statarb

July 26, 2025

1 Statistical Arbitrage

Alyssa Chen

2 Abstract

Statistical arbitrage is a quantitative trading strategy that seeks to identify and exploit market inefficiencies through mathematical models. Leveraging Python's robust computational capabilities, this project examines the application of statistical arbitrage in high-frequency trading. Key methodologies include mean reversion, cointegration analysis, and pairs trading. Using historical equity data, a Python-based model was developed to backtest and evaluate the strategy's performance. Insights from seminal works, such as "Statistical Arbitrage in the U.S. Equities Market" by Andrew Pole and "Pairs Trading: Performance of a Relative-Value Arbitrage Rule" by Gatev, Goetzmann, and Rouwenhorst, serve as the theoretical foundation. Results reveal the viability of statistical arbitrage in generating consistent returns while managing risk.

2.1 Introduction

High-frequency trading enables traders to execute complex strategies at unprecedented speeds. Among these strategies, statistical arbitrage stands out for its ability to utilize quantitative models to identify and capitalize on price discrepancies between related financial instruments. Unlike traditional arbitrage, which exploits outright price differences in identical assets across markets, statistical arbitrage relies on patterns, correlations, and mean reversion to uncover opportunities.

This project focuses on implementing a Python-based statistical arbitrage framework, drawing inspiration from Andrew Pole's "Statistical Arbitrage in the U.S. Equities Market" and Gatev, Goetzmann, and Rouwenhorst's "Pairs Trading: Performance of a Relative-Value Arbitrage Rule." The primary objective is to develop, backtest, and evaluate a pairs trading strategy to demonstrate its effectiveness in high-frequency trading scenarios.

[89]:
```
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.stattools import coint
import itertools
```

[90]:
```python
# tech
aapl = yf.download('AAPL', start='2020-01-01', end='2023-01-01',␣
 ↪auto_adjust=True)
msft = yf.download('MSFT', start='2020-01-01', end='2023-01-01',␣
 ↪auto_adjust=True)
goog = yf.download('GOOG', start='2020-01-01', end='2023-01-01',␣
 ↪auto_adjust=True)
amzn = yf.download('AMZN', start='2020-01-01', end='2023-01-01',␣
 ↪auto_adjust=True)
meta = yf.download('META', start='2020-01-01', end='2023-01-01',␣
 ↪auto_adjust=True)
nvda = yf.download('NVDA', start='2020-01-01', end='2023-01-01',␣
 ↪auto_adjust=True)
tsla = yf.download('TSLA', start='2020-01-01', end='2023-01-01',␣
 ↪auto_adjust=True)
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

[91]:
```python
# banks
jpm = yf.download('JPM', start='2020-01-01', end='2023-01-01', auto_adjust=True)
bac = yf.download('BAC', start='2020-01-01', end='2023-01-01', auto_adjust=True)
gs = yf.download('GS', start='2020-01-01', end='2023-01-01', auto_adjust=True)
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

[92]:
```python
# etfs
spy = yf.download('spy', start='2020-01-01', end='2023-01-01', auto_adjust=True)
voo = yf.download('voo', start='2020-01-01', end='2023-01-01', auto_adjust=True)
ivv = yf.download('ivv', start='2020-01-01', end='2023-01-01', auto_adjust=True)
qqq = yf.download('qqq', start='2020-01-01', end='2023-01-01', auto_adjust=True)
xlk = yf.download('xlk', start='2020-01-01', end='2023-01-01', auto_adjust=True)
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

```
[93]: # oil
xom = yf.download('xom', start='2020-01-01', end='2023-01-01', auto_adjust=True)
cvx = yf.download('cvx', start='2020-01-01', end='2023-01-01', auto_adjust=True)
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

2.1.1 Dataset

The analysis utilizes historical price data for equities and ETFs sourced from public APIs, such as Yahoo Finance and Alpha Vantage. The dataset includes:

Daily closing prices for a selected group of liquid stocks over a 5-year period.

Sector-specific filtering to identify stocks with similar characteristics.

Preprocessing steps include handling missing data, normalizing prices, and computing log returns.

