Cross-Asset Market Regime Monitor

MSc in Financial Markets and Investments
Python Programming for Finance
Academic Year 2025/26 - Fall 2025 Semester

Alexandre Landi - Skema Business School

Overview

In this session, we built a prototype of a market dashboard using Python. The objective was to fetch market data (equities, FX, indices), validate it, transform it into returns, and plot aligned series.

Throughout, we emphasized:

- fetching data robustly with `yfinance`,
- saving intermediate results to CSV,
- validating data types and missing values,
- aligning multiple series by their common start,
- computing cumulative returns for comparability,
- producing quick visualizations with `matplotlib`.

1 Fetching Market Data

We started by importing the required libraries:

- `pandas` for data manipulation,
- `numpy` for numerical operations,
- `yfinance` for data fetching,
- `matplotlib` for plotting.

```
# ============================
# Market Dashboard Prototype
# Python Version
# ============================

import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
```

We then downloaded SPY (the ETF tracking the S&P 500).

```
# 1. Fetch SPY
spy_df = yf.download("SPY")  # full history by default
spy = spy_df[["Close"]].rename(columns={"Close": "SPY_Close"})

print("SPY head:")
print(spy.head())
print("SPY date range:", spy.index.min(), "->", spy.index.max())
```

Narrative: At this stage we confirm the data loads, we check its structure, and ensure the date range is reasonable. This is an essential first validation step.

2 Saving and Reloading Data

Next, we saved the data locally to a CSV file. This ensures reproducibility if the API becomes unavailable.

```
# 2. Save and reload locally (robustness)
spy.to_csv("spy_data.csv")
spy_from_file = pd.read_csv("spy_data.csv", index_col=0, parse_dates=
    True)

print("Index types:", type(spy.index), type(spy_from_file.index))
```

Narrative: We notice that when reloading from CSV, date indices must be parsed properly. Otherwise, they default to strings. The parameter `parse_dates=True` ensures the index remains a `DatetimeIndex`.

3 Validating Data

Always check for missing values before proceeding.

```
# 3. Validate data (missing check)
missing_counts = spy.isna().sum()
print("Missing counts:\n", missing_counts)
```

Narrative: This quick diagnostic step reassures us there are no gaps in the SPY series. Missing values could arise from holidays, API outages, or formatting issues.

4 Quick Plot

A fast visualization helps spot anomalies.

```
# 4. Quick plot (SPY only)
spy.plot(y="SPY_Close", title="SPY Close (full history)")
plt.xlabel("Date"); plt.ylabel("Price")
plt.show()
```

Narrative: This is our "quick win"—seeing SPY's historical behavior instantly assures us the pipeline is working.

5 Fetching Additional Assets

The dashboard requires more than equities: we added the U.S. Dollar Index (DXY) and the EUR/USD exchange rate.

```
# 6. Fetch DXY and EURUSD
dxy_df = yf.download("DX-Y.NYB")  # full history by default
dxy = dxy_df[["Close"]].rename(columns={"Close": "DXY_Close"})

eurusd_df = yf.download("EURUSD=X")
eurusd = eurusd_df[["Close"]].rename(columns={"Close": "EURUSD_Close"})

print("Fetched series:", list([c for c in [spy.columns, dxy.columns,
    eurusd.columns]]))
```

Narrative: Note that ticker symbols vary by provider. We used `"DX-Y.NYB"` for DXY and `"EURUSD=X"` for the EUR/USD exchange rate.

6 From Prices to Returns

We introduced a helper function to transform prices into cumulative returns. This makes different assets comparable.

```
# 7. Returns and cumulative returns
def pct_to_cumret(series):
    """
    Convert a price series to cumulative returns.
    Starts at 1.0, grows over time.
    """
    r = series.pct_change()
    cumret = (1.0 + r).cumprod()
    cumret.iloc[0] = 1.0
    return cumret

dxy_cum = pct_to_cumret(dxy["DXY_Close"])
eurusd_cum = pct_to_cumret(eurusd["EURUSD_Close"])
```

Narrative: The key insight: cumulative returns normalize all assets to start at 1, letting us track their performance side by side.

7 Aligning Series

Before comparison, all series must be aligned to a common start date.

```
# 8. Align multiple series safely
series_map = {
    "SPY_Close": spy["SPY_Close"],
    "DXY_Close": dxy["DXY_Close"],
    "EURUSD_Close": eurusd["EURUSD_Close"]
}

# latest first-valid date across series
first_dates = [s.dropna().index.min() for s in series_map.values()]
common_start = max(first_dates)

multi = pd.concat(series_map, axis=1, join="outer")
aligned = multi[multi.index >= common_start]

print("Common start date:", common_start)
print(aligned.head())
```

Narrative: This prevents errors from series that begin on different dates. We take the maximum of all start dates, ensuring fairness in comparison.

8 Normalization

We normalize aligned series into cumulative returns.

```
# 9. Normalize all series to cumulative returns
multi_cum = aligned.apply(lambda col: (1 + col.pct_change()).cumprod(),
    axis=0)
multi_cum.iloc[0] = 1.0
```

Narrative: Now each series begins at 1.0 on the common start date, which makes comparisons across SPY, DXY, and EUR/USD meaningful.

9 Visualization

Finally, we plot aligned prices and cumulative returns.

```
# 10. Plot aligned series (prices and cumrets)
aligned.plot(title="Aligned Prices (SPY, DXY, EURUSD)")
plt.xlabel("Date"); plt.ylabel("Price")
plt.show()

multi_cum.plot(title="Aligned Cumulative Returns (SPY, DXY, EURUSD)")
plt.xlabel("Date"); plt.ylabel("Cumulative Return (base=1)")
plt.show()
```

Narrative: This is the first version of our dashboard: fast, reproducible, and easy to extend.

10 Exporting Results

We saved outputs to CSV for later reuse.

```
# 11. Save outputs
aligned.to_csv("aligned_prices.csv")
multi_cum.to_csv("aligned_cumrets.csv")

print("Saved aligned prices and cumulative returns.")
```

Narrative: This ensures we can reload data later without fetching from the API again.

Key Takeaways

- Always validate data (types, missing values, ranges).
- Save intermediate CSVs for robustness against outages.
- Align series by common start dates to ensure fairness.
- Normalize to cumulative returns for comparability.