PYTHON FOR QUANT TRADERS · PART 0

รากฐาน — ก่อนเขียนโค้ดบรรทัดแรก

คณิตศาสตร์ · สถิติ · ตลาดการเงิน · Quant Mindset · Roadmap

ออกแบบสำหรับทุกคน — ไม่ว่าจะจบสายอะไร

วิสัยทัศน์

โลกเปลี่ยนแล้ว — ไม่ต้องจบสายการเงินหรือ CS เพื่อเขียนโค้ดวิเคราะห์ตลาดได้
หมอที่อ่านหนังสือเล่มนี้จบจะเห็นว่า p-value ในงานวิจัยยา กับ p-value ใน regression ของราคาหุ้น คือสิ่งเดียวกัน ทนายที่อ่านจบจะเห็นว่า LP constraint $x_1 + x_2 \leq 1$ คือ "เงื่อนไขสัญญา" ในภาษาคณิตศาสตร์

Part 0 ปูพื้น 5 เรื่องที่จำเป็น — ทั้งหมดเชื่อมตรงกับโค้ดที่จะเขียนใน Part I ถึง VI

แผนที่หนังสือ — 35 บท 7 ส่วน

Part 0	Part I	Part II	Part III	Part IV	Part V	Part VI
รากฐาน	Python Basics	Math Tools	OOP	Backtesting	AI-Assisted	Live Trading
บท 1–5	บท 6–10	บท 11–17	บท 18–21	บท 22–25	บท 26–30	บท 31–35
◀ อยู่ที่นี่						

บทที่ 1: คณิตศาสตร์ที่ต้องรู้จริงๆ

แก่นของบทนี้

คณิตศาสตร์ทั้งหมดใน Quant สรุปได้เป็น 4 ความคิด: **ฟังก์ชัน** (input→output), **ผลรวม** Σ (บวกหลายค่า), **เวกเตอร์** (รายการเลข), **เมทริกซ์** (ตารางเลข) — ทั้งหมดแปลเป็น Python ได้ตรงตัว

1.1 ฟังก์ชัน — เครื่องแปลงค่า

ฟังก์ชันคือกฎที่บอกว่า "ใส่อะไรเข้าไป ได้อะไรออกมา" เหมือนสูตรอาหาร

$$f(x) = 2x + 3$$

อ่านว่า: "ฟังก์ชัน f ของ x เท่ากับ สองเท่าของ x บวกสาม"
ตัวอย่าง: $f(5) = 2(5) + 3 = 13$ | $f(10) = 2(10) + 3 = 23$

Running Example: ฟังก์ชันใน Quant

ผลตอบแทนรายวัน (Return) เป็นฟังก์ชันของราคา:

$$r_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

อ่านว่า: "r ณ เวลา t เท่ากับ ราคาวันนี้ ลบ ราคาเมื่อวาน หารด้วย ราคาเมื่อวาน"
ตัวอย่าง: $P_{t-1} = 100$, $P_t = 103 \rightarrow r_t = \frac{103-100}{100} = 3\%$

ใน Python: `r = (price_today - price_yesterday) / price_yesterday` — เราจะเขียนสิ่งนี้ใน Part I

ฟังก์ชันหลายตัวแปร

ฟังก์ชันรับ input ได้มากกว่า 1 ตัว เช่น ความดันโลหิตเป็นฟังก์ชันของหลายปัจจัย:

$$BP = \alpha + \beta_1 \cdot \text{น้ำหนัก} + \beta_2 \cdot \text{อายุ} + \beta_3 \cdot \text{ออกกำลังกาย}$$

ใน Quant เราทำสิ่งเดียวกัน: ผลตอบแทนหุ้น = ฟังก์ชันของ (ตลาดรวม, อุตสาหกรรม, ข่าว, ...)

1.2 ผลรวม Σ — บวกหลายค่าพร้อมกัน

$$S = \sum_{i=1}^{n} x_i = x_1 + x_2 + x_3 + \cdots + x_n$$

อ่านว่า: "ซิกมาของ x_i ตั้งแต่ i=1 ถึง n" = "บวก x ทุกตัวเข้าด้วยกัน"

ตัวอย่าง: ผลกำไรรายสัปดาห์

กำไร 5 วัน: $x = [+200, -150, +300, -80, +120]$ บาท

$$\sum_{i=1}^{5} x_i = 200 + (-150) + 300 + (-80) + 120 = +390 \text{ บาท}$$

ใน Python: `sum([200, -150, 300, -80, 120])` คืนค่า 390

▶ **แบบฝึกหัด: คำนวณกำไรรวม**

ผลตอบแทนรายวัน 4 วัน: [+1%, −0.5%, +2%, −0.3%]
$\sum_{i=1}^{4} r_i = 1 + (-0.5) + 2 + (-0.3) = +2.2\%$
กำไรรวม (แบบ simple) = 2.2% ของเงินทุนเริ่มต้น

1.3 เวกเตอร์ — รายการตัวเลขที่เรียงลำดับ

เวกเตอร์คือ **รายการตัวเลข** ไม่มีอะไรน่ากลัวกว่านั้น ใน Python คือ `list` หรือ numpy `array`

$$\mathbf{P} = [100,\ 105,\ 98,\ 110,\ 107]$$

แปลว่า: ราคาหุ้น 5 วัน (เวกเตอร์ขนาด 5 มิติ)

Operation พื้นฐานบนเวกเตอร์

Operation	ความหมาย	Python
$\mathbf{x} + \mathbf{y}$	บวกทีละ element	`x + y` (numpy)
$c \cdot \mathbf{x}$	คูณทุก element ด้วย c	`c * x`
$\mathbf{x} \cdot \mathbf{y}$	dot product: $\sum x_i y_i$	`np.dot(x, y)`
$\lVert\mathbf{x}\rVert$	ความยาวเวกเตอร์: $\sqrt{\sum x_i^2}$	`np.linalg.norm(x)`

1.4 เมทริกซ์ — ตาราง Excel ที่มีตัวเลขทุกช่อง

เมทริกซ์คือตาราง $m \times n$ (m แถว, n คอลัมน์) ใน Python คือ pandas `DataFrame` หรือ `numpy 2D array`

$$\mathbf{A} = \begin{pmatrix} 100 & 200 & 150 \\ 105 & 195 & 148 \\ 98 & 210 & 155 \end{pmatrix}$$

แถว = วัน | คอลัมน์ = หุ้น (AAPL, MSFT, NVDA)
$A[0][0] = 100$ หมายถึง AAPL วันที่ 1 ราคา 100

Matrix ที่สำคัญที่สุดใน Quant: Covariance Matrix

เมทริกซ์ความแปรปรวนร่วมบอกว่าแต่ละคู่หุ้นเดินสัมพันธ์กันแค่ไหน:

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_A^2 & \sigma_{AB} \\ \sigma_{AB} & \sigma_B^2 \end{pmatrix}$$

ตัวเลขในแนวทแยง = ความแปรปรวนของแต่ละหุ้น | นอกแนวทแยง = ความแปรปรวนร่วมของคู่หุ้น
ใช้ใน Part II สำหรับ portfolio optimization

สรุปความเชื่อมโยงกับ Python

คณิตศาสตร์	ใน Python	เริ่มเรียนที่
ฟังก์ชัน $f(x)$	`def f(x): return ...`	Part I บทที่ 7
ผลรวม Σ	`sum()` / `np.sum()`	Part I บทที่ 9
เวกเตอร์	`list` / `np.array`	Part I บทที่ 9
เมทริกซ์	`pd.DataFrame`	Part I บทที่ 9

บทที่ 2: สถิติสำหรับมนุษย์

แก่นของบทนี้

สถิติคือการ "สรุปข้อมูลจำนวนมากให้เป็นตัวเลขไม่กี่ตัว" 5 concept ที่ต้องรู้: ค่าเฉลี่ย, ส่วนเบี่ยงเบน, การกระจายปกติ, correlation, และ regression — ทั้งหมด Python คำนวณได้ใน 1–2 บรรทัด แต่ต้องรู้ *ความหมาย* ก่อนจึงจะตีความผลลัพธ์ได้ถูก

2.1 ค่าเฉลี่ยและส่วนเบี่ยงเบน — อยู่ตรงไหน กระจายแค่ไหน?

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \qquad \sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

อ่านว่า: "$\bar{x}$ (x-bar) = ค่าเฉลี่ย" | "σ (sigma) = ส่วนเบี่ยงเบนมาตรฐาน = ค่าเฉลี่ยของ*ระยะห่าง* จากค่าเฉลี่ย"

Running Example: ผลตอบแทนรายวัน

ผลตอบแทน 5 วัน: $r = [+1\%,\ -0.5\%,\ +2\%,\ -1\%,\ +0.5\%]$

$$\bar{r} = \frac{1 + (-0.5) + 2 + (-1) + 0.5}{5} = \frac{2}{5} = 0.4\%/\text{วัน}$$

$$\sigma \approx 1.08\%/\text{วัน}$$

แปลว่า: วันปกติผลตอบแทนอยู่ระหว่างประมาณ -0.68% ถึง $+1.48\%$ (±1σ จาก $\bar{r}$)

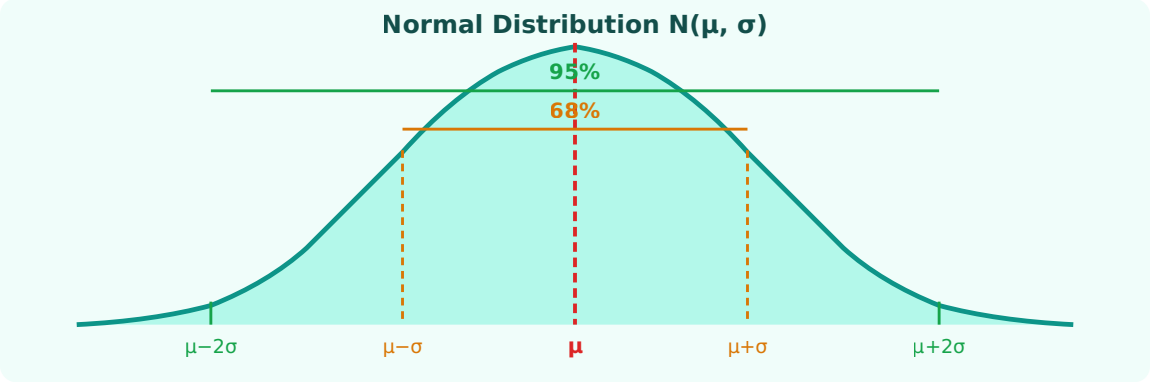
ใน Python: `np.mean(r)` และ `np.std(r)`

ความหมายของ σ ในการเทรด

สินทรัพย์	σ รายวัน	แปลว่า
US Treasury Bond	~0.05%	นิ่งมาก, ผลตอบแทนต่ำ
หุ้นขนาดใหญ่ (S&P500)	~1%	ปกติสำหรับหุ้น
BTC	~3–5%	ผันผวนสูง, โอกาส/ความเสี่ยงมาก
Altcoin ขนาดเล็ก	~10%+	เก็งกำไรสูงมาก

2.2 Normal Distribution — รูประฆัง

ข้อมูลหลายอย่างในธรรมชาติกระจายเป็น "รูประฆัง" (Bell Curve) รอบค่าเฉลี่ย μ ด้วยความกว้าง σ



กฎ 68–95–99.7: ข้อมูล 68% อยู่ใน μ±σ, 95% อยู่ใน μ±2σ, 99.7% อยู่ใน μ±3σ

ตัวอย่าง: กฎ 68–95–99.7 กับหุ้น

BTC มีผลตอบแทนรายวัน $\mu = 0\%$, $\sigma = 3\%$
→ วันส่วนใหญ่ (68%) BTC เคลื่อนที่ระหว่าง −3% ถึง +3%
→ วันที่ตก >6% (เกิน 2σ) เกิดได้ประมาณ 5% ของวันทั้งหมด = ประมาณ 12–13 วัน/ปี
→ วันที่ตก >9% (เกิน 3σ) เกิดได้ประมาณ 0.3% = ประมาณ 1 วัน/ปี

2.3 Correlation — สองตัวแปรเดินไปด้วยกันไหม?

$$r_{XY} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \cdot \sum(y_i - \bar{y})^2}} \quad \in [-1,\ +1]$$

$r = +1$: เดินทิศเดียวกันสมบูรณ์ | $r = 0$: ไม่สัมพันธ์ | $r = -1$: เดินสวนทางสมบูรณ์

Correlation ≠ Causation (ความสัมพันธ์ ≠ เป็นสาเหตุ)

ไอศกรีมขาย correlate กับอัตราจมน้ำ ($r \approx +0.9$) — แต่ไม่ใช่เพราะกินไอศกรีมแล้วจมน้ำ แต่เพราะ "อากาศร้อน" เป็นตัวแปรที่ซ่อนอยู่ (confounding variable)
ใน Quant: สองหุ้นอาจ correlate กันโดย *ไม่* cointegrate กัน — ต่างกันมาก เราจะเรียนใน Part II

2.4 p-value — มั่นใจแค่ไหนว่าไม่ใช่ความบังเอิญ?

