Lesson 12: Time Series Basics

Data Science with Python – BSc Course

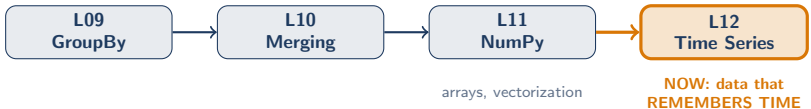
Data Science Program

BSc Course

45 Minutes

Where We Are

Last time: NumPy gave you fast array math for portfolio analysis



Financial data has a **timestamp**. This changes everything – order matters, gaps have meaning, and patterns repeat over time.

Module 2 capstone – combining everything from L07–L12

Learning Objectives

After this lesson, you will be able to:

- **Create** and manipulate DatetimeIndex for time-indexed data
- **Parse** date strings into datetime objects using `pd.to_datetime()`
- **Resample** data between frequencies (daily $\rightarrow$ weekly $\rightarrow$ monthly)
- **Apply** rolling and expanding windows for moving statistics
- **Compute** returns, lags, and drawdowns with `shift()` and `pct_change()`

Finance Application: Moving averages, drawdown analysis, rolling correlations, seasonality.

Time series analysis is THE core skill for financial data science

Why Time Series Are Special

Financial data has a TIMESTAMP. This changes everything.

Regular DataFrame:

- Row order does not matter
- No concept of "next" or "previous"
- Cannot ask "what happened last month?"

Time Series DataFrame:

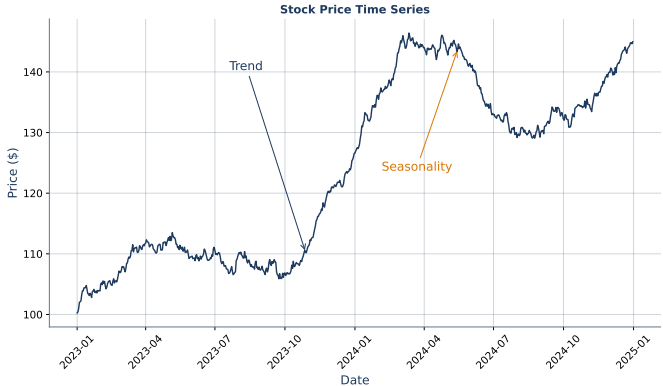
- Row order IS the data
- shift() gives yesterday's value
- Resample daily $\rightarrow$ monthly
- Rolling windows for trends

```
df = pd.read_csv("prices.csv", parse_dates=["Date"],
    index_col="Date")  # DatetimeIndex unlocks time powers
```

A DatetimeIndex turns a regular DataFrame into a time machine.

DatetimeIndex = the key that unlocks resampling, rolling windows, and date slicing

Basic Time Series



Time-indexed data enables date-based slicing: df.loc['2024-01':'2024-03']

Parsing Dates

Three ways to parse:

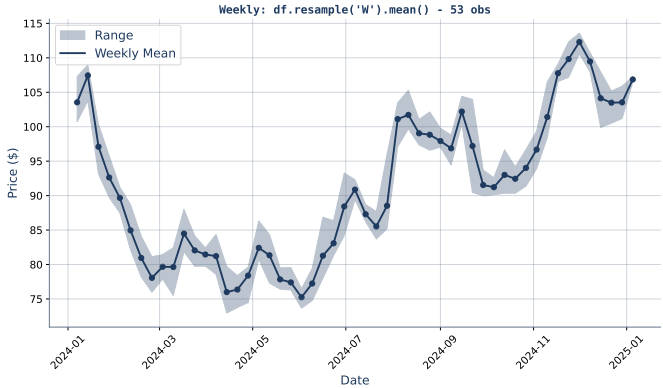
1. `pd.to_datetime('2024-01-15')` – auto-detect format
2. `pd.to_datetime(df['date'], format='%Y-%m-%d')` – explicit (faster)
3. `pd.read_csv('file.csv', parse_dates=['date'])` – on import

Useful accessors:

