应的影响，因为极端市场条件下(如牛熊转换)这些效应可能增强或减弱。

研究局限性

1. **显著性不足**：大多数组合的效应在统计上不显著(p 值>0.05)，说明结果可靠性有限。
2. **时间变化**：建议进一步分析不同市场阶段(如牛市/熊市)下这些效应的表现，以检验其稳健性。