BOCCONI UNIVERSITY

MASTER OF QUANTITATIVE FINANCE AND RISK MANAGEMENT

XXI Edition 2024-2025

Research Project

Multi-Spread Statistical Arbitrage: Modeling Mean Reversion in Commodities by Cointegration

Student: Maxime Gressé 3323429 **Supervisor:** Prof. Barbara Alemanni

Abstract: This paper explores a multi-spread statistical arbitrage strategy applied to a diversified set of commodities. Using Johansen's cointegration methodology, we identify long-term equilibrium relationships between commodity prices and construct stationary spreads. Trading signals are generated when these spreads deviate significantly from equilibrium, with entry and exit rules based on z-score thresholds.

This trading algorithm was developed using Python. The lines of code are added in an appendix for each part.

Keywords: *Statistical Arbitrage, Commodities, Mean Reversion, Multi-Spread, Cointegration, Algorithmic Trading*

Contents

1 Introduction

Statistical arbitrage has become one of the pillars of modern quantitative trading, exploiting short-term mispricing between assets under the assumption of long-term equilibrium. First appearing on the stock markets in the late 1980s thanks to pair trading strategies developed by Nunzio Tartaglia's team at Morgan Stanley, StatArb has since expanded to various asset classes, including equities, ETFs, fixed income securities and commodities. The central idea is to identify stationary linear combinations of non-stationary time series in order to construct portfolios whose deviations from equilibrium can be systematically traded.

While equity pair trading is well documented (see Gatev, 2006; Vidyamurthy, 2004), its extension to the commodities market is less explored. Compared to pair trading, multi-spread strategies are much less well documented. Constructing mean-reverting spreads with more than two assets using Johansen cointegration is more complex, and empirical guidance is limited. Some recent studies have begun to explore these strategies, notably Jungblut (2024) on multivariate cointegration for statistical arbitrage and Lee (2023) on diversification frameworks for multiple pairs. This highlights an opportunity for further research, particularly in the commodities space, where multi-asset relationships may reveal exploitable mean-reverting spreads.

Energy products such as crude oil, refined oil and natural gas, as well as precious and industrial metals such as gold, silver and copper, are influenced by global macroeconomic cycles, supply chain shocks and geopolitical risks. Agricultural markets, particularly wheat, cocoa and corn, also exhibit strong correlations related to seasonality, demand and climate shocks. These interdependencies suggest that a cointegration-based framework could effectively capture the long-term equilibrium between multiple commodities, facilitating the design of profitable arbitrage portfolios.

2 Data & Statistical Tests

2.1 Data

The dataset consists of daily closing prices for a selection of commodities from 1990-04-03 to 2025-08-31, sourced from Refinitiv LSEG. It includes daily settlement prices and log returns for a panel of ten liquid commodity futures contracts. To construct the strategy, we adopt a two-step approach by dividing the dataset into an in-sample period (1990-04-03 to 2021-12-31) and an out-of-sample period (2022-01-01 to 2025-08-31). During the in-sample period, we estimate cointegration relationships, identify statistically stationary spreads, and calibrate strategy parameters such as entry/exit thresholds. Once the spreads and parameters are calibrated, the out-of-sample period is used for backtesting, allowing an unbiased evaluation of the strategy's performance under previously unseen market conditions, including periods of increased volatility. This methodology ensures that the strategy is statistically robust, avoids overfitting, and provides a realistic assessment of potential profitability and risk for the multi-spread strategy.

The panel covers three main sectors: energy (WTI crude oil, Brent crude oil, refined oil, natural gas), metals (gold, silver, copper) and agricultural commodities (wheat, cocoa, corn). All commodity futures are priced in dollars on US exchanges, primarily the Chicago Mercantile Exchange. This cross-sectional diversity provides a solid basis for identifying long-term equilibrium relationships between heterogeneous but economically

linked markets.

To stabilize variance and facilitate comparability across assets, we work with logarithmic returns:

$$x_{i,t} = \ln(P_{i,t}) \tag{1}$$

where $P_{i,t}$ denotes the daily closing price of commodity i on day t. The log returns are computed as:

$$r_{i,t} = x_{i,t} - x_{i,t-1} = \ln\left(\frac{P_{i,t}}{P_{i,t-1}}\right) \tag{2}$$

which are used for stationarity tests.

2.2 Statistical Tests

To ensure that the constructed spreads are suitable for mean-reversion strategies, we first test the stationarity of both prices and log returns using standard unit root tests. Specifically, we apply the *Augmented Dickey-Fuller (ADF)* test to check for the presence of a unit root, and the *Kwiatkowski-Phillips-Schmidt-Shin (KPSS)* test to verify stationarity around a deterministic trend. These complementary tests allow us to rigorously assess whether the time series exhibit the statistical properties required for reliable cointegration-based multi-spread strategy.

2.2.1 Augmented Dickey-Fuller Test

The ADF regression model is defined as:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \phi_i \Delta y_{t-i} + \varepsilon_t, \tag{3}$$

where $\Delta y_t = y_t - y_{t-1}$, α is a drift term, βt a deterministic trend, γ the coefficient of interest, ε_t is stationary noise and p lags are included to remove autocorrelation.

The hypotheses are:

$$\begin{aligned} H_0 &: \gamma = 0 \quad \text{(unit root, non-stationary)} \\ H_1 &: \gamma < 0 \quad \text{(stationary)} \end{aligned}$$

The test statistic is:

$$ADF = \frac{\hat{\gamma}}{\mathrm{SE}(\hat{\gamma})}, \tag{4}$$

where $\hat{\gamma}$ is the estimated coefficient of y_{t-1}. If the ADF statistic is less than the critical value (1%, 5%, 10%), we reject H_0 and conclude that the series is stationary.

Table 1: ADF Test Results

Series	ADF Stat	p-value	Critical 1%	Critical 5%	Critical 10%	Interpretation
WTI_Close	-2.140968	0.228363	-3.431184	-2.861909	-2.566966	Non-Stationary at 1%
Brent_Close	-1.955037	0.306629	-3.431184	-2.861909	-2.566966	Non-Stationary at 1%
Oil_Close	-1.540263	0.513508	-3.431182	-2.861907	-2.566966	Non-Stationary at 1%
Gas_Close	-3.095993	0.026851	-3.431184	-2.861909	-2.566966	Stationary at 5%
Gold_Close	0.095673	0.965764	-3.431183	-2.861908	-2.566966	Non-Stationary at 1%
Silver_Close	-1.310771	0.624230	-3.431185	-2.861909	-2.566966	Non-Stationary at 1%
Copper_Close	-1.003378	0.752030	-3.431185	-2.861909	-2.566966	Non-Stationary at 1%
Wheat_Close	-2.131189	0.232160	-3.431185	-2.861909	-2.566966	Non-Stationary at 1%
Cocoa_Close	-1.706842	0.427591	-3.431184	-2.861909	-2.566966	Non-Stationary at 1%
Corn_Close	-2.062571	0.259829	-3.431185	-2.861909	-2.566966	Non-Stationary at 1%
WTI_Log Return	-14.586852	0.000000	-3.431185	-2.861909	-2.566966	Stationary at 1%
Brent_Log Return	-19.769547	0.000000	-3.431183	-2.861908	-2.566966	Stationary at 1%
Oil_Log Return	-21.351440	0.000000	-3.431183	-2.861908	-2.566966	Stationary at 1%
Gas_Log Return	-42.190559	0.000000	-3.431182	-2.861908	-2.566966	Stationary at 1%
Gold_Log Return	-19.464729	0.000000	-3.431184	-2.861909	-2.566966	Stationary at 1%
Silver_Log Return	-20.252011	0.000000	-3.431184	-2.861909	-2.566966	Stationary at 1%
Copper_Log Return	-26.270179	0.000000	-3.431182	-2.861908	-2.566966	Stationary at 1%
Wheat_Log Return	-15.835060	0.000000	-3.431185	-2.861909	-2.566966	Stationary at 1%
Cocoa_Log Return	-24.611922	0.000000	-3.431183	-2.861908	-2.566966	Stationary at 1%
Corn_Log Return	-18.943781	0.000000	-3.431184	-2.861908	-2.566966	Stationary at 1%

The Augmented Dickey–Fuller (ADF) test results show a clear distinction between raw closing prices and log returns of commodity futures. For nearly all closing price series, the ADF statistics are above the critical thresholds at 1%, 5%, and 10%, with high p-values (often greater than 0.20). This indicates the presence of a unit root, meaning that prices are non-stationary over the full sample. The only exception is Natural Gas, which rejects the unit root at the 5% level. This relative stationarity may be attributed to the stronger seasonality and mean-reverting dynamics of natural gas markets, often driven by storage cycles and weather-related demand shocks.

In contrast, all log-return series show strongly negative ADF statistics and p-values equal to zero, decisively rejecting the unit root hypothesis at the 1% significance level. This confirms that log returns are stationary, consistent with financial theory, where returns are expected to oscillate around a constant mean with stable variance, while price levels follow non-stationary processes such as random walks.

2.2.2 Kwiatkowski-Phillips-Schmidt-Shin Test

To complement the ADF test, we apply the KPSS test for stationarity. The KPSS test reverses the usual null hypothesis: it assumes that the series $\{y_t\}_{t=1}^{T}$ is stationary around a deterministic trend and tests against the alternative that it contains a unit root (non-stationary).

The model is:

$$y_t = \delta t + r_t + \varepsilon_t \tag{5}$$

where δt is a deterministic trend, ε_t is stationary noise, and r_t is a random walk component:

$$r_t = r_{t-1} + u_t, \quad u_t \sim \text{i.i.d. } N(0, \sigma_u^2) \tag{6}$$

The hypotheses are:

$$\begin{aligned} H_0 &: \sigma_u^2 = 0 \quad \text{(stationary around deterministic trend)} \\ H_1 &: \sigma_u^2 > 0 \quad \text{(non-stationary due to random walk component)} \end{aligned} \tag{7}$$

The KPSS statistic is computed from the cumulative sum of residuals:

$$KPSS = \frac{1}{T^2 \hat{\sigma}^2} \sum_{t=1}^{T} S_t^2, \quad S_t = \sum_{i=1}^{t} \hat{\varepsilon}_i \tag{8}$$

where $\hat{\varepsilon}_i$ are residuals from regressing y_t on the deterministic trend, $\hat{\sigma}^2$ is a long-run variance estimate, and T is the sample size. If the KPSS statistic exceeds the critical value at a chosen significance level (1%, 5%, 10%), we reject H_0 and conclude that the series is non-stationary.

