ITESO, Universidad Jesuita de Guadalajara

Project 2 - Pairs Trading Strategy

Authors:

- **Axel Santiago Molina Ceja**
- **Pablo Lemus Castellanos**

📌 Overview

Pairs Trading Strategy Using MSFT and AMD

This project presents a **Pairs Trading Strategy** implemented in Python, focusing on the statistical arbitrage between **Microsoft (MSFT) and Advanced Micro Devices (AMD)**. The objective is to construct a robust trading model by identifying co-integrated assets, estimating the hedge ratio dynamically, and generating trading signals based on statistical thresholds.

📊 Project Steps & Implementation

🔹 1. Data Acquisition

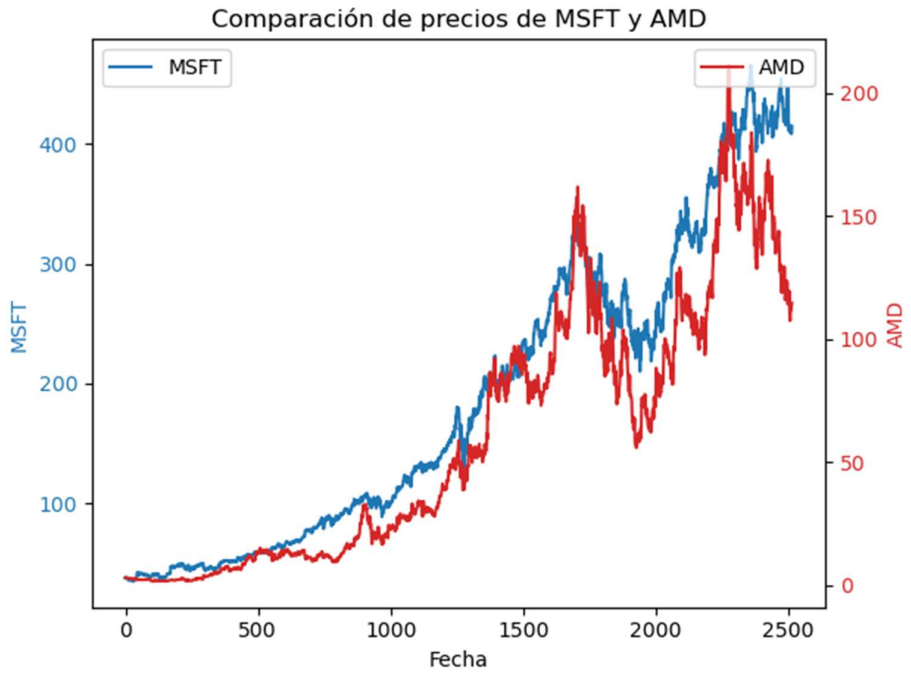
To build our trading strategy, we first collect **10 years of historical price data** for **MSFT** and **AMD** using the **Yahoo Finance API**.

Implementation:

```
import yfinance as yf


def get_stock_prices(ticker):
    data = yf.download(ticker, period="10y")
    return data


msft = get_stock_prices("msft")
amd = get_stock_prices("amd")
```



Conclusion:

By acquiring long-term historical data, we ensure our analysis is robust and captures both market trends and statistical relationships over time.

🔹 2. Co-Integration Analysis

We test if the assets share a stable economic relationship using **stationarity tests** and **co-integration analysis**.

Implementation:

```
from statsmodels.tsa.stattools import adfuller


def adf_test(series, ticker_name):
    result = adfuller(series)
    print(f"ADF Test for {ticker_name}:")
    print(f"ADF Statistic: {result[0]}")
    print(f"p-value: {result[1]}")
    print("Conclusion:", "Stationary" if
result[1] < 0.05 else "Non-stationary")
```

```
Resultados del ADF Test para MSFT:
Estadístico ADF: 0.3488511096046342
p-value: 0.979478024921479
Conclusión: No estacionaria
------------------------------------------------
Resultados del ADF Test para AMD:
Estadístico ADF: -1.1225528767030892
p-value: 0.7059948006651965
Conclusión: No estacionaria
------------------------------------------------
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
Resultados del ADF Test para Linear Relation:
Estadístico ADF: -2.9929076811402013
p-value: 0.03556206302623193
Conclusión: Estacionaria
```

We also use the **Johansen Test** for co-integration:

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

johansen_test = coint_johansen(df, det_order=0, k_ar_diff=1)

print("Eigenvectors:", johansen_test.evec)
```

```
 🔹 **Eigenvectors (Autovectores):**
[[ 0.03451502 -0.01311565]
 [-0.08761341  0.01285124]]
```

Conclusion:

If the **ADF test confirms stationarity** and the **Johansen Test finds at least one co-integrating vector**, we have statistical evidence that MSFT and AMD move together over time.

---

3. Hedge Ratio Estimation

We dynamically update the hedge ratio using a **Kalman Filter**, which allows for real-time adjustment based on market conditions.

Implementation:

