StatisticalArbitrage

May 25, 2025

1 Evaluating Statistical Properties of a Multi-Asset Spread

Statistical Arbitrage is a trading strategy that aims to identify a temporary deviation from historical/expected statistical relationship between a group of assets and makes a bet that the deviation will revert, thereby generating profit. A common simple starting point for such a strategy is referred to as a pairs-trade. Suppose there is some statistical relationship X and Y (for example, we might expect the prices of two technology stocks to move together). This relationship may take the form $Y = \beta X + \alpha$ and if the magnitude of the spread $S = Y - \beta X - \alpha$ is high enough, this may signal a temporary deviation from the relationship that we can exploit.

In this report, we will investigate the process of identifying such a relationship, and rather than focusing on a pairs trade, we will look at a 'basket' of three assets. In other words, we will trade some asset Y against two assets, $X1, X2$, though the approaches explored here should work for a basket of arbitrary size as well. In this example, let us analyze the following basket of cryptocurrency perpetuals.

$$\text{ADA}_t = s + \beta_1 \text{AVAX}_t + \beta_2 \text{XLM}_t + \epsilon_t$$

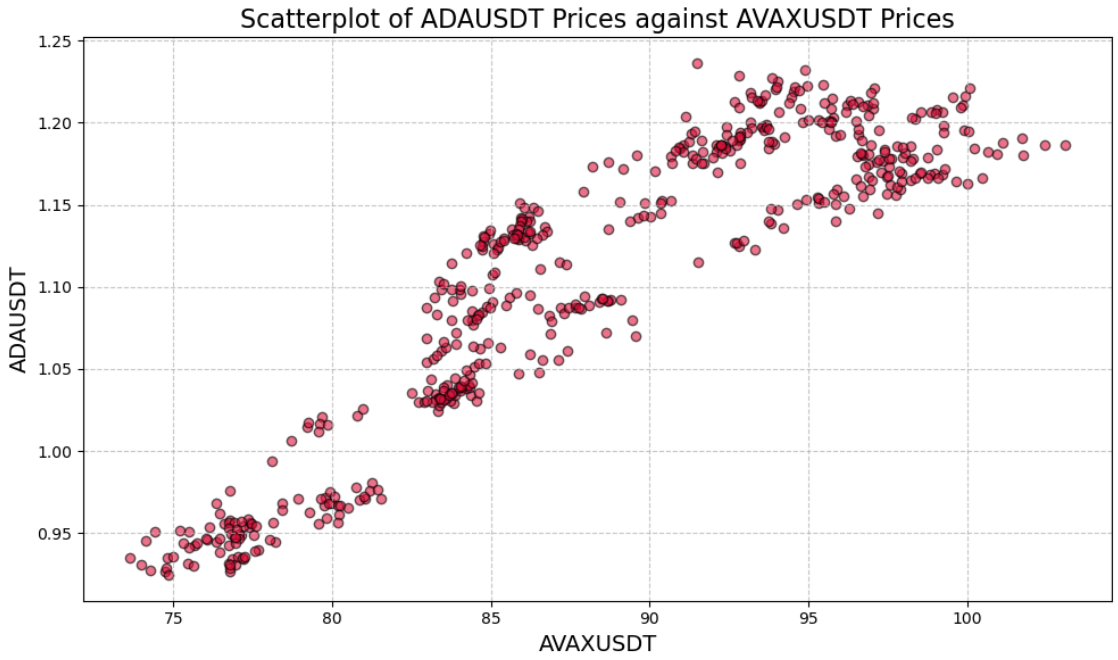
[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import sys

TOP_LEVEL_DIR = os.path.abspath(os.path.join(os.getcwd(), '..'))
sys.path.append(TOP_LEVEL_DIR)

from utils.data_vis import plot_series, plot_histogram
```

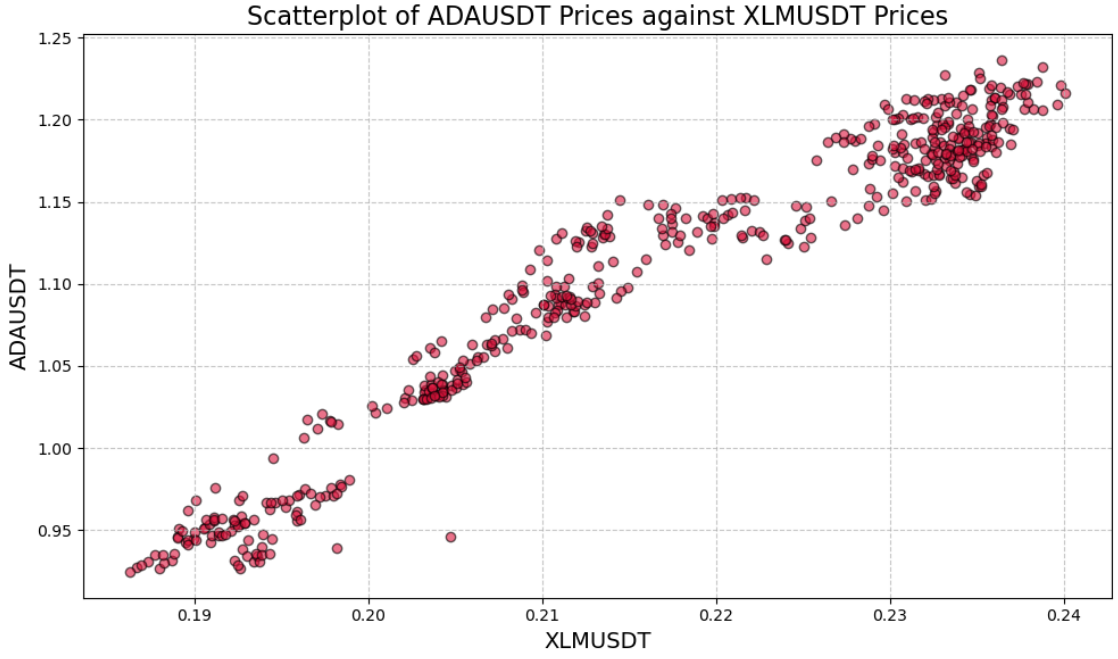
[ ]:
```
DIR = '../data/cryptocurrencies/candlestick/'
SYMBOLS = [symbol for symbol in os.listdir(DIR)]
dfs = {}
for symbol in SYMBOLS:
    sub_dir = os.path.join(DIR, symbol)
    csv_files = [os.path.join(sub_dir, f) for f in os.listdir(sub_dir) if f.
 ↪endswith('.csv')]
```

```
    dfs[symbol] = pd.concat([pd.read_csv(f) for f in csv_files],␣
 ↪ignore_index=True)

for symbol in dfs.keys():
    df = dfs.get(symbol)
    df['open_time'] = pd.to_datetime(df['open_time'], unit='ms')
    df['mid'] = (df['high'] + df['low'])/2
    df = df.sort_values('open_time')
    dfs[symbol] = df

def prepare_and_resample(df, interval='60T'):
    df = df.copy()
    df['open_time'] = pd.to_datetime(df['open_time'])
    df.set_index('open_time', inplace=True)

    df_resampled = df.resample(interval).agg({
        'open': 'first',
        'high': 'max',
        'low': 'min',
        'close': 'last',
        'volume': 'sum',
        'quote_volume': 'sum',
        'count': 'sum',
        'taker_buy_base_volume': 'sum',
        'taker_buy_quote_volume': 'sum',
        'mid': 'mean'
    }).dropna()

    return df_resampled

Y = prepare_and_resample(dfs.get('ADAUSDT'))
X1 = prepare_and_resample(dfs.get('AVAXUSDT'))
X2 = prepare_and_resample(dfs.get('XLMUSDT'))
```

[21]:
```
plt.figure(figsize=(10, 6))
plt.scatter(X1['mid'][2000:2500], Y['mid'][2000:2500], c='crimson', alpha=0.6,␣
 ↪edgecolors='k')
plt.title('Scatterplot of ADAUSDT Prices against AVAXUSDT Prices', fontsize=16)
plt.xlabel('AVAXUSDT', fontsize=14)
plt.ylabel('ADAUSDT', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(X2['mid'][2000:2500], Y['mid'][2000:2500], c='crimson', alpha=0.6,␣
 ↪edgecolors='k')
```

```
plt.title('Scatterplot of ADAUSDT Prices against XLMUSDT Prices', fontsize=16)
plt.xlabel('XLMUSDT', fontsize=14)
plt.ylabel('ADAUSDT', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```





As we see above, the prices of our dependent variable price series and our independent variables display some linearity and correlation. I've only displayed a small segment of the data to avoid cluttering but the full scatterplot also has a clear positive correlation ($r^2 = 0.960153$). However, price correlation alone is not a sufficient statistical indication that this portfolio is suitable for a statistical arbitrage strategy. First, we will test the individual price series of ADAUSDT, AVAXUSDT and XRPUSDT; we want to statistically test whether these series are $I(1)$. We want the individual series to be $I(1)$, non-stationary processes, while we want the residuals of the linear-regression equation to be stationary (mean-reverting) within a certain statistical level of confidence.

We will apply the Engle-Granger Test, the Augmented-Dickey-Fuller, and the Kwiatkowski–Phillips–Schmidt–Shin Test to the respective series. As mentioned above, we want the price series of each security to be an $I(1)$ non-stationary process. This means, for the first two tests, we wish to fail to reject the null hypothesis, and in the KPPS test, reject the null hypothesis.

Then for the constructed spread, we want this series to be stationary, which means we want the opposite results for the statistical tests.

[27]:
```python
from arch.unitroot import ADF, PhillipsPerron, KPSS

def unit_root_summary(series, name="Series"):
    print(f"\n Unit Root Tests for: {name}")
    print("="*60)

    # ADF Test
    adf = ADF(series)
    print("\n Augmented Dickey-Fuller (ADF) Test")
    print(f"Test Statistic: {adf.stat:.4f}")
    print(f"p-value       : {adf.pvalue:.4f}")
    print(f"Lags Used     : {adf.lags}")
    print(f"Null Hypothesis: Non-stationary (has unit root)")

    # Phillips-Perron Test
    pp = PhillipsPerron(series)
    print("\n Phillips-Perron (PP) Test")
    print(f"Test Statistic: {pp.stat:.4f}")
    print(f"p-value       : {pp.pvalue:.4f}")
    print(f"Lags Used     : {pp.lags}")
    print(f"Null Hypothesis: Non-stationary (has unit root)")

    # KPSS Test
    kpss = KPSS(series)
    print("\n KPSS Test")
    print(f"Test Statistic: {kpss.stat:.4f}")
    print(f"p-value       : {kpss.pvalue:.4f}")
    print(f"Lags Used     : {kpss.lags}")
    print(f"Null Hypothesis: Stationary")
```

```
    print("="*60)
```

[24]:
```
unit_root_summary(Y['mid'])
```

```
 Unit Root Tests for: Series
============================================================

 Augmented Dickey-Fuller (ADF) Test
Test Statistic: -3.2887
p-value       : 0.0154
Lags Used     : 48
Null Hypothesis: Non-stationary (has unit root)

 Phillips-Perron (PP) Test
Test Statistic: -3.1661
p-value       : 0.0220
Lags Used     : 50
Null Hypothesis: Non-stationary (has unit root)

 KPSS Test
Test Statistic: 4.0424
p-value       : 0.0001
Lags Used     : 99
Null Hypothesis: Stationary
============================================================
```

[25]:
```
unit_root_summary(X1['mid'])
```

```
 Unit Root Tests for: Series
============================================================

 Augmented Dickey-Fuller (ADF) Test
Test Statistic: -4.1811
p-value       : 0.0007
Lags Used     : 47
Null Hypothesis: Non-stationary (has unit root)

 Phillips-Perron (PP) Test
Test Statistic: -3.9613
p-value       : 0.0016
Lags Used     : 50
Null Hypothesis: Non-stationary (has unit root)

 KPSS Test
Test Statistic: 4.2713
```

```
p-value        : 0.0001
Lags Used      : 99
Null Hypothesis: Stationary
============================================================
```

[26]:
```
unit_root_summary(X2['mid'])
```

```
 Unit Root Tests for: Series
============================================================

 Augmented Dickey-Fuller (ADF) Test
Test Statistic: -2.3757
p-value        : 0.1487
Lags Used      : 50
Null Hypothesis: Non-stationary (has unit root)

 Phillips-Perron (PP) Test
Test Statistic: -2.1247
p-value        : 0.2347
Lags Used      : 50
Null Hypothesis: Non-stationary (has unit root)

 KPSS Test
Test Statistic: 6.1057
p-value        : 0.0001
Lags Used      : 99
Null Hypothesis: Stationary
============================================================
```

[28]:
```
import statsmodels.api as sm
```

[29]:
```
basket = [Y, X1, X2]
y = basket[0]['mid']
X1 = basket[1]['mid']
X2 = basket[2]['mid']

df_reg = pd.concat([y, X1, X2], axis=1)
df_reg.columns = ['ADA', 'XLM', 'AVAX']
df_reg = df_reg.dropna()

y = df_reg['ADA']
X = df_reg[['XLM', 'AVAX']]
X = sm.add_constant(X)

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

```
print(results.params)

a = results.params.get('const')
b1 = results.params.get('XLM')
b2 = results.params.get('AVAX')

spread = y - b1*X1 - b2*X2
```

```
const    0.070233
XLM      0.007867
AVAX     1.512114
dtype: float64
```

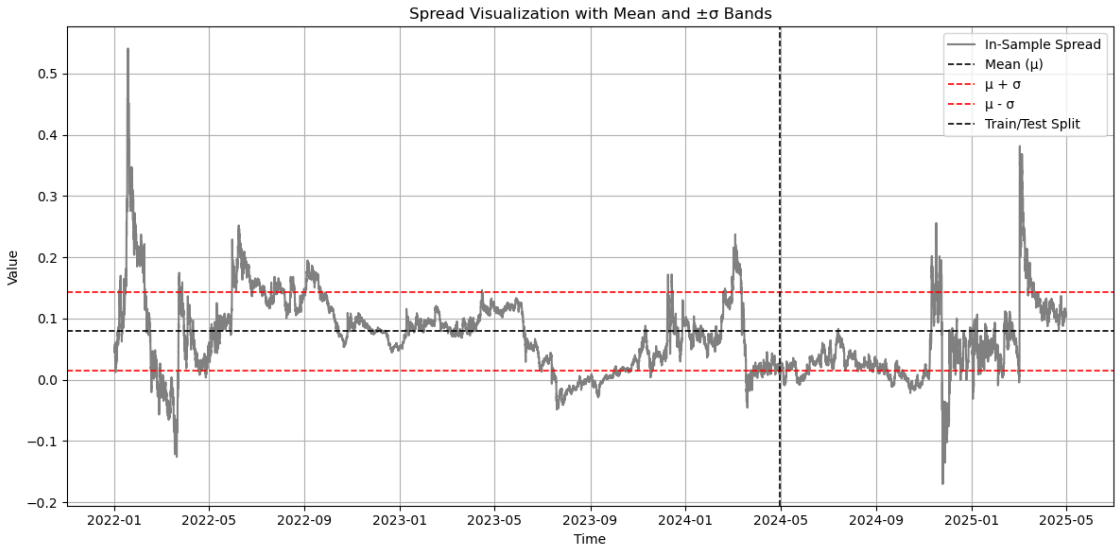
[30]:
```
unit_root_summary(spread)
```

```
 Unit Root Tests for: Series
============================================================

 Augmented Dickey-Fuller (ADF) Test
Test Statistic: -5.7281
p-value       : 0.0000
Lags Used     : 50
Null Hypothesis: Non-stationary (has unit root)

 Phillips-Perron (PP) Test
Test Statistic: -5.7105
p-value       : 0.0000
Lags Used     : 50
Null Hypothesis: Non-stationary (has unit root)

 KPSS Test
Test Statistic: 4.1037
p-value       : 0.0001
Lags Used     : 99
Null Hypothesis: Stationary
============================================================
```

From these test results, we can reasonably say that our spread series is mean-reverting and thereby suitable for a statistical-arbitrage strategy. Let us visualize the spread series below.

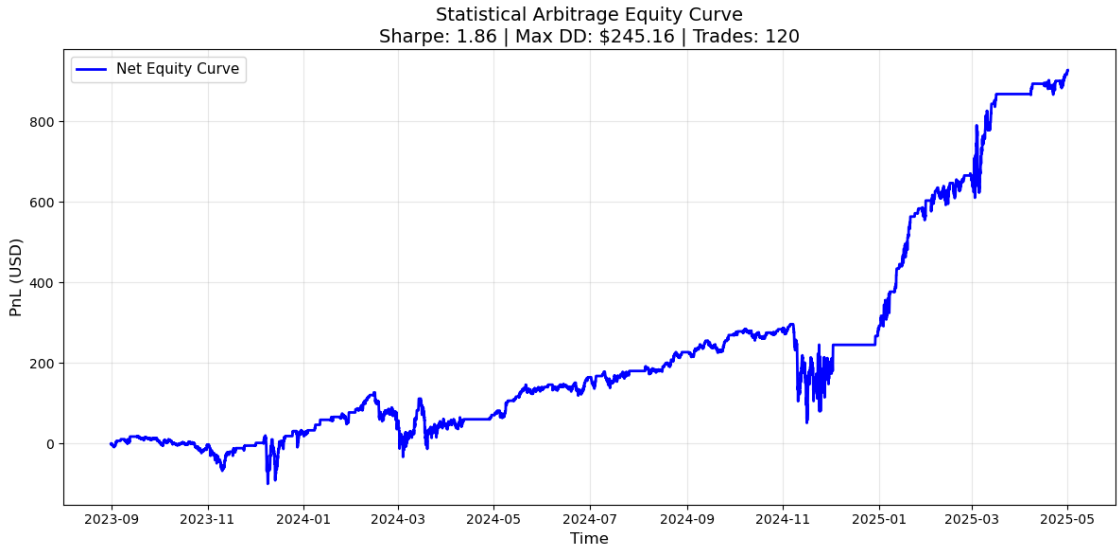
[33]:
```
sample_size = 0.7
in_sample = spread[: int(sample_size * len(spread))]
mu = in_sample.mean()
sigma = in_sample.std()

plt.figure(figsize=(12, 6))
plt.plot(spread.index, spread.values, color='grey', linewidth=1.5,␣
 ↪label='In-Sample Spread')
```

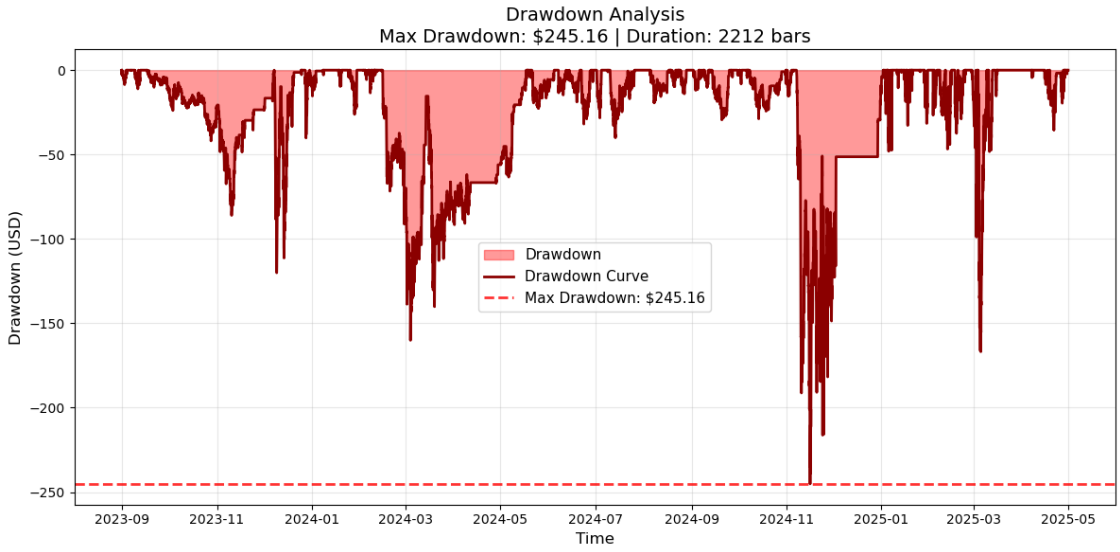
```
plt.axhline(mu, color='black', linestyle='--', linewidth=1.2, label='Mean (μ)')

plt.axhline(mu + sigma, color='red', linestyle='--', linewidth=1.2, label='μ +␣
↪σ')
plt.axhline(mu - sigma, color='red', linestyle='--', linewidth=1.2, label='μ -␣
↪σ')

split_index = int(sample_size * len(spread))
split_time = spread.index[split_index]
plt.axvline(x=split_time, color='black', linestyle='--', linewidth=1.2,␣
↪label='Train/Test Split')

plt.title("Spread Visualization with Mean and ±σ Bands")
plt.xlabel("Time")
plt.ylabel("Value")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

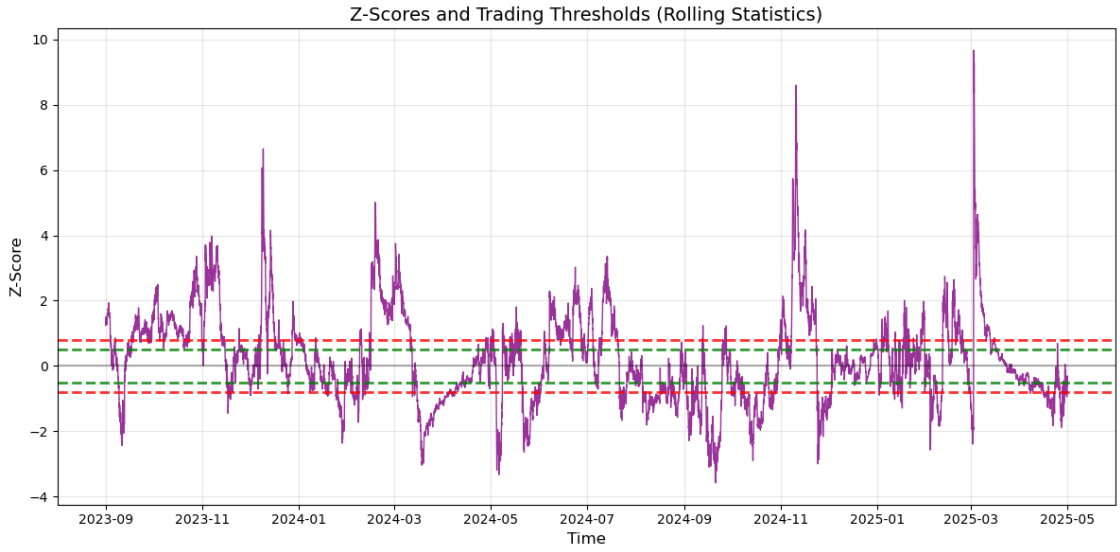


The general strategy is when the spread is a certain threshold above the average value of the spread, we short one unit of the spread expecting the value of the spread to revert to the mean. Shorting one unit of the spread means shorting one unit of Y, and longing β_1 units of X_1 and β_2 units of X_2. Similarly, if the spread is a certain threshold below the average value of the spread, we long, expecting the spread to revert to the mean. Longing one unit of the spread means longing one unit of Y, and shorting β_1 units of X_1 and β_2 units of X_2. The mean and z-scores are often taken to be rolling metrics; ie, we construct our signal based on the z-score's position relative to some mean over a rolling window. Below are some diagrams of an out-of-sample backtest for the given spread,

with modeled transaction costs.







One major issue with this backtest is the severe drawdowns. The maximum drawdown is over 20% of the total invested capital, which is far from ideal. In another report, we will explore reducing drawdowns using a stop-loss mechanism for risk-management.