

A股上市公司股票收益率分析

分析目标：研究股票收益率与市盈率(PE)、成交量(Volume)、换手率(Turnover Rate)之间的关系

数据区间：2025.7.1 ~ 2026.3.31

第〇步：从Tushare获取数据并保存到CSV

```
In [3]: import tushare as ts
import pandas as pd
import os

# 设置Tushare Token
ts.set_token('d4ade4dc2337bb63733c3ec90deb49bfe013ac6b81452bdba73641e7')
pro = ts.pro_api()

# 选择一只A股：贵州茅台 (600519.SH)
ts_code = '600519.SH'
start_date = '20250701'
end_date = '20260331'

# 获取日线行情数据 (含成交量等)
df_daily = pro.daily(
    ts_code=ts_code,
    start_date=start_date,
    end_date=end_date,
    fields='ts_code,trade_date,open,high,low,close,vol,amount,pct_chg,change'
)

# 获取每日指标数据 (含市盈率PE、换手率)
df_basic = pro.daily_basic(
    ts_code=ts_code,
    start_date=start_date,
    end_date=end_date,
    fields='ts_code,trade_date,pe,pe_ttm,turnover_rate,volume_ratio'
)

# 合并两个数据集 (从daily_basic取pe、pe_ttm、turnover_rate)
df = pd.merge(df_daily, df_basic[['trade_date', 'pe', 'pe_ttm', 'turnover_rate']],
              on='trade_date', how='inner')

# 按日期升序排列
df = df.sort_values('trade_date').reset_index(drop=True)

# 重命名列
df.rename(columns={
    'pct_chg': 'return_rate',      # 收益率 (%)
    'vol': 'volume',              # 成交量 (手)
    'turnover_rate': 'turnover',  # 换手率 (%)
})
```

```

    'pe': 'pe_ratio'                                # 市盈率
}, inplace=True)

# 保存到CSV
csv_path = 'stock_data.csv'
df.to_csv(csv_path, index=False, encoding='utf-8-sig')
print(f'数据已保存到 {csv_path}')
print(f'数据形状: {df.shape}')
print(f'日期范围: {df["trade_date"].min()} ~ {df["trade_date"].max()}')
df.head()

```

数据已保存到 stock_data.csv

数据形状: (182, 13)

日期范围: 20250701 ~ 20260331

Out[3]:

	ts_code	trade_date	open	high	low	close	volume	amount
0	600519.SH	20250701	1409.00	1411.94	1403.31	1405.10	19878.19	2794423.203
1	600519.SH	20250702	1409.50	1414.80	1400.00	1409.60	26218.25	3689246.316
2	600519.SH	20250703	1412.00	1422.69	1408.00	1415.60	24412.67	3456733.002
3	600519.SH	20250704	1415.70	1431.89	1410.01	1422.22	28766.91	4086547.268
4	600519.SH	20250707	1422.28	1423.50	1410.01	1410.70	23854.67	3370562.659

第一步：读取CSV数据并进行描述性统计分析

```

In [4]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# 设置中文显示
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 从CSV读取数据
df = pd.read_csv('stock_data.csv')
print(f'数据形状: {df.shape}')
print(f'\n数据列名: {df.columns.tolist()}')
print(f'\n缺失值统计:')
print(df.isnull().sum())

# 删除含缺失值的行
df.dropna(subset=['return_rate', 'pe_ratio', 'volume', 'turnover'], inplace=True)
df.reset_index(drop=True, inplace=True)
print(f'\n清洗后数据形状: {df.shape}')
df.head()

```

数据形状: (182, 13)

数据列名: ['ts_code', 'trade_date', 'open', 'high', 'low', 'close', 'volume', 'amount', 'return_rate', 'change', 'pe_ratio', 'pe_ttm', 'turnover']

缺失值统计:

```
ts_code      0
trade_date   0
open         0
high         0
low          0
close        0
volume       0
amount       0
return_rate  0
change       0
pe_ratio     0
pe_ttm       0
turnover     0
dtype: int64
```

清洗后数据形状: (182, 13)

Out[4]:

	ts_code	trade_date	open	high	low	close	volume	amount
0	600519.SH	20250701	1409.00	1411.94	1403.31	1405.10	19878.19	2794423.203
1	600519.SH	20250702	1409.50	1414.80	1400.00	1409.60	26218.25	3689246.316
2	600519.SH	20250703	1412.00	1422.69	1408.00	1415.60	24412.67	3456733.002
3	600519.SH	20250704	1415.70	1431.89	1410.01	1422.22	28766.91	4086547.268
4	600519.SH	20250707	1422.28	1423.50	1410.01	1410.70	23854.67	3370562.659