p-value ตอบคำถามว่: *"ถ้าสมมติว่าไม่มีผลจริง (*null hypothesis*) โอกาสที่จะเห็นข้อมูลแบบนี้โดยบังเอิญคือเท่าไหร่?"*

Running Example: ทดสอบกลยุทธ์

กลยุทธ์ทำกำไรเฉลี่ย +0.1%/วัน ใน 252 วัน (1 ปี)
p-value = 0.03 หมายความว่า:
→ "ถ้ากลยุทธ์ไม่มีประสิทธิภาพจริง (random) โอกาสที่จะเห็นผลดีขนาดนี้โดยบังเอิญ = 3%"
→ ต่ำกว่า 5% = **มีนัยสำคัญทางสถิติ** (statistically significant)

ใน Python: หลาย library คืน p-value ให้โดยอัตโนมัติ เช่น
`statsmodels.OLS().fit().pvalues`

2.5 Regression — หาเส้นที่พอดีที่สุด

สูตรหลัก — OLS REGRESSION

$$y = \alpha + \beta x + \varepsilon$$

α (alpha) = intercept = ค่า y เมื่อ x = 0
β (beta) = slope = ถ้า x เพิ่ม 1 หน่วย y เปลี่ยนไป β หน่วย
ε (epsilon) = error = ส่วนที่ model ทำนายไม่ได้

OLS (Ordinary Least Squares) หาค่า α, β ที่ทำให้ $\sum \varepsilon_i^2$ น้อยที่สุด — เรียนลึกใน Part II บทที่ 13

ตัวอย่าง: Hedge Ratio ระหว่างสองหุ้น

เราต้องการหาว่า AAPL เคลื่อนที่ตาม MSFT แค่ไหน:

$$r_{\text{AAPL}} = \alpha + \beta \cdot r_{\text{MSFT}} + \varepsilon$$

ผล OLS: $\beta = 0.85$ หมายความว่า
→ ทุกครั้งที่ MSFT ขึ้น 1% AAPL มักขึ้น 0.85%
→ ถ้าต้องการ hedge AAPL $100K ด้วย MSFT → ต้อง short MSFT $85K

```
# Python (Part II) import statsmodels.api as sm X =
sm.add_constant(r_msft) model = sm.OLS(r_aapl, X).fit() beta =
model.params['r_msft'] # = 0.85
```

▶ แบบฝึกหัด: อ่านผล regression

สมมติ regression ผลตอบแทน BTC บน ETH ได้: $\beta = 1.12$, $\alpha = 0.001$, $R^2 = 0.78$

แปลว่า:
• ETH ขึ้น 1% → BTC มักขึ้น 1.12% (BTC volatile กว่า ETH เล็กน้อย)
• $R^2 = 0.78$ = 78% ของความผันผวน BTC อธิบายได้ด้วย ETH
• 22% ที่เหลือ (ε) คือ "idiosyncratic move" ที่ไม่สัมพันธ์ — นี่คือ spread ที่จะ trade

บทที่ 3: ตลาดการเงิน 101

แก่นของบทนี้

ไม่ต้องรู้จักหุ้นทุกตัว แค่รู้ 4 concept: **สินทรัพย์** (มีอะไรซื้อขายได้บ้าง), **Return** (วัดผลกำไร/ขาดทุนอย่างไร), **Risk = Volatility** (วัดความไม่แน่นอน), **Arbitrage** (หลักการหากำไรไร้ความเสี่ยง) — 4 concept นี้รองรับโค้ดที่จะเขียนตลอดเล่ม

3.1 ประเภทสินทรัพย์

ประเภท	ความหมายง่ายๆ	ตัวอย่าง
หุ้น (Stocks)	ส่วนแบ่งความเป็นเจ้าของบริษัท — กำไรขาดทุนตามบริษัท	AAPL, PTT, ADVANC
พันธบัตร (Bonds)	กู้ยืมเงิน ได้ดอกเบี้ยแน่นอน — ความเสี่ยงต่ำกว่าหุ้น	US Treasury, พันธบัตรรัฐบาลไทย
FX / Crypto	แลกเปลี่ยนระหว่างสกุลเงิน — ราคาขึ้นลงตามอุปสงค์อุปทาน	USD/THB, BTC/USDT
Derivatives	สัญญาที่ราคาขึ้นกับสินทรัพย์อื่น — leverage สูง, ซับซ้อน	Futures, Options, Perpetuals

3.2 Price vs Return

นักวิเคราะห์ทำงานกับ Return มากกว่า Price เพราะ Return เปรียบเทียบข้ามสินทรัพย์ได้

$$r_t^{\text{simple}} = \frac{P_t - P_{t-1}}{P_{t-1}} \qquad r_t^{\log} = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

Simple vs Log Return — เลือกอันไหน?

	Simple Return	Log Return
สูตร	$(P_t - P_{t-1})/P_{t-1}$	$\ln(P_t/P_{t-1})$
บวกข้ามเวลา	ไม่ได้โดยตรง	ได้ (additive)
กระจายเป็น Normal	ใกล้เคียง	ดีกว่า
ใช้เมื่อ	ผลตอบแทนระยะสั้น (ต่ำกว่า 5%)	วิเคราะห์เชิงสถิติ, cointegration

สำหรับค่าน้อยๆ: log return ≈ simple return (ต่างกันน้อยมาก)

3.3 Risk = Volatility

ใน Finance **Risk = ความไม่แน่นอน = Standard Deviation ของ Return** เรียกว่า Volatility (σ)

ตัวอย่าง: Annualized Volatility

BTC มี $\sigma_{\text{daily}} = 3\%$ ต่อวัน
แปลงเป็นรายปี: $\sigma_{\text{annual}} = \sigma_{\text{daily}} \times \sqrt{T}$ โดย T = จำนวนวันซื้อขายต่อปี

$$\sigma_{\text{annual}} = 3\% \times \sqrt{252} \approx 47.6\%/\text{ปี}$$

แปลว่า: ใน 1 ปี ราคา BTC มักเคลื่อนที่ไปมา ±47.6% จากค่าเฉลี่ย
หมายเหตุ: หุ้นไทย/สหรัฐใช้ $\sqrt{252}$ (252 วันทำการ) — แต่ **Crypto เปิด 24/7 ใช้** $\sqrt{365}$
BTC: $3\% \times \sqrt{365} \approx 57.3\%/\text{ปี}$ — ตัวเลขต่างกันไม่ใช่ผิด แต่ต้องใช้ให้สม่ำเสมอ

3.4 Arbitrage — ซื้อถูก ขายแพง พร้อมกัน

Arbitrage คือการทำกำไรโดย *ไม่มีความเสี่ยงสุทธิ* จากราคาที่ไม่สมดุลกันระหว่างตลาด

ทำไม Pure Arbitrage ถึงหายากขึ้นเรื่อยๆ?

เมื่อมีคนเห็น Arb → รีบซื้อตลาด A + ขายตลาด B → ราคา A สูงขึ้น, ราคา B ลง → ส่วนต่างหด → หายภายใน ms–วินาที
โอกาสที่เหลือต้องการ: **ความเร็ว** (automated bot), **ข้อมูลที่คนอื่นไม่มี**, หรือ **capital ขนาดใหญ่** สำหรับโอกาสที่ margin น้อย

3.5 Portfolio และ Diversification

$$R_p = \sum_{i=1}^{n} w_i R_i \qquad \sum_{i=1}^{n} w_i = 1$$

อ่านว่า: "ผลตอบแทน portfolio เท่ากับ ผลรวมถ่วงน้ำหนักของผลตอบแทนแต่ละสินทรัพย์"

Diversification: กระจายเพื่อลด Risk โดยไม่ลด Return

ถ้าหุ้น A และ B มี $r = -1$ (สวนทางสมบูรณ์) → ถือ 50%/50% → portfolio ไม่ขึ้นไม่ลง
ถ้า $r = 0$ (ไม่สัมพันธ์) → $\sigma_p = \sigma/\sqrt{2}$ = ลด risk ได้ 30% โดยไม่ลด return
กุญแจ: หา assets ที่ correlation ต่ำ เพื่อกระจาย risk ได้จริง

บทที่ 4: คิดแบบ Quant

แก่นของบทนี้

ทักษะ coding และ math เป็นแค่เครื่องมือ สิ่งที่แยก Quant ออกจากคนทั่วไป คือ **วิธีคิด** — ตัดสินใจด้วยตัวเลข ไม่ใช่ความรู้สึก, รู้ว่าโมเดลมีขีดจำกัด, คิดเรื่อง expected value ก่อนเสมอ

4.1 ทุกอย่างคือ Data

สิ่งที่เห็น	Quant มองว่า	ใช้ทำอะไร
ราคาหุ้น	Time series ของ returns	หา pattern, signal
ข่าว	Sentiment score [−1, +1]	NLP model ทำนายทิศ
ปริมาณซื้อขาย	Volume series	ยืนยัน signal, หา liquidity
Fear ตลาด	VIX index	ปรับ position size
คำพูด Fed	Hawkish/Dovish score	ปรับ model สำหรับ macro regime

Running Example: หมอกับ Quant คิดคล้ายกัน

หมอไม่ "รู้สึก" ว่าคนไข้เป็นอย่างไร — *วัด*: BP, HR, SpO_2, GCS, Temp
Quant ไม่ "รู้สึก" ว่าตลาดจะขึ้นหรือลง — ดู data: price, volume, σ, correlation, macro

ทั้งคู่ **ตัดสินใจจาก signal ที่วัดได้** ไม่ใช่ intuition — แต่ทั้งคู่ก็รู้ว่า signal ไม่สมบูรณ์เสมอ

4.2 Model = แผนที่ ≠ ดินแดน

> **"All models are wrong, but some are useful" — George E.P. Box**
>
> แผนที่กรุงเทพฯ ไม่ใช่กรุงเทพฯ จริง — ไม่มีต้นไม้ ไม่มีเสียง ไม่มีกลิ่น
> แต่แผนที่ช่วยนำทางได้ ซึ่งคือจุดประสงค์
>
> โมเดล Quant เช่นกัน: ไม่ได้อธิบายตลาดครบทุกมิติ แต่ช่วยตัดสินใจได้ดีกว่าไม่มีโมเดล
> **อันตรายคือการลืมว่าโมเดลคือแผนที่ แล้วเชื่อว่ามันคือความจริงสมบูรณ์**

4.3 Expected Value — ตัดสินใจด้วยค่าคาดหวัง

สูตรหลัก

$$E[X] = \sum_i p_i \cdot x_i$$

p_i = ความน่าจะเป็นที่ผลลัพธ์ i จะเกิด (ทุก p_i รวมกัน = 1)
x_i = มูลค่าของผลลัพธ์ i (บวก = กำไร, ลบ = ขาดทุน)

Quant ดูที่ E[X] ก่อนเสมอ ไม่ใช่แค่ "น่าจะชนะ"

ตัวอย่าง: เปรียบเทียบสองกลยุทธ์ด้วย Expected Value

กลยุทธ์ A
ชนะ 70% ได้ +฿100 | แพ้ 30% เสีย −฿300

ชนะ 70%	แพ้ 30%
+฿100 → +฿70	−฿300 → −฿90

E[A] = +70 − 90 = −฿20

กลยุทธ์ B
ชนะ 40% ได้ +฿800 | แพ้ 60% เสีย −฿200

ชนะ 40%	แพ้ 60%
+฿800 → +฿320	−฿200 → −฿120

E[B] = +320 − 120 = +฿200 ✓

กลยุทธ์ A ชนะบ่อยกว่า (70%) แต่ E[X] ติดลบ — กลยุทธ์ B แพ้บ่อยกว่า (60%) แต่ E[X] เป็นบวกมาก

4.4 Backtesting — ทดสอบก่อนใช้เงินจริง

Backtesting คือการนำกลยุทธ์ไป "เล่นซ้ำ" กับข้อมูลราคาในอดีต เพื่อดูว่าถ้าใช้ในอดีตจะได้ผลลัพธ์อย่างไร

อันตราย: Overfitting = การเรียน "คำตอบสอบ" ไม่ใช่ "วิชา"

ถ้าปรับ parameter มากพอ กลยุทธ์ใดๆ ก็ดูดีในข้อมูลเก่าได้เสมอ
เหมือนจำคำตอบข้อสอบ — ผ่านข้อสอบเก่าได้ แต่ข้อสอบจริง (live market) ไม่ได้

วิธีป้องกัน: ทดสอบกับข้อมูลที่โมเดลไม่เคยเห็น (out-of-sample) — เราจะเรียนใน Part IV

▶ แบบฝึกหัด: คิดแบบ Quant

กลยุทธ์นี้ดีไหม? "ทำกำไร 8 ใน 10 ครั้ง"

ข้อมูลไม่พอ ต้องถามเพิ่ม:
1. ครั้งที่แพ้ 2 ครั้ง ขาดทุนเท่าไหร่? → ถ้าแพ้ครั้งละ −5,000 บาท และชนะครั้งละ +100 บาท → E[X] ติดลบ
2. ทดสอบกับข้อมูล out-of-sample ไหม? → ถ้าทดสอบแค่ in-sample อาจ overfit
3. ค่า transaction cost รวมอยู่ด้วยไหม? → ถ้าไม่รวม ผลจริงอาจแย่กว่ามาก