Table 2: KPSS Test Results

Series	KPSS Stat	p-value	Critical 1%	Critical 5%	Critical 10%	Interpretation	
WTI_Close	7.901490	0.01	0.347	0.463	0.574	0.739	Non-Stationary at 1%
Brent_Close	8.276989	0.01	0.347	0.463	0.574	0.739	Non-Stationary at 1%
Oil_Close	8.813093	0.01	0.347	0.463	0.574	0.739	Non-Stationary at 1%
Gas_Close	2.526046	0.01	0.347	0.463	0.574	0.739	Non-Stationary at 1%
Gold_Close	12.138888	0.01	0.347	0.463	0.574	0.739	Non-Stationary at 1%
Silver_Close	9.506068	0.01	0.347	0.463	0.574	0.739	Non-Stationary at 1%
Copper_Close	9.827823	0.01	0.347	0.463	0.574	0.739	Non-Stationary at 1%
Wheat_Close	5.262003	0.01	0.347	0.463	0.574	0.739	Non-Stationary at 1%
Cocoa_Close	10.216916	0.01	0.347	0.463	0.574	0.739	Non-Stationary at 1%
Corn_Close	5.879773	0.01	0.347	0.463	0.574	0.739	Non-Stationary at 1%
WTI_Log Return	0.045758	0.10	0.347	0.463	0.574	0.739	Stationary at 1%
Brent_Log Return	0.045844	0.10	0.347	0.463	0.574	0.739	Stationary at 1%
Oil_Log Return	0.051443	0.10	0.347	0.463	0.574	0.739	Stationary at 1%
Gas_Log Return	0.040860	0.10	0.347	0.463	0.574	0.739	Stationary at 1%
Gold_Log Return	0.225735	0.10	0.347	0.463	0.574	0.739	Stationary at 1%
Silver_Log Return	0.087928	0.10	0.347	0.463	0.574	0.739	Stationary at 1%
Copper_Log Return	0.136894	0.10	0.347	0.463	0.574	0.739	Stationary at 1%
Wheat_Log Return	0.041916	0.10	0.347	0.463	0.574	0.739	Stationary at 1%
Cocoa_Log Return	0.025592	0.10	0.347	0.463	0.574	0.739	Stationary at 1%
Corn_Log Return	0.055367	0.10	0.347	0.463	0.574	0.739	Stationary at 1%

The KPSS test results for the period covered by the test indicate that all commodity closing price series are non-stationary, as their KPSS statistics greatly exceed the critical values at the 1% significance level. In contrast, the logarithmic return series for all commodities exhibit stationarity, with KPSS statistics well below the critical thresholds and p-values equal to 0.10, implying that the null hypothesis of stationarity cannot be rejected. These results are consistent with those found in the previous test and appear appropriate for constructing mean reversion spreads and applying market-neutral statistical arbitrage strategy.

3 Strategy

In the next step of the strategy, the appropriate lag order will be determined based on information criteria, which is a necessary prerequisite for conducting the Johansen cointegration test. This test will identify the cointegration vectors between the selected assets and thereby construct the corresponding spreads. Once the spreads are obtained, they will be evaluated over the testing period to assess their robustness. Furthermore,

the strategy parameters, such as the z-score threshold used for trading signals, will be optimised in order to enhance performance.

3.1 Johansen Cointegration

Lag selection: It is necessary to determine the optimal lag length of the underlying Vector Autoregression (VAR) because an inappropriate choice may lead to residual autocorrelation (too few lags) or inefficiency and loss of degrees of freedom (too many lags).

Let Y_t denote a $(k \times 1)$ vector of commodities prices. A VAR(p) model is defined as:

$$Y_t = A_1 Y_{t-1} + A_2 Y_{t-2} + \cdots + A_p Y_{t-p} + \varepsilon_t \tag{9}$$

where A_i are $(k \times k)$ coefficient matrices and $\varepsilon_t \sim \mathcal{N}(0, \Sigma)$ is the error term.

The lag order p is selected by minimizing an information criterion. We rely primarily on the Bayesian Information Criterion (BIC),

$$\mathrm{BIC}(p) = -2 \ln \mathcal{L}_p + m_p \ln T \tag{10}$$

where $\mathcal{L}_p$ is the maximized likelihood of the VAR(p), m_p is the number of estimated parameters, and T is the sample size.

Table 3: VAR Lag Order Selection Results

Lag	AIC	BIC	FPE	HQIC
0	26.28	26.28	2.578e+11	26.28
1	-20.28	-20.18*	1.559e-09	-20.25
2	-20.34	-20.15	1.466e-09	-20.28*
3	-20.35	-20.07	1.454e-09	-20.25
4	-20.35	-19.98	1.456e-09	-20.22
5	-20.35	-19.89	1.457e-09	-20.19
6	-20.37	-19.83	1.425e-09	-20.18
7	-20.37	-19.74	1.426e-09	-20.15
8	-20.41	-19.70	1.362e-09	-20.17
9	-20.42	-19.62	1.350e-09	-20.15
10	-20.43*	-19.53	1.342e-09*	-20.12

Compared to AIC (Akaike Information Criterion), FPE (Final Prediction Error) or HQIC (Hannan–Quinn Information Criterion), the BIC imposes a stronger penalty for model complexity, making it consistent in large samples and less prone to overfitting. In our empirical analysis (Table 3), we compute AIC, BIC, HQIC, and FPE for lag orders $p = 0, \ldots, 10$. The BIC reaches its minimum at $\hat{p} = 1$, while HQIC suggests $\hat{p} = 2$ and AIC/FPE indicate $\hat{p} = 10$. Following BIC, we adopt $\hat{p} = 1$ as the baseline specification for the Johansen test.

Cointegration Analysis: The VAR can be rewritten in its Vector Error Correction Model (VECM) form:

$$\Delta Y_t = \Pi Y_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta Y_{t-i} + \varepsilon_t \tag{11}$$

where Π contains information about the long-run relationships. The rank of Π, denoted r, corresponds to the number of cointegration vectors.

The Johansen procedure tests sequential null hypotheses using two statistics:

- Trace Statistic:

$$\begin{aligned} H_0 &: \text{rank}(\Pi) \leq r \\ H_1 &: \text{rank}(\Pi) > r \end{aligned} \tag{12}$$

$$\text{Trace} = -T \sum_{i=r+1}^{k} \ln(1 - \hat{\lambda}_i) \tag{13}$$

- Maximum Eigenvalue Statistic:

$$\begin{aligned} H_0 &: \text{rank}(\Pi) = r \\ H_1 &: \text{rank}(\Pi) = r + 1 \end{aligned} \tag{14}$$

$$\text{Max-Eigen} = -T \ln(1 - \hat{\lambda}_{r+1}) \tag{15}$$

where $\hat{\lambda}_i$ are the estimated eigenvalues of Π and T is the sample size. Both statistics are compared to critical values at chosen significance levels (90%, 95%, 99%).

Results: For the commodity price system (1990-04-03 to 2021-12-31), the test results are:

- $r = 0$: Trace = 487.92 $>$ 99% critical value 253.25, Max-Eigen = 174.61 $>$ 99% critical value 71.25 $\Rightarrow$ Reject H_0, rank > 0
- $r = 1$: Trace = 313.31 $>$ 99% critical value 210.04, Max-Eigen = 88.40 $>$ 99% critical value 64.99 $\Rightarrow$ Reject H_0, rank > 1
- $r = 2$: Trace = 224.91 $>$ 99% critical value 171.09, Max-Eigen = 79.86 $>$ 99% critical value 58.66 $\Rightarrow$ Reject H_0, rank > 2
- $r = 3$: Trace = 145.05 $>$ 99% critical value 135.98, Max-Eigen = 42.75 $<$ 99% critical value 52.31 $\Rightarrow$ Fail to reject H_0, rank ≤ 3

Accordingly, the Johansen test indicates a cointegration rank of $r = 3$ among the ten commodity prices. This implies that there exist three linearly independent cointegration vectors describing long-run equilibrium relationships among the prices, which can be used to construct stationary spreads for further analysis.

Table 4: Cointegration Vectors

Commodity	Vector 1	Vector 2	Vector 3
WTI_Close	0.266390	-0.196724	-0.209332
Brent_Close	-0.577726	-0.018592	0.070552
Oil_Close	13.068810	8.351650	2.983389
Gas_Close	-0.498361	-0.069193	0.143231
Gold_Close	-0.003984	-0.001863	-0.001201
Silver_Close	0.436534	-0.155194	0.179298
Copper_Close	-0.940053	0.181234	0.224656
Wheat_Close	0.030334	-0.356970	1.063560
Cocoa_Close	-0.000372	0.000867	0.000009
Corn_Close	-0.450743	0.636123	-1.554753

3.2 Parameters & Calibration

3.2.1 Spreads

Normalized Cointegration Vectors: To interpret the cointegration vectors consistently, we normalize them by setting the first coefficient to 1. This allows us to express the stationary spread as a linear combination of the prices:

$$s_t = y_{1,t} + \beta_2 y_{2,t} + \cdots + \beta_k y_{k,t} \tag{16}$$

where $y_{i,t}$ are the individual commodity prices and β_i are the normalized coefficients.

The stationary linear combinations are represented by the normalized cointegration vectors. For each vector $i = 1, 2, 3$, the stationary spread is defined as:

$$\text{Spread}_i(t) = \sum_{j=1}^{10} \beta_{ij}\, P_j(t) \tag{17}$$

where $P_j(t)$ denotes the price of commodity j at time t, and β_{ij} is the coefficient in the i-th cointegration vector.