1.1 各变量的描述性统计 (均值、标准差、分位数)

```
In [5]: # 选取分析变量
analysis_vars = ['return_rate', 'pe_ratio', 'volume', 'turnover']
desc_stats = df[analysis_vars].describe(percentiles=[0.25, 0.5, 0.75]).T
desc_stats.columns = ['样本数', '均值', '标准差', '最小值', '25%分位', '中位数', '75%分位']

print('=' * 80)
print('各变量描述性统计')
print('=' * 80)
print(f'\n收益率(return_rate): 日涨跌幅(%)')
print(f'市盈率(pe_ratio): PE值')
print(f'成交量(volume): 成交量(手)')
print(f'换手率(turnover): 换手率(%)')
print()
desc_stats
```

=====

各变量描述性统计

=====

收益率(return_rate): 日涨跌幅(%)
市盈率(pe_ratio): PE值
成交量(volume): 成交量(手)
换手率(turnover): 换手率(%)

```
Out[5]:
```

	样本数	均值	标准差	最小值	25%分位	中位数
return_rate	182.0	0.032432	1.246455	-2.5717	-0.639500	-0.12205
pe_ratio	182.0	20.911808	0.551723	19.2236	20.609575	20.84851
volume	182.0	39420.846978	19242.024347	17802.9900	27501.410000	35176.43000
turnover	182.0	0.314573	0.153632	0.1422	0.219625	0.28041

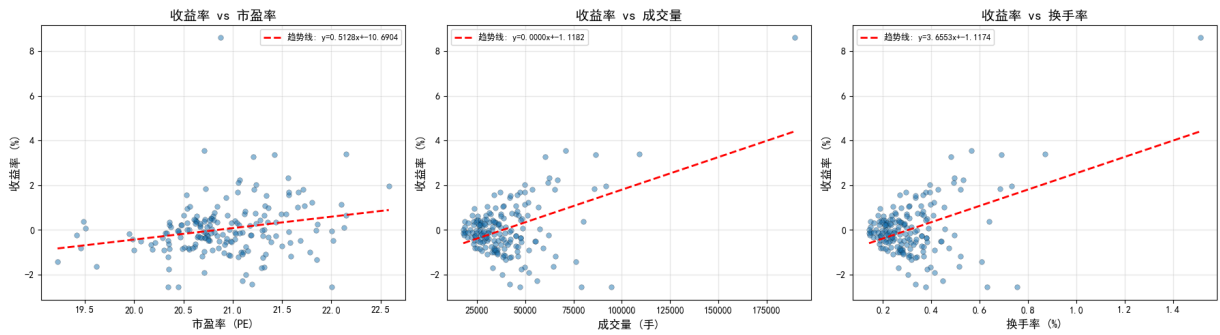
1.2 绘制收益率与每个自变量的散点图

```
In [6]: fig, axes = plt.subplots(1, 3, figsize=(18, 5))

independent_vars = ['pe_ratio', 'volume', 'turnover']
titles = ['收益率 vs 市盈率', '收益率 vs 成交量', '收益率 vs 换手率']
xlabels = ['市盈率 (PE)', '成交量 (手)', '换手率 (%)']

for i, (var, title, xlabel) in enumerate(zip(independent_vars, titles, xlabels)):
    axes[i].scatter(df[var], df['return_rate'], alpha=0.5, edgecolors='k', linewidth=1)
    # 添加趋势线
    z = np.polyfit(df[var], df['return_rate'], 1)
    p = np.poly1d(z)
    x_line = np.linspace(df[var].min(), df[var].max(), 100)
    axes[i].plot(x_line, p(x_line), 'r--', linewidth=2, label=f'趋势线: y={z[0]:.4f}x+{z[1]:.4f}')
    axes[i].set_xlabel(xlabel, fontsize=12)
    axes[i].set_ylabel('收益率 (%)', fontsize=12)
    axes[i].set_title(title, fontsize=14)
    axes[i].legend(fontsize=9)
    axes[i].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('scatter_plots.png', dpi=150, bbox_inches='tight')
plt.show()
print('散点图分析: 观察收益率与各自变量之间是否存在明显的线性趋势')
```