บทที่ 5: Roadmap & วิธีอ่านหนังสือเล่มนี้

แก่นของบทนี้

หนังสือเล่มนี้ออกแบบให้ **อ่านตามลำดับ** — แต่ละ Part สร้างบนของเดิม ถ้าติดในบท X ให้อ่านต่อไปก่อน เพราะบางครั้ง context ข้างหน้าช่วยให้เข้าใจย้อนกลับมาได้ดีกว่า

5.1 AI ช่วยได้ที่ไหนบ้าง?

หนังสือเล่มนี้ไม่ได้สอนให้ "จำ syntax" — สอนให้ **รู้ว่าต้องการอะไร และตรวจสอบ AI ได้**

AI ทำได้ดี	คุณต้องรู้เอง
เขียน boilerplate / template code	ว่า algorithm ที่ต้องการคืออะไร
แปลง logic เป็น Python syntax	ว่า logic นั้นถูกต้องทาง math ไหม
Debug error message	ว่า root cause จริงคืออะไร
เขียน OLS / optimization ให้	ว่า β หรือ output ที่ได้มีความหมายอะไร
หา library และ function ที่เหมาะ	ว่าผลลัพธ์สมเหตุสมผลไหม

Running Example: Prompt ที่ดีสำหรับ Quant

```
# Prompt ที่ไม่ดี: "เขียน trading bot ให้หน่อย" # Prompt ที่ดี: "เขียน Python
function ที่รับ DataFrame ของราคา 2 assets คืนค่า hedge ratio beta จาก OLS
regression (log prices) และ z-score ของ spread: log(P1) - beta *
log(P2) ใช้ statsmodels.OLS, normalize z-score ด้วย rolling window 60 วัน"
```

Prompt ที่ดีต้องระบุ: input, output, method, library — เราจะเรียนใน Part V

5.2 Learning Path ตามอาชีพ

พื้นฐานที่มี	Part ที่ต้องใส่ใจเป็นพิเศษ	ข้อได้เปรียบ
หมอ / นักวิทยาศาสตร์	Part II (statistics ลึก)	p-value, hypothesis testing คุ้นเคย
วิศวกร / IT	Part III, VI (OOP, async)	programming concepts คุ้นเคย
นักการเงิน / Trader	Part I (Python basics)	ตลาด, risk management คุ้นเคย
ทนาย / นักบัญชี	Part II (LP constraints)	คิด constraint, rule-based ได้ดี
ไม่มีพื้นฐานใดเลย	Part 0 อ่านช้าๆ ทำแบบฝึกหัดทุกข้อ	ไม่มี bad habit จากสาขาอื่น

5.3 Environment ที่ต้องติดตั้งก่อน Part I

1. **Python 3.10+** — ภาษาหลัก ดาวน์โหลดจาก python.org
2. **VS Code** — โปรแกรมเขียนโค้ด (ฟรี) + Python extension
3. **Conda หรือ pip** — จัดการ library
4. **AI Assistant** (Claude, ChatGPT, Gemini) — คู่หูตลอดการเรียน

บทที่ 6 สอนขั้นตอนการติดตั้งทีละขั้นตอนพร้อม screenshot — ไม่ต้องทำตอนนี้

5.4 Checklist ก่อนเริ่ม Part I

▶ ตรวจสอบความพร้อม — คลิกเพื่อดู

ถ้าตอบ "ใช่" ได้ทุกข้อ พร้อมแล้ว:

- ฟังก์ชัน $f(x) = 2x + 3$ แปลว่า "ใส่ x เข้าไป คูณ 2 บวก 3 ออกมา" ✓
- $\sum_{i=1}^{3} x_i$ เมื่อ $x = [10, 20, 30]$ มีค่าเท่ากับ 60 ✓
- Standard Deviation ใหญ่ = ข้อมูลกระจายมาก = ผันผวนมาก ✓
- Correlation $r = -0.8$ หมายถึง X ขึ้นเมื่อ Y ลง (สวนทางกัน) ✓
- Return รายวัน 3% = $(P_t - P_{t-1})/P_{t-1} = 0.03$ ✓
- Arbitrage ต้องซื้อและขายพร้อมกัน ไม่ใช่ซื้อก่อนแล้วรอขาย ✓
- $E[X] = 0.7 \times (+100) + 0.3 \times (-300) = -20$ ✓

ถ้ายังไม่แน่ใจบางข้อ → กลับไปอ่านบทนั้นซ้ำได้เลย ไม่ต้องรีบ

จบ Part 0 — คุณพร้อมแล้ว

ตอนนี้คุณมีเครื่องมือทางความคิดครบ: **Math** (ฟังก์ชัน, Σ, เวกเตอร์, เมทริกซ์) + **สถิติ** (mean, σ, correlation, regression) + **Finance** (assets, return, risk, arb) + **Quant mindset** (EV, model = map, backtest)

ใน Part I เราจะเริ่มแปลง concept เหล่านี้เป็น Python code ที่รันได้จริง — บรรทัดแรกของคุณรอคุณอยู่ในบทที่ 6

PYTHON FOR QUANT TRADERS · PART I

Python Basics — เริ่มเขียนโค้ดบรรทัดแรก

บทที่ 6–10: Setup · Variables · Control Flow · Data Structures · NumPy/Pandas · Visualization

จบ Part นี้ → โหลดข้อมูลตลาด วิเคราะห์ และ plot กราฟได้จริง

สิ่งที่จะทำได้เมื่อจบ PART I

- โหลดข้อมูลราคาหุ้น/crypto จาก CSV เข้า Python
- คำนวณ daily return, rolling mean, volatility ได้ใน 3–5 บรรทัด
- กรองข้อมูล หาวันที่ผลตอบแทนเกิน threshold ได้
- plot กราฟราคาและ return ได้ พร้อม label และ title

ทุกอย่างนี้เป็นพื้นฐานของ quant analysis ทั้งหมดที่จะตามมา

บทที่ 6: Setup & Python Basics

แก่นของบทนี้

Python คือภาษาที่ "อ่านออกเหมือนภาษาอังกฤษ" — ไม่ต้องประกาศ type, ไม่ต้อง compile, รันได้ทันที บทนี้ตั้ง environment และสร้างความคุ้นเคยกับ variable, type, และ operator ซึ่งเป็นอิฐก้อนแรกของทุก program

6.1 ติดตั้ง Environment

ขั้นตอน 4 ขั้น (ทำครั้งเดียว)

1. **Python 3.10+** — ดาวน์โหลดจาก `python.org` เลือก "Add to PATH" ตอนติดตั้ง
2. **VS Code** — ดาวน์โหลดจาก `code.visualstudio.com` → ติดตั้ง Extension "Python" ของ Microsoft
3. **Terminal** — เปิด Terminal ใน VS Code (Ctrl+`) แล้วรัน: `pip install numpy pandas matplotlib`
4. **ทดสอบ** — สร้างไฟล์ `test.py` พิมพ์ `print("Hello Quant")` แล้วรัน (F5 หรือ `python test.py`)

Running Example: ตลอด Part I — วิเคราะห์ราคา BTC

ทุกบทใน Part I จะต่อยอดจากตัวอย่างเดียวกัน: โหลดราคา BTC 1 ปี → คำนวณ return → หา signal → plot กราฟ
บท 6 เริ่มจาก variable พื้นฐาน → บท 9 จะเป็น DataFrame จริง → บท 10 จะ plot ออกมาให้เห็น

6.2 Variables และ Types

ใน Python ไม่ต้องประกาศ type — Python รู้เองว่าตัวแปรเก็บอะไร

```
# กำหนดค่าตัวแปร price = 65000 # int (จำนวนเต็ม) return_pct = 0.032 # float (ทศนิยม) symbol = "BTCUSDT" # str (ข้อความ) is_long = True # bool (True/False)
```

```
# ดู type print(type(price)) # <class 'int'> print(type(return_pct))# <class 'float'>
```

Type	ตัวอย่าง	ใช้กับ
`int`	`65000` , `-3` , `0`	จำนวนหุ้น, วัน, index
`float`	`0.032` , `65000.5`	ราคา, ผลตอบแทน, volatility
`str`	`"AAPL"` , `"2024-01-01"`	ชื่อหุ้น, วันที่, label
`bool`	`True` , `False`	signal on/off, position open/close

ชื่อตัวแปรที่ดีใน Quant Code

- ใช้ snake_case: `daily_return` ไม่ใช่ `dailyReturn`
- ชื่อบอกความหมาย: `btc_price_usd` ดีกว่า `p`
- หลีกเลี่ยง `l` , `O` , `I` (สับสนกับ 1, 0)
- ค่า constant ใช้ตัวใหญ่: `RISK_FREE_RATE = 0.05`

6.3 อ่านโค้ดให้ออกเสียง — ทักษะแรกก่อนหัดเขียน

คนอ่านหนังสือออกเพราะรู้ว่าตัวอักษรแต่ละตัว "ออกเสียง" อย่างไร โค้ดก็เหมือนกัน — **ทุกสัญลักษณ์มีเสียงอ่านในหัว** ถ้ารู้เสียงอ่าน บรรทัดโค้ดจะกลายเป็นประโยคภาษาไทยธรรมดา และนี่คือทักษะที่สำคัญที่สุดในยุค AI: ต่อให้ไม่เคยเขียนเอง คุณก็**อ่านโค้ดที่ AI เขียนให้ออก**ได้ เหมือนเด็กที่อ่านหนังสือออกก่อนจะเขียนเรียงความเป็น

```
a = "hello world"
```

🗣**อ่านว่า:** เอา `"hello world"` มาใส่ใน `a` — เครื่องหมาย `=` ทำงานจาก**ขวาไปซ้าย**เสมอ: ค่าทางขวา ถูกใส่เข้า "กล่อง" ทางซ้าย

```
price = price + 10
```

🗣**อ่านว่า:** เอาค่า `price` เดิม บวก 10 แล้วใส่กลับเข้า `price` — บรรทัดนี้**ไม่ใช่สมการคณิตศาสตร์** (ในคณิต price = price + 10 เป็นไปไม่ได้!) แต่คือคำสั่ง "อัปเดตค่า"

ตารางเสียงอ่าน — ใช้ตลอดทั้งเล่ม

สัญลักษณ์	อ่านว่า	ตัวอย่าง	อ่านทั้งบรรทัดว่า
`=`	"เอา...มาใส่ใน..."	`z = 2.5`	เอา 2.5 มาใส่ใน z
`==`	"ถามว่า...เท่ากับ...ไหม"	`z == 2.5`	ถามว่า z เท่ากับ 2.5 ไหม (ได้คำตอบ True/False)
`!=`	"ถามว่า...ไม่เท่ากับ...ไหม"	`signal != 0`	ถามว่า signal ไม่เท่ากับ 0 ใช่ไหม
`>=`	"ถามว่า...อย่างน้อย...ไหม"	`z >= 2.0`	ถามว่า z มากกว่าหรือเท่ากับ 2.0 ไหม
`.`	"ของ" (อ่านย้อนจากขวา)	`df.mean()`	ค่าเฉลี่ย**ของ** df
`[ ]`	"หยิบตัวที่..."	`prices[0]`	หยิบตัวแรกจาก prices
`( )` หลังชื่อ	"สั่งให้ทำงาน"	`calc_return()`	สั่งให้ calc_return ทำงาน
`:` ท้ายบรรทัด	"แล้วทำต่อไปนี้"	`if z > 2:`	ถ้า z มากกว่า 2 แล้วทำต่อไปนี้

⚠ กับดักอันดับ 1 ของมือใหม่: `=` กับ `==`

```
if signal = 1: #    SyntaxError! ใช้ "ใส่ค่า" ในที่ที่ต้อง "ถาม" if signal == 1:
# ✓ "ถามว่า signal เท่ากับ 1 ไหม"
```

จำง่าย ๆ: **หนึ่งขีด = สั่ง (ใส่ค่า), สองขีด = ถาม (เปรียบเทียบ)** — โชคดีที่ Python ฟ้อง error ทันทีถ้าใช้ผิดใน if แต่ภาษาอื่นบางภาษาปล่อยผ่าน แล้ว bug จะตามหลอกหลอน

วิธีฝึกอ่านโค้ด

ทุกครั้งที่เจอโค้ดในเล่มนี้ ลอง**อ่านออกเสียงเป็นภาษาไทยทีละบรรทัด**ก่อนดูคำอธิบาย ถ้าอ่านแล้วเป็นประโยคที่เข้าใจได้ แปลว่าคุณเข้าใจโค้ดบรรทัดนั้นแล้ว

จริง ๆ — จากบทนี้ไป จะมีกล่อง 🗣 "อ่านว่า" แทรกใต้โค้ดสำคัญ เพื่อให้เทียบเสียงอ่านของตัวเองได้

6.4 Operators และ Expressions

```
# Arithmetic operators price_today = 65000 price_yesterday = 63000
simple_return = (price_today - price_yesterday) / price_yesterday
print(f"Return: {simple_return:.2%}") # Return: 3.17% # Operators ที่ใช้บ่อย
profit = 1000 tax_rate = 0.15 net = profit * (1 - tax_rate) # = 850.0 days
= 252 annual_vol = 0.03 * (days ** 0.5) # ** = ยกกำลัง (ทำไมไม่ใช้ ^ ดูกล่องด้านล่าง)
shares = 1000 remainder = shares % 100 # % = เศษจากหาร: 1000 % 100 = 0 lots =
shares // 100 # // = หารปัดทิ้งทศนิยม: 1000 // 100 = 10
```

คำถามที่พบบ่อย: ทำไม `**` ไม่ใช้ `^` แบบคณิต?