We normalize the cointegration vectors by WTI because it serves as a highly liquid benchmark in the energy market. Choosing WTI as the reference allows for an intuitive interpretation of the spreads: each coefficient of the other commodities represents their relative exposure per unit of WTI. While any other commodity could be used for normalization without affecting the statistical properties of the spreads, WTI provides a practical and standard reference for trading and portfolio construction.

Large coefficients indicate a stronger contribution of the corresponding commodity to the spread, and negative signs imply that the series moves in the opposite direction in the equilibrium relationship.

- **Vector 1:** $\text{Spread}_1 = 1 \cdot \text{WTI} - 2.168724 \cdot \text{Brent} + 49.058995 \cdot \text{Oil} - 1.870795 \cdot \text{Gas} + \ldots$
- **Vector 2:** $\text{Spread}_2 = 1 \cdot \text{WTI} + 0.094509 \cdot \text{Brent} - 42.453626 \cdot \text{Oil} + 0.351725 \cdot \text{Gas} + \ldots$
- **Vector 3:** $\text{Spread}_3 = 1 \cdot \text{WTI} - 0.337033 \cdot \text{Brent} - 14.251961 \cdot \text{Oil} - 0.684232 \cdot \text{Gas} + \ldots$

3.2.2 Z-scores

For trading purposes, each spread is standardized as a z-score:

$$\text{Zscore}_i(t) = \frac{\text{Spread}_i(t) - \bar{\text{Spread}}_i}{\sigma_{\text{Spread}_i}} \tag{18}$$

This standardization allows detection of extreme deviations from equilibrium, providing potential buy or sell signals when the z-score exceeds a threshold.

Table 5: Spreads and Z-scores

Date	**Spread 1**	**Spread 2**	**Spread 3**	**Z-score**	**Z-score**	**Z-score**
1990-04-03	-1.8456	3.8705	24.7660	0.2016	0.1485	0.1442
1990-04-04	-1.8857	3.8354	22.7377	0.1880	0.1425	0.0785
1990-04-05	-1.8861	3.8403	24.1260	0.1878	0.1433	0.1235
1990-04-06	-2.4569	3.6329	24.6753	-0.0056	0.1075	0.1413
1990-04-09	-2.4881	3.1179	24.6499	-0.0162	0.0186	0.1404
⋮	⋮	⋮	⋮	⋮	⋮	⋮
2021-12-27	-2.6313	6.8887	11.7675	-0.0647	0.6695	-0.2768
2021-12-28	-1.9908	6.4002	15.2277	0.1524	0.5852	-0.1647
2021-12-29	-1.9126	6.8214	18.0988	0.1789	0.6579	-0.0717
2021-12-30	-0.2664	5.3368	19.8203	0.7368	0.4016	-0.0160
2021-12-31	-1.3771	6.7748	14.3739	0.3604	0.6499	-0.1924

3.2.3 Parameters Optimization

Let $S_1, \ldots, S_n$ denote the n stationary spreads obtained from the normalized cointegration vectors. For each spread i at time t, we define its z-score $z_t^{(i)}$.

The trading positions are determined according to the following rules:

$$\text{Position}_t^{(i)} = \begin{cases} 1, & \text{if } z_t^{(i)} < -\text{entry}_z \text{ (long spread)}, \\ -1, & \text{if } z_t^{(i)} > \text{entry}_z \text{ (short spread)}, \\ 0, & \text{if } |z_t^{(i)}| < \text{exit}_z, \\ \text{previous position}, & \text{otherwise.} \end{cases} \tag{19}$$

To evaluate strategy performance, we compute the annualized Sharpe ratio:

$$\text{Sharpe Ratio} = \frac{E[\text{PnL}] \cdot 252}{\text{Std}[\text{PnL}] \cdot \sqrt{252}}. \tag{20}$$

The optimal entry and exit thresholds $(\text{entry}_z, \text{exit}_z)$ are determined via a grid search over a feasible range:

$$\text{entry}_z \in [0.5, 2.5], \quad \text{exit}_z \in [0.05, 0.5], \quad \text{with exit}_z < \text{entry}_z,$$

that selecting the combination that maximizes the Sharpe ratio.

Therefore, a position is entered when the Z-score exceeds the entry threshold, |Z-score| $>$ 0.8, and exited when the Z-score reverts below the exit threshold, |Z-score| $<$ 0.3.

3.3 Trade Example

For each commodity in the spread, the position size is proportional to the weight in the cointegration vector and the total notional capital allocated to the spread. We invest a total capital of $100,000, allocating $33,333 to each of the three spreads, to analyze the profitability of the strategy.

The profit and loss (PnL) for each commodity is calculated as:

$$\text{PnL}_i = \text{Capital Invested}_i \times \frac{\text{Price Exit}_i - \text{Price Entry}_i}{\text{Price Entry}_i} \tag{21}$$

Table 6: Trade 1 of Spread 1, Entry: 1990-08-20 | Exit: 1990-08-31

Commodity	Weight	Capital Invested ($)	Price Entry	Price Exit	PnL Before Cost ($)
WTI_Close	0.0164	-545.66	28.56	27.32	23.69
Brent_Close	-0.0355	1183.38	27.38	26.75	-27.23
Oil_Close	0.8031	-26769.42	0.83	0.77	1960.48
Gas_Close	-0.0306	1020.81	1.42	1.50	59.79
Gold_Close	-0.0002	8.16	410.10	382.30	-0.55
Silver_Close	0.0268	-894.17	5.13	4.76	64.50
Copper_Close	-0.0578	1925.55	1.29	1.26	-43.88
Wheat_Close	0.0019	-62.13	2.76	2.62	3.21
Cocoa_Close	-0.0000	0.76	1178.00	1319.00	0.09
Corn_Close	-0.0277	923.28	2.52	2.43	-32.94

In this trade, a short position in the spread means selling commodities with positive weights and buying those with negative weights according to the cointegration vector. The PnL for the spread is the sum of the individual commodity PnLs. The trade resulting in a net trade PnL of $1,973.82. Thus, the capital increased from $33,333.33 at entry to $34,940.07 at exit.

3.4 Test Period

The strategy focuses on cointegrated spreads constructed from normalized vectors, with trading signals based on a spread z-score entry threshold of 0.8 and an exit threshold of 0.3, a minimum holding period of 1 day, and a transaction cost of 0.1% per turnover unit. The initial capital is $100,000, equally allocated across the selected spreads and fully reinvested after each trade.

Table 7: Test Period : 1991-03-04 to 2021-12-31

Spread	Initial ($)	Final ($)	Total PnL ($)	Return (%)	N Trades	Win %
Spread 1	33,333	375,336	342,002	1026.01	182	61.54
Spread 2	33,333	907,027	873,694	2621.08	70	61.43
Spread 3	33,333	45,298	11,964	35.89	71	57.75
TOTAL	99,999	1,327,660	1,227,660	1227.66	323	60.68

Spread 3 shows a moderate return of 35.89% over approximately 30 years, with a success rate of 57.75% and a very low average PnL per trade, making this spread less attractive. Therefore, for the backtest, we will focus only on Spread 1 and Spread 2, applying the same trading rules and parameters. This setup concentrate capital on the most promising spreads ($50,000 per spread).

4 Backtest

4.1 Results

To rigorously evaluate the trading strategy, we conduct a backtest over the out-of-sample period from 2022-01-01 to 2025-08-31, using price data carefully sorted by date. Spread 1 and Spread 2 were selected for trading. The strategy parameters are already stated above.

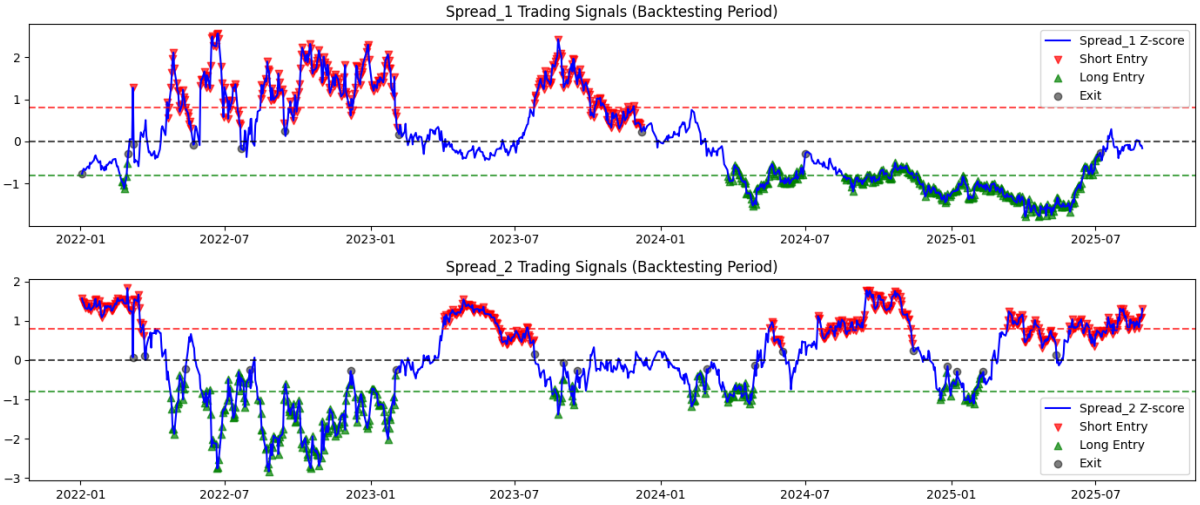

Figure 1: Spread 1 & 2

Table 8: Out-of-Sample Measures per Spread

Spread	Total PnL ($)	Number of Trades	Win Rate (%)
Spread 1	18,285.83	9	77.78
Spread 2	45,331.31	17	64.71
TOTAL	63,617.14	26	69.23

The backtest results indicate that the trading strategy achieved a positive total PnL of $63,617.14 over the out-of-sample period. Spread 2 contributed the largest portion of profits ($45,331.31), while Spread 1 provided a profitable complement ($18,285.83), achieving win rates of 64.71% and 77.78% respectively. The total number of executed trades across both spreads is 26. A table with all trades is available in the appendix.