散点图分析：观察收益率与各自变量之间是否存在明显的线性趋势

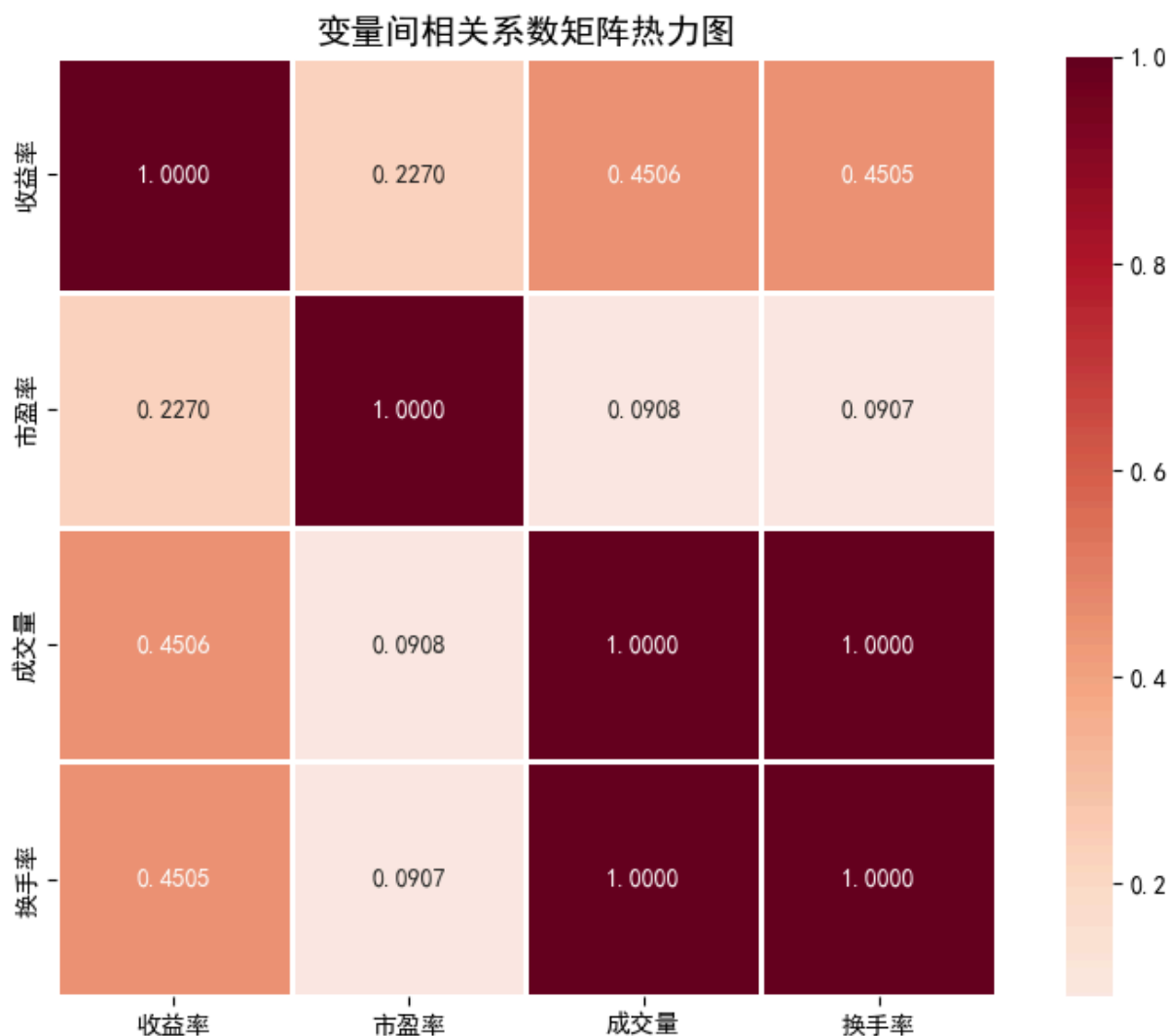
1.3 相关系数矩阵与热力图

```
In [7]: # 计算相关系数矩阵
corr_matrix = df[analysis_vars].corr()
print('相关系数矩阵: ')
print(corr_matrix.round(4))

# 绘制热力图
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, fmt='.4f', cmap='RdBu_r', center=0,
            square=True, linewidths=1, ax=ax,
            xticklabels=['收益率', '市盈率', '成交量', '换手率'],
            yticklabels=['收益率', '市盈率', '成交量', '换手率'])
ax.set_title('变量间相关系数矩阵热力图', fontsize=14)
plt.tight_layout()
plt.savefig('correlation_heatmap.png', dpi=150, bbox_inches='tight')
plt.show()
```

相关系数矩阵:

	return_rate	pe_ratio	volume	turnover
return_rate	1.0000	0.2270	0.4506	0.4505
pe_ratio	0.2270	1.0000	0.0908	0.0907
volume	0.4506	0.0908	1.0000	1.0000
turnover	0.4505	0.0907	1.0000	1.0000



第二步：划分数据集并拟合OLS回归模型

```
In [8]: from sklearn.model_selection import train_test_split
import statsmodels.formula.api as smf

# 划分训练集(80%)和测试集(20%)
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
print(f'训练集大小: {len(train_df)}')
print(f'测试集大小: {len(test_df)}')

# 使用 statsmodels.formula.api.ols 拟合多元线性回归模型
formula = 'return_rate ~ pe_ratio + volume + turnover'
model = smf.ols(formula=formula, data=train_df).fit()

# 输出回归结果摘要
print(model.summary())
```