`df.index.year` | `df.index.month` | `df.index.dayofweek` | `df.index.quarter`

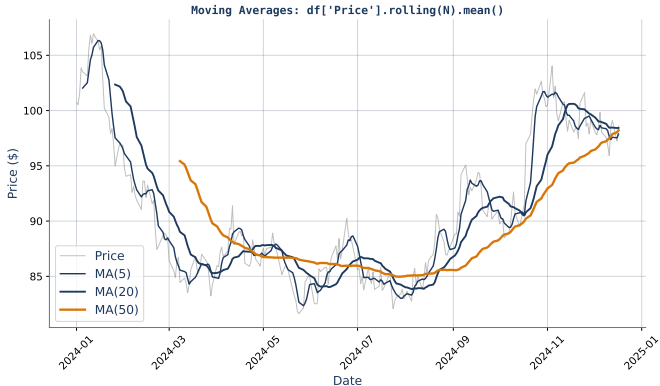
Always parse dates on import – string dates cannot be resampled

Resampling: Changing Frequency



df.resample('W').mean() converts daily data to weekly averages

Rolling Windows: Moving Statistics



df.rolling(20).mean() computes a 20-day moving average

Expanding Windows: Cumulative Statistics

Rolling (fixed window):

`df.rolling(20).mean()`

Uses the **last 20 observations** only. Window slides forward each day.

Expanding (growing window):

`df.expanding().mean()`

Uses **all data up to that point**. Window grows each day.

When to use which:

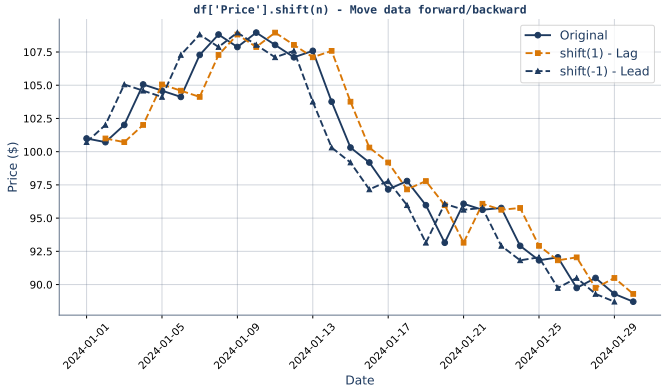
Rolling	Expanding
Recent trend	All-time average
20-day MA	Running mean
Current vol	Cumulative max
Recent Sharpe	Drawdown calc

```
# Cumulative maximum price
df['Peak'] = df['Close']
    .expanding().max()

# Drawdown from peak
df['DD'] = df['Close']
    / df['Peak'] - 1
```

Rolling = recent snapshot; Expanding = historical accumulation

Shift: Creating Lags



df.shift(1) gives yesterday's value – the building block for returns

Computing Returns

Simple returns:

```
df['Return'] = df['Close'].pct_change()
```

Equivalent to: $(P_t - P_{t-1})/P_{t-1}$

Log returns:

```
df['LogRet'] = np.log(df['Close'] /
df['Close'].shift(1))
```

Time-additive (can sum across days).

Multi-period returns:

```
df['Ret_5d'] = df['Close'].pct_change(5)
```

Weekly return (5 trading days).

Cumulative returns:

```
(1 + df['Return']).cumprod() - 1
```

Growth of $1 invested.

First value is NaN because there is no previous day to compare.

---

pct_change() = shift + divide in one step. Always check for NaN at row 0.

Checkpoint: Test Your Understanding

Q1: What is the difference between `rolling(20).mean()` and `expanding().mean()`?

Q2: `df.shift(1)` moves data forward or backward? What value appears in the first row?

Q3: You have daily prices. How do you get monthly average prices?

Think for 30 seconds. Answers: Q1: Rolling uses last 20 only; expanding uses all data up to that point. Q2: Forward (NaN in first row). Q3: df.resample('ME').mean().