4.2 Performance Metrics

The strategy's annualized return of 13.86% with an annualized volatility of 19.74% leads to a Sharpe ratio of 0.70, which reflects a moderate risk-adjusted performance for cointegration-based spreads trading. The maximum drawdown of -16.56% demonstrates that the strategy is able to limit downside risk, benefiting from diversification across the two selected spreads. Overall, these metrics show that the strategy generates consistent positive returns while maintaining controlled risk levels.

Table 9: Out-of-Sample Strategy Performance Metrics (2022-01-01 to 2025-08-31)

Metric	Value
Initial Capital ($)	100,000.00
Final Capital ($)	163,617.14
Total Return (%)	63.62
Annualized Return (%)	13.83
Annualized Volatility (%)	19.74
Sharpe Ratio	0.70
Maximum Drawdown (%)	-16.56

4.3 Risks & Improvement

To enhance the robustness of the multi-spread statistical arbitrage strategy, several layers of risk control can be introduced.

First, dynamic position sizing ensures that capital allocation across spreads is proportional to their individual risk levels. By scaling positions based on volatility or risk contribution, the strategy avoids concentration in highly volatile spreads and achieves a more balanced risk-adjusted profile. This approach can be further refined by applying stop-loss rules or limiting exposure when a spread deviates excessively from its equilibrium, thereby preventing outsized losses.

Second, at the portfolio level, risk can be managed by imposing drawdown constraints and monitoring tail risk measures such as Value-at-Risk or Expected Shortfall. In addition, reducing exposure when correlations among spreads increase helps to mitigate systemic risks and avoid redundant positions.

Finally, the statistical arbitrage framework can also be extended to a stochastic intraday setting, where spreads are monitored and traded at high frequency (down to milliseconds). By capturing arbitrage opportunities more often in shorter horizons, the strategy reduces exposure time to market risk, improves turnover efficiency, and allows for more frequent mean-reversion trades with lower downside volatility. Together, these measures strengthen both the resilience and profitability of the strategy.

5 Conclusion

The out-of-sample backtest of our strategy over 965 trading days demonstrates strong performance across multiple commodities. While individual commodities such as WTI, Brent, Gas, and Corn experienced negative cumulative returns ranging from -37.74% to -3.00%, our strategy achieved a cumulative return of 63.62%, substantially outperforming the BCI index (Bloomberg Commodity Index), which recorded a modest 3.34% return over the same period.

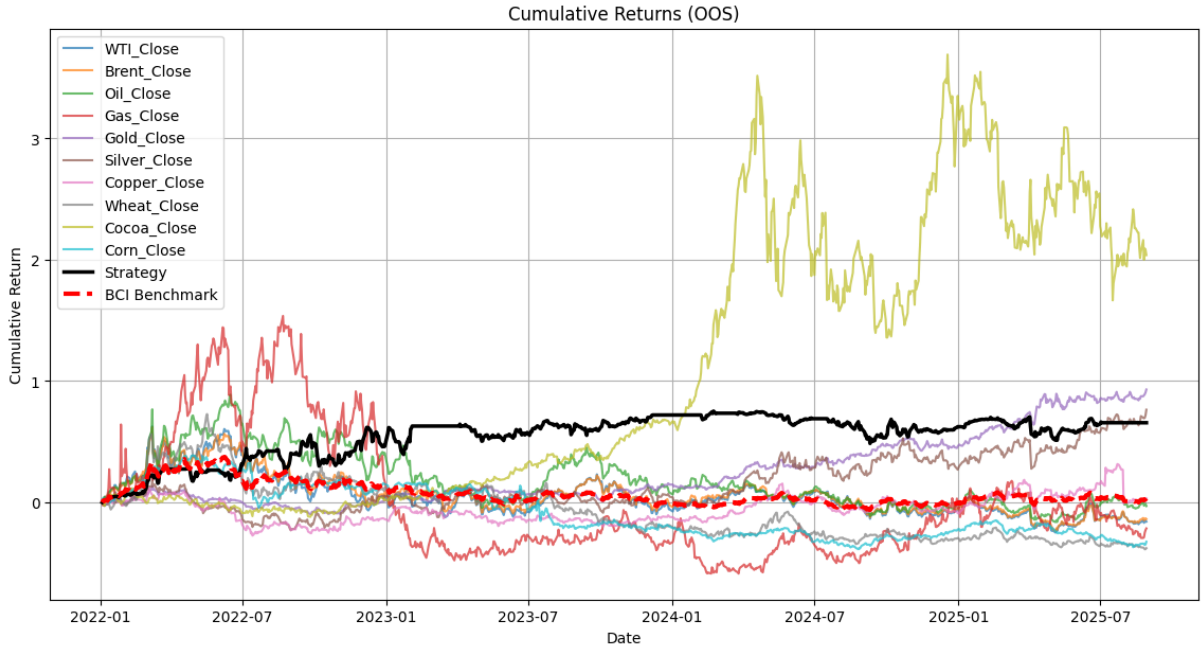

Figure 2: Cumulative Returns

To quantify the performance relative to the benchmark, we can compute the strategy's alpha and beta. Using the standard Capital Asset Pricing Model (CAPM) framework, the beta with respect to the BCI index is approximately

$$\beta = \frac{\text{Cov}(\text{Strategy Returns}, \text{BCI Returns})}{\text{Var}(\text{BCI Returns})} \approx 0.12$$

indicating low correlation with the benchmark. The excess return (alpha) relative to BCI is:

$$\alpha = R_{\text{Strategy}} - \beta \cdot R_{\text{BCI}} \approx 63.62\% - 0.12 \cdot 3.34\% \approx 63.22\%$$

confirming that the strategy generates significant independent gains beyond the benchmark. Overall, the results suggest that the cointegration-based spread strategy provides robust risk-adjusted returns and captures opportunities not reflected in broad commodity indices.

6 Bibliography

1. Dickey, D. A., & Fuller, W. A. (1979). *Distribution of the estimators for autoregressive time series with a unit root.* Journal of the American Statistical Association, 427–431.

2. Kwiatkowski, D., Phillips, P. C. B., Schmidt, P., & Shin, Y. (1992). *Testing the null hypothesis of stationarity against the alternative of a unit root.* Journal of Econometrics, 159–178.

3. Johansen, S. (1988). *Statistical analysis of cointegration vectors.* Journal of Economic Dynamics and Control, 231–254.

4. Gatev, E., Goetzmann, W. N., & Rouwenhorst, K. G. (2006). *Pairs trading: Performance of a relative-value arbitrage rule.* Review of Financial Studies, 797–827.

5. Vidyamurthy, G. (2004). *Pairs Trading: Quantitative Methods and Analysis.* John Wiley & Sons.

6. Jungblut, M. (2024). *Multivariate cointegration for statistical arbitrage.*

7. Lee, S. (2023). *Diversification frameworks for multiple pairs in commodities markets.*

8. Mohandas, G. (2023). *Cointegration-Based Pairs Trading Strategy in Commodity Markets: A Novel Approach to Statistical Arbitrage.*

9. Stübinger, J., & Schneider, L. (2019). *Statistical Arbitrage with Mean-Reverting Overnight Price Gaps on High-Frequency Data of the S&P 500.* Journal of Risk and Financial Management, 51.

10. Verbeek, J., *Statistical Arbitrage in Commodity Markets: A Cointegration Approach.*

11. Ungever, C., *Pairs Trading to the Commodities Futures Market Using Cointegration Method.*

7 Appendix

7.1 Data

```python
import pandas as pd
import numpy as np

commodity_files = [ "WTI.xlsx", "Brent.xlsx", "Oil.xlsx", "Gas.xlsx", "Gold.xlsx",
    "Silver.xlsx", "Copper.xlsx", "Wheat.xlsx", "Cocoa.xlsx", "Corn.xlsx" ]

commodity_data = {}
for file in commodity_files:
    try:
        df = pd.read_excel(file)
        df['Exchange Date'] = pd.to_datetime(df['Exchange Date'])
        df = df.set_index('Exchange Date')
        df_close = df[['Close']].copy()
        commodity_name = file.split('.')[0]
        df_close = df_close.rename(columns={'Close': f'{commodity_name}_Close'})
        commodity_data[commodity_name] = df_close
    except FileNotFoundError:
        print(f"File not found: {file}")
    except Exception as e:
        print(f"Error processing file {file}: {e}")

merged_df = pd.concat(commodity_data.values(), axis=1, join='inner')

for commodity_name in commodity_data.keys():
    close_col = f'{commodity_name}_Close'
    log_return_col = f'{commodity_name}_Log Return'
    merged_df[log_return_col] = np.log(merged_df[close_col] / merged_df[close_col].shift(1))

merged_df = merged_df.reset_index()
merged_df = merged_df.dropna(subset=[col for col in merged_df.columns if 'Log Return' in col])

display(merged_df)
```

