```
# !pip install pandas-datareader


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()
plt.style.use('fivethirtyeight')
import datetime

from pandas_datareader import data as pdr
import yfinance as yf
# yf.pdr_override()


import yfinance as yf

end_date = datetime.date.today().strftime('%Y-%m-%d')
apple = yf.Ticker("AAPL")
AAPL = apple.history(start = "2020-01-01", end= end_date)
AAPL.head()
```

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2020-01-02 00:00:00-05:00	71.721026	72.776606	71.466820	72.716080	135480400	0.0	0.0
2020-01-03 00:00:00-05:00	71.941328	72.771745	71.783962	72.009117	146322800	0.0	0.0

Next steps: Generate code with AAPL | View recommended plots | New interactive sheet

Get the Balance Sheet and Income Statements

```
balance_sheet = apple.balance_sheet
```

```
print("Balance Sheet:")
print(balance_sheet.head())

income_statement = apple.financials
print("\nIncome Statement:")
print(income_statement.head())

# Information about Apple:
info = apple.info
print(f"\nCompany: {info['longName']}")
print(f"Sector: {info['sector']}")
print(f"Industry: {info['industry']}")
print(f"Market Cap: {info['marketCap']}")
print(f"P/E Ratio: {info['trailingPE']}")

# dividend data
dividends = apple.dividends
print("Dividends:")
print(dividends.tail())
```

```
Balance Sheet:
                            2024-09-30      2023-09-30      2022-09-30  \
Treasury Shares Number             NaN             0.0             NaN
Ordinary Shares Number   15116786000.0   15550061000.0   15943425000.0
Share Issued             15116786000.0   15550061000.0   15943425000.0
Net Debt                 76686000000.0   81123000000.0   96423000000.0
Total Debt              106629000000.0  111088000000.0  132480000000.0

                            2021-09-30 2020-09-30
Treasury Shares Number             NaN        NaN
Ordinary Shares Number   16426786000.0        NaN
Share Issued             16426786000.0        NaN
Net Debt                 89779000000.0        NaN
Total Debt              136522000000.0        NaN

Income Statement:
                                                        2024-09-30  \
Tax Effect Of Unusual Items                                    0.0
Tax Rate For Calcs                                           0.241
Normalized EBITDA                                   134661000000.0
Net Income From Continuing Operation Net Minori...   93736000000.0
Reconciled Depreciation                              11445000000.0

                                                        2023-09-30  \
Tax Effect Of Unusual Items                                    0.0
Tax Rate For Calcs                                           0.147
Normalized EBITDA                                   125820000000.0
Net Income From Continuing Operation Net Minori...   96995000000.0
```

```
Reconciled Depreciation                                11519000000.0

                                                             2022-09-30  \
Tax Effect Of Unusual Items                                         0.0
Tax Rate For Calcs                                                0.162
Normalized EBITDA                                     130541000000.0
Net Income From Continuing Operation Net Minori...     99803000000.0
Reconciled Depreciation                                11104000000.0

                                                             2021-09-30 2020-09-30
Tax Effect Of Unusual Items                                         0.0        NaN
Tax Rate For Calcs                                                0.133        NaN
Normalized EBITDA                                     123136000000.0        NaN
Net Income From Continuing Operation Net Minori...     94680000000.0        NaN
Reconciled Depreciation                                11284000000.0        NaN

Company: Apple Inc.
Sector: Technology
Industry: Consumer Electronics
Market Cap: 3149833961472
P/E Ratio: 33.335453
Dividends:
Date
2024-02-09 00:00:00-05:00    0.24
2024-05-10 00:00:00-04:00    0.25
2024-08-12 00:00:00-04:00    0.25
2024-11-08 00:00:00-05:00    0.25
2025-02-10 00:00:00-05:00    0.25
Name: Dividends, dtype: float64
```

```
apple = yf.Ticker("AAPL")

tickers = ["SPY", "AAL", "ZM", "NFLX", "META", 'AAPL']

end_date = datetime.date.today().strftime('%Y-%m-%d')
apple = yf.Ticker("AAPL")
AAPL = apple.history(start = "2020-01-01", end= "2024-12-31")

for ticker in tickers:
    globals()[ticker] = yf.Ticker(ticker)
    globals()[ticker] = globals()[ticker].history(start = "2020-01-01", end= "202
```

```
print(META.Close.mean())
META.describe()
```

```
300.24229519160474
```