Rolling, shifting, and resampling are the three pillars of time series work

Time Zones

Setting a time zone:

```
# Localize naive timestamps
df.index = df.index
    .tz_localize('US/Eastern')
# Convert between zones
df.index = df.index
    .tz_convert('UTC')
# Check current zone
df.index.tz
```

Why it matters in finance:

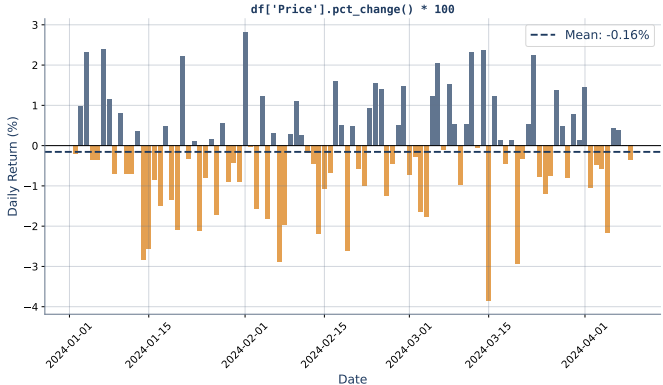
- NYSE opens 9:30 ET
- London opens 8:00 GMT
- Merging cross-market data requires aligned timestamps
- `tz_localize` first, then `tz_convert`

Common zones:

US/Eastern, Europe/London, UTC, Asia/Tokyo

Always localize before converting – mixing naive and aware timestamps causes errors

Finance: Daily Returns



Daily returns cluster – high volatility follows high volatility

Finance: Drawdown Calculation

How far has the price fallen from its peak?

```
# Step 1: Cumulative returns
cumulative = (1 + df['Return']).cumprod()
# Step 2: Running maximum (expanding window)
peak = cumulative.expanding().max()
# Step 3: Drawdown
drawdown = cumulative / peak - 1
# Step 4: Maximum drawdown
max_dd = drawdown.min()
print(f"Max drawdown: {max_dd:.1%}")
```

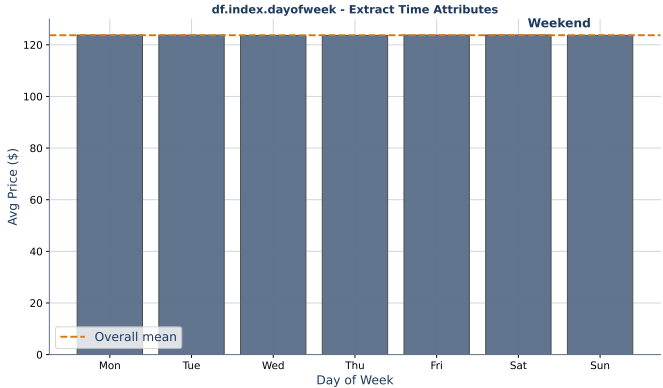
Interpretation: A max drawdown of -35% means the portfolio lost 35% from its highest point. This is the **risk metric investors care about most**.

Drawdown = how much you lost from the peak. Uses expanding().max().

Finance: Rolling Volatility

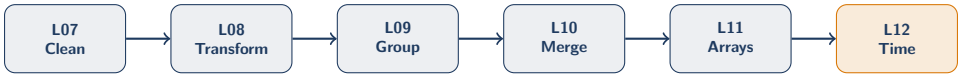
Rolling volatility reveals risk regimes – calm vs. turbulent periods

Finance: Seasonal Patterns



Day-of-week effects: use df.index.dayofweek to analyze calendar patterns

Module 2 Complete!



Clean → Transform → Group → Merge → Compute → Time

Your complete data pipeline is ready!

You can wrangle ANY financial dataset.

Six lessons: from messy raw data to time series analysis

Resampling Reference

Aggregation functions:

```
df.resample('ME').mean()  # Average monthly price
df.resample('ME').last()  # Month-end closing price
df.resample('ME').agg({'Close': 'last', 'Volume': 'sum'})
```

OHLC shortcut: `df.resample('ME').ohlc()` gives Open/High/Low/Close.