Pair selection criteria involve identifying asset pairs with high historical correlations and conducting cointegration tests (e.g., Augmented Dickey-Fuller and Johansen tests) to establish long-term relationships.

```
[94]: # Merge the data on the date and use 'Close' column for closing prices
data = pd.concat(
    [aapl['Close'], msft['Close'], goog['Close'], amzn['Close'], meta['Close'],␣
 ↪nvda['Close'], tsla['Close'],
     jpm['Close'], bac['Close'], gs['Close'],
     spy['Close'], voo['Close'], ivv['Close'], qqq['Close'], xlk['Close'],
     xom['Close'], cvx['Close'],],
    axis=1,
    keys=['AAPL', 'MSFT', 'GOOG', 'AMZN', 'META', 'NVDA', 'TSLA', 'JPM', 'BAC',␣
 ↪'GS', 'SPY', 'VOO', 'IVV', 'QQQ', 'XLK', 'XOM', 'CVX']
)

if isinstance(data.columns, pd.MultiIndex):
    data.columns = [col[0] if isinstance(col, tuple) else col for col in data.
 ↪columns]

data.dropna(inplace=True)

data
```

[94]:

	AAPL	MSFT	GOOG	AMZN	META	\
Date						
2020-01-02	72.620857	153.042297	67.964500	94.900497	208.635391	
2020-01-03	71.914818	151.136658	67.630989	93.748497	207.531479	
2020-01-06	72.487854	151.527313	69.298576	95.143997	211.440033	
2020-01-07	72.146935	150.145706	69.255341	95.343002	211.897507	
2020-01-08	73.307518	152.537308	69.801094	94.598503	214.045731	
…	…	…	…	…	…	

2022-12-23	130.173782	233.975861	89.279305	85.250000	117.395966
2022-12-27	128.367188	232.241135	87.410423	83.040001	116.242294
2022-12-28	124.428223	229.859528	85.949112	81.820000	114.989159
2022-12-29	127.952568	236.210480	88.424393	84.180000	119.603851
2022-12-30	128.268448	235.044189	88.205704	84.000000	119.683411

	NVDA	TSLA	JPM	BAC	GS	\
Date						
2020-01-02	5.971747	28.684000	120.154701	31.092318	205.282990	
2020-01-03	5.876163	29.534000	118.569092	30.446732	202.882507	
2020-01-06	5.900805	30.102667	118.474815	30.403116	204.958847	
2020-01-07	5.972244	31.270666	116.460686	30.202461	206.307983	
2020-01-08	5.983446	32.809334	117.369179	30.507811	208.296661	
…	…	…	…	…	…	
2022-12-23	15.192497	123.150002	122.518608	30.336744	323.748169	
2022-12-27	14.108459	109.099998	122.947891	30.392799	320.431061	
2022-12-28	14.023535	112.709999	123.619881	30.617031	319.400421	
2022-12-29	14.590030	121.820000	124.329140	30.962721	321.799133	
2022-12-30	14.601022	123.180000	125.150406	30.944040	321.752289	

	SPY	VOO	IVV	QQQ	XLK	\
Date						
2020-01-02	299.406464	274.314545	299.997070	209.091110	88.947235	
2020-01-03	297.139221	272.310577	297.689545	207.175797	87.947174	
2020-01-06	298.272858	273.330963	298.866302	208.510666	88.156700	
2020-01-07	297.434265	272.577240	298.048096	208.481674	88.118599	
2020-01-08	299.019379	273.983643	299.565033	210.048706	89.061508	
…	…	…	…	…	…	
2022-12-23	370.189240	339.575409	371.110718	263.268097	122.269638	
2022-12-27	368.729401	338.224365	369.682587	259.545898	121.111717	
2022-12-28	364.146881	334.074554	365.195587	256.119232	119.158920	
2022-12-29	370.701630	340.000122	371.516083	262.362213	122.279442	
2022-12-30	369.725159	339.063965	370.744141	262.204559	122.112648	