ใน Python เครื่องหมาย `^` ถูกสงวนไว้สำหรับ "XOR" (bitwise operation ที่ใช้ใน cryptography) จึงต้องใช้ `**` แทนการยกกำลัง — ไม่ผิดปกติ แค่ convention ของภาษา

สำหรับ `//` และ `%` : ถ้า `1000 / 100 = 10.0` (float) แล้ว `1000 // 100 = 10` (int, ปัดทิ้งทศนิยม) และ `1000 % 100 = 0` (เศษ) — สามสิ่งนี้เป็นชุดเดียวกัน: ผลหาร, ผลหารปัดลง, และเศษที่เหลือ
`range(1, 5)` ให้ค่า 1, 2, 3, 4 (ไม่รวมตัวสุดท้าย 5) — rule เดียวกับ slicing `[1:5]` นี่เป็น convention ของ Python เพื่อให้ `len(range(1, 5)) == 5 - 1 == 4` นับออกง่าย

6.5 f-string — แสดงผลแบบมีรูปแบบ

f-string คือวิธี "ประกอบข้อความ" สำหรับแสดงผล — **ในเล่มนี้เราใช้ f-string เฉพาะใน `print()` ตอนแสดงผลลัพธ์สุดท้ายเท่านั้น** การคำนวณจริงจะแยกเป็นตัวแปรชื่อชัด ๆ เสมอ เพื่อให้อ่านง่าย (AI ชอบยัดการคำนวณเข้าไปใน f-string เลย — เดี๋ยวบทหลังจะฝึกอ่านแบบนั้น)

```
symbol = "BTC" price = 65432.789 ret = 0.0317 # f-string: ใส่ตัวแปรใน { } ได้เลย
print(f"{symbol} ราคา: ${price:,.2f}") # BTC ราคา: $65,432.79 print(f"Return:
{ret:.2%}") # Return: 3.17% print(f"Annualized: {ret * 252:.1f}%") #
Annualized: 8.0%
```

Format codes ที่ใช้บ่อยใน Quant

Code	ความหมาย	ตัวอย่าง	ผลลัพธ์
`:.2f`	ทศนิยม 2 ตำแหน่ง	`f"{65432.789:.2f}"`	`65432.79`
`:.0f`	ไม่มีทศนิยม	`f"{65432:.0f}"`	`65432`
`:,`	คั่นหลักพัน	`f"{65432:,}"`	`65,432`
`:.2%`	แปลงเป็น % ทศนิยม 2	`f"{0.0317:.2%}"`	`3.17%`
`:.4f`	ทศนิยม 4 ตำแหน่ง	`f"{0.0317:.4f}"`	`0.0317`

ตัวอย่าง: คำนวณและแสดงผล **Trade Summary**

```
entry_price = 63000.0 exit_price = 65450.0 position_size = 0.5 # BTC
pnl = (exit_price - entry_price) * position_size ret = (exit_price -
entry_price) / entry_price print(f"Entry: ${entry_price:,.2f}")
print(f"Exit: ${exit_price:,.2f}") print(f"P&L: ${pnl:,.2f}")
print(f"Return: {ret:.2%}")
```

```
▶ OUTPUT Entry: $63,000.00 Exit: $65,450.00 P&L: $1,225.00 Return:
3.89%
```

▶ แบบฝึกหัด: คำนวณ **Annualized Return**

ถ้า daily return = 0.1% (0.001) คำนวณ simple annualized return (252 วัน) และแสดงผลแบบ %

```
daily_ret = 0.001 annual_ret = daily_ret * 252 # simple annualization
print(f"Annualized Return: {annual_ret:.1%}") # Annualized Return:
25.2% # หรือแบบ compound: annual_compound = (1 + daily_ret) ** 252 - 1
print(f"Compound Annual: {annual_compound:.1%}") # Compound Annual:
28.4%
```

บทที่ 7: Control Flow & Functions

แก่นของบทนี้

Control flow คือการบอก program ว่า "ทำอะไรเมื่อเงื่อนไขแบบไหน" และ "ทำซ้ำกี่ครั้ง" — ใน Quant ใช้สร้าง signal logic, loop ผ่านข้อมูล, และ function ที่ reuse ได้ทั้ง codebase

7.1 เงื่อนไขคืออะไร — คำถามที่ตอบได้แค่ True/False

ก่อนรู้จัก if ต้องเข้าใจก่อนว่า "เงื่อนไข" คืออะไร: **เงื่อนไขคือคำถามที่คำตอบมีแค่ 2 แบบ — True (จริง) หรือ False (เท็จ)**

```
z = 2.3 z > 2.0 # คำถาม: z มากกว่า 2.0 ไหม? → คำตอบ: True z == 2.0 # คำถาม: z
เท่ากับ 2.0 ไหม? → คำตอบ: False z < -2.0 # คำถาม: z น้อยกว่า -2.0 ไหม? → คำตอบ:
False
```

คำตอบ True/False เป็น "ค่า" จริง ๆ ใน Python — เอามาใส่ตัวแปรได้เหมือนตัวเลข:

```
is_overbought = z > 2.0 # เอาคำตอบ (True) มาใส่ใน is_overbought
print(is_overbought) # True
```

> **อ่านว่า:** ถามว่า z มากกว่า 2.0 ไหม แล้วเอาคำตอบมาใส่ใน is_overbought — สังเกตว่า `=` (ใส่ค่า) กับ `>` (ถาม) ทำงานคนละหน้าที่ในบรรทัดเดียวกัน: ฝั่งขวาถามก่อน ได้คำตอบแล้วค่อยใส่

Comparison Operators — เครื่องมือสร้างคำถาม

Operator	อ่านว่า	ตัวอย่าง
`==`	เปรียบเทียบว่า...เท่ากับ... (ได้ True/False)	`signal == "LONG"`
`!=`	ไม่เท่ากับ...ใช่ไหม	`position != 0`
`>` , `<`	มากกว่า/น้อยกว่า...ไหม	`z > 2.0`
`>=` , `<=`	อย่างน้อย / ไม่เกิน...ไหม	`drawdown <= -0.1`

7.2 if / elif / else — เครื่องจักรถามคำถามตามลำดับ

if/elif/else คือเครื่องจักรที่**ถามคำถามจากบนลงล่างทีละข้อ** — เจอข้อไหนตอบ "ใช่" ทำงานของข้อนั้นแล้ว**ออกจากเครื่องทันที** ข้อที่เหลือไม่ถูกถามอีกเลย

```
z = 2.3 if z > 2.0: # ถามข้อ 1: z มากกว่า 2.0 ไหม? signal = "SHORT" # ใช่ → ทำบรรทัดนี้ แล้วออกเลย ไม่ถามต่อ elif z < -2.0: # ถามข้อ 2 (เฉพาะเมื่อข้อ 1 ตอบ "ไม่") signal = "LONG" # ใช่ → ทำบรรทัดนี้ แล้วออก elif abs(z) < 0.5: # ถามข้อ 3 (เฉพาะเมื่อข้อ 1 และ 2 ตอบ "ไม่") signal = "EXIT" # ใช่ → ทำบรรทัดนี้ แล้วออก else: # ทุกข้อข้างบนตอบ "ไม่" ทั้งหมด signal = "HOLD" # ทำบรรทัดนี้ (else ไม่มีคำถาม = "ที่เหลือทั้งหมด") print(signal)
```

```
▶ OUTPUT SHORT
```

🗣 **อ่านว่า:** ถ้า z มากกว่า 2.0 แล้วเอา "SHORT" ใส่ใน signal — ไม่งั้นถ้า z น้อยกว่า -2.0 แล้วเอา "LONG" ใส่ใน signal — ไม่งั้นถ้าขนาดของ z น้อยกว่า 0.5 แล้วเอา "EXIT" ใส่ — ไม่งั้น (ที่เหลือทั้งหมด) เอา "HOLD" ใส่

เส้นทางของ z = 2.3 ในเครื่องจักรนี้:

ข้อ 1: `z > 2.0` ? (2.3 > 2.0)

│ ตอบ "ใช่" ↓

`signal = "SHORT"` **→ ออกจากเครื่องทันที**

ข้อ 2, ข้อ 3, else **ไม่ถูกถามเลย** แม้ว่า abs(2.3) < 0.5 จะเป็น False อยู่แล้วก็ตาม

⚠ elif ไม่เท่ากับ if สองอัน — ตัวอย่างที่ผลต่างกันจริง

```
z = 3.0 # —— แบบ elif: ถามต่อเมื่อข้อบนตอบ "ไม่" —— if z > 2.0: action = "SHORT" # เข้าข้อนี้ แล้วออกเลย elif z > 1.0: action = "WARN" # ไม่ถูกถาม # ผล: action = "SHORT" ✓ # —— แบบ if สองอัน: ถามทุกข้อเสมอ —— if z > 2.0: action = "SHORT" # เข้าข้อนี้ if z > 1.0: action = "WARN" # ถูกถามอีก! 3.0 > 1.0 จริง → ทับค่าเดิม # ผล: action = "WARN"    SHORT หายไปเฉย ๆ
```

ถ้าเงื่อนไขเป็น "ทางเลือกที่เกิดได้ทีละอย่าง" (signal มีได้ค่าเดียว) ต้องใช้ `elif` เสมอ — bug ชนิดนี้ Python ไม่ฟ้อง error โปรแกรมรันได้ปกติแต่ให้ signal ผิด ซึ่งใน trading คือเสียเงินจริง

ลำดับเงื่อนไขสำคัญ — เช็คข้อแคบก่อนข้อกว้าง

```
#    ผิด: ข้อกว้าง (z > 1) อยู่บน → "กิน" ทุกกรณี z > 2 ไปด้วย if z > 1.0: level = "WARN" # z = 3.0 ก็เข้าข้อนี้! ไม่มีวันถึง EXTREME elif z > 2.0: level = "EXTREME" # โค้ดตายซาก — ไม่มีทางถูกเรียก # ✓ ถูก: ข้อแคบ/รุนแรงกว่า ขึ้นก่อน if z > 2.0: level = "EXTREME" elif z > 1.0: level = "WARN"
```

เชื่อมหลายเงื่อนไข: and / or / not

ตัวเชื่อม	อ่านว่า	ผลเป็น **True** เมื่อ
`and`	"และ"	จริง**ทั้งคู่**
`or`	"หรือ"	จริง**อย่างน้อยหนึ่งข้อ**
`not`	"ไม่/กลับด้าน"	ของเดิมเป็น False

```
# เข้า trade เมื่อ: z สูงพอ "และ" ยังไม่มี position if z > 2.0 and position == 0: position = -1 # ออกจาก trade เมื่อ: z กลับใกล้ 0 "หรือ" ขาดทุนเกิน limit if abs(z) < 0.5 or loss > max_loss: position = 0
```

🗣 **อ่านว่า:** ถ้า z มากกว่า 2.0 **และ** position เท่ากับ 0 (ทั้งสองข้อต้องจริง) แล้วเอา -1 ใส่ใน position

⚠ **กับดักการเขียนเงื่อนไข 2 จุด**

```
# 1) เงื่อนไขช่วง — ต้องเขียนตัวแปรซ้ำทั้งสองฝั่ง if z > 1.0 and < 3.0: # SyntaxError if z > 1.0 and z < 3.0: # ✓ if 1.0 < z < 3.0: # ✓ Python ยอมให้เขียนแบบคณิตได้ด้วย # 2) indent (ย่อหน้า) กำหนดว่าบรรทัดไหนอยู่ "ใน" if if z > 2.0: signal = -1 log_trade() # อยู่ใน if — ทำเฉพาะเมื่อ z > 2.0 if z > 2.0: signal = -1 log_trade() # อยู่นอก if — ทำทุกครั้ง! ต่างแค่ย่อหน้าเดียว
```

7.3 for loop — วนซ้ำ

for loop คือคำสั่ง "หยิบของจากรายการมาทีละตัว แล้วทำชุดคำสั่งเดิมซ้ำกับทุกตัว"

```
prices = [63000, 65000, 64500] for p in prices: # หยิบสมาชิกจาก prices มาใส่ p ทีละตัว print(p) # ทำบรรทัดนี้กับ p ทุกรอบ
```

🗣 **อ่านว่า:** สำหรับแต่ละตัวใน prices — เอาตัวนั้นมาใส่ใน p แล้วทำต่อไปนี้: print ค่า p

ดูการทำงาน**ทีละรอบ** — เทคนิคสำคัญ: เวลางง loop ให้ "**เดินเครื่องด้วยมือ**" ตามตาราง — จดทั้ง *ค่าตัวแปร* loop (i) และ state *ของ* list *ที่กำลังสะสม* (returns):