Resampling = groupby for time – aggregate within each time period

Hands-on Exercise (25 min)

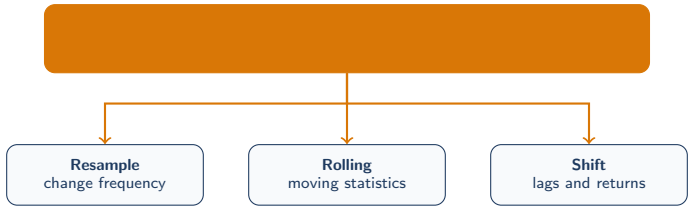
Analyze stock price time series:

1. Load stock data with `parse_dates` and `index_col='Date'`
2. Compute daily returns with `pct_change()`
3. Resample to monthly returns using `resample('ME').last()`
4. Plot 20-day and 50-day moving averages
5. Compute rolling 20-day volatility: `rolling(20).std()`
6. Calculate maximum drawdown using `expanding().max()`
7. Analyze day-of-week return patterns using `index.dayofweek`

Bonus: Compute rolling 60-day correlation between two stocks.

This exercise uses every time series tool from today's lesson

The Big Idea



With DatetimeIndex, your DataFrame becomes a time machine.

Resample + rolling + shift: the time series toolkit

Key Takeaways

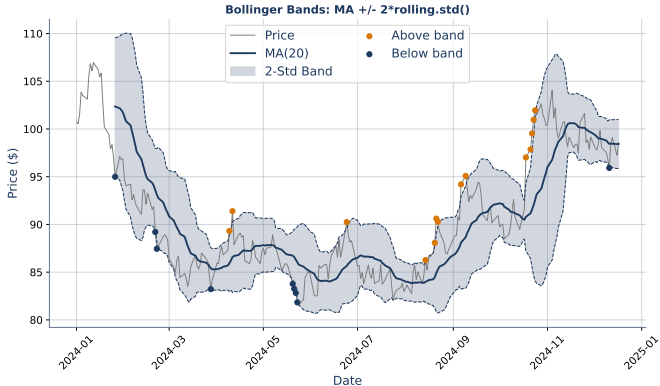
What you learned today:

- DatetimeIndex enables date slicing, resampling, and time-based operations
- `resample()` changes data frequency (daily → weekly → monthly)
- `rolling(N)` computes moving statistics over the last N observations
- `shift()` creates lags; `pct_change()` computes period returns
- Expanding windows power drawdown calculations and cumulative metrics

Up Next: Module 3 – Statistics and Visualization
L13: Descriptive Statistics. You have clean, merged, time-indexed data. Now you learn to **describe it** – mean, median, variance, skewness, and more.

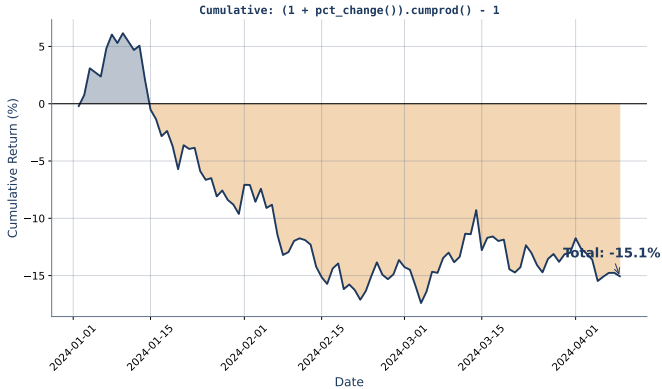
Module 2 complete – you can clean, transform, group, merge, and time-analyze data!