	XOM	CVX
Date		
2020-01-02	54.634792	95.001534
2020-01-03	54.195545	94.672951
2020-01-06	54.611656	94.352180
2020-01-07	54.164719	93.147362
2020-01-08	53.347893	92.083351
…	…	…
2022-12-23	99.805168	159.900543
2022-12-27	101.191872	161.910583
2022-12-28	99.529663	159.521973
2022-12-29	100.282722	160.729813
2022-12-30	101.292892	161.784409

```
[756 rows x 17 columns]
```

[95]:
```python
# Calculate daily returns
returns = data.pct_change().dropna()
```

Next, we will perform a cointegration test to identify pairs of assets suitable for statistical arbitrage. We will test for cointegration using the Engle-Granger two-step method, which involves estimating a long-term equilibrium relationship and testing the residuals for stationarity.

[96]:
```python
def half_life(spread: pd.Series) -> float:
    """
    Estimate OU half-life via regression of Δspread on lagged spread.
    Returns half-life in the same units as your index (trading days).
    """
    s = spread.dropna()
    s_lag = s.shift(1).iloc[1:]
    delta = s.diff().iloc[1:]
    beta  = np.polyfit(s_lag, delta, 1)[0]
    if beta >= 0:
        return np.nan
    return -np.log(2) / beta

# skip ETF-duplicates
etf_groups = [
    {'SPY','VOO','IVV'},
    {'XOM','CVX','COP'}
]

results = []
tickers = list(data.columns)

for i in range(len(tickers)):
    for j in range(i+1, len(tickers)):
        t1, t2 = tickers[i], tickers[j]
        s1, s2 = data[t1], data[t2]

        if any({t1,t2} <= grp for grp in etf_groups):
            continue

        # skip if their returns correlate > 0.98
        r1 = s1.pct_change()
        r2 = s2.pct_change()
        if r1.corr(r2) > 0.98:
            continue

        score, pval, _ = coint(s1, s2)
```

```
        if pval >= 0.05:
            continue

        X = sm.add_constant(s1)
        beta  = sm.OLS(s2, X).fit().params[t1]

        spread = s2 - beta * s1

        hl = half_life(spread)
        if np.isnan(hl) or hl > 252:
            # drop if not mean-reverting or half-life > 1 year
            continue

        results.append({
            'pair':     (t1, t2),
            'pval':     pval,
            'half_life': hl,
            'beta':     beta
        })

res_df = pd.DataFrame(results)
res_df = res_df.sort_values('half_life').reset_index(drop=True)

print("=== Top Cointegrated Pairs (by half-life) ===")
print(res_df[['pair','pval','half_life','beta']].head(10))
```

```
=== Top Cointegrated Pairs (by half-life) ===
         pair      pval  half_life      beta
0  (GOOG, BAC)  0.017625  23.613625  0.242902
1   (BAC, XLK)  0.044241  27.987227  2.814301
2  (MSFT, BAC)  0.014156  29.072570  0.129018
```

A p-value below 0.05 indicates that the two series are cointegrated, which makes them suitable for pairs trading. Now that we've filtered out all pairs with a p-value less than 0.05, we can go ahead and calculate the spread and implement our pairs trading strategy.

[97]:
```
split = '2022-01-01'
data_in  = data.loc[:split]
data_out = data.loc[split:]
```

2.2 Methods (Modeling)

Mean Reversion and Cointegration Analysis

- Mean Reversion: The hypothesis that asset prices tend to revert to their historical mean.
- Cointegration: Statistical techniques, including the Augmented Dickey-Fuller and Johansen tests, are applied to identify pairs of assets that maintain a stable, long-term relationship.

Pairs Trading Strategy

- Entry Criteria: Z-score thresholds are used to signal potential trading opportunities based on deviations from the historical price spread.
- Exit Criteria: Positions are closed when the spread returns to the mean or other pre-defined conditions are met.

Backtesting Framework

- A Python-based backtesting environment was implemented to evaluate the strategy.

[98]:
```
def safe_sharpe(returns, ann_factor=252):
     = returns.std()
    if   <= 0 or np.isnan( ):
        return np.nan
     = returns.mean()
    return  /  * np.sqrt(ann_factor)
```

[99]:
```
def backtest_pair(data, t1, t2, beta, lookback, entry_z, exit_z, capital=100_000):
    spread = data[t2] - beta * data[t1]
          = spread.rolling(lookback).mean()
          = spread.rolling(lookback).std()
    z      = (spread -  ) / 

    sig = np.where(z < -entry_z,  +1,
          np.where(z >  entry_z,  -1, np.nan))
    pos = pd.Series(sig, index=data.index).ffill().fillna(0)
    pos[z.abs() < exit_z] = 0

    pnl     = pos.shift(1) * spread.diff().fillna(0)
    equity  = capital + pnl.cumsum()
    returns = pnl / capital

    sr  = safe_sharpe(returns)
    mdd = (equity.cummax() - equity).max()
    wr  = (returns > 0).mean()

    return {
        'Spread':       spread,
        'Z':            z,
        'Position':     pos,
        'Equity':       equity,
        'Returns':      returns,
        'Sharpe':       sr,
        'MaxDrawdown':  mdd,
        'WinRate':      wr
    }
```

```
[100]: lookback  = int(round(res_df.loc[0,'half_life']))
entry_z   = 2.0
exit_z    = 0.5
```

```
[101]: results = []
for idx, row in res_df.iterrows():
    t1, t2 = row['pair']
    beta   = row['beta']
    hl     = int(round(row['half_life']))
    out    = backtest_pair(data_out, t1, t2, beta, hl, 2.0, 0.5)
    results.append({
        'pair':     (t1,t2),
        'Sharpe':   out['Sharpe'],
        'Drawdown': out['MaxDrawdown'],
        'WinRate':  out['WinRate']
    })

perf = pd.DataFrame(results).set_index('pair')
print(perf)
```

```
               Sharpe   Drawdown   WinRate
pair
(GOOG, BAC)  0.667604   9.515065  0.358566
(BAC, XLK)   0.733761  19.073711  0.294821
(MSFT, BAC) -0.221997   8.942470  0.310757
```

```
[102]: TOP_N = 5

all_results = []

for _, row in res_df.iterrows():
    t1, t2 = row['pair']
    beta   = row['beta']
    hl     = int(round(row['half_life']))

    lookbacks = [max(5, int(hl * fac)) for fac in (0.5, 1.0, 1.5)]
    entry_zs  = [1.5, 2.0, 2.5, 3.0]
    exit_zs   = [0.3, 0.5, 0.7, 1.0]

    rows = []
    for L, E, X in itertools.product(lookbacks, entry_zs, exit_zs):
        out = backtest_pair(data_out, t1, t2, beta, L, E, X)
        rows.append({
            'pair':      (t1, t2),
            'beta':      beta,
            'lookback':  L,
            'entry_z':   E,
```

```python
            'exit_z':    X,
            'Sharpe':    out['Sharpe'],
            'Drawdown':  out['MaxDrawdown'],
            'WinRate':   out['WinRate']
        })

    grid = pd.DataFrame(rows)
    top_grid = grid.sort_values('Sharpe', ascending=False).head(TOP_N)
    all_results.append(top_grid)

final_grid = pd.concat(all_results).reset_index(drop=True)

print("=== Top parameter combos per pair ===")
print(final_grid.to_string(index=False))
```

```
=== Top parameter combos per pair ===
       pair      beta  lookback  entry_z  exit_z   Sharpe  Drawdown  WinRate
(GOOG, BAC) 0.242902        24      2.0     0.7 1.189194  7.800160 0.310757
(GOOG, BAC) 0.242902        24      2.0     1.0 0.680744  8.278505 0.254980
(GOOG, BAC) 0.242902        24      2.0     0.5 0.667604  9.515065 0.358566
(GOOG, BAC) 0.242902        24      2.5     0.7 0.579409  9.505317 0.278884
(GOOG, BAC) 0.242902        24      2.0     0.3 0.551356 10.333151 0.406375
 (BAC, XLK) 2.814301        42      2.5     0.3 2.544526 10.868481 0.215139
 (BAC, XLK) 2.814301        42      2.5     0.5 2.108341  9.470009 0.191235
 (BAC, XLK) 2.814301        42      2.5     0.7 1.782134  9.470009 0.179283
 (BAC, XLK) 2.814301        42      2.5     1.0 1.773013  7.670079 0.151394
 (BAC, XLK) 2.814301        42      1.5     0.3 1.197273 23.156735 0.422311
(MSFT, BAC) 0.129018        43      3.0     0.5 1.644935  3.326943 0.219124
(MSFT, BAC) 0.129018        43      3.0     1.0 1.535757  2.509105 0.167331
(MSFT, BAC) 0.129018        43      3.0     0.3 1.504251  3.672011 0.243028
(MSFT, BAC) 0.129018        43      3.0     0.7 1.468301  3.326943 0.183267
(MSFT, BAC) 0.129018        29      2.5     0.3 1.457165  3.475392 0.235060
```

GOOG-BAC has a good sharpe of 1.189 with a decent win rate. MFST-BAC with sharpe of 1.197 is also interesting because the win rate is ~42% but the drawback is 23, so we could use it is we prioritize capturing frequent small wins. I think we will go with the GOOG-BAC.

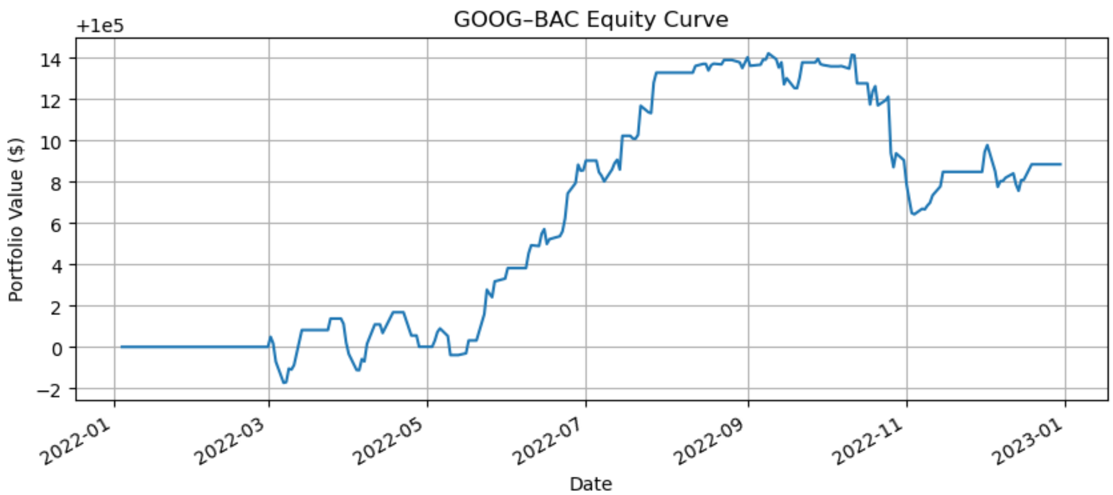
[103]:
```python
# optimized values based off the above selection

t1, t2      = 'GOOG', 'BAC'
beta        = 0.242902
lookback    = 24
entry_z     = 2.0
exit_z      = 0.7
capital     = 100_000
```

[104]:
```python
res = backtest_pair(data_out, t1, t2, beta, lookback, entry_z, exit_z, capital)

# unpack
sharpe     = res['Sharpe']
drawdown   = res['MaxDrawdown']
win_rate   = res['WinRate']
equity     = res['Equity']      # pd.Series of equity curve
returns    = res['Returns']     # pd.Series of daily returns
position = res['Position']
zscore   = res['Z']
```

[105]:
```python
print(f"GOOG-BAC  Sharpe: {sharpe:.2f}   "
      f"Max Drawdown: {drawdown:.1f}   "
      f"Win Rate: {win_rate:.1%}")
```

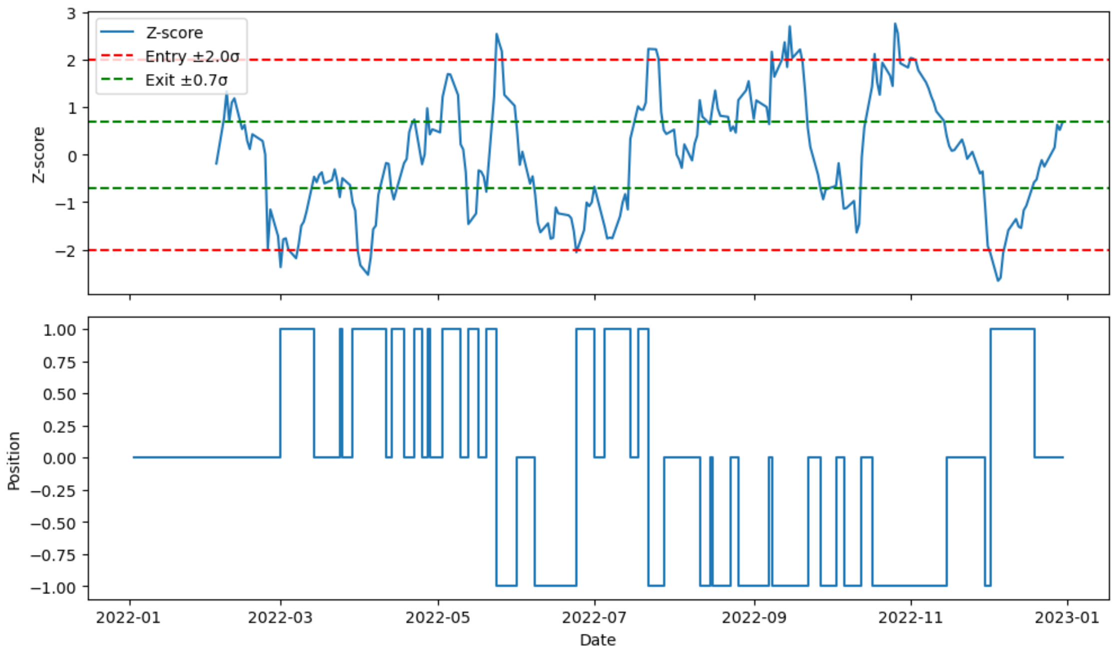
```
GOOG-BAC  Sharpe: 1.19   Max Drawdown: 7.8   Win Rate: 31.1%
```

[106]:
```python
plt.figure(figsize=(10,4))
equity.plot()
plt.title("GOOG-BAC Equity Curve")
plt.ylabel("Portfolio Value ($)")
plt.xlabel("Date")
plt.grid(True)
plt.show()
```



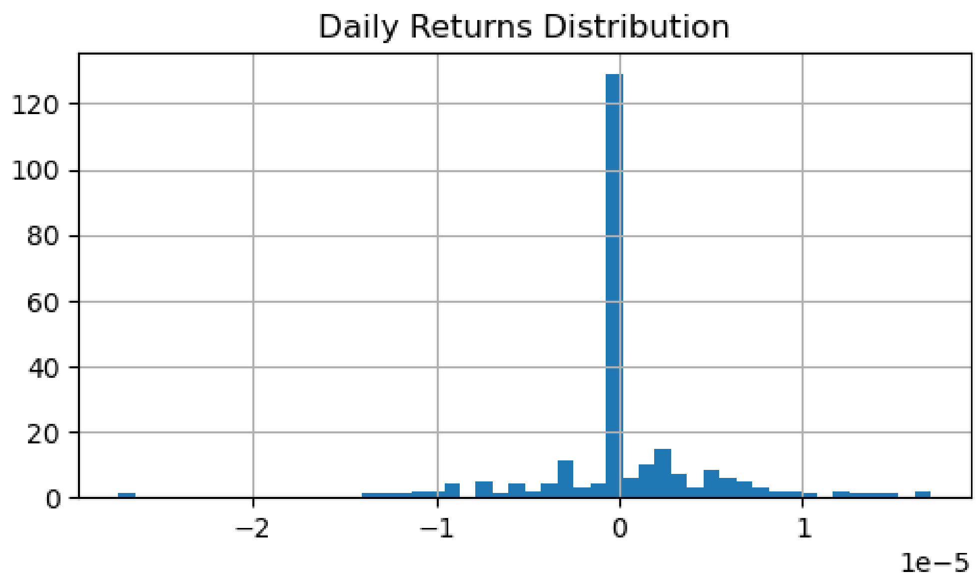
[107]:
```python
fig, ax = plt.subplots(2, 1, figsize=(10, 6), sharex=True)

ax[0].plot(zscore.index, zscore, label="Z-score")
ax[0].axhline( entry_z, color='r', linestyle='--', label=f"Entry ±{entry_z} ")
ax[0].axhline(-entry_z, color='r', linestyle='--')
```

```
ax[0].axhline( exit_z, color='g', linestyle='--', label=f"Exit ±{exit_z} ")
ax[0].axhline(-exit_z, color='g', linestyle='--')
ax[0].set_ylabel("Z-score")
ax[0].legend(loc='upper left')

ax[1].step(position.index, position, where='post')
ax[1].set_ylabel("Position")
ax[1].set_xlabel("Date")

plt.tight_layout()
plt.show()
```