รอบที่	i	prices[i]	returns (list สะสม)
1	1	65000	[0.0317]
2	2	64500	[0.0317, -0.0077]
3	3	66000	[0.0317, -0.0077, 0.0233]
4	4	65800	[0.0317, -0.0077, 0.0233, -0.0030] → loop จบ

ตัวอย่างจริง: คำนวณ return รายวัน — ต้องใช้ "ราคาวันนี้" คู่กับ "ราคาเมื่อวาน" จึง loop ด้วยตำแหน่ง (index) แทน:

```
prices = [63000, 65000, 64500, 66000, 65800] returns = [] # เตรียม list เปล่าไว้เก็บผล for i in range(1, len(prices)): # i ไล่จาก 1, 2, 3, 4 (เริ่มวันที่สอง) today = prices[i] # หยิบราคาวันนี้ yesterday = prices[i-1] # หยิบราคาเมื่อวาน (ตัวก่อนหน้า) r = (today - yesterday) / yesterday # สูตร simple return returns.append(r) # เก็บผลต่อท้าย list print(returns) # [0.0317, -0.0077, 0.0233, -0.0030]
```

```
# enumerate: ได้ทั้งตำแหน่งและค่า พร้อมกัน for i, price in enumerate(prices): print(f"วันที่ {i+1}: ${price:,}") # zip: loop สองรายการพร้อมกัน (จับคู่ตัวต่อตัว) symbols
```

```
= ["BTC", "ETH", "SOL"] weights = [0.5, 0.3, 0.2] for sym, w in
zip(symbols, weights): print(f"{sym}: {w:.0%}")
```

while loop — วนจนกว่าเงื่อนไขจะเป็นเท็จ
for loop กับ while loop แตกต่างกันที่ **"รู้ล่วงหน้าไหมว่าจะวนกี่รอบ"**:

	for loop	while loop
ใช้เมื่อ	รู้จำนวนรอบล่วงหน้า	ไม่รู้จำนวนรอบ — วนจนเงื่อนไขเปลี่ยน
หยุดเมื่อ	หมด list / range	เงื่อนไขกลายเป็น False
ตัวอย่าง Quant	คำนวณ return 252 วัน (รู้ว่า 252 วัน)	เทรดต่อจนทุนหมด (ไม่รู้ว่ากี่วัน)

```
equity = 100000 # เงินเริ่มต้น 100,000 บาท max_loss = 90000 # ถ้าเงินเหลือน้อยกว่า 90,000
หยุดเทรด daily_return = 0.005 # สมมติกำไร/ขาดทุนวันละ 0.5% (จริง ๆ จะได้จาก strategy)
while equity > max_loss: # ก่อนทุกรอบ ถาม: equity ยังมากกว่า max_loss ไหม? profit =
equity * daily_return # คำนวณกำไร/ขาดทุนวันนี้ (ใช้ค่าสุ่มแทนจริง ๆ) equity = equity +
profit # อัปเดตเงินทุน # ไม่ → ออกจาก loop (หยุดเทรด) print("หยุดเทรด — ขาดทุนถึง limit")
```

🗣อ่านว่า: ตราบใดที่ equity ยังมากกว่า max_loss ก็ทำต่อไปนี้: เทรดหนึ่งวันแล้วอัปเดต equity — เครื่องจะกลับขึ้นไป**ถามคำถามเดิมซ้ำ**ก่อนเริ่มรอบใหม่ทุกครั้ง

⚠ Infinite loop — while ที่ไม่มีวันจบ

ถ้าในตัว loop **ไม่มีอะไรทำให้เงื่อนไขเปลี่ยนเป็น False ได้เลย** โปรแกรมจะวนตลอดกาล (ค้าง) เช่นลืมอัปเดต `equity` ในตัวอย่างบน — เช็คเสมอว่า "ตัวแปรในเงื่อนไขถูกเปลี่ยนค่าข้างใน loop หรือยัง" ถ้าโปรแกรมค้าง กด Ctrl+C เพื่อหยุด

7.4 List Comprehension — Loop ในบรรทัดเดียว (ฝึกอ่านโค้ดแบบ AI ครั้งแรก)

ทุกอย่างที่ทำได้ด้วย loop ธรรมดา เขียนย่อในบรรทัดเดียวได้ด้วย "list comprehension" — **คุณไม่จำเป็นต้องเขียนเป็น แต่ต้องอ่านออก** เพราะ AI เขียนแบบนี้ตลอด

```
prices = [63000, 65000, 64500, 66000, 65800] # แบบที่เราเขียน (อ่านออก 100%)
returns = [] for i in range(1, len(prices)): today = prices[i] yesterday =
prices[i-1] returns.append((today - yesterday) / yesterday)
```

AI เขียนแบบนี้ — ฝึกอ่านกัน

```
returns = [(prices[i]-prices[i-1])/prices[i-1] for i in range(1, len(pr
```

สูตรอ่าน list comprehension: **อ่านครึ่งหลังก่อน แล้วย้อนกลับมาครึ่งแรก**

1. `for i in range(1, len(prices))` — "สำหรับแต่ละ i ตั้งแต่ 1 ถึงท้ายรายการ" (นี่คือ loop เดิมนั่นเอง)
2. `(prices[i]-prices[i-1])/prices[i-1]` — "คำนวณค่านี้ในแต่ละรอบ" (คือ 3 บรรทัดในตัว loop ยุบเหลือนิพจน์เดียว)
3. `[ ... ]` ครอบทั้งหมด — "เก็บผลทุกรอบลง list" (คือ `returns = []` + `.append()`)

ทั้งบรรทัดอ่านว่า: *"สร้าง* list *ของ* return *โดยคำนวณจากราคาคู่วันติดกัน ทุกวันตั้งแต่วันที่สอง"* — ความหมายเหมือนโค้ด 5 บรรทัดข้างบนเป๊ะ แค่ย่อ

```
# comprehension + if = กรองของ loss_days = [r for r in returns if r < 0]
print(f"วันขาดทุน: {len(loss_days)} วัน")
```

อ่านว่า: สร้าง list จาก r แต่ละตัวใน returns **เฉพาะตัวที่** r น้อยกว่า 0 — แล้วเอามาใส่ใน loss_days

เมื่อ AI ใช้ if ใน comprehension + ternary — 2 รูปแบบที่ต่างกัน

```
# รูปแบบที่ 1: กรอง (filter) — if อยู่หลัง for
loss_days = [r for r in returns if r < 0]
# รูปแบบที่ 2: เลือกค่า (ternary) — if อยู่หน้า for
labels = ["loss" if r < 0 else "gain" for r in returns]
```

ทั้งสองรูปแบบมี `if` แต่**ตำแหน่งต่างกัน → ความหมายต่างกันสิ้นเชิง**:

1. **`if หลัง for`** = กรองของ: "เอาเฉพาะ r ที่ r < 0" → list สั้นลง
2. **`if...else หน้า for`** = เลือกค่า: "ถ้า r < 0 ให้เป็น 'loss' ไม่งั้น 'gain'" → list ยาวเท่าเดิม แค่เปลี่ยนค่า

จำง่าย: *ก่อน* for = *เลือกว่าแต่ละตัวจะเป็นอะไร, หลัง* for = *กรองว่าตัวไหนเข้าหรือออก*

7.5 Functions — def

```
# นิยาม function def calc_return(price_now, price_prev): """คำนวณ simple
return ระหว่างสองราคา""" return (price_now - price_prev) / price_prev # เรียกใช้ r
```

```
= calc_return(65000, 63000) print(f"Return: {r:.2%}") # Return: 3.17% # Default arguments def annualize(daily_vol, periods=252): """แปลง daily volatility เป็น annualized""" return daily_vol * (periods ** 0.5) print(annualize(0.03)) # 0.4762 (252 วัน = หุ้น) print(annualize(0.03, 365)) # 0.5733 (365 วัน = crypto)
```

Docstring — เขียนทุก function ใน Quant code

บรรทัดแรกใน function (ในเครื่องหมาย `"""`) คือ docstring อธิบาย function นั้น ช่วยเมื่อ copy code ไปให้ AI ตรวจ — AI จะเข้าใจ intent ได้ถูกต้อง และช่วย debug ได้แม่นยำขึ้น

ตัวอย่าง: Z-score Function แบบ Reusable

```
def zscore(series, window=60): """ คำนวณ rolling z-score ของ series Args: series: list หรือ numpy array ของราคา/spread window: rolling window (default 60 วัน) Returns: list ของ z-score """ results = [] # เตรียม list เก็บผลลัพธ์ for i in range(len(series)): # เดินผ่านข้อมูลทีละวัน if i < window: # วันแรก ๆ ข้อมูลย้อนหลังยังไม่ครบ window results.append(None) # → ใส่ None แทน (คำนวณไม่ได้) continue # ข้ามไปวันถัดไปเลย window_data = series[i-window:i] # หยิบข้อมูล window วันล่าสุด mean = sum(window_data) / window # ค่าเฉลี่ยของหน้าต่างนั้น variance = sum((x - mean)**2 for x in window_data) / window # ความแปรปรวน std = variance ** 0.5 # ส่วนเบี่ยงเบนมาตรฐาน = รากของ variance if std == 0: # กันหารด้วยศูนย์ (ราคานิ่งสนิท) results.append(0) else: z = (series[i] - mean) / std # สูตร z-score: ห่างจากค่าเฉลี่ยกี่ std results.append(z) return results # ทดสอบ spread = [100, 102, 98, 105, 99, 103, 110, 97, 101, 104] z = zscore(spread, window=5) print(z[-3:]) # z-score 3 วันล่าสุด
```

AI เขียนฟังก์ชันเดียวกันนี้ใน 1 บรรทัด — ฝึกอ่านกัน

```
z = (s - s.rolling(60).mean()) / s.rolling(60).std()
```

(`s` คือ pandas Series — จะเรียนบทที่ 9) อ่านทีละชิ้นจากในออกนอก:

1. `s.rolling(60)` — "หน้าต่างเลื่อน 60 วัน **ของ** s" (จุด = "ของ")
2. `s.rolling(60).mean()` — "ค่าเฉลี่ย **ของ** หน้าต่าง 60 วัน" = ตัวแปร `mean` ในฟังก์ชันเรา
3. `s.rolling(60).std()` — "std ของหน้าต่าง 60 วัน" = ตัวแปร `std` ของเรา
4. `(s - mean) / std` — สูตร z-score เดิมเป๊ะ แค่ทำกับ**ทุกวันพร้อมกัน** (ไม่ต้อง loop เอง)

โค้ด 15 บรรทัดของเรากับบรรทัดเดียวนี้ *ทำสิ่งเดียวกัน* — pandas ซ่อน loop ไว้ข้างใน (และเร็วกว่า ~100 เท่า) เราเขียนแบบยาวก่อนเพื่อให้รู้ว่า "ข้างในมันทำอะไร" พอไปบทที่ 9 จะใช้แบบสั้นได้อย่างมั่นใจ

7.6 lambda — Function ย่อ

```
# lambda: function 1 บรรทัด ไม่มีชื่อ calc_ret = lambda p_now, p_prev: (p_now - p_prev) / p_prev # ใช้ป่อยกับ sorted() และ map() trades = [("BTC", 500), ("ETH", -200), ("SOL", 300)] by_pnl = sorted(trades, key=lambda t: t[1], reverse=True) print(by_pnl) # [('BTC', 500), ('SOL', 300), ('ETH', -200)]
```

▶ แบบฝึกหัด: เขียน function คำนวณ Sharpe Ratio

```
def sharpe_ratio(returns, risk_free=0.0, periods=252): """ คำนวณ Sharpe Ratio รายปี returns: list ของ daily return (float) risk_free: annual risk-free rate (default 0) periods: วันซื้อขายต่อปี """ n = len(returns) mean_r = sum(returns) / n variance = sum((r - mean_r)**2 for r in returns) / n daily_std = variance ** 0.5 daily_rf = risk_free / periods sharpe = (mean_r - daily_rf) / daily_std * (periods ** 0.5) return sharpe # ทดสอบ daily_returns = [0.001, -0.005, 0.008, 0.002, -0.003] s = sharpe_ratio(daily_returns) print(f"Sharpe: {s:.2f}")
```

บทที่ 8: **Data Structures**

แก่นของบทนี้

Python มี 4 data structure หลัก — **list** (รายการเรียงลำดับ), **dict** (key-value), **tuple** (ค่าคงที่), **set** (ค่าไม่ซ้ำ) แต่ละอย่างมีจุดแข็งต่างกัน รู้จักเลือกใช้ถูกตัว code จะสั้นและเร็วขึ้นมาก

8.1 list — รายการที่เรียงลำดับ

```
prices = [63000, 65000, 64500, 66000, 65800] # Indexing (เริ่มจาก 0) print(prices[0]) # 63000 (ตัวแรก) print(prices[-1]) # 65800 (ตัวสุดท้าย — เลขติดลบ = นับจากท้าย) # Slicing [start:stop:step] print(prices[1:4]) # [65000, 64500, 66000] print(prices[:3]) # [63000, 65000, 64500] (3 ตัวแรก) print(prices[::2]) # [63000, 64500, 65800] (ทุก 2 ตัว) # เพิ่ม/ลบ prices.append(67000) # เพิ่มต่อท้าย prices.insert(0, 62000) # แทรกที่ตำแหน่ง 0 last = prices.pop() # ดึงตัวสุดท้ายออก # ข้อมูลพื้นฐาน print(len(prices)) # จำนวนสมาชิก print(min(prices), max(prices)) # ต่ำสุด สูงสุด print(sum(prices) / len(prices)) # ค่าเฉลี่ย
```