训练集大小: 145
测试集大小: 37

OLS Regression Results

```
=====
Dep. Variable:          return_rate    R-squared:                0.113
Model:                  OLS            Adj. R-squared:          0.095
Method:                 Least Squares   F-statistic:              6.014
Date:                  Fri, 10 Apr 2026 Prob (F-statistic):       0.000694
Time:                  11:52:46         Log-Likelihood:           -209.19
No. Observations:      145             AIC:                     426.4
Df Residuals:          141             BIC:                     438.3
Df Model:               3
Covariance Type:       nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	-11.1856	3.238	-3.454	0.001	-17.587	-4.784
pe_ratio	0.5144	0.156	3.292	0.001	0.205	0.823
volume	0.0010	0.002	0.615	0.539	-0.002	0.004
turnover	-122.1189	200.704	-0.608	0.544	-518.896	274.658

```
=====
Omnibus:                9.776    Durbin-Watson:                1.890
Prob(Omnibus):          0.008    Jarque-Bera (JB):            18.351
Skew:                   -0.232    Prob(JB):                    0.000104
Kurtosis:               4.680    Cond. No.:                   9.67e+07
=====
```

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 9.67e+07. This might indicate that there are strong multicollinearity or other numerical problems.

```
In [9]: # 重点观察
print('=' * 60)
print('回归分析重点观察')
print('=' * 60)

print('\n(1) 回归系数及其符号: ')
for name, coef in model.params.items():
    sign = '+' if coef > 0 else '-'
    print(f'    {name:>15s}: {coef:>12.6f} ({sign})')

print(f'\n(2) R-squared = {model.rsquared:.6f}')
print(f'    Adj. R-squared = {model.rsquared_adj:.6f}')

print('\n(3) 各系数的p值 (显著性水平  $\alpha=0.05$ ): ')
for name, pval in model.pvalues.items():
    sig = '*** 显著' if pval < 0.01 else ('** 显著' if pval < 0.05 else ('* 边际显著'
    print(f'    {name:>15s}: p = {pval:.6f} {sig}')

print(f'\n(4) F检验: F = {model.fvalue:.4f}, p = {model.f_pvalue:.6f}')
if model.f_pvalue < 0.05:
    print('    → 模型整体显著 (p < 0.05) ')
else:
    print('    → 模型整体不显著 (p >= 0.05) ')

```

回归分析重点观察

(1) 回归系数及其符号:

```
Intercept:  -11.185635 (-)
pe_ratio:    0.514388 (+)
volume:      0.000986 (+)
turnover:    -122.118916 (-)
```

(2) R-squared = 0.113446

Adj. R-squared = 0.094583

(3) 各系数的p值 (显著性水平 $\alpha=0.05$):

```
Intercept: p = 0.000729 *** 显著
pe_ratio:  p = 0.001257 *** 显著
volume:    p = 0.539268 不显著
turnover:  p = 0.543865 不显著
```

(4) F检验: F = 6.0142, p = 0.000694

→ 模型整体显著 ($p < 0.05$)

第三步：模型诊断检验

3.1 多重共线性检验 (VIF)

```
In [10]: from statsmodels.stats.outliers_influence import variance_inflation_factor

# 计算VIF
X_vars = train_df[['pe_ratio', 'volume', 'turnover']]
vif_data = pd.DataFrame()
vif_data['变量'] = X_vars.columns
vif_data['VIF'] = [variance_inflation_factor(X_vars.values, i) for i in range(X_var