[108]:

```
plt.figure(figsize=(6,3))
returns.hist(bins=50)
plt.title("Daily Returns Distribution")
plt.show()
print("Mean:", returns.mean(), "Std:", returns.std())
```

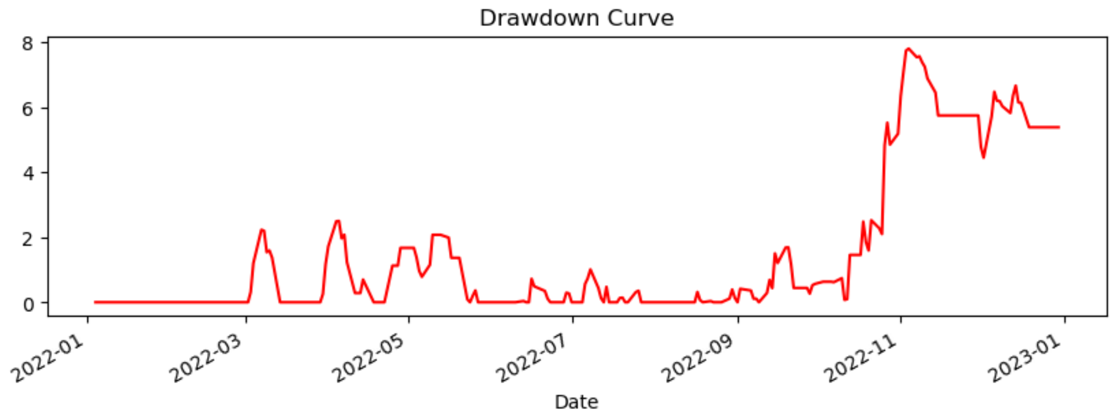



```
Mean: 3.5388291147827103e-07 Std: 4.723968959570621e-06
```

[109]:

```
trades = res['Position'].diff().abs().sum()
print("Number of Trades:", int(trades))

holds = res['Position'].ne(0)
holding_days = holds.groupby((holds != holds.shift()).cumsum()).
 ↪transform('size')
print("Avg Holding Period (days):", holding_days[holds].mean())

dd = res['Equity'].cummax() - res['Equity']
plt.figure(figsize=(10,3))
dd.plot(color='red')
plt.title("Drawdown Curve")
plt.show()
```

```
Number of Trades: 48
Avg Holding Period (days): 10.735294117647058
```



2.3 Conclusion

This project evaluates the potential of statistical arbitrage as a trading strategy in high-frequency markets. By leveraging Python for data analysis and model implementation, traders can harness advanced techniques such as mean reversion and cointegration to identify profitable opportunities. The insights derived from this analysis align with findings from foundational research, reinforcing the relevance of statistical arbitrage in modern financial markets.

Resources used: 1. Statistical Arbitrage in the U.S. Equities Market by Andrew Pole 2. Pairs Trading: Performance of a Relative-Value Arbitrage Rule by Gatev, Goetzmann, and Rouwenhorst 3. Quantitative Finance Stack Exchange