Finance: Bollinger Bands



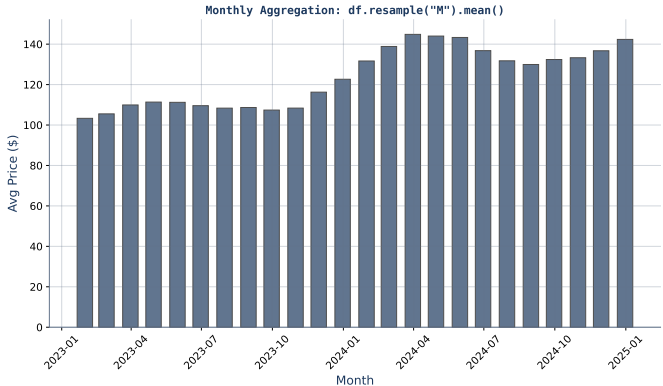
Bollinger Bands = rolling mean +/- 2 * rolling std – a volatility envelope

Finance: Cumulative Returns



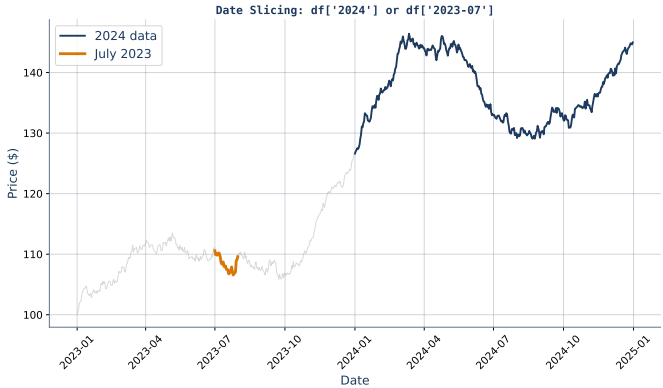
(1 + daily_returns).cumprod() – growth of $1 invested over time

Monthly Aggregation



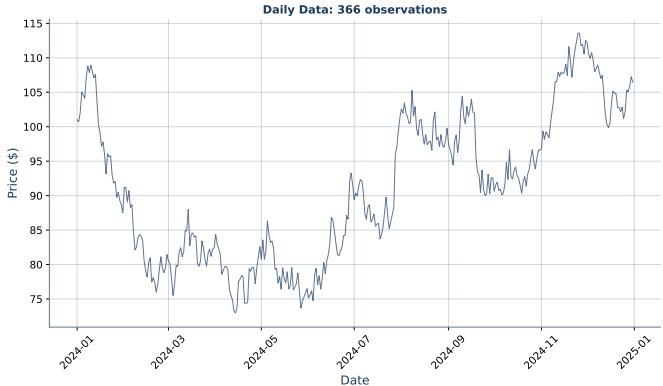
Summarizing daily data into monthly periods

Date Slicing



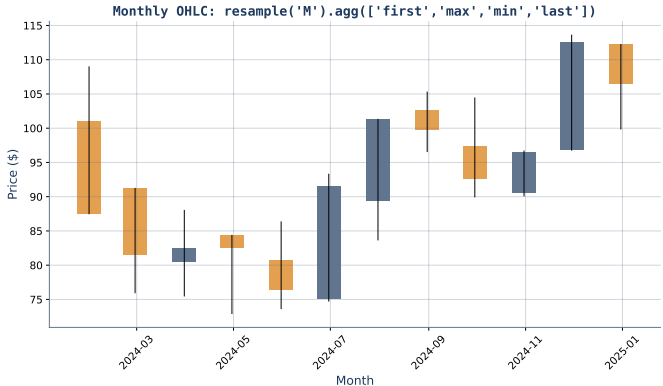
df.loc['2024-01':'2024-06'] for date-range selection