print('方差膨胀因子(VIF): ')
print(vif_data.to_string(index=False))
print()
for _, row in vif_data.iterrows():
    if row['VIF'] > 10:
        print(f"△ {row['变量']} 的 VIF = {row['VIF']:.2f} > 10, 存在严重多重共线性! ")
    elif row['VIF'] > 5:
        print(f"△ {row['变量']} 的 VIF = {row['VIF']:.2f} > 5, 存在一定共线性")
    else:
        print(f"✓ {row['变量']} 的 VIF = {row['VIF']:.2f}, 无严重共线性")
```


方差膨胀因子(VIF):

变量	VIF
pe_ratio	7.043924
volume	595784.704121
turnover	595414.905403

△ pe_ratio 的 VIF = 7.04 > 5, 存在一定共线性

△ volume 的 VIF = 595784.70 > 10, 存在严重多重共线性!

△ turnover 的 VIF = 595414.91 > 10, 存在严重多重共线性!

3.2 异方差检验

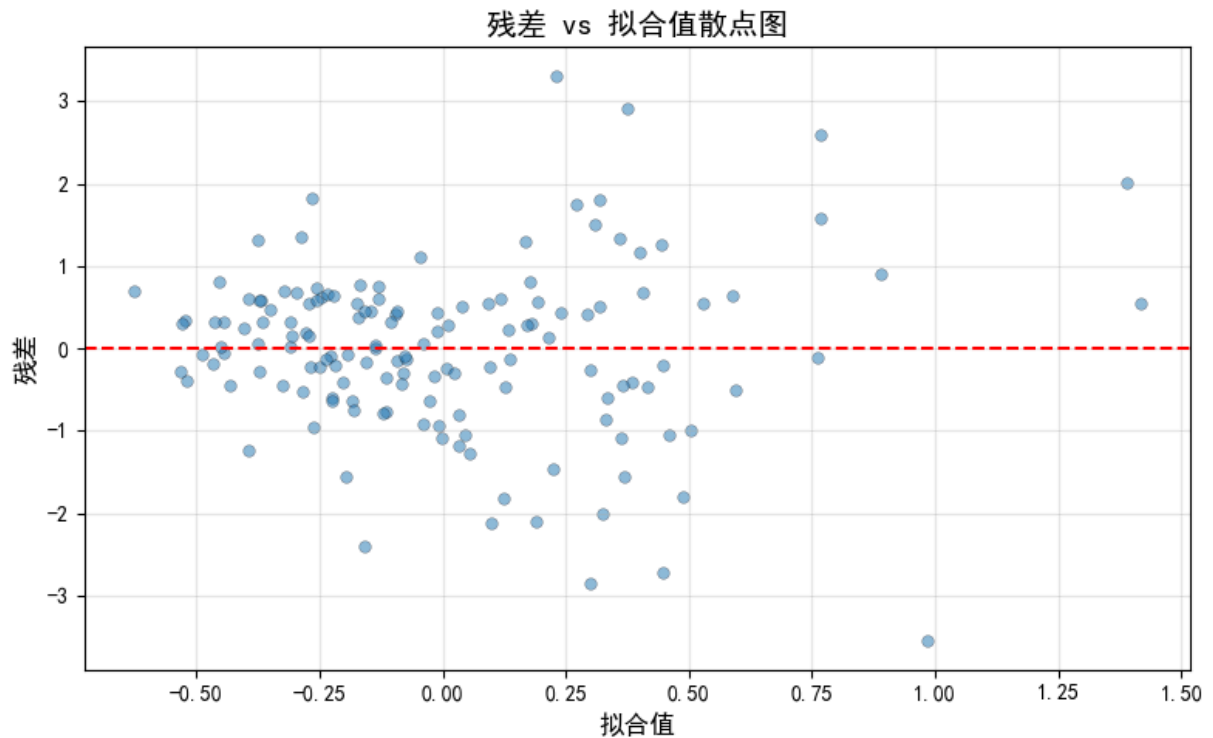
```
In [11]: from statsmodels.stats.diagnostic import het_breuschpagan

# 残差 vs 拟合值散点图
fitted_vals = model.fittedvalues
residuals = model.resid

fig, ax = plt.subplots(figsize=(8, 5))
ax.scatter(fitted_vals, residuals, alpha=0.5, edgecolors='k', linewidth=0.3, s=30)
ax.axhline(y=0, color='r', linestyle='--', linewidth=1.5)
ax.set_xlabel('拟合值', fontsize=12)
ax.set_ylabel('残差', fontsize=12)
ax.set_title('残差 vs 拟合值散点图', fontsize=14)
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('residual_vs_fitted.png', dpi=150, bbox_inches='tight')
plt.show()

# Breusch-Pagan 检验
bp_test = het_breuschpagan(residuals, model.model.exog)
labels = ['LM统计量', 'LM检验p值', 'F统计量', 'F检验p值']
print('\nBreusch-Pagan异方差检验结果: ')
for label, val in zip(labels, bp_test):
    print(f' {label}: {val:.6f}')

if bp_test[1] < 0.05:
    print('\n→ p < 0.05, 拒绝同方差假设, 存在异方差性')
else:
    print('\n→ p >= 0.05, 不拒绝同方差假设, 不存在明显异方差性')
```



Breusch-Pagan异方差检验结果:

LM统计量: 49.532757

LM检验p值: 0.000000

F统计量: 24.385743

F检验p值: 0.000000

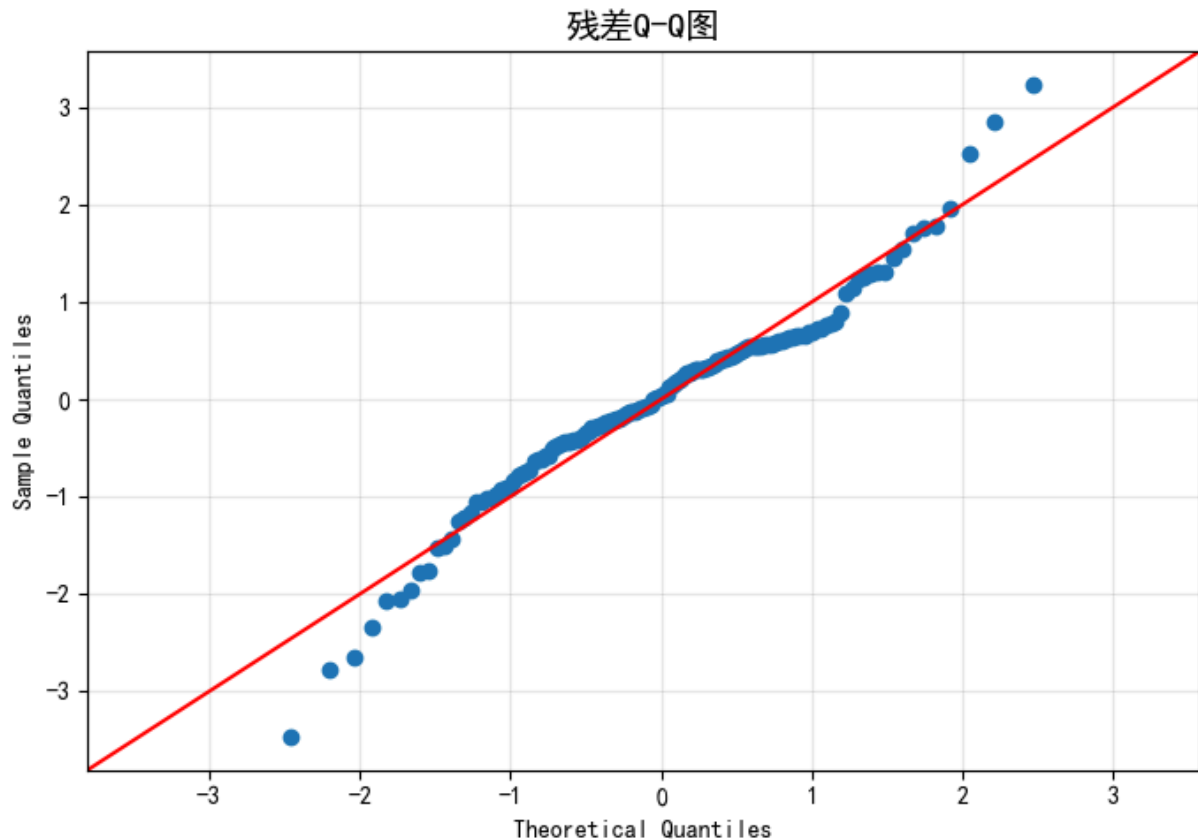
→ $p < 0.05$, 拒绝同方差假设, 存在异方差性

3.3 残差正态性检验

```
In [12]: from scipy import stats
import statsmodels.api as sm

# Q-Q图
fig, ax = plt.subplots(figsize=(7, 5))
sm.qqplot(residuals, line='45', fit=True, ax=ax)
ax.set_title('残差Q-Q图', fontsize=14)
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('qq_plot.png', dpi=150, bbox_inches='tight')
plt.show()

# Shapiro-Wilk检验
stat_sw, p_sw = stats.shapiro(residuals)
print(f'Shapiro-Wilk检验: 统计量 = {stat_sw:.6f}, p值 = {p_sw:.6f}')
if p_sw < 0.05:
    print('→  $p < 0.05$ , 拒绝正态性假设, 残差不服从正态分布')
else:
    print('→  $p \geq 0.05$ , 不拒绝正态性假设, 残差近似服从正态分布')
```



Shapiro-Wilk检验：统计量 = 0.966484, p值 = 0.001289

→ $p < 0.05$, 拒绝正态性假设, 残差不服从正态分布

3.4 尝试改进：删除不显著变量后重新建模

```
In [13]: # 根据上面p值结果, 删除不显著的变量重新拟合
# 先检测哪些变量不显著
insignificant = [name for name, pval in model.pvalues.items()
                  if pval >= 0.05 and name != 'Intercept']
print(f'不显著变量 (p >= 0.05): {insignificant}')

significant = [name for name, pval in model.pvalues.items()
               if pval < 0.05 and name != 'Intercept']
print(f'显著变量 (p < 0.05): {significant}')

if len(significant) > 0:
    formula_improved = 'return_rate ~ ' + ' + '.join(significant)
    print(f'\n改进模型公式: {formula_improved}')
    model_improved = smf.ols(formula=formula_improved, data=train_df).fit()
    print(model_improved.summary())
else:
    print('\n所有变量均不显著, 保留原模型进行后续分析')
    formula_improved = formula
    model_improved = model

# 比较两个模型
print('\n' + '=' * 60)
print('模型对比')
print('=' * 60)
```

```
print(f'{"指标":<25s} {"原始模型":>15s} {"改进模型":>15s}')
print(f'{"R-squared":<25s} {model.rsquared:>15.6f} {model_improved.rsquared:>15.6f}')
print(f'{"Adj. R-squared":<25s} {model.rsquared_adj:>15.6f} {model_improved.rsquared_adj:>15.6f}')
print(f'{"AIC":<25s} {model.aic:>15.2f} {model_improved.aic:>15.2f}')
print(f'{"BIC":<25s} {model.bic:>15.2f} {model_improved.bic:>15.2f}')
```

不显著变量 (p >= 0.05): ['volume', 'turnover']

显著变量 (p < 0.05): ['pe_ratio']

改进模型公式: return_rate ~ pe_ratio

```

                                OLS Regression Results
=====
Dep. Variable:                  return_rate    R-squared:                  0.085
Model:                            OLS         Adj. R-squared:              0.079
Method:                 Least Squares         F-statistic:                13.32
Date:                  Fri, 10 Apr 2026        Prob (F-statistic):          0.000366
Time:                  11:52:48               Log-Likelihood:             -211.46
No. Observations:        145                 AIC:                      426.9
Df Residuals:            143                 BIC:                      432.9
Df Model:                 1
Covariance Type:          nonrobust
=====
               coef      std err          t      P>|t|      [0.025      0.975]
-----
Intercept    -11.8574      3.250     -3.648      0.000     -18.283      -5.432
pe_ratio       0.5677      0.156      3.650      0.000       0.260       0.875
=====
Omnibus:                 10.424    Durbin-Watson:              1.828
Prob(Omnibus):            0.005    Jarque-Bera (JB):            18.603
Skew:                     0.292    Prob(JB):                    9.13e-05
Kurtosis:                 4.655    Cond. No.                     783.
=====
```

Notes:

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

模型对比

```

=====
指标                                原始模型                                改进模型
R-squared                          0.113446                          0.085239
Adj. R-squared                     0.094583                          0.078842
AIC                                426.38                             426.93
BIC                                438.29                             432.88
=====
```

第四步：测试集预测与误差评估

```
In [14]: from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# 在测试集上预测
y_test = test_df['return_rate'].values
y_pred = model_improved.predict(test_df)
```

```

# 计算多种预测误差指标
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = np.mean(np.abs((y_test - y_pred) / (y_test + 1e-8))) * 100

print('=' * 50)
print('测试集预测误差指标')
print('=' * 50)
print(f'MSE (均方误差): {mse:.6f}')
print(f'RMSE (均方根误差): {rmse:.6f}')
print(f'MAE (平均绝对误差): {mae:.6f}')
print(f'MAPE (平均绝对百分比误差): {mape:.2f}%')
print(f'R² (决定系数): {r2:.6f}')

```

=====
测试集预测误差指标
=====

```

MSE (均方误差):      2.974299
RMSE (均方根误差):   1.724616
MAE (平均绝对误差):   0.998297
MAPE (平均绝对百分比误差): 328.64%
R² (决定系数):      -0.008753

```

```

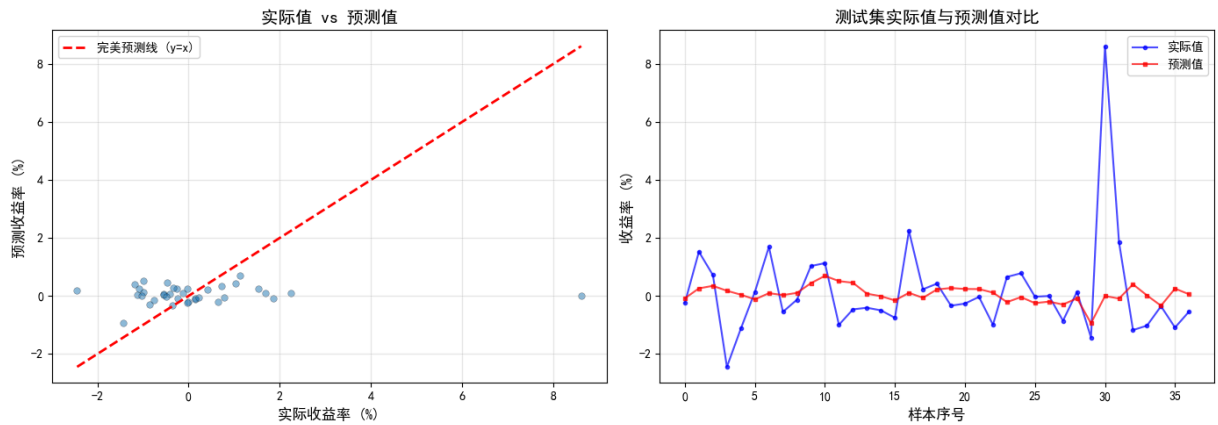
In [15]: # 绘制实际值与预测值的散点图
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# 散点图: 实际 vs 预测
axes[0].scatter(y_test, y_pred, alpha=0.5, edgecolors='k', linewidth=0.3, s=30)
min_val = min(y_test.min(), y_pred.min())
max_val = max(y_test.max(), y_pred.max())
axes[0].plot([min_val, max_val], [min_val, max_val], 'r--', linewidth=2, label='完美')
axes[0].set_xlabel('实际收益率 (%)', fontsize=12)
axes[0].set_ylabel('预测收益率 (%)', fontsize=12)
axes[0].set_title('实际值 vs 预测值', fontsize=14)
axes[0].legend(fontsize=10)
axes[0].grid(True, alpha=0.3)

# 折线图: 对比走势
test_sorted = test_df.sort_values('trade_date').reset_index(drop=True)
y_test_sorted = test_sorted['return_rate'].values
y_pred_sorted = model_improved.predict(test_sorted)
axes[1].plot(range(len(y_test_sorted)), y_test_sorted, 'b-o', markersize=3, label='实际')
axes[1].plot(range(len(y_pred_sorted)), y_pred_sorted, 'r-s', markersize=3, label='预测')
axes[1].set_xlabel('样本序号', fontsize=12)
axes[1].set_ylabel('收益率 (%)', fontsize=12)
axes[1].set_title('测试集实际值与预测值对比', fontsize=14)
axes[1].legend(fontsize=10)
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('prediction_results.png', dpi=150, bbox_inches='tight')
plt.show()

```



第五步：结论与总结

```
In [16]: print('=' * 70)
print('最终分析报告')
print('=' * 70)

# 1. 最终模型公式与系数
print('\n【1. 最终模型的公式和系数】')
print(f'模型公式: {formula_improved}')
print('\n回归系数: ')
for name, coef in model_improved.params.items():
    print(f' {name}: {coef:.6f}')

formula_str = f'return_rate = {model_improved.params["Intercept"]:.4f}'
for name, coef in model_improved.params.items():
    if name != 'Intercept':
        sign = '+' if coef > 0 else '-'
        formula_str += f' {sign}{coef:.4f}*{name}'
print(f'\n数学表达式: {formula_str}')

# 2. 显著因素的经济含义
print('\n【2. 显著影响收益率的因素及其经济含义】')
for name, pval in model_improved.pvalues.items():
    if name == 'Intercept':
        continue
    coef = model_improved.params[name]
    if pval < 0.05:
        direction = '正向' if coef > 0 else '负向'
        if name == 'pe_ratio':
            meaning = f'市盈率每增加1个单位, 收益率{direction}变动{abs(coef):.4f}%. 高'
        elif name == 'volume':
            meaning = f'成交量每增加1手, 收益率{direction}变动{abs(coef):.6f}%. 成交量'
        elif name == 'turnover':
            meaning = f'换手率每增加1%, 收益率{direction}变动{abs(coef):.4f}%. 高换手率'
        print(f' - {name} (p={pval:.4f}): {meaning}')

# 3. 模型局限性
print('\n【3. 模型存在的局限性】')
print(' (1) R²较低: 线性模型可能无法充分捕捉股票收益率的非线性特征。')
```

```
print('      股票市场受到宏观经济、政策、市场情绪等多种复杂因素影响。')
print(' (2) 变量有限: 仅使用了市盈率、成交量、换手率三个变量, ')
print('      未涵盖宏观经济指标、行业因素、市场情绪等重要变量。')
print(' (3) 线性假设: OLS假设自变量与因变量之间为线性关系, ')
print('      实际金融市场中可能存在非线性关系。')
print(' (4) 时间序列特性: 股票数据具有时间序列特性 (自相关性), ')
print('      OLS未考虑时间依赖性, 可能导致估计偏差。')
print(' (5) 市场有效性: 根据有效市场假说, 历史数据难以预测未来收益率, ')
print('      因此线性回归预测能力天然受限。')
```

最终分析报告

【1. 最终模型的公式和系数】

模型公式: $\text{return_rate} \sim \text{pe_ratio}$

回归系数:

Intercept: -11.857443

pe_ratio: 0.567740

数学表达式: $\text{return_rate} = -11.8574 + 0.5677 \times \text{pe_ratio}$

【2. 显著影响收益率的因素及其经济含义】

- pe_ratio ($p=0.0004$): 市盈率每增加1个单位, 收益率正向变动0.5677%。高PE可能反映市场对公司的乐观预期。

【3. 模型存在的局限性】

- (1) R^2 较低: 线性模型可能无法充分捕捉股票收益率的非线性特征。
股票市场受到宏观经济、政策、市场情绪等多种复杂因素影响。
- (2) 变量有限: 仅使用了市盈率、成交量、换手率三个变量,
未涵盖宏观经济指标、行业因素、市场情绪等重要变量。
- (3) 线性假设: OLS假设自变量与因变量之间为线性关系,
实际金融市场中可能存在非线性关系。
- (4) 时间序列特性: 股票数据具有时间序列特性 (自相关性),
OLS未考虑时间依赖性, 可能导致估计偏差。
- (5) 市场有效性: 根据有效市场假说, 历史数据难以预测未来收益率,
因此线性回归预测能力天然受限。