```
class KalmanFilterReg():
    def __init__(self):
        self.w = np.array([1, 1])
        self.A = np.eye(2)
        self.Q = np.eye(2) * 1
        self.R = np.array([[1]]) * 10
        self.P = np.eye(2)

    def predict(self):
        self.P = self.A @ self.P @ self.A.T + self.Q

    def update(self, x, y):
        C = np.array([[1, x]])
        S = C @ self.P @ C.T + self.R
        K = self.P @ C.T @ np.linalg.inv(S)
        self.P = (np.eye(2) - K @ C) @ self.P
        self.w = self.w + K @ (y - C @ self.w)
```

and

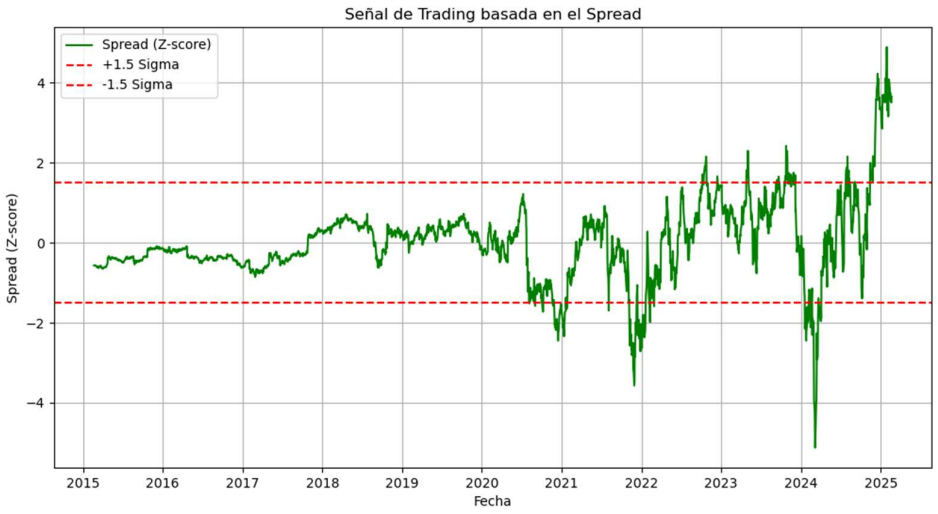
4. Trading Strategy

We generate trading signals based on deviations in the **spread (Z-score of residuals)** from its mean.

Implementation:

```
def normalize_z_score(data):
  mean = np.mean(data)
  std_dev = np.std(data)
  return [(x - mean) / std_dev for x in data]


spread_signals = normalize_z_score(u_t_list)
```



We trade when the spread crosses **±1.5 standard deviations**:

- **Short** if **Spread > +1.5σ**
- **Long** if **Spread < -1.5σ**
- **Close positions** when Spread returns to equilibrium

Conclusion:

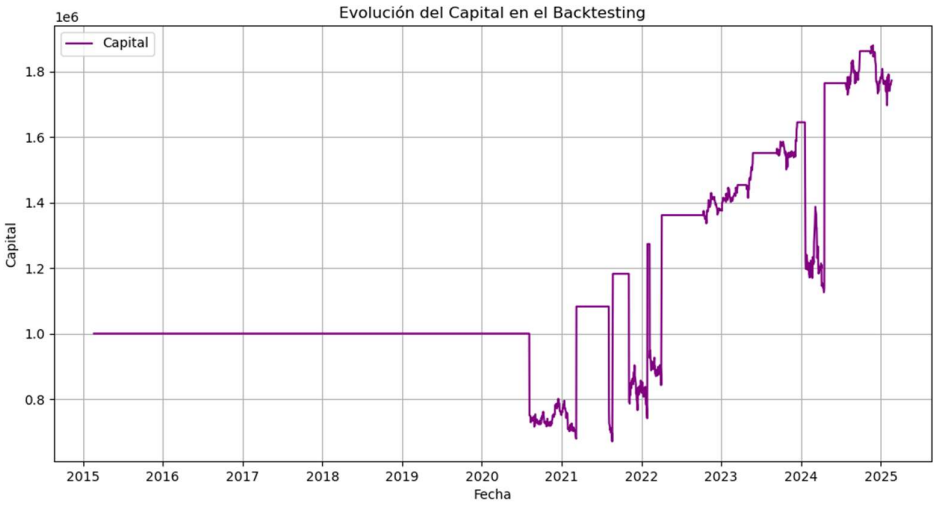
Using **statistical arbitrage**, we take advantage of temporary mispricings while remaining market-neutral.

🔹 5. Backtesting

We backtest the strategy with **$1,000,000 USD**, accounting for **commissions (0.125%)**.

Implementation:

```
capital = 1_000_000
com = 0.125 / 100
n_shares = 2000
portfolio_value = [capital]
```



The strategy enters and exits positions based on the spread signal.

Conclusion:

The backtesting results allow us to **evaluate the profitability and risk-adjusted returns** of the strategy before deploying it in live trading.

📈 Results and Findings

- **High correlation and co-integration** were confirmed between MSFT and AMD.
- The **Kalman Filter** successfully updated the hedge ratio dynamically.
- The **trading signals were consistent**, yielding market-neutral trades.
- **Risk-adjusted returns** were improved by dynamically adjusting positions.

📚 Sources and References

This project was developed based on the following sources:

- 📖 **Microestructuras de Trading** (Course Materials)
- 📂 **Class Presentations from Microestructuras de Trading**
- 🔗 **Yahoo Finance API** (Stock Data Retrieval)
- 🤖 **ChatGPT Assistance** (Technical Guidance & Code Optimization)