Daily Data



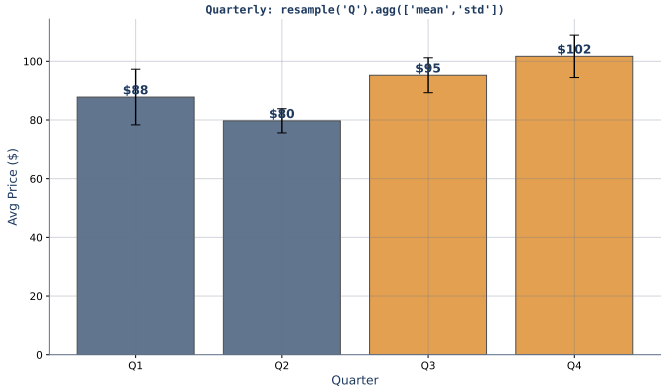
Working with daily frequency data

Monthly OHLC



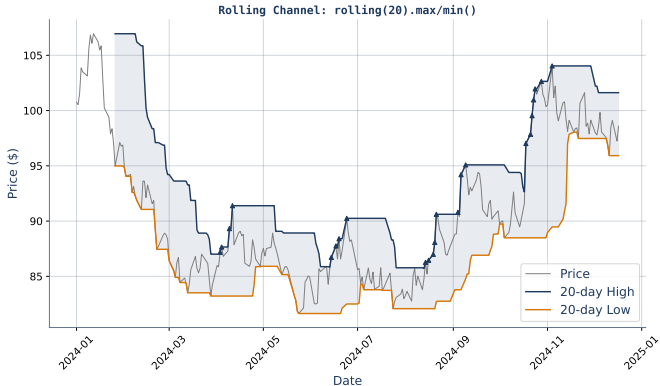
Open-High-Low-Close monthly bars via resample('ME').ohlc()

Quarterly Resampling



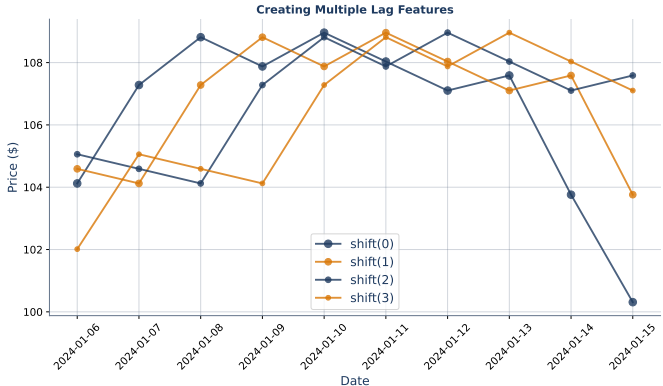
Aggregating to quarterly frequency for earnings analysis

Rolling Channel



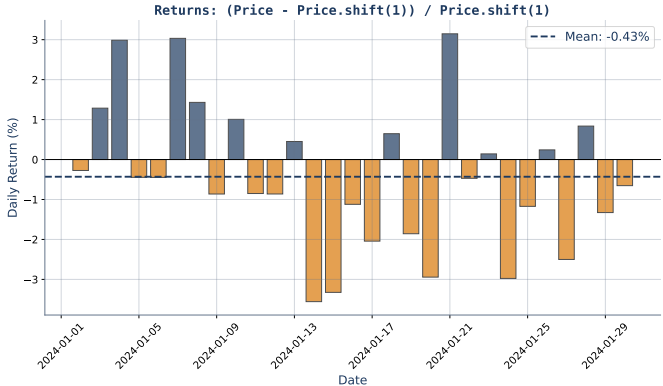
Price channels with rolling min/max

Multiple Lags



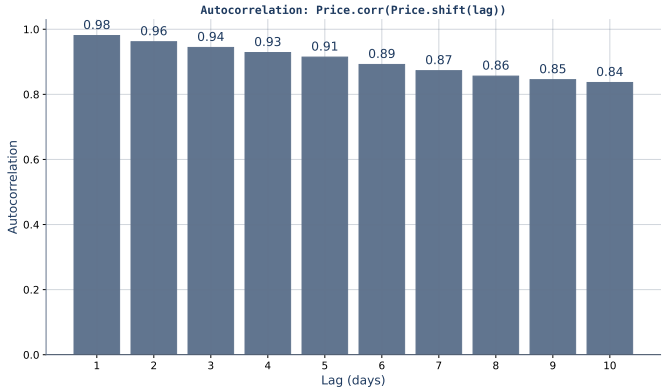
Creating multiple lagged features for predictive models