	Open	High	Low	Close	Volume	Dividends	S
count	1257.000000	1257.000000	1257.000000	1257.000000	1.257000e+03	1257.000000	
mean	300.078645	304.300186	296.060907	300.242295	2.315541e+07	0.001591	
std	124.856234	125.814469	123.529762	124.713439	1.572882e+07	0.028172	
min	89.737209	90.115760	87.754781	88.571663	4.726100e+06	0.000000	
25%	208.045272	210.794803	205.724130	208.981705	1.453120e+07	0.000000	
50%	278.097672	284.144576	275.228648	279.761322	1.938320e+07	0.000000	
75%	345.310970	350.760134	341.874050	344.972229	2.711680e+07	0.000000	
max	630.991005	637.885465	626.704485	632.170044	2.323166e+08	0.500000	

Now, let us keep only the closing prices for our analysis.

```
## keep only column close for all tickers
for ticker in tickers:
    globals()[ticker] = globals()[ticker].Close
```

SPY

Date	Close
2020-01-02 00:00:00-05:00	301.194946
2020-01-03 00:00:00-05:00	298.914215
2020-01-06 00:00:00-05:00	300.054535
2020-01-07 00:00:00-05:00	299.210876
2020-01-08 00:00:00-05:00	300.805603
...	...
2024-12-23 00:00:00-05:00	594.690002
2024-12-24 00:00:00-05:00	601.299988
2024-12-26 00:00:00-05:00	601.340027
2024-12-27 00:00:00-05:00	595.010010
2024-12-30 00:00:00-05:00	588.219971

1257 rows × 1 columns

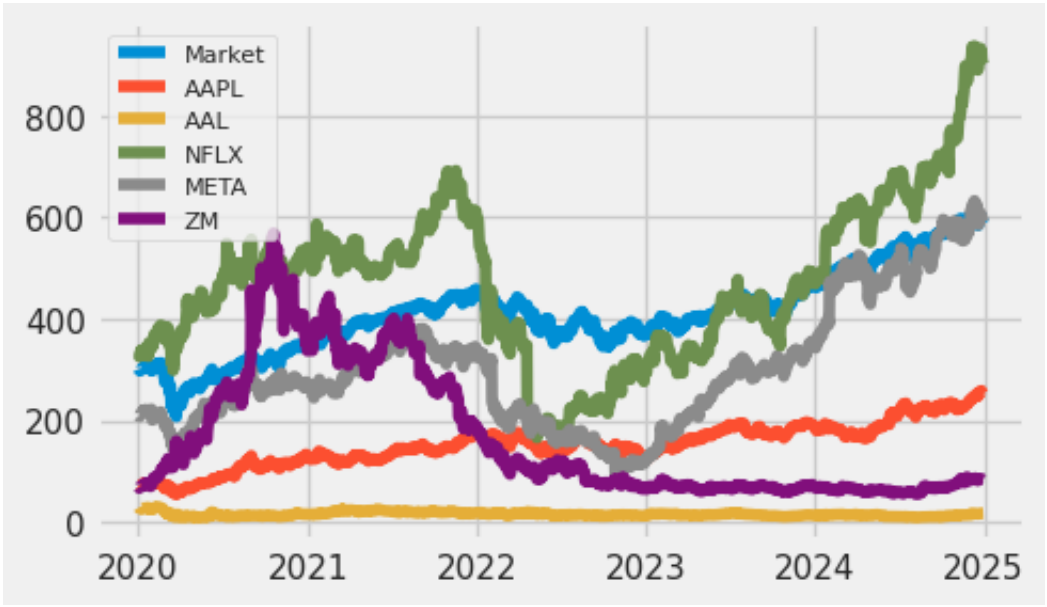
dtype: float64

```
df = pd.DataFrame({'Market': SPY, 'AAPL':AAPL, 'AAL':AAL, 'NFLX':NFLX, 'META':META
df.tail()
```

Date	Market	AAPL	AAL	NFLX	META	ZM
2024-12-23 00:00:00-05:00	594.690002	254.989655	17.250000	911.450012	599.849976	85.269997
2024-12-24 00:00:00-05:00	601.299988	257.916443	17.350000	932.119995	607.750000	85.669998
2024-12-26 00:00:00-	601.340027	258.735504	17.350000	924.140015	603.349976	85.440002

```
# print(AAPL)
# print(AAPL.shift(1))


plt.style.use('fivethirtyeight')
plt.figure(figsize=(5, 3))
plt.plot(df, label=df.columns)
plt.legend(loc='upper left',fontsize=8)
plt.show()
```



For financial analysis, we require the log returns (daily), rather than the raw stock prices. The formula for log returns is:

log(Today's Price/yesterday's price - 1)

```
# create new columns that are log returns of the columns
data = np.log(df/df.shift(1))
# data = (df-df.shift(1))/df.shift(1)
# replace first row with zeroes
data.iloc[0] = 0
data.head(5)
```

Date	Market	AAPL	AAL	NFLX	META	ZM
2020-01-02 00:00:00-05:00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2020-01-03 00:00:00-05:00	-0.007601	-0.009770	-0.050769	-0.011926	-0.005305	-0.021177
2020-01-06 00:00:00-05:00	0.003808	0.007937	-0.012007	0.030014	0.018658	0.044193
2020-01-07						

Next steps: Generate code with data | View recommended plots | New interactive sheet

Find the betas of the stocks. The formula is shown below:

```
beta_aapl = (data[['Market','AAPL']].cov()/data['Market'].var()).iloc[0].iloc[1]
beta_aapl
```

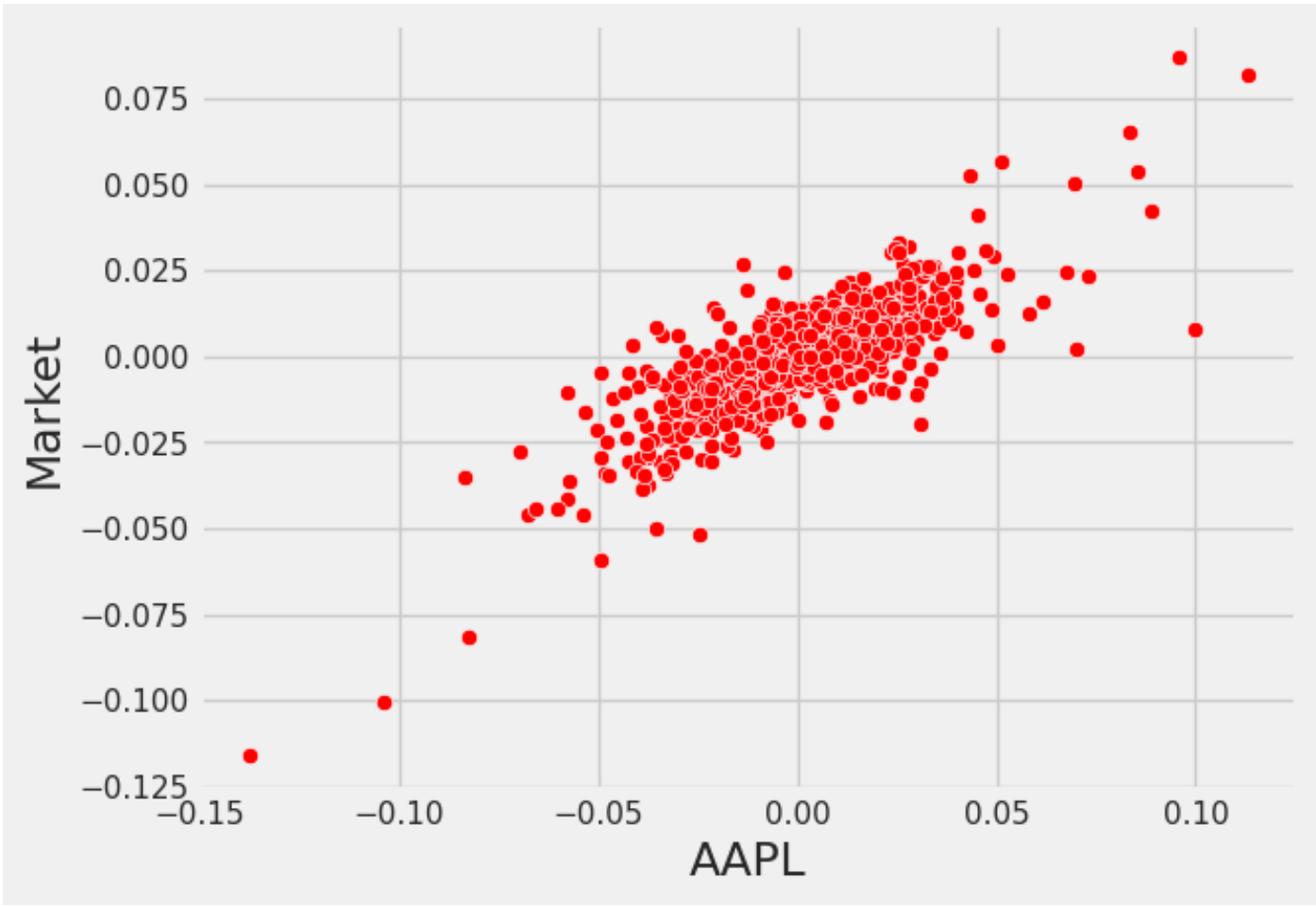
```
1.1898311725984072
```

Calculate beta using regression line.

```
beta, alpha = np.polyfit(data['Market'], data['AAPL'], 1)
alpha
beta
```

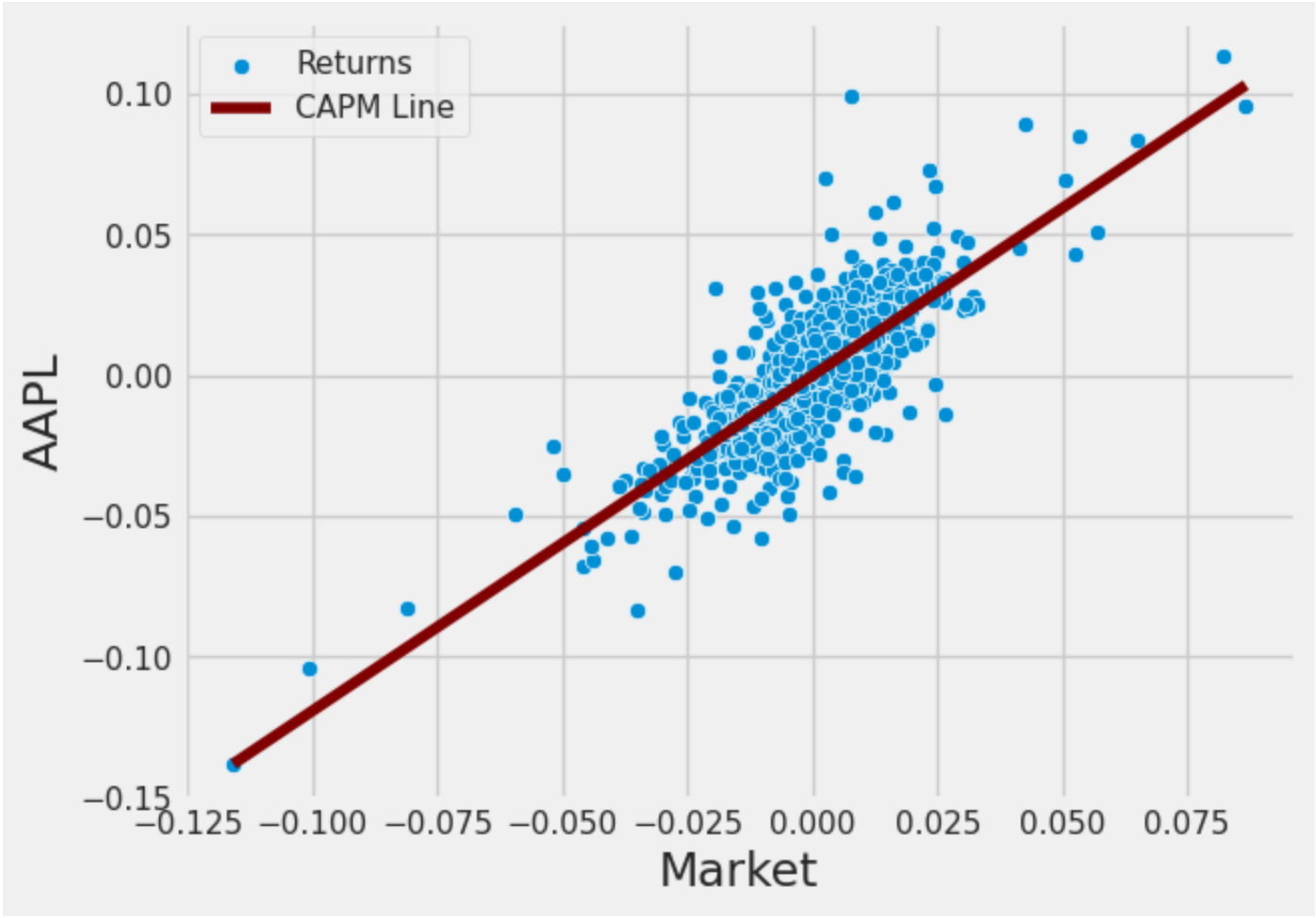
```
1.1898311725984076
```

```
# plt.axvline(0, color='grey', alpha = 0.5)
# plt.axhline(0, color='grey', alpha = 0.5)
sns.scatterplot(y = 'Market', x = 'AAPL', data = data, color = 'red')
plt.show()
```



```
sns.scatterplot(y = 'AAPL', x = 'Market', data = data, label = 'Returns')
sns.lineplot(x = data['Market'], y = alpha + (data['Market']-alpha)*beta_aapl, colo
```

```
<Axes: xlabel='Market', ylabel='AAPL'>
```



Convert Daily Stock Market Returns to Annualized Returns (assuming 252 trading days in a year).

```
rm = data['Market'].mean()*252
rm
cov = data[['Market','AAPL']].cov() *252
cov_aapl_market = cov.iloc[0,1]
cov_aapl_market
market_var = data['Market'].var()*252
market_var

AAPL_beta_annual = cov_aapl_market / market_var
print('The annualized beta will equal the one calculated at daily returns:',AAPL_

rf = 0.025
riskpremium = rm - rf

## CAPM
AAPL_capm_return = rf + AAPL_beta_annual*riskpremium

print(f"The annualized CAPM return of AAPL is {AAPL_capm_return*100:.2f}%")
```

```
The annualized beta will equal the one calculated at daily returns: 1.1898311
The annualized CAPM return of AAPL is 15.49%
```

```
sharperatio = (rm-rf)/(data['AAPL'].std()*np.sqrt(252))
sharperatio
print(f"Sharpe Ratio: {round(sharperatio,4)}")
```

```
Sharpe Ratio: 0.345
```

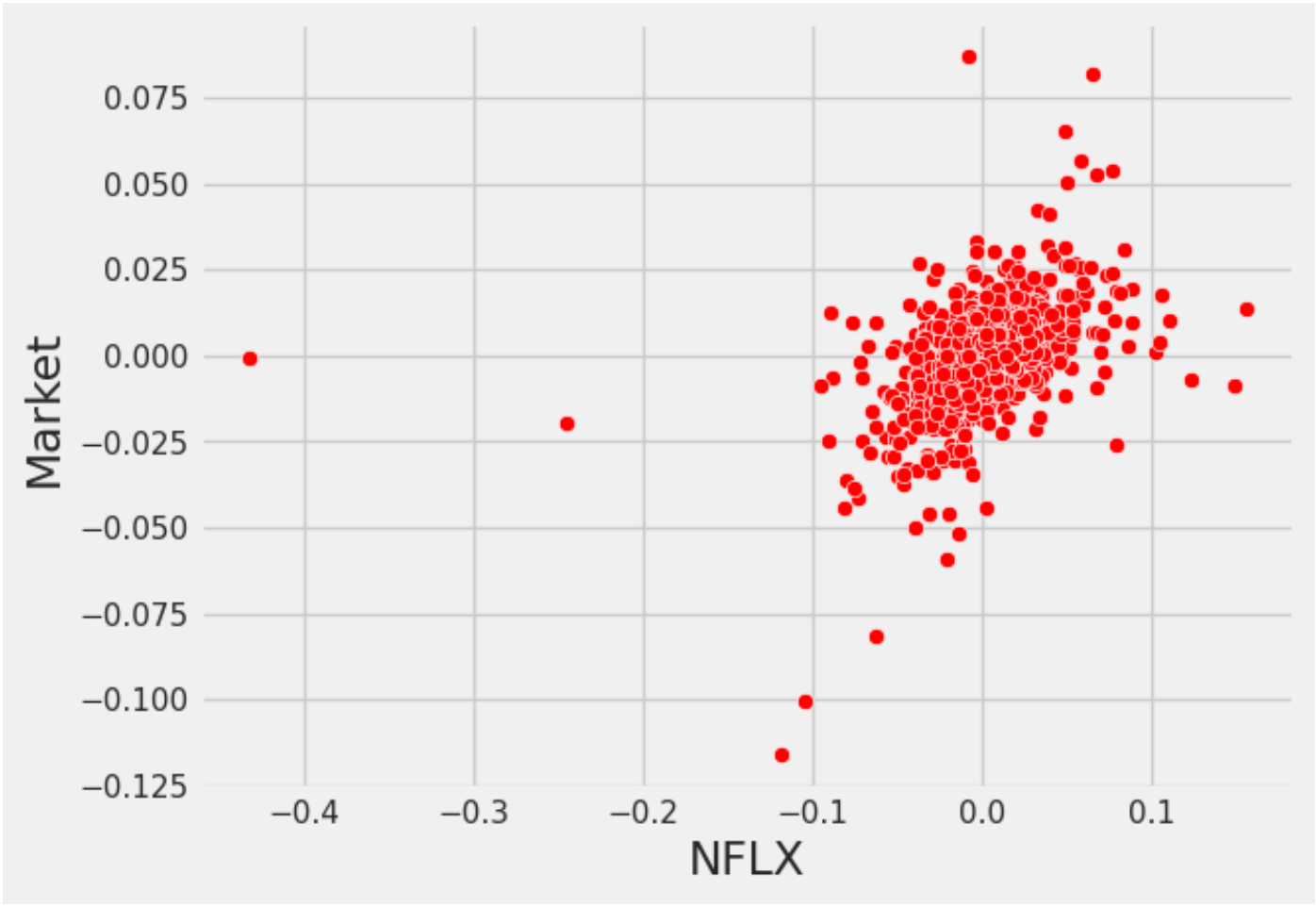
```
beta_NFLX = (data[['Market','NFLX']].cov()/data['Market'].var()).iloc[0].iloc[1]
beta_NFLX
```

```
1.047975443217553
```

```
beta, alpha = np.polyfit(data['Market'], data['NFLX'], 1)
alpha
beta
```

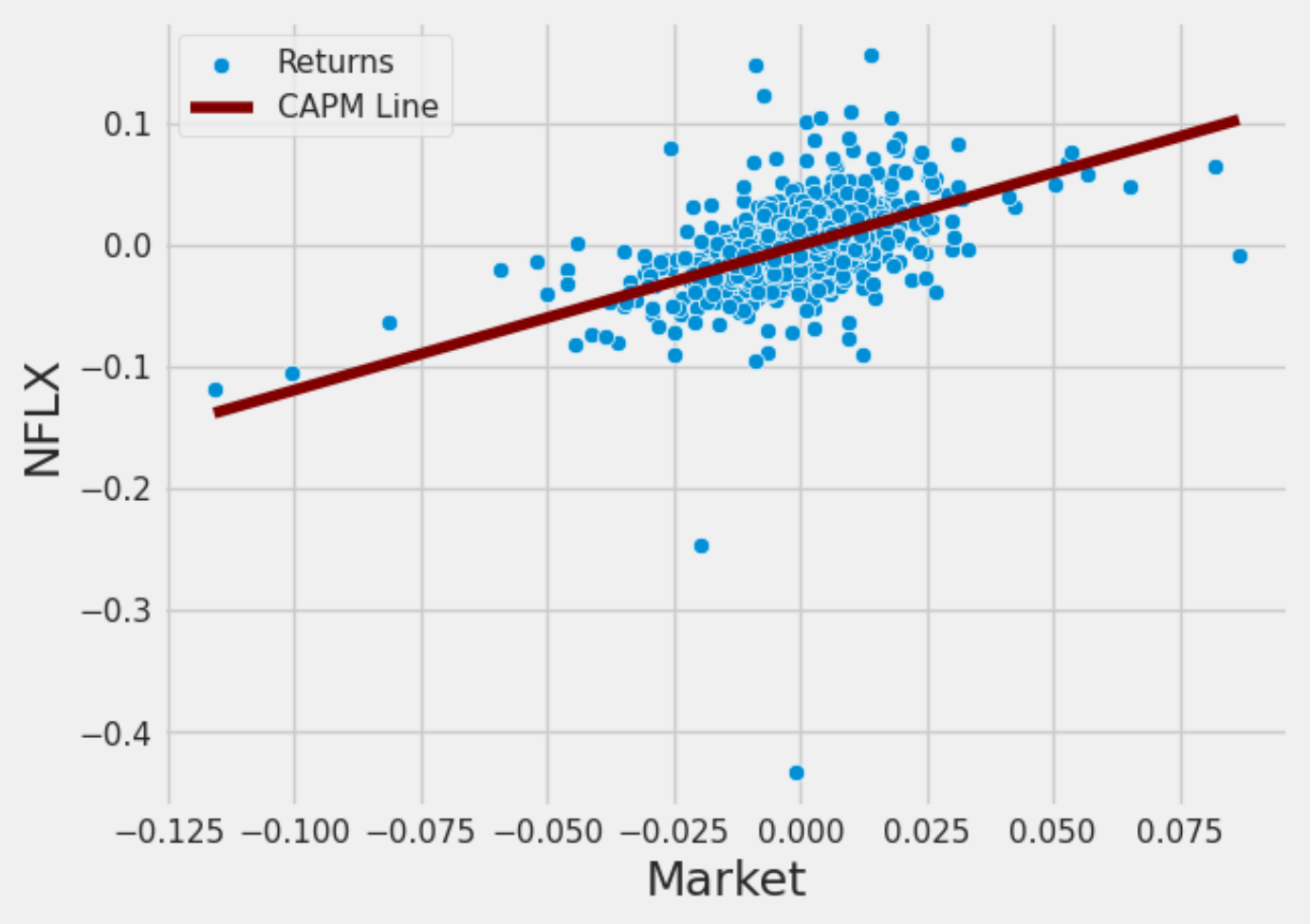
```
1.0479754432175543
```

```
# plt.axvline(0, color='grey', alpha = 0.5)
# plt.axhline(0, color='grey', alpha = 0.5)
sns.scatterplot(y = 'Market', x = 'NFLX', data = data, color = 'red')
plt.show()
```



```
sns.scatterplot(y = 'NFLX', x = 'Market', data = data, label = 'Returns')
sns.lineplot(x = data['Market'], y = alpha + (data['Market']-alpha)*beta_aapl, co
```

```
<Axes: xlabel='Market', ylabel='NFLX'>
```



```
rm = data['Market'].mean()*252
rm
cov = data[['Market','NFLX']].cov() *252
cov_NFLX_market = cov.iloc[0,1]
cov_NFLX_market
market_var = data['Market'].var()*252
market_var

NFLX_beta_annual = cov_NFLX_market / market_var
print('The annualized beta will equal the one calculated at daily returns:',NFLX_

rf = 0.025
riskpremium = rm - rf

## CAPM
NFLX_capm_return = rf + NFLX_beta_annual*riskpremium

print(f"The annualized CAPM return of NFLX is {NFLX_capm_return*100:.2f}%")
```

```
The annualized beta will equal the one calculated at daily returns: 1.0479754
The annualized CAPM return of NFLX is 13.94%
```

```
sharperatio = (rm-rf)/(data['NFLX'].std()*np.sqrt(252))
sharperatio
print(f"Sharpe Ratio: {round(sharperatio,4)}")
```

```
Sharpe Ratio: 0.233
```