8.2 dict — Key-Value Store

dict คือโครงสร้างที่ใช้มากที่สุดในการเก็บข้อมูลตลาด — key คือชื่อ/symbol, value คือข้อมูล

```
# OHLCV ของวันหนึ่ง candle = { "open": 64000.0, "high": 66500.0, "low": 63800.0, "close": 65450.0, "volume": 28500.0 } # เข้าถึงค่า print(candle["close"]) # 65450.0 print(candle.get("vwap", None)) # None (ไม่มี key นั้น — ไม่ error) # เพิ่ม/อัปเดต candle["body"] = candle["close"] - candle["open"] # = 1450.0 # loop ผ่าน dict for key, value in candle.items(): print(f"{key:10s}: {value:,.1f}") # Portfolio เป็น dict ของ dict portfolio = { "BTC": {"size": 0.5, "entry": 63000, "side": "long"}, "ETH": {"size": 2.0, "entry": 3200, "side": "long"}, "SOL": {"size": 10.0, "entry": 145, "side": "short"}, }
```

8.3 tuple — ค่าคงที่

```
# tuple เหมือน list แต่ immutable (เปลี่ยนไม่ได้) config = (2.0, 0.5, 60) # (entry_z, exit_z, window) entry_z, exit_z, window = config # unpacking # ใช้เป็น key ของ dict ได้ (list ทำไม่ได้) pair_params = { ("BTC", "ETH"): {"beta": 0.85,
```

```
"half_life": 4.2}, ("BTC", "SOL"): {"beta": 1.20, "half_life": 6.8}, }
print(pair_params[("BTC", "ETH")]["beta"]) # 0.85
```

8.4 set — ค่าไม่ซ้ำ

```
# set เก็บเฉพาะค่าที่ไม่ซ้ำ signals_day1 = {"AAPL", "MSFT", "NVDA"} signals_day2 =
{"MSFT", "NVDA", "TSLA"} both_days = signals_day1 & signals_day2 #
intersection = {"MSFT", "NVDA"} either_day = signals_day1 | signals_day2 #
union only_day1 = signals_day1 - signals_day2 # difference = {"AAPL"} # ลบ
duplicate ออกจาก list trades = ["BTC", "ETH", "BTC", "SOL", "ETH", "BTC"]
unique_symbols = list(set(trades)) # ["BTC", "ETH", "SOL"]
```

เลือก **Data Structure** ไหน?

ถ้าต้องการ	ใช้	เหตุผล
เก็บ series ราคาเรียงเวลา	`list`	เรียงลำดับ, append ง่าย
เก็บ OHLCV ของ 1 candle	`dict`	access ด้วยชื่อ field ได้
ค่า config ที่ไม่เปลี่ยน	`tuple`	immutable, ใช้เป็น dict key ได้
รายการ symbol ที่เคยเทรด	`set`	ลบ duplicate อัตโนมัติ
ข้อมูลตลาดหลายวัน	`pd.DataFrame`	Part I บทที่ 9

ตัวอย่าง: สร้าง **Return Map** ของหลายเหรียญ

```
close_prices = { "BTC": [63000, 65000, 64500], "ETH": [3100, 3200,
3150], "SOL": [140, 148, 145], } # แบบที่เราเขียน: loop ผ่านทุกเหรียญ คำนวณ
return ล่าสุด latest_returns = {} # เตรียม dict เปล่า for sym, prices in
close_prices.items(): # หยิบ (ชื่อเหรียญ, list ราคา) ทีละคู่ last = prices[-1]
# ราคาล่าสุด (ตัวสุดท้าย) prev = prices[-2] # ราคาก่อนหน้า (ตัวรองสุดท้าย)
latest_returns[sym] = (last - prev) / prev # เก็บ return โดยใช้ชื่อเหรียญเป็น
key for sym, r in latest_returns.items(): print(f"{sym}: {r:.2%}")
```

```
▶ OUTPUT BTC: -0.77% ETH: -1.56% SOL: -2.03%
```

AI เขียนแบบนี้ — dict comprehension

```
latest_returns = {
    sym: (prices[-1] - prices[-2]) / prices[-2]
    for sym, prices in close_prices.items()
}
```

สูตรเดียวกับ list comprehension: **อ่านบรรทัด for ก่อน แล้วย้อนขึ้นไปดูว่าแต่ละรอบสร้างอะไร**

1. `for sym, prices in close_prices.items()` — loop เดิม: หยิบ (ชื่อเหรียญ, ราคา) ทีละคู่
2. `sym: (prices[-1] - prices[-2]) / prices[-2]` — แต่ละรอบสร้างคู่ `key: value` = ชื่อเหรียญ : return ล่าสุด
3. `{ ... }` ปีกกาครอบ — เก็บทุกคู่ลง dict (ปีกกา = dict, วงเล็บเหลี่ยม = list)

ความหมายเหมือนโค้ด 5 บรรทัดข้างบนทุกประการ

▶ แบบฝึกหัด: สร้าง **Portfolio Summary**

```
positions = { "BTC": {"entry": 60000, "current": 65000, "size": 0.5},
"ETH": {"entry": 3000, "current": 3200, "size": 2.0}, } # คำนวณ P&L
แต่ละ position for sym, pos in positions.items(): pnl = (pos["current"]
- pos["entry"]) * pos["size"] ret = (pos["current"] - pos["entry"]) /
pos["entry"] print(f"{sym}: P&L=${pnl:,.0f} ({ret:.1%})")
```

```
▶ OUTPUT BTC: P&L=$2,500 (8.3%) ETH: P&L=$400 (6.7%)
```

บทที่ 9: **NumPy & Pandas**

แก่นของบทนี้

NumPy และ Pandas เป็น "engine" ของ Quant Python — NumPy ทำ math บน array เร็วกว่า loop ธรรมดา 100x+, Pandas จัดการ table ข้อมูลพร้อม label วันที่ ทั้งสองรวมกันทำให้วิเคราะห์ข้อมูลตลาดหลักหมื่นแถวได้ใน milliseconds

9.1 NumPy — คณิตศาสตร์บน Array

```
import numpy as np # สร้าง array prices = np.array([63000, 65000, 64500, 66000, 65800]) # Operations ทำทั้ง array พร้อมกัน (vectorized) returns = np.diff(prices) / prices[:-1] # return ทุกวัน print(returns) # [0.0317 -0.0077 0.0233 -0.0030] # Statistics print(f"Mean: {np.mean(returns):.4f}") print(f"Std: {np.std(returns):.4f}") print(f"Max: {np.max(returns):.4f}") print(f"Min: {np.min(returns):.4f}") # Annualized Vol annual_vol = np.std(returns) * np.sqrt(252) print(f"Annual Vol: {annual_vol:.1%}")
```

ทำไม NumPy เร็วกว่า Python loop?

Python loop: อ่านแต่ละ element → แปลง type → คำนวณ → เก็บ → ไปต่อ (overhead สูง)
NumPy: ส่ง array ทั้งก้อนให้ C code → คำนวณ parallel → ได้ผล (overhead ต่ำมาก)
ผลลัพธ์: array ขนาด 1,000,000 element → NumPy เร็วกว่า loop ประมาณ 100–1,000x

9.2 ทำไม "ตารางธรรมดา" ถึงไม่พอ

ก่อนตอบว่า DataFrame คืออะไร ลองดูก่อนว่า**ถ้าไม่มีมัน** ชีวิตเป็นยังไง — เก็บราคา 2 เหรียญด้วยเครื่องมือที่เรามีจากบทที่ 8 (list ซ้อน list = "ตารางทำมือ"):

```
# ตารางธรรมดา: list ซ้อน list table = [ # [วันที่, BTC, ETH] ["2024-01-01", 63000, 3100], ["2024-01-02", 65000, 3250], ["2024-01-03", 64500, 3200], ] # คำถามง่าย ๆ: "ราคา BTC วันที่ 2024-01-02 คือเท่าไหร่?" # → ต้องเขียน loop ค้นเอง: for row in table:
```

```
if row[0] == "2024-01-02": # ช่องที่ 0 คือวันที่ (ต้องจำเอง) print(row[1]) # ช่องที่ 1 คือ BTC (ต้องจำเอง)
```

ใช้งานได้ แต่มีปัญหา 3 ข้อที่จะกัดเราแน่นอนเมื่อข้อมูลโตขึ้น:

> **ปัญหาของตารางทำมือ**
>
> 1. **ต้องจำตำแหน่งเอง** — "ช่องที่ 1 คือ BTC" ไม่มีเขียนไว้ในโค้ด ถ้าสลับคอลัมน์ในไฟล์ข้อมูลวันไหน โค้ดจะหยิบเลขผิดช่อง**โดยไม่มี error**
> 2. **ค้นหาต้อง loop ทุกครั้ง** — ข้อมูล 100,000 แถว อยากรู้ราคาวันเดียว ต้องไล่ดูแสนแถว
> 3. **ข้อมูลหายแล้วแถวเลื่อน** — สมมติ ETH ไม่มีข้อมูลวันที่ 2 ถ้าเก็บแยกสอง list แล้วเอามาคู่กัน ตำแหน่งที่ 1 ของ BTC จะจับคู่กับ ETH ผิดวัน → คำนวณ correlation ผิดทั้งชุด **แบบเงียบ ๆ**

pandas ถูกสร้างมาแก้ 3 ปัญหานี้ตรง ๆ — และกุญแจของมันคือสิ่งที่เรียกว่า **index**

9.3 Index คืออะไร — ป้ายชื่อประจำแถว

list ธรรมดาเรียกข้อมูลด้วย "ตำแหน่ง": ตัวที่ 0, ตัวที่ 1, ตัวที่ 2 — เหมือนล็อกเกอร์ที่มีแต่เลขตู้
Series ของ pandas คือ list ที่**ทุกค่ามีป้ายชื่อติดอยู่** — ป้ายชื่อเหล่านั้นรวมกันเรียกว่า index:

```
import pandas as pd prices = pd.Series( [63000, 65000, 64500], # ค่า (values) index=["2024-01-01", "2024-01-02", "2024-01-03"], # ป้ายชื่อ (index) name="BTC" ) print(prices)
```

```
▶ OUTPUT 2024-01-01 63000 2024-01-02 65000 2024-01-03 64500 Name: BTC, dtype: int64
```

สังเกตว่า pandas พิมพ์ป้ายชื่อ (ซ้าย) คู่กับค่า (ขวา) เสมอ — ทีนี้คำถาม "ราคาวันที่ 2 ม.ค.?" ตอบได้บรรทัดเดียว ไม่ต้อง loop:

```
print(prices["2024-01-02"]) # 65000 — ถามด้วยป้ายชื่อตรง ๆ print(prices.iloc[1]) # 65000 — หรือถามด้วยตำแหน่งแบบเดิมก็ยังได้ (iloc)
```

> 🗣 **อ่านว่า:** หยิบค่าที่ป้ายชื่อ "2024-01-02" จาก prices — index ทำให้เราคุยกับข้อมูลด้วย "ภาษาของข้อมูล" (วันที่) แทนตำแหน่ง

และพลังที่สำคัญที่สุดของ index — **จับคู่แถวให้อัตโนมัติ (alignment)** ซึ่งแก้ปัญหาข้อ 3 แบบถาวร:

```
# ETH ไม่มีข้อมูลวันที่ 2 (ตลาดล่ม) btc = pd.Series([63000, 65000, 64500], index=["2024-01-01", "2024-01-02", "2024-01-03"]) eth = pd.Series([3100, 3200], index=["2024-01-01", "2024-01-03"]) # ขาดวันที่ 2! ratio = btc / eth # pandas จับคู่ด้วย "ป้ายชื่อ" ไม่ใช่ตำแหน่ง print(ratio)
```

```
▶ OUTPUT 2024-01-01 20.322581 2024-01-02 NaN 2024-01-03 20.156250
```

วันที่ 2 ได้ `NaN` (Not a Number = "ไม่มีข้อมูล") — pandas **บอกตรง ๆ ว่าวันนี้คำนวณไม่ได้** แทนที่จะจับคู่ผิดแถวเงียบ ๆ แบบ list ถ้าเป็นตารางทำมือ ราคา ETH วันที่ 3 จะถูกจับคู่กับ BTC วันที่ 2 — ผิดโดยไม่มีใครรู้