Returns with Shift



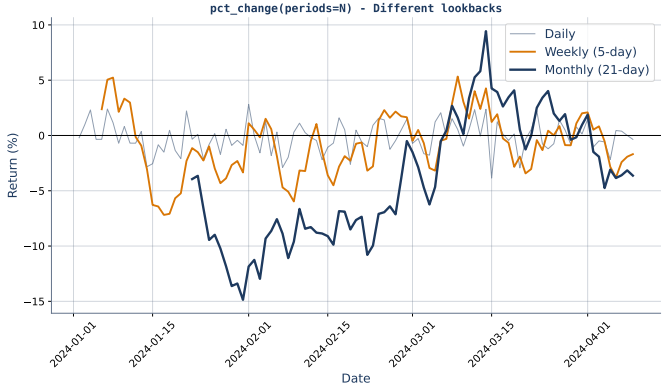
Computing returns manually: (price - price.shift(1)) / price.shift(1)

Autocorrelation



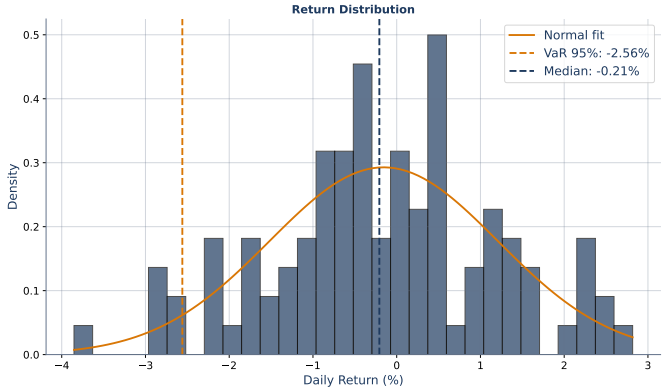
Serial correlation in returns – do past returns predict future returns?

Return Periods Comparison



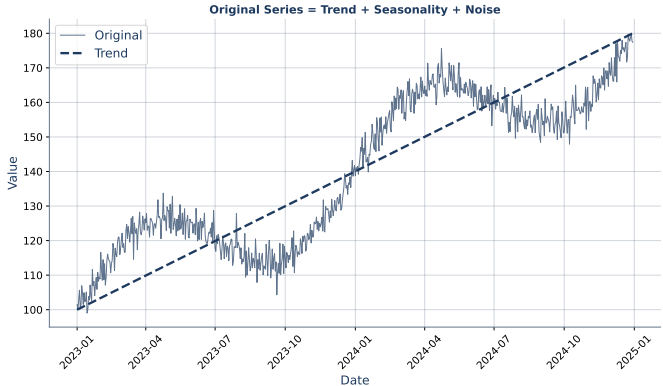
Daily, weekly, and monthly return distributions

Return Distribution



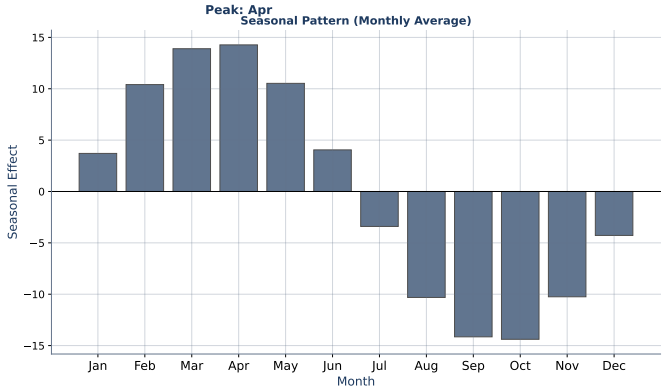
Fat tails: real returns have more extreme events than normal distribution

Trend Decomposition



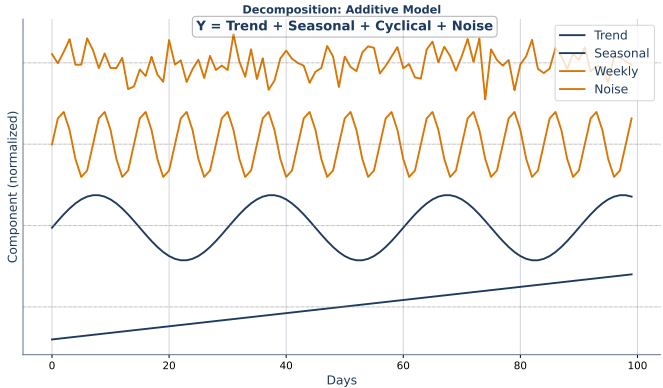
Identifying the long-term trend component

Seasonal Pattern



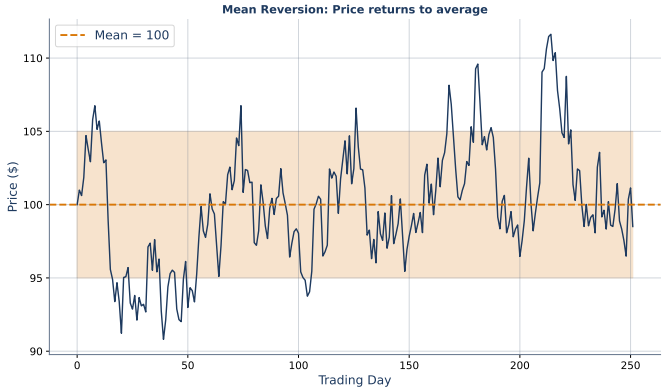
Recurring patterns within periods

Full Decomposition



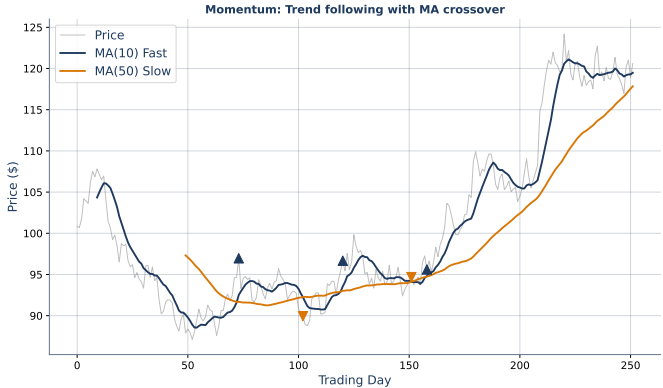
Trend + Seasonal + Residual = full decomposition

Mean Reversion



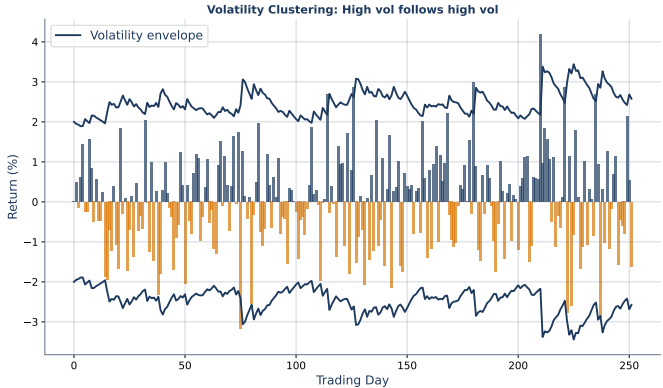
Price returning to its long-term average

Momentum



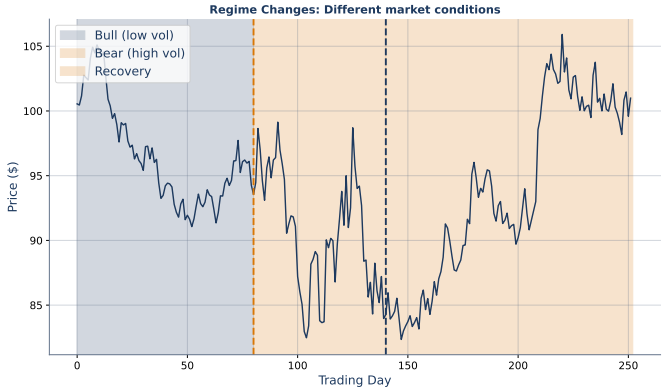
Trend-following patterns in asset prices

Volatility Clustering



High volatility follows high volatility – a key stylized fact

Regime Changes



Markets switch between calm and turbulent regimes