Listing 1: Data - Close Price & Log Return

7.2 ADF Test

```python
from statsmodels.tsa.stattools import adfuller

start_date = '1990-04-03'
end_date = '2021-12-31'
filtered_df = merged_df.loc[start_date:end_date].copy()

def run_adf_test(series):
    """Runs ADF test on a time series and returns critical values at 1%, 5%, 10%."""
    results = {}
    series = series.dropna()
```

```
    adf_stat, adf_pvalue, _, _, crit_values, _ = adfuller(series,
        autolag='AIC')
    results["ADF Stat"] = adf_stat
    results["ADF p-value"] = adf_pvalue
    results["ADF Critical 1%"] = crit_values['1%']
    results["ADF Critical 5%"] = crit_values['5%']
    results["ADF Critical 10%"] = crit_values['10%']

    if adf_stat < crit_values['1%']:
        results["Interpretation"] = "Stationary at 1%"
    elif adf_stat < crit_values['5%']:
        results["Interpretation"] = "Stationary at 5%"
    elif adf_stat < crit_values['10%']:
        results["Interpretation"] = "Stationary at 10%"
    else:
        results["Interpretation"] = "Non-Stationary"
    return results

test_results = []
for col in filtered_df.columns:
    if "_Close" in col or "_Log Return" in col:
        res = run_adf_test(filtered_df[col])
        res["Series"] = col
        test_results.append(res)

results_df = pd.DataFrame(test_results)
results_df = results_df[["Series", "ADF Stat", "ADF p-value",
                         "ADF Critical 1%", "ADF Critical 5%", "ADF
                            Critical 10%",
                         "Interpretation"]]

print("\nAugmented Dickey-Fuller (ADF) Test Results with Confidence
    Levels Test Period:")
with pd.option_context('display.float_format', '{:.6f}'.format):
    display(results_df)
```

Listing 2: ADF Test

7.3 KPSS Test

```

from statsmodels.tsa.stattools import kpss

start_date = '1990-04-03'
end_date = '2021-12-31'
merged_df = merged_df.sort_index()
filtered_df = merged_df.loc[start_date:end_date].copy()

def run_kpss_test(series):
    """Runs KPSS test on a time series and returns critical values at
        10%, 5%, 2.5%, 1%."""
    results = {}
    series = series.dropna()
    kpss_stat, kpss_pvalue, lags, crit_values = kpss(series, regression=
        'c', nlags='auto')
    results["KPSS Stat"] = kpss_stat
    results["KPSS p-value"] = kpss_pvalue
```

```
    results["KPSS Critical 10%"] = crit_values['10%']
    results["KPSS Critical 5%"] = crit_values['5%']
    results["KPSS Critical 1%"] = crit_values['1%']
    if kpss_stat > crit_values['1%']:
        results["Interpretation"] = "Non-Stationary at 1%"
    elif kpss_stat > crit_values['5%']:
        results["Interpretation"] = "Non-Stationary at 5%"
    elif kpss_stat > crit_values['10%']:
        results["Interpretation"] = "Non-Stationary at 10%"
    else:
        results["Interpretation"] = "Stationary"
    return results

kpss_test_results = []

for col in filtered_df.columns:
    if "_Close" in col or "_Log Return" in col:
        res = run_kpss_test(filtered_df[col])
        res["Series"] = col
        kpss_test_results.append(res)

kpss_results_df = pd.DataFrame(kpss_test_results)
kpss_results_df = kpss_results_df[[
    "Series", "KPSS Stat", "KPSS p-value",
    "KPSS Critical 10%", "KPSS Critical 5%", "KPSS Critical 1%",
    "Interpretation" ]]

print("\nKPSS Test Results for 1990-04-03 to 2022-12-31:")
display(kpss_results_df)
```

Listing 3: KPSS Test

7.4 Johansen Cointegration

```
from statsmodels.tsa.api import VAR

start_date = '1990-04-03'
end_date = '2021-12-31'

merged_df = merged_df.sort_index()
filtered_df = merged_df.loc[start_date:end_date].copy()

price_cols = [col for col in filtered_df.columns if "_Close" in col]
price_data = filtered_df[price_cols].copy()
price_data = price_data.dropna()
max_lag = 10
model = VAR(price_data)
lag_selection_results = model.select_order(maxlags=max_lag, )

print(f"Recommended lag order based on BIC: {lag_selection_results.bic}"
    )
print("\nLag Selection Results (BIC):")
display(lag_selection_results.summary())
```

Listing 4: Lag Selection

```python
from statsmodels.tsa.vector_ar.vecm import coint_johansen

start_date = '1990-04-03'
end_date = '2021-12-31'

merged_df = merged_df.sort_index()
filtered_df = merged_df.loc[start_date:end_date].copy()
price_cols = [col for col in filtered_df.columns if "_Close" in col]
price_data = filtered_df[price_cols].copy()
price_data = price_data.dropna()
lag_order = 1
n_series = price_data.shape[1]
johansen_test_result = coint_johansen(price_data, det_order=0, k_ar_diff=lag_order)

print(f"Johansen Cointegration Test Results for Commodity Prices ({start_date} to {end_date}):")
print("-" * 50)

for r in range(n_series):
    trace_stat = johansen_test_result.lr1[r]
    trace_crit_90 = johansen_test_result.cvt[r, 0]
    trace_crit_95 = johansen_test_result.cvt[r, 1]
    trace_crit_99 = johansen_test_result.cvt[r, 2]
    max_eigen_stat = johansen_test_result.lr2[r]
    max_eigen_crit_90 = johansen_test_result.cvm[r, 0]
    max_eigen_crit_95 = johansen_test_result.cvm[r, 1]
    max_eigen_crit_99 = johansen_test_result.cvm[r, 2]
    print(f"Testing for rank r <= {r}:")
    print(f"  Trace Statistic: {trace_stat:.4f}")
    print(f"  Trace Critical Values (90%, 95%, 99%): {trace_crit_90:.4f}, {trace_crit_95:.4f}, {trace_crit_99:.4f}")
    if trace_stat > trace_crit_99:
        trace_interp = f"Reject H0 at 99% significance (rank > {r})"
    elif trace_stat > trace_crit_95:
        trace_interp = f"Reject H0 at 95% significance (rank > {r})"
    elif trace_stat > trace_crit_90:
        trace_interp = f"Reject H0 at 90% significance (rank > {r})"
    else:
        trace_interp = f"Fail to reject H0 (rank <= {r})"
    print(f"  Trace Test Interpretation (H0: rank <= {r}): {trace_interp}")
    print(f"  Max-Eigen Statistic: {max_eigen_stat:.4f}")
    print(f"  Max-Eigen Critical Values (90%, 95%, 99%): {max_eigen_crit_90:.4f}, {max_eigen_crit_95:.4f}, {max_eigen_crit_99:.4f}")
    if max_eigen_stat > max_eigen_crit_99:
        max_eigen_interp = f"Reject H0 at 99% significance (rank = {r + 1})"
    elif max_eigen_stat > max_eigen_crit_95:
        max_eigen_interp = f"Reject H0 at 95% significance (rank = {r + 1})"
    elif max_eigen_stat > max_eigen_crit_90:
        max_eigen_interp = f"Reject H0 at 90% significance (rank = {r + 1})"
    else:
        max_eigen_interp = f"Fail to reject H0 (rank <= {r})"

```

```
    print(f"  Max-Eigenvalue Test Interpretation (H0: rank <= {r}): {
        max_eigen_interp}")
    print("-" * 50)
```

Listing 5: Johansen Cointegration

```
r = 3
raw_vectors = johansen_test_result.evec[:, :r]
commodities = price_data.columns.tolist()
for i in range(r):
    print(f"\nCointegration Vector {i+1} (raw, before normalization):")
    for j, commodity in enumerate(commodities):
        print(f"  {commodity}: {raw_vectors[j, i]:.6f}")
```

Listing 6: Cointegration Vectors

7.5 Parameters & Calibration

```
start_date = '1990-04-03'
end_date = '2021-12-31'

merged_df = merged_df.sort_index()
filtered_df = merged_df.loc[start_date:end_date].copy()
price_cols = [col for col in filtered_df.columns if "_Close" in col]
price_data = filtered_df[price_cols].copy()
price_data = price_data.dropna()

lag_order = 1
johansen_test_result = coint_johansen(price_data, det_order=0, k_ar_diff
    =lag_order)

trace_rank = sum(johansen_test_result.lr1 > johansen_test_result.cvt[:,
    1])
maxeig_rank = sum(johansen_test_result.lr2 > johansen_test_result.cvm[:,
     1])

r = min(trace_rank, maxeig_rank)
print(f"Cointegration Rank (conservateur) : {r}\n")

coint_vectors = johansen_test_result.evec[:, :r]
normalized_vectors = coint_vectors / coint_vectors[0, :]
commodities = price_data.columns.tolist()
for i in range(normalized_vectors.shape[1]):
    print(f"Normalized Cointegration Vector {i+1}:")
    for j, commodity in enumerate(commodities):
        print(f"  {commodity}: {normalized_vectors[j, i]:.6f}")
    print("-"*40)
spreads = pd.DataFrame(
    np.dot(price_data.values, normalized_vectors),
    index=price_data.index,
    columns=[f"Spread_{i+1}" for i in range(normalized_vectors.shape[1])
        ])
spread_zscores = (spreads - spreads.mean()) / spreads.std()
print("\nSpreads et Z-scores :")
spread_zscores_renamed = spread_zscores.add_suffix("_zscore")
display(pd.concat([spreads, spread_zscores_renamed], axis=1))
```

Listing 7: Normalize Vectors - Spreads & Z-scores

```
import numpy as np

entry_range = np.arange(0.5, 2.5, 0.1)
exit_range  = np.arange(0.05, 0.5, 0.05)

best_sharpe = -np.inf
best_params = None

for entry_z in entry_range:
    for exit_z in exit_range:
        sharpe = backtest_zscore_strategy(spread_zscores, unit_vecs,
           price_data, entry_z, exit_z)
        if sharpe > best_sharpe:
            best_sharpe = sharpe
            best_params = (entry_z, exit_z)

print("Optimal entry_z, exit_z:", best_params)
```

Listing 8: Parameters Optimization

```
entry_z = 0.799999999999999
exit_z = 0.3
min_holding = 1
transaction_cost = 0.001
max_leverage = 2.0
notional_per_spread = 1.0
n_dates = spread_zscores.shape[0]
dates = spread_zscores.index
n_spreads = spread_zscores.shape[1]
assets = price_data.columns.tolist()
n_assets = len(assets)
vecs = normalized_vectors.copy()
tiny_thresh = 1e-3
vecs[np.abs(vecs) < tiny_thresh] = 0.0
abs_sum = np.sum(np.abs(vecs), axis=0)
abs_sum[abs_sum == 0] = 1.0
unit_vecs = vecs / abs_sum

positions = pd.DataFrame(index=dates, columns=assets, dtype=float).
   fillna(0.0)
spread_positions = pd.DataFrame(0, index=dates, columns=[f"Spread_{i+1}"
    for i in range(n_spreads)])  # +1, -1, 0
last_entry_day = {s: -9999 for s in range(n_spreads)}

for t, date in enumerate(dates):
    z = spread_zscores.iloc[t].values
    if t == 0:
        prev_sp_pos = np.zeros(n_spreads, dtype=int)
    else:
        prev_sp_pos = spread_positions.iloc[t-1].values.astype(int)
    sp_pos = prev_sp_pos.copy()
    for s in range(n_spreads):
        zt = z[s]
        prev = prev_sp_pos[s]
        if prev == 0:
            if zt > entry_z:
                sp_pos[s] = -1
                last_entry_day[s] = t
            elif zt < -entry_z:
```

```
                sp_pos[s] = 1
                last_entry_day[s] = t
        else:
            if (abs(zt) < exit_z) and (t - last_entry_day[s] >=
               min_holding):
                sp_pos[s] = 0
            else:
                sp_pos[s] = prev

    spread_positions.iloc[t] = sp_pos
    pos_today = np.dot(unit_vecs, sp_pos.astype(float) *
       notional_per_spread)
    positions.iloc[t] = pos_today

gross = positions.abs().sum(axis=1)
scale = np.minimum(1.0, max_leverage / (gross.replace(0, np.nan)))
scale = scale.fillna(1.0)
positions = positions.mul(scale, axis=0)

returns = price_data.pct_change().fillna(0.0)
pos_shifted = positions.shift(1).fillna(0.0)
turnover = (positions - pos_shifted).abs().sum(axis=1)
daily_pnl = (pos_shifted * returns).sum(axis=1)
daily_cost = turnover * transaction_cost
net_daily_pnl = daily_pnl - daily_cost
cumulative_pnl = net_daily_pnl.cumsum()
sharpe = net_daily_pnl.mean() / net_daily_pnl.std() * np.sqrt(252) if
   net_daily_pnl.std() > 0 else np.nan
annual_return = net_daily_pnl.mean() * 252
annual_vol = net_daily_pnl.std() * np.sqrt(252)
max_dd = (cumulative_pnl.cummax() - cumulative_pnl).max()
display(spread_positions)
dates = spread_positions.index
n_spreads = spread_positions.shape[1]
assets = price_data.columns.tolist()

spread_portfolios = {}
for s in range(n_spreads):
    sig = spread_positions.iloc[:, s]
    vec = unit_vecs[:, s]
    pos_s = np.outer(sig.values, vec)
    df_s = pd.DataFrame(
        pos_s,
        index=dates,
        columns=assets   )
    spread_portfolios[f"Spread_{s+1}"] = df_s
for name, df in spread_portfolios.items():
    print(f"\n{name} - Positions par actif")
    display(df)
```

Listing 9: Spreads Signals & Weights Commodities

7.6 In-Sample Test Period

```
trade_log = []
initial_capital = 100000
capital_per_spread = initial_capital / n_spreads
```

```
4
5  for s in range(n_spreads):
6      spread_name = f"Spread_{s+1}"
7      pos_series = spread_positions.iloc[:, s]
8      current_trade = None
9
10     for t, date in enumerate(dates):
11         sig = pos_series.iloc[t]
12
13         if current_trade is None and sig != 0:
14             current_trade = {
15                 "Spread": spread_name,
16                 "Entry_Date": date,
17                 "Direction": sig,
18                 "EntryIdx": t
19             }
20
21         elif current_trade is not None and sig == 0:
22             current_trade["Exit_Date"] = date
23             current_trade["ExitIdx"] = t
24
25             pos_slice = spread_portfolios[spread_name].iloc[
                   current_trade["EntryIdx"]:t]
26             ret_slice = returns.iloc[current_trade["EntryIdx"]+1:t+1]
27             pnl_by_day = (pos_slice.shift(1).fillna(0) * ret_slice).sum(
                   axis=1)
28             gross_pnl = pnl_by_day.sum()
29             turnover_trade = (pos_slice - pos_slice.shift(1).fillna(0)).
                   abs().sum(axis=1).sum()
30             cost_trade = turnover_trade * transaction_cost
31             current_trade["PnL"] = gross_pnl - cost_trade
32             current_trade["HoldingDays"] = t - current_trade["EntryIdx"]
33             trade_log.append(current_trade)
34             current_trade = None
35
36 trade_log_df = pd.DataFrame(trade_log)
37 trade_log_df["Entry_Capital"] = np.nan
38 trade_log_df["Exit_Capital"] = np.nan
39
40 for s_name in trade_log_df["Spread"].unique():
41     cap = capital_per_spread
42     for i in trade_log_df[trade_log_df["Spread"] == s_name].index:
43         trade_log_df.at[i, "Entry_Capital"] = cap
44         cap = cap * (1 + trade_log_df.at[i, "PnL"])
45         trade_log_df.at[i, "Exit_Capital"] = cap
46
47 n_trades_display = 3
48 selected_spreads = ["Spread_1", "Spread_2"]
49
50 for s_name in selected_spreads:
51     trades = trade_log_df[trade_log_df["Spread"] == s_name].reset_index(
           drop=True).head(n_trades_display)
52     print(f"\n{s_name} - Premiers {n_trades_display} trades\n")
53
54     spread_idx = int(s_name.split("_")[1]) - 1
55     weights_trade = unit_vecs[:, spread_idx]
56     weights_trade_norm = weights_trade / np.sum(np.abs(weights_trade))
57
```

```
    for idx, trade in trades.iterrows():
        direction = int(trade["Direction"])  # +1 long, -1 short
        entry_cap = trade["Entry_Capital"]
        capital_by_asset = entry_cap * direction * weights_trade_norm
        price_entry = price_data.loc[trade["Entry_Date"]].values
        price_exit  = price_data.loc[trade["Exit_Date"]].values
        pnl_by_asset = capital_by_asset * (price_exit - price_entry) /
            price_entry
        cost_trade = np.sum(np.abs(capital_by_asset)) * transaction_cost
        trade_pnl = np.sum(pnl_by_asset) - cost_trade
        df_trade = pd.DataFrame({
            "Commodity": assets,
            "Weight": np.round(weights_trade_norm, 4),
            "Capital_Invested($)": np.round(capital_by_asset, 2),
            "Price_Entry": np.round(price_entry, 2),
            "Price_Exit": np.round(price_exit, 2),
            "PnL_Before_Cost($)": np.round(pnl_by_asset, 2)      })

        print(f"\nTrade {idx+1}: {'Long' if direction==1 else 'Short'} "
              f"| Entry: {trade['Entry_Date']} | Exit: {trade['Exit_Date
                  ']}")
        display(df_trade)
        print(f"Total PnL before cost: {np.sum(pnl_by_asset):.2f}$")
        print(f"Transaction cost: {cost_trade:.2f}$")
        print(f"Trade PnL (net of cost): {trade_pnl:.2f}$")
        print(f"Entry capital: {entry_cap:.2f}$ | Exit capital: {trade['
            Exit_Capital']:.2f}$")
        print("-"*80)
```

Listing 10: All Trades Test Period

```
initial_capital = 100000
capital_per_spread = initial_capital / n_spreads  # capital initial par
    spread
trade_log = []

for s in range(n_spreads):
    spread_name = f"Spread_{s+1}"
    pos = spread_positions.iloc[:, s]
    current_trade = None
    cap = capital_per_spread
    for t, date in enumerate(dates):
        sig = pos.iloc[t]
        if current_trade is None and sig != 0:
            current_trade = {
                "Spread": spread_name,
                "EntryDate": date,
                "Direction": sig,
                "EntryIdx": t,
                "Entry_Capital": cap    }
        elif current_trade is not None and sig == 0:
            current_trade["ExitDate"] = date
            current_trade["ExitIdx"] = t
            pos_slice = spread_portfolios[spread_name].iloc[
                current_trade["EntryIdx"]:t]
            ret_slice = returns.iloc[current_trade["EntryIdx"]+1:t+1]
            pnl_by_day = (pos_slice.shift(1).fillna(0) * ret_slice).sum(
                axis=1)
            gross_pnl = pnl_by_day.sum() * current_trade["Entry_Capital"
```

```
                ]
            turnover_trade = (pos_slice - pos_slice.shift(1).fillna(0)).
                abs().sum(axis=1).sum() * current_trade["Entry_Capital"]
            cost_trade = turnover_trade * transaction_cost
            trade_pnl = gross_pnl - cost_trade
            current_trade["PnL"] = trade_pnl
            cap = current_trade["Entry_Capital"] + trade_pnl
            current_trade["Exit_Capital"] = cap
            current_trade["HoldingDays"] = t - current_trade["EntryIdx"]
            trade_log.append(current_trade)
            current_trade = None

trade_log_df = pd.DataFrame(trade_log)
summary_list = []

for s in range(n_spreads):
    spread_name = f"Spread_{s+1}"
    trades = trade_log_df[trade_log_df["Spread"] == spread_name].
        reset_index(drop=True)
    if not trades.empty:
        initial_cap = trades["Entry_Capital"].iloc[0]
        final_cap = trades["Exit_Capital"].iloc[-1]
        pnl_total = final_cap - initial_cap
        pnl_list = trades["PnL"]
        n_trades = len(trades)
        win_rate = (pnl_list > 0).sum() / n_trades * 100
        avg_pnl = pnl_list.mean()
        vol_pnl = pnl_list.std()
        sharpe = avg_pnl / vol_pnl * np.sqrt(252) if vol_pnl != 0 else
            np.nan
        capital_curve = pd.concat([pd.Series([initial_cap]), trades["
            Exit_Capital"]], ignore_index=True)
        running_max = capital_curve.cummax()
        drawdowns = (capital_curve - running_max) / running_max
        max_dd = drawdowns.min() * 100
    else:
        initial_cap = final_cap = capital_per_spread
        pnl_total = 0.0
        n_trades = 0
        win_rate = avg_pnl = vol_pnl = sharpe = max_dd = 0.0

    summary_list.append({
        "Spread": spread_name,
        "Initial_Capital($)": round(initial_cap,2),
        "Final_Capital($)": round(final_cap,2),
        "Total_PnL($)": round(pnl_total,2),
        "Return(%)": round(pnl_total/initial_cap*100,2),
        "N_Trades": n_trades,
        "WinRate(%)": round(win_rate,2),
        "Avg_PnL($)": round(avg_pnl,2),
        "Vol_PnL($)": round(vol_pnl,2),
        "Sharpe": round(sharpe,2),
        "MaxDD(%)": round(max_dd,2)   })

summary_df = pd.DataFrame(summary_list).set_index("Spread")
total_initial = summary_df["Initial_Capital($)"].sum()
total_final = summary_df["Final_Capital($)"].sum()
total_pnl = total_final - total_initial
```

```
total_trades = summary_df["N_Trades"].sum()
summary_df.loc["TOTAL"] = {
    "Initial_Capital($)": round(total_initial,2),
    "Final_Capital($)": round(total_final,2),
    "Total_PnL($)": round(total_pnl,2),
    "Return(%)": round(total_pnl/total_initial*100,2),
    "N_Trades": total_trades,
    "WinRate(%)": round((trade_log_df["PnL"] > 0).mean()*100,2),
    "Avg_PnL($)": round(trade_log_df["PnL"].mean(),2),
    "Vol_PnL($)": round(trade_log_df["PnL"].std(),2),
    "Sharpe": round((trade_log_df["PnL"].mean()/trade_log_df["PnL"].std()*np.sqrt(252))
                    if trade_log_df["PnL"].std()!=0 else np.nan,2),
    "MaxDD(%)": round(((trade_log_df["Exit_Capital"].cummax() - trade_log_df["Exit_Capital"])
                       / trade_log_df["Exit_Capital"].cummax()).max()*100,2)}

print("===== Tableau Spreads =====")
display(summary_df)
```

Listing 11: Spreads Performance Test Period

7.7 Out-of-Sample Backtest Period

```
import matplotlib.pyplot as plt

entry_z = 0.7999999999
exit_z = 0.3

selected_spreads_to_plot = ["Spread_1", "Spread_2"]
zscores_to_plot = spread_zscores[selected_spreads_to_plot]
fig, axes = plt.subplots(len(selected_spreads_to_plot), 1, figsize=(14, 3 * len(selected_spreads_to_plot)))

if len(selected_spreads_to_plot) == 1:
    axes = [axes]

for i, spread_name in enumerate(selected_spreads_to_plot):
    z = zscores_to_plot[spread_name]
    axes[i].plot(z.index, z, label=f"{spread_name} Z-score", color='blue')
    axes[i].axhline(0, color='black', linestyle='--', alpha=0.7)
    axes[i].axhline(entry_z, color='red', linestyle='--', alpha=0.7)
    axes[i].axhline(-entry_z, color='green', linestyle='--', alpha=0.7)
    spread_pos = spread_positions[spread_name]
    axes[i].scatter(z.index[spread_pos < 0], z[spread_pos < 0], color='red', marker='v', label="Short Entry", alpha=0.7)
    axes[i].scatter(z.index[spread_pos > 0], z[spread_pos > 0], color='green', marker='^', label="Long Entry", alpha=0.7)
    exit_dates = spread_pos[(spread_pos.shift(1) != 0) & (spread_pos == 0)].index
    axes[i].scatter(exit_dates, z.loc[exit_dates], color='black', marker='o', label="Exit", alpha=0.5)
    axes[i].set_title(f"{spread_name} Trading Signals (Backtesting Period)")
    axes[i].legend()
```

```

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

Listing 12: Graphs Spreads

```
merged_df = pd.concat(commodity_data.values(), axis=1, join='inner')
price_cols = [col for col in merged_df.columns if "_Close" in col]
price_data = merged_df[price_cols].copy()
price_data = price_data.sort_index()
start_date_oos = '2022-01-01'
end_date_oos   = '2025-08-31'
price_data_oos = price_data.loc[start_date_oos:end_date_oos].copy()
dates_oos = price_data_oos.index

selected_spreads_idx = [0, 1]
entry_z = 0.8
exit_z  = 0.3
min_holding = 1
transaction_cost = 0.001
initial_capital = 100000
capital_per_spread = initial_capital / len(selected_spreads_idx)

spreads_oos = pd.DataFrame(
    np.dot(price_data_oos.values, unit_vecs[:, selected_spreads_idx]),
    index=dates_oos,
    columns=[f"Spread_{i+1}" for i in selected_spreads_idx])
spread_zscores_oos = (spreads_oos - spreads_oos.mean()) / spreads_oos.
   std()

spread_positions_oos = pd.DataFrame(0, index=dates_oos, columns=
   spreads_oos.columns)
last_entry_day = {s: -9999 for s in spreads_oos.columns}

for t, date in enumerate(dates_oos):
    z = spread_zscores_oos.iloc[t]
    if t == 0:
        prev = pd.Series(0, index=spreads_oos.columns)
    else:
        prev = spread_positions_oos.iloc[t-1]

    sp_pos = prev.copy()
    for s in spreads_oos.columns:
        zt = z[s]
        if prev[s] == 0:
            if zt > entry_z:
                sp_pos[s] = -1
                last_entry_day[s] = t
            elif zt < -entry_z:
                sp_pos[s] = 1
                last_entry_day[s] = t
        else:
            if abs(zt) < exit_z and (t - last_entry_day[s] >=
               min_holding):
                sp_pos[s] = 0
            else:
                sp_pos[s] = prev[s]
    spread_positions_oos.iloc[t] = sp_pos

```

```
print("Signaux de trading (-1,0,1) pour les spreads s lectionn s :")
display(spread_positions_oos)

assets = price_data.columns.tolist()
spread_portfolios_oos = {}
for idx, s in enumerate(selected_spreads_idx):
    spread_name = f"Spread_{s+1}"
    vec = unit_vecs[:, s]
    pos_s = np.outer(spread_positions_oos.iloc[:, idx].values, vec)
    df_s = pd.DataFrame(pos_s, index=dates_oos, columns=assets)
    spread_portfolios_oos[spread_name] = df_s

returns_oos = price_data_oos.pct_change().fillna(0.0)
trade_log = []

for idx, s in enumerate(selected_spreads_idx):
    spread_name = f"Spread_{s+1}"
    pos = spread_positions_oos.iloc[:, idx]
    current_trade = None
    cap = capital_per_spread
    for t, date in enumerate(dates_oos):
        sig = pos.iloc[t]
        if current_trade is None and sig != 0:
            current_trade = {
                "Spread": spread_name,
                "EntryDate": date,
                "Direction": sig,
                "EntryIdx": t,
                "Entry_Capital": cap }
        elif current_trade is not None and sig == 0:
            current_trade["ExitDate"] = date
            current_trade["ExitIdx"] = t
            pos_slice = spread_portfolios_oos[spread_name].iloc[
                current_trade["EntryIdx"]:t]
            ret_slice = returns_oos.iloc[current_trade["EntryIdx"]+1:t
                +1]
            pnl_by_day = (pos_slice.shift(1).fillna(0) * ret_slice).sum(
                axis=1)
            gross_pnl = pnl_by_day.sum() * current_trade["Entry_Capital"
                ]
            turnover_trade = (pos_slice - pos_slice.shift(1).fillna(0)).
                abs().sum(axis=1).sum() * current_trade["Entry_Capital"]
            cost_trade = turnover_trade * transaction_cost
            trade_pnl = gross_pnl - cost_trade
            current_trade["PnL"] = trade_pnl
            cap = current_trade["Entry_Capital"] + trade_pnl
            current_trade["Exit_Capital"] = cap
            current_trade["HoldingDays"] = t - current_trade["EntryIdx"]
            trade_log.append(current_trade)
            current_trade = None

trade_log_df_oos = pd.DataFrame(trade_log)

spread_pnl_summary_oos = {}
for idx, s in enumerate(selected_spreads_idx):
    spread_name = f"Spread_{s+1}"
    trades = trade_log_df_oos[trade_log_df_oos["Spread"]==spread_name].
        reset_index(drop=True)
```

```
    total_pnl_spread = 0.0

    print(f"\n=== Tous les trades pour {spread_name} ===\n")
    for i, trade in trades.iterrows():
        direction = int(trade["Direction"])
        entry_cap = trade["Entry_Capital"]
        weights_trade = unit_vecs[:, s]
        weights_trade_norm = weights_trade / np.sum(np.abs(weights_trade))
        capital_by_asset = entry_cap * direction * weights_trade_norm
        price_entry = price_data_oos.loc[trade["EntryDate"]].values
        price_exit  = price_data_oos.loc[trade["ExitDate"]].values
        pnl_by_asset = capital_by_asset * (price_exit - price_entry) / price_entry
        total_pnl_before_cost = np.sum(pnl_by_asset)
        cost_trade = np.sum(np.abs(capital_by_asset)) * transaction_cost
        trade_pnl = total_pnl_before_cost - cost_trade
        total_pnl_spread += trade_pnl

        df_trade = pd.DataFrame({
            "Commodity": assets,
            "Weight": np.round(weights_trade_norm, 4),
            "Capital_Invested($)": np.round(capital_by_asset, 2),
            "Price_Entry": np.round(price_entry, 2),
            "Price_Exit": np.round(price_exit, 2),
            "PnL_Before_Cost($)": np.round(pnl_by_asset, 2)   })
        print(f"Trade {i+1}: {'Long' if direction==1 else 'Short'} "
              f"| Entry: {trade['EntryDate']} | Exit: {trade['ExitDate']}")
        display(df_trade)
        print(f"Total PnL before cost: {total_pnl_before_cost:.2f}$")
        print(f"Transaction cost: {cost_trade:.2f}$")
        print(f"Trade PnL (net of cost): {trade_pnl:.2f}$")
        print(f"Entry capital: {entry_cap:.2f}$ | Exit capital: {trade['Exit_Capital']:.2f}$")
        print("-"*80)

    initial_cap_spread = trade_log_df_oos[trade_log_df_oos["Spread"] == spread_name]["Entry_Capital"].iloc[0]
    final_cap_spread = trade_log_df_oos[trade_log_df_oos["Spread"] == spread_name]["Exit_Capital"].iloc[-1]
    total_pnl_spread_calculated = final_cap_spread - initial_cap_spread
    print(f"Calculated total PnL for {spread_name} (using Capital method): {total_pnl_spread_calculated:.2f}$")
    manual_pnl_sum = trades["PnL"].sum()
    print(f"Calculated total PnL for {spread_name} (summing trade PnLs): {manual_pnl_sum:.2f}$")
    spread_pnl_summary_oos[spread_name] = total_pnl_spread_calculated
    print(f">>> PnL total pour {spread_name} (net de co ts) : {total_pnl_spread_calculated:.2f}$")

summary_list_oos = []
for idx, s in enumerate(selected_spreads_idx):
    spread_name = f"Spread_{s+1}"
    trades = trade_log_df_oos[trade_log_df_oos["Spread"] == spread_name].reset_index(drop=True)

    if not trades.empty:
```

```
        initial_cap = trades["Entry_Capital"].iloc[0]
        final_cap = trades["Exit_Capital"].iloc[-1]
        pnl_total = final_cap - initial_cap
        pnl_list = trades["PnL"]
        n_trades = len(trades)
        win_rate = (pnl_list > 0).sum() / n_trades * 100 if n_trades > 0
            else 0
        summary_list_oos.append({
            "Spread": spread_name,
            "Total_PnL($)": round(pnl_total,2),
            "N_Trades": n_trades,
            "WinRate(%)": round(win_rate,2) })
    else:
         summary_list_oos.append({
            "Spread": spread_name,
            "Total_PnL($)": 0.0,
            "N_Trades": 0,
            "WinRate(%)": 0.0  })

summary_df_oos = pd.DataFrame(summary_list_oos).set_index("Spread")

print(f"\n=== PnL total global (tous spreads OOS) : {summary_df_oos['
    Total_PnL($)'].sum():.2f}$ ===")
display(summary_df_oos)
```

Listing 13: Trades & Measures per Spread

Table 10: Tableau récapitulatif de tous les trades OOS

Spread	EntryDate	ExitDate	Direction	Entry_Capital	Exit_Capital	PnL	HoldingDays
Spread_1	2022-02-22	2022-03-02	1	50000.00	53260.38	3260.38	6
Spread_1	2022-03-08	2022-03-09	-1	53260.38	53207.12	-53.26	1
Spread_1	2022-04-20	2022-05-23	-1	53207.12	53626.46	419.34	23
Spread_1	2022-06-01	2022-07-22	-1	53626.46	57822.01	4195.55	35
Spread_1	2022-08-16	2022-09-15	-1	57822.01	59157.79	1335.78	21
Spread_1	2022-09-22	2023-02-06	-1	59157.79	65085.21	5927.41	93
Spread_1	2023-07-26	2023-12-08	-1	65085.21	69834.72	4749.52	95
Spread_1	2024-03-26	2024-07-01	1	69834.72	67097.46	-2737.26	66
Spread_1	2024-08-19	2025-07-07	1	67097.46	68285.83	1188.36	220
Spread_2	2022-01-03	2022-03-08	-1	50000.00	69056.48	19056.48	44
Spread_2	2022-03-09	2022-03-23	-1	69056.48	74049.49	4993.01	10
Spread_2	2022-04-25	2022-05-13	1	74049.49	72648.39	-1401.09	14
Spread_2	2022-06-03	2022-08-02	1	72648.39	83998.14	11349.75	40
Spread_2	2022-08-12	2022-12-07	1	83998.14	94727.85	10729.71	81
Spread_2	2022-12-13	2023-02-02	1	94727.85	97045.86	2318.01	34
Spread_2	2023-04-03	2023-07-26	-1	97045.86	101462.35	4416.50	78
Spread_2	2023-08-18	2023-09-01	1	101462.35	101898.83	436.48	10
Spread_2	2023-09-11	2023-09-18	1	101898.83	101075.51	-823.32	5
Spread_2	2024-02-07	2024-02-28	1	101075.51	102436.94	1361.43	14
Spread_2	2024-03-22	2024-04-29	1	102436.94	105054.41	2617.47	25
Spread_2	2024-05-17	2024-06-03	-1	105054.41	100400.96	-4653.44	10
Spread_2	2024-07-17	2024-11-14	-1	100400.96	92068.24	-8332.72	85
Spread_2	2024-12-18	2024-12-27	1	92068.24	93472.47	1404.23	6
Spread_2	2024-12-31	2025-01-08	1	93472.47	92324.42	-1148.05	5
Spread_2	2025-01-17	2025-02-10	1	92324.42	98201.86	5877.44	15
Spread_2	2025-03-12	2025-05-13	-1	98201.86	95331.31	-2870.55	43

```
strategy_equity_curve = initial_capital_strategy + strategy_daily_pnl.
    cumsum()
total_return = (strategy_equity_curve.iloc[-1] / strategy_equity_curve.
    iloc[0]) - 1
total_return_percent = total_return * 100
start_date_oos = '2022-01-01'
```

```
end_date_oos   = '2025-08-31'
num_years = (pd.to_datetime(end_date_oos) - pd.to_datetime(
    start_date_oos)).days / 365.25
annualized_return = (1 + total_return)**(1 / num_years) - 1
annualized_return_percent = annualized_return * 100


strategy_daily_returns = strategy_daily_pnl / strategy_equity_curve.
    shift(1).fillna(initial_capital_strategy)
annualized_volatility = strategy_daily_returns.std() * np.sqrt(252) *
    100
sharpe_ratio = annualized_return / (annualized_volatility / 100)
rolling_max = strategy_equity_curve.cummax()
drawdown = (strategy_equity_curve - rolling_max) / rolling_max
max_drawdown = drawdown.min() * 100

print("=== Strategy Performance Metrics (Out-of-Sample Period:
    2022-01-01 to 2025-08-31) ===")
print(f"Initial Capital: ${initial_capital_strategy:,.2f}")
print(f"Final Capital: ${strategy_equity_curve.iloc[-1]:,.2f}")
print(f"Total Return: {total_return_percent:.2f}%")
print(f"Annualized Return: {annualized_return_percent:.2f}%")
print(f"Annualized Volatility: {annualized_volatility:.2f}%")
print(f"Sharpe Ratio: {sharpe_ratio:.2f}")
print(f"Maximum Drawdown: {max_drawdown:.2f}%")
```

Listing 14: Performance Metrics

```

bci_df = pd.read_excel("BCI.xlsx")
bci_df['Exchange Date'] = pd.to_datetime(bci_df['Exchange Date'])
bci_df.set_index('Exchange Date', inplace=True)
bci_df = bci_df.sort_index()
bci_df = bci_df.loc[price_data_oos.index[0]:price_data_oos.index[-1]]
bci_daily_ret = bci_df['Close'].pct_change().fillna(0)
bci_cum_ret = (1 + bci_daily_ret).cumprod() - 1

strategy_daily_pnl = pd.Series(0, index=price_data_oos.index)
for s_name in ['Spread_1','Spread_2','Spread_6']:
    trades_s = trade_log_df_oos[trade_log_df_oos['Spread']==s_name]
    for _, trade in trades_s.iterrows():
        try:
            entry_idx = price_data_oos.index.get_loc(trade['EntryDate'])
            exit_idx  = price_data_oos.index.get_loc(trade['ExitDate'])
        except KeyError:
            continue
        pos_slice = spread_portfolios_oos[s_name].iloc[entry_idx:
            exit_idx]
        ret_slice = price_data_oos.pct_change().iloc[entry_idx:exit_idx]
        pnl_daily = (pos_slice.shift(1).fillna(0) * ret_slice).sum(axis
            =1) * trade['Entry_Capital']
        strategy_daily_pnl.iloc[entry_idx:exit_idx] += pnl_daily.values

initial_capital_strategy = initial_capital
strategy_cum_ret = (strategy_daily_pnl.cumsum() /
    initial_capital_strategy)

aligned_data = pd.DataFrame({
    'Strategy': strategy_cum_ret.pct_change().fillna(0),
```

```
    'BCI': bci_daily_ret}).dropna()

X = sm.add_constant(aligned_data['BCI'])
y = aligned_data['Strategy']
model = sm.OLS(y, X).fit()
alpha = model.params['const']
beta = model.params['BCI']

plt.figure(figsize=(14,7))
for col in price_data_oos.columns:
    cum_ret = (1 + price_data_oos[col].pct_change().fillna(0)).cumprod()
        - 1
    plt.plot(cum_ret.index, cum_ret, alpha=0.7, label=col)

plt.plot(strategy_cum_ret.index, strategy_cum_ret, color='black',
    linewidth=2.5, label='Strategy')
plt.plot(bci_cum_ret.index, bci_cum_ret, color='red', linewidth=3,
    linestyle='--', label='BCI Benchmark') # Modified line

plt.title(f"Cumulative Returns (OOS)")
plt.xlabel("Date")
plt.ylabel("Cumulative Return")
plt.grid(True)
plt.legend()
plt.show()

final_cum_returns = pd.DataFrame({
    'Commodity_CumReturn(%)': ((1 + price_data_oos.pct_change().fillna
        (0)).cumprod() - 1).iloc[-1]*100,
    'Strategy_CumReturn(%)': strategy_cum_ret.iloc[-1]*100,
    'BCI_CumReturn(%)': bci_cum_ret.iloc[-1]*100})

print("=== Final Cumulative Returns (OOS) ===")
display(final_cum_returns)
```

Listing 15: Graph Cumulative Returns