9.4 DataFrame — หลาย Series ที่ใช้ป้ายชื่อชุดเดียวกัน

พอเข้าใจ Series แล้ว DataFrame ตรงไปตรงมามาก: **DataFrame = ตารางที่เกิดจากหลาย Series (หลายคอลัมน์) มาแชร์ index ชุดเดียวกัน** — หน้าตาเหมือน Excel sheet แต่อยู่ใน Python:

```
data = { "BTC": [63000, 65000, 64500, 66000, 65800], "ETH": [3100, 3250, 3200, 3300, 3280], "SOL": [140, 148, 145, 152, 149], } dates = pd.date_range("2024-01-01", periods=5) # สร้างป้ายวันที่ 5 วัน df = pd.DataFrame(data, index=dates) print(df)
```

```
▶ OUTPUT BTC ETH SOL 2024-01-01 63000 3100 140 2024-01-02 65000 3250 148 2024-01-03 64500 3200 145 2024-01-04 66000 3300 152 2024-01-05 65800 3280 149
```

กายวิภาคของ DataFrame มี 3 ส่วน — รู้ 3 ชื่อนี้แล้วอ่านเอกสาร pandas ออกทั้งหมด:

ส่วน	คืออะไร	ในตัวอย่างบน
index	ป้ายชื่อแถว (แนวตั้งซ้ายสุด)	วันที่ 2024-01-01 ถึง 05
columns	ป้ายชื่อคอลัมน์ (แนวนอนบนสุด)	BTC, ETH, SOL
values	ตัวข้อมูลข้างใน (NumPy array)	ตัวเลขราคาทั้งหมด

> **แล้วใช้ Excel แทนได้ไหม?**
>
> ได้ — สำหรับ "ดูข้อมูล" Excel ดีมาก แต่สำหรับงาน Quant มี 3 จุดที่ DataFrame ชนะขาด:
> 1. **ทำซ้ำได้:** โค้ด 10 บรรทัดรันกับข้อมูลใหม่ทุกวันอัตโนมัติ — Excel ต้องลากสูตรใหม่ทุกครั้ง
> 2. **ขนาด:** ข้อมูล 1 นาทีย้อนหลัง 3 ปี ≈ 1.5 ล้านแถว — Excel เริ่มค้าง pandas ทำงานเป็นวินาที
> 3. **ตรวจสอบย้อนหลังได้:** ทุกขั้นตอนคำนวณเขียนเป็นโค้ดอ่านได้ ส่งให้คนอื่น (หรือ AI) ตรวจได้ — สูตรที่ฝังใน cell ของ Excel ตรวจยากมาก
>
> คิดง่าย ๆ: **Excel = ทำกับข้าวด้วยมือ, DataFrame = โรงงานอาหาร** — ทำกินเองมื้อเดียวใช้มือได้ แต่ถ้าต้องทำทุกวัน วันละหมื่นจาน ต้องใช้โรงงาน

"แล้วมันทำอะไรได้บ้าง?" — ตัวอย่างงานที่ใช้บ่อยที่สุดใน Quant แต่ละงานใช้โค้ด**บรรทัดเดียว** (ทุกอันได้ผลถูกต้องแม้ข้อมูลขาดบางวัน เพราะ index จับคู่ให้):

อยากได้	โค้ด
return รายวันของทุกเหรียญ	`df.pct_change()`
ค่าเฉลี่ยเคลื่อนที่ 20 วันของ BTC	`df["BTC"].rolling(20).mean()`
เฉพาะวันที่ BTC ตกเกิน 2%	`df[df["BTC"].pct_change() < -0.02]`
correlation ระหว่างทุกคู่เหรียญ	`df.pct_change().corr()`
กราฟราคาทุกเหรียญ	`df.plot()`

9.5 Operations ที่ใช้บ่อยใน Quant

```
# --- ดูข้อมูลเบื้องต้น --- df.head(3) # 3 แถวแรก df.tail(3) # 3 แถวสุดท้าย
df.describe() # mean, std, min, max, quartile df.shape # (5, 3) = 5 แถว 3
คอลัมน์ df.dtypes # type ของแต่ละคอลัมน์ # --- เลือกข้อมูล --- df["BTC"] # คอลัมน์ BTC
(Series) df[["BTC","ETH"]] # สองคอลัมน์ (DataFrame) df.loc["2024-01-03"]
# .loc: เลือกด้วย "ป้ายชื่อ" (label) = วันที่ df.iloc[0] # .iloc: เลือกด้วย "ตำแหน่ง"
(position) = แถวที่ 0 คือแถวแรก df.iloc[-1] # .iloc[-1] = แถวสุดท้าย (เลขลบนับจากท้าย
เหมือน list) # --- คำนวณ Return --- returns = df.pct_change() # daily return
ทุก column (แถวแรกเป็น NaN เพราะไม่มีวันก่อน) log_returns = np.log(df/df.shift(1)) #
log return: shift(1) เลื่อนข้อมูลลงหนึ่งแถว (= ราคาเมื่อวาน) # --- Rolling Statistics
--- rolling_mean = df["BTC"].rolling(window=3).mean() # ค่าเฉลี่ย 3 วันล่าสุด (2
```

```
แถวแรกเป็น NaN) rolling_std = df["BTC"].rolling(window=3).std() # std 3 วันล่าสุด
zscore = (df["BTC"] - rolling_mean) / rolling_std # z-score: ห่างจากค่าเฉลี่ยกี่ std
```

.loc vs .iloc — เลือกตัวไหน?

	`.loc["2024-01-03"]`	`.iloc[2]`
เลือกด้วย	ป้ายชื่อ (วันที่, ชื่อหุ้น)	ตำแหน่ง (0, 1, 2, -1)
ข้อดี	ชัดเจน ไม่ผิดแม้เรียงลำดับใหม่	เร็ว ใช้ได้แม้ไม่รู้ชื่อ index
ใช้เมื่อ	รู้วันที่/label ที่ต้องการ	ต้องการ "ตัวแรก" หรือ "ตัวสุดท้าย"

NaN คืออะไร? "Not a Number" = "ไม่มีข้อมูล ณ จุดนี้" — pandas ใส่อัตโนมัติเมื่อคำนวณไม่ได้ (เช่น แถวแรกของ `pct_change()` หรือหน้าต่างแรกของ `rolling()`) — `dropna()` ลบแถวที่มี NaN ออก, `fillna(0)` แทน NaN ด้วยศูนย์ — pandas ส่วนใหญ่ข้าม NaN ให้อัตโนมัติ (`mean()`, `std()`) แต่บางงาน (เช่น `corr()`) ต้องการ `dropna()` ก่อน

```
# --- Filter: boolean indexing — "กรองแถวด้วยเงื่อนไข" --- btc_returns =
df["BTC"].pct_change() # แทนที่จะ loop แล้ว if เองทีละแถว pandas ทำให้ในบรรทัดเดียว:
crash_days = df[btc_returns < -0.02] # เลือกเฉพาะแถวที่ return < -2% print(f"BTC
crash days (<-2%): {len(crash_days)}") # เงื่อนไขสองข้อ: ต้องใช้ & และ ครอบแต่ละเงื่อนไขด้วย
() both_up = df[(btc_returns > 0) & (df["ETH"].pct_change() > 0)]
```

ไฟล์ btc_daily.csv มาจากไหน?

ดาวน์โหลดข้อมูลราคา BTC จาก **CoinGecko** (ฟรี) หรือ **Binance API**:
วิธีง่ายที่สุด: เปิด `https://www.coingecko.com/en/coins/bitcoin/historical_data` → Export CSV → เปลี่ยนชื่อไฟล์เป็น `btc_daily.csv`
หรือใช้ Python: `pip install yfinance` แล้วรัน `yf.download("BTC-USD", start="2023-01-01").to_csv("btc_daily.csv")`

ตัวอย่างด้านล่างใช้โค้ดจำลอง (สร้างข้อมูลด้วย numpy) เพื่อรันได้ทันทีโดยไม่ต้องดาวน์โหลด

Running Example: โหลดข้อมูลจาก CSV และคำนวณ Metrics

```
# สมมติไฟล์ btc_daily.csv มีคอลัมน์ date, open, high, low, close, volume df
= pd.read_csv("btc_daily.csv", index_col="date", parse_dates=True)
df = df.sort_index() # เรียงวันที่จากเก่าไปใหม่ close = df["close"] # หยิบคอลัมน์
```

```
ราคาปิดมาตั้งชื่อสั้น ๆ # 1) return รายวัน df["return"] = close.pct_change() #
เปอร์เซ็นต์เปลี่ยนแปลงจากวันก่อน # 2) log return prev_close = close.shift(1) #
ราคาเมื่อวาน (เลื่อนลง 1 แถว) df["log_ret"] = np.log(close / prev_close) # 3)
annualized volatility จากหน้าต่าง 20 วัน daily_vol_20 =
df["return"].rolling(20).std() # std ของ return 20 วันล่าสุด df["vol_20"]
= daily_vol_20 * np.sqrt(252) # คูณ √252 เพื่อแปลงเป็นรายปี # 4) moving
average 50 วัน df["ma_50"] = close.rolling(50).mean() # 5) z-score:
ราคาวันนี้ห่างค่าเฉลี่ย 60 วัน กี่ std mean_60 = close.rolling(60).mean() # ค่าเฉลี่ย
เคลื่อนที่ 60 วัน std_60 = close.rolling(60).std() # std เคลื่อนที่ 60 วัน
df["zscore"] = (close - mean_60) / std_60 # สูตร z-score เดิมจากบทที่ 7
print(df.tail()) # ดูผลลัพธ์ 5 วันล่าสุด print(f"\nAnnualized Vol (20d):
{df['vol_20'].iloc[-1]:.1%}") print(f"Current Z-score:
{df['zscore'].iloc[-1]:.2f}")
```

AI จะยุบ 5 ขั้นนี้เหลือแบบนี้ — ฝึกอ่าน method chain

```
df["vol_20"] = df["close"].pct_change().rolling(20).std() * np.sqrt(252
```

เทคนิคอ่าน chain: **อ่านจากซ้ายไปขวา เหมือนสายพานโรงงาน** — ข้อมูลผ่านเครื่องจักรทีละตัว จุดคือ "ส่งต่อให้"

1. `df["close"]` — เริ่มจากราคาปิด
2. `.pct_change()` — ส่งเข้าเครื่องคำนวณ return รายวัน
3. `.rolling(20)` — ส่งต่อเข้าหน้าต่างเลื่อน 20 วัน
4. `.std()` — คำนวณ std ของแต่ละหน้าต่าง
5. `* np.sqrt(252)` — สุดท้ายคูณ √252 แปลงเป็นรายปี

คือขั้นที่ 1 + 3 ของเรารวมกันในบรรทัดเดียว — chain สั้น ๆ (2–3 จุด) อ่านสบาย แต่ถ้ายาวกว่านั้นแล้วเริ่มงง **แตกเป็นตัวแปรกลางแบบที่เราเขียนได้เสมอ ผลเหมือนกันเป๊ะ** และ debug ง่ายกว่าเพราะ print ดูค่ากลางทางได้

▶ แบบฝึกหัด: คำนวณ Correlation Matrix

```
# สร้าง DataFrame ของราคา 3 สินทรัพย์ data = { "BTC": [63000, 65000, 64500, 66000, 65800, 67000, 66500], "ETH": [3100, 3250, 3200, 3300, 3280, 3350, 3320], "GOLD":[1900, 1895, 1905, 1910, 1900, 1915, 1908], } df = pd.DataFrame(data) # Correlation matrix ของ returns returns = df.pct_change().dropna() corr = returns.corr() print(corr.round(3))
```

ผลลัพธ์จะแสดง matrix 3×3 — BTC/ETH มักมี correlation สูง, GOLD มัก correlation ต่ำกับ crypto

บทที่ 10: **Visualization**

แก่นของบทนี้

กราฟที่ดีบอกเรื่องราวได้ภายใน 3 วินาที — ใน Quant ใช้กราฟเพื่อ debug signal, ตรวจสอบ backtest, และนำเสนอผลลัพธ์ บทนี้ใช้ matplotlib เป็นหลัก (standard, ทำได้ทุกอย่าง) และแนะนำ plotly สำหรับ interactive charts

10.1 matplotlib พื้นฐาน

```
import matplotlib.pyplot as plt import numpy as np import pandas as pd # สร้างข้อมูลตัวอย่าง dates = pd.date_range("2024-01-01", periods=60) prices = 65000 * np.cumprod(1 + np.random.normal(0.001, 0.02, 60)) # Plot พื้นฐาน fig, ax = plt.subplots(figsize=(10, 4)) ax.plot(dates, prices, color="#0d9488", linewidth=1.5, label="BTC Price") ax.set_title("BTC/USDT Daily Close", fontsize=14, fontweight="bold") ax.set_xlabel("Date") ax.set_ylabel("Price (USD)") ax.legend() ax.grid(True, alpha=0.3) plt.tight_layout() plt.savefig("btc_price.png", dpi=150) plt.show()
```

10.2 Multiple Subplots — Price + Return + Z-score

```
# 3 panels: ราคา, return, z-score fig, axes = plt.subplots(3, 1, figsize=(10, 10), sharex=True) # สมมติ df มีคอลัมน์ close, return, zscore # Panel 1: ราคา + MA50 axes[0].plot(df.index, df["close"], label="Close", color="#0d9488") axes[0].plot(df.index, df["ma_50"], label="MA50", color="#f59e0b", linestyle="--") axes[0].set_ylabel("Price ($)") axes[0].legend() axes[0].set_title("BTC Analysis Dashboard") # Panel 2: Daily Return — เลือกสีแท่งทีละวัน: เขียวถ้าบวก แดงถ้าลบ bar_colors = [] for r in df["return"]: if r > 0: bar_colors.append("#16a34a") # เขียว else: bar_colors.append("#dc2626") # แดง axes[1].bar(df.index, df["return"], color=bar_colors, alpha=0.7, width=0.8) axes[1].axhline(0, color="black", linewidth=0.5) axes[1].set_ylabel("Daily Return") # Panel 3: Z-score + threshold lines axes[2].plot(df.index, df["zscore"], color="#7c3aed", linewidth=1) axes[2].axhline(2.0, color="#dc2626", linestyle="--", alpha=0.7, label="Entry (±2σ)") axes[2].axhline(-2.0, color="#dc2626", linestyle="--", alpha=0.7) axes[2].axhline(0, color="black", linewidth=0.5) axes[2].set_ylabel("Z-score") axes[2].legend() plt.tight_layout() plt.savefig("btc_dashboard.png", dpi=150) plt.show()
```

AI จะยัด loop เลือกสีเข้าไปในวงเล็บเลย — ฝึกอ่าน

```
axes[1].bar(df.index, df["return"],
            color=["#16a34a" if r > 0 else "#dc2626" for r in df["retur
```

ตรง `color=[...]` คือ comprehension ที่มี **เงื่อนไขเลือกค่า** (`A if เงื่อนไข else B`) ฝังอยู่:

1. `for r in df["return"]` — หยิบ return ทีละวัน (loop เดิม)
2. `"#16a34a" if r > 0 else "#dc2626"` — อ่านว่า "เอาสีเขียว **ถ้า** r บวก **ไม่งั้น**เอาสีแดง" (รูปแบบนี้เรียก ternary — ค่าที่ได้อยู่หน้า if)
3. ผลคือ list ของสี ส่งให้ `color=` โดยตรง ไม่ต้องตั้งชื่อตัวแปร

ความหมายเหมือน loop 6 บรรทัดของเรา — ถ้าเจอแบบนี้ในโค้ด AI แล้วงง ให้ลากมันออกมาตั้งเป็นตัวแปรข้างนอกก่อน แล้วค่อยอ่าน

10.3 Scatter Plot — Correlation

```
# สร้าง DataFrame ที่มีราคา BTC และ ETH (จำลองข้อมูล 100 วัน) import numpy as np, pandas
as pd np.random.seed(42) dates = pd.date_range("2024-01-01", periods=100)
df_multi = pd.DataFrame({ "BTC": 65000 * np.cumprod(1 +
np.random.normal(0.001, 0.02, 100)), "ETH": 3200 * np.cumprod(1 +
np.random.normal(0.001, 0.025, 100)), }, index=dates) # วิเคราะห์ correlation
ระหว่าง BTC และ ETH returns btc_ret = df_multi["BTC"].pct_change().dropna() #
return รายวัน ลบแถวแรกที่เป็น NaN eth_ret = df_multi["ETH"].pct_change().dropna()
fig, ax = plt.subplots(figsize=(6, 6)) ax.scatter(btc_ret, eth_ret,
alpha=0.4, color="#0d9488", s=20) # เพิ่ม regression line m, b =
np.polyfit(btc_ret, eth_ret, 1) # slope, intercept x_line =
np.linspace(btc_ret.min(), btc_ret.max(), 100) ax.plot(x_line, m * x_line +
b, color="#dc2626", linewidth=2, label=f"β={m:.2f}") ax.set_xlabel("BTC
Daily Return") ax.set_ylabel("ETH Daily Return") ax.set_title(f"BTC vs ETH
Returns (r={btc_ret.corr(eth_ret):.2f})") ax.legend() plt.tight_layout()
plt.show()
```

Matplotlib Cheat Sheet — สำหรับ Quant

Function	ใช้ทำ
`ax.plot(x, y)`	เส้นราคา, equity curve
`ax.bar(x, y)`	return รายวัน, volume
`ax.scatter(x, y)`	correlation, scatter plot
`ax.fill_between(x, y1, y2)`	Bollinger Bands, confidence interval
`ax.axhline(y)`	เส้น threshold (entry/exit level)
`ax.axvspan(x1, x2)`	ไฮไลต์ช่วงเวลา (crisis, drawdown)

ตัวอย่าง: Equity Curve + Drawdown

```
import matplotlib.pyplot as plt import numpy as np # จำลอง equity
curve np.random.seed(42) daily_ret = np.random.normal(0.0005, 0.015,
252) equity = 100000 * np.cumprod(1 + daily_ret) # คำนวณ drawdown
running_max = np.maximum.accumulate(equity) drawdown = (equity -
running_max) / running_max # Plot fig, (ax1, ax2) = plt.subplots(2,
1, figsize=(10, 6), sharex=True) ax1.plot(equity, color="#0d9488",
linewidth=1.5) ax1.set_ylabel("Portfolio Value ($)")
ax1.set_title("Strategy Equity Curve") ax1.grid(True, alpha=0.3)
ax2.fill_between(range(len(drawdown)), drawdown, 0, color="#dc2626",
alpha=0.5) ax2.set_ylabel("Drawdown") ax2.set_xlabel("Trading Day")
ax2.grid(True, alpha=0.3) max_dd = drawdown.min() final = equity[-1]
print(f"Final Equity: ${final:,.0f}") print(f"Total Return: {(final/
100000-1):.1%}") print(f"Max Drawdown: {max_dd:.1%}")
plt.tight_layout() plt.show()
```

10.4 plotly — Interactive Charts (แนะนำ)

```
# pip install plotly import plotly.graph_objects as go fig = go.Figure() #
Candlestick chart fig.add_trace(go.Candlestick( x=df.index,
open=df["open"], high=df["high"], low=df["low"], close=df["close"],
name="BTC/USDT" )) # เพิ่ม MA fig.add_trace(go.Scatter( x=df.index,
y=df["ma_50"], line=dict(color="#f59e0b", width=1.5), name="MA50" ))
fig.update_layout( title="BTC/USDT Candlestick", yaxis_title="Price (USD)",
```

```
xaxis_rangeslider_visible=False, template="plotly_dark" ) fig.show() # เปิดใน browser — zoom, pan ได้
```

▶ แบบฝึกหัด: Plot Spread Z-score พร้อม Entry/Exit Signals

```
import matplotlib.pyplot as plt import numpy as np # จำลอง z-score np.random.seed(0) z = np.cumsum(np.random.normal(0, 0.2, 200)) # OU-like entry = 2.0 exit_z = 0.5 fig, ax = plt.subplots(figsize=(12, 4)) ax.plot(z, color="#0d9488", linewidth=1, label="Z-score") ax.axhline(entry, color="#dc2626", linestyle="--", label=f"Short entry (>{entry})") ax.axhline(-entry, color="#16a34a", linestyle="--", label=f"Long entry (<{-entry})") ax.axhline(exit_z, color="gray", linestyle=":", alpha=0.7) ax.axhline(-exit_z, color="gray", linestyle=":", alpha=0.7) ax.fill_between(range(len(z)), z, 0, where=(np.abs(z) > entry), color="#dc2626", alpha=0.2, label="Active position") ax.set_title("Spread Z-score with Entry/Exit Signals") ax.legend() ax.grid(True, alpha=0.3) plt.tight_layout() plt.show()
```

จบ Part I — ทักษะที่ได้

ตอนนี้คุณสามารถ:

- อ่านโค้ดออกเสียงเป็นภาษาไทยได้ทีละบรรทัด — รวมถึง comprehension และ method chain ที่ AI ชอบเขียน ✓
- เขียน Python function, loop, และ conditional logic สำหรับ signal generation ✓
- เก็บข้อมูลด้วย list, dict, tuple, set เลือกถูกตามสถานการณ์ ✓
- โหลดข้อมูลตลาดเข้า pandas DataFrame คำนวณ return, rolling vol, z-score ✓
- plot equity curve, drawdown, scatter plot, candlestick ด้วย matplotlib/plotly ✓

ใน **Part II** เราจะนำ Python ที่รู้มาต่อเข้ากับ math จาก Part 0: OLS regression, optimization, linear programming, cointegration

⚠ **ก่อนไป Part II — สิ่งสำคัญที่ต้องจำ**

1. **Lookahead bias:** signal ที่คำนวณจากข้อมูลวัน t ต้องถูก `.shift(1)` ก่อน execute วัน t+1 — Part IV จะอธิบายละเอียด แต่จำหลักนี้ไว้ตั้งแต่ตอนนี้

2. **Code ใน Part I ยังไม่ optimize:** loop แบบ manual ที่เขียน เร็วพอสำหรับเรียน แต่ช้ากว่า pandas vectorized ใน Part II ประมาณ 10–100× สำหรับข้อมูลหลายหมื่นแถว

3. **ตัวเลข "ดูดี" ยังไม่น่าเชื่อ**: Sharpe หรือ Return ที่คำนวณในตัวอย่าง Part I–II เป็น in-sample ทั้งหมด Part IV จะสอน Walk-Forward เพื่อยืนยันว่าดีจริงหรือ overfit

PYTHON FOR QUANT TRADERS · PART II

Math Tools — เครื่องมือคณิตศาสตร์สำหรับ Quant

บทที่ 11–17: Statistics · Time Series · OLS Regression · Cointegration · Optimization · Linear Programming · Signals

จบ Part นี้ → สร้าง pair trading strategy ตั้งแต่ต้นจนได้ signal พร้อม trade

สิ่งที่จะทำได้เมื่อจบ PART II

- หาคู่หุ้นที่ cointegrate กัน ด้วย ADF test และ Johansen test
- ประมาณ hedge ratio β ด้วย OLS regression (statsmodels)
- สร้าง spread และคำนวณ z-score สำหรับ entry/exit signal
- Maximize Sharpe ratio ด้วย `scipy.optimize`
- กำหนด portfolio weight constraints ด้วย Linear Programming

ทุกอย่างนี้คือ core engine ของ Statistical Arbitrage

Running Example ตลอด Part II — Pair Trading: BTC Perp Bybit ↔ BTC Perp Binance

เราจะสร้าง pair trading strategy ทีละขั้นตอนตลอด 7 บท:
บท 11 → ตรวจสอบ distribution ของ spread · บท 12 → ทดสอบ stationarity · บท 13 → ประมาณ β ด้วย OLS
บท 14 → ยืนยัน cointegration · บท 15 → optimize position size · บท 16 → ใส่ capital constraint · บท 17 → สร้าง signal pipeline

บทที่ 11: Statistics & Probability เชิงลึก

แก่นของบทนี้

สถิติใน Part 0 เป็นแนวคิด — บทนี้แปลงเป็น Python จริง ด้วย `scipy.stats` และ `numpy` เน้น 3 เรื่องที่ Quant ใช้จริง: hypothesis testing (กลยุทธ์นี้ดีจริงหรือแค่โชค?), VaR/CVaR (ความเสี่ยงสูงสุดคือเท่าไหร่?), และ fat tail (ทำไมต้องระวัง Black Swan?)

11.1 Hypothesis Testing — กลยุทธ์ดีจริงหรือแค่โชค?

สมมติฐาน: H_0 (null) = "กลยุทธ์ไม่ดีกว่าสุ่ม" vs H_1 = "กลยุทธ์ดีกว่าสุ่มจริงๆ"

```
from scipy import stats import numpy as np # ผลตอบแทนรายวัน 252 วัน จากกลยุทธ์
np.random.seed(42) strategy_returns = np.random.normal(loc=0.0008,
scale=0.015, size=252) # t-test: H0 = mean return = 0 (ไม่ดีกว่าสุ่ม) t_stat,
p_value = stats.ttest_1samp(strategy_returns, popmean=0) print(f"Mean
Return: {strategy_returns.mean():.4f} ({strategy_returns.mean()*252:.1%}/
year)") print(f"t-statistic: {t_stat:.3f}") print(f"p-value: {p_value:.
4f}") if p_value < 0.05: print("✓ มีนัยสำคัญ — กลยุทธ์น่าจะดีกว่าสุ่ม (p < 5%)") else:
print("✗ ไม่มีนัยสำคัญ — อาจเป็นแค่โชค")
```

```
▶ OUTPUT
Mean Return: 0.0008 (20.2%/year)
t-statistic: 0.893
p-value:     0.3726
✗ ไม่มีนัยสำคัญ — อาจเป็นแค่โชค
```

ทำไม Return ดูดีแต่ p-value ยังสูง?

20% ต่อปีดูน่าสนใจ แต่ถ้า σ สูง (1.5%/วัน) — noise มากกว่า signal มาก ต้องมีข้อมูลมากพอ (หลาย 100 วัน) ถึงจะ "แยก" signal ออกจาก noise ได้ $t = \frac{\bar{r}}{\sigma/\sqrt{n}}$ → ยิ่ง n มาก t ยิ่งใหญ่ → p-value ยิ่งเล็ก

11.2 VaR และ CVaR — ความเสี่ยงสูงสุดคือเท่าไหร่?

$$\text{VaR}_{95\%} = \text{percentile}_{5\%}(r) \qquad \text{CVaR}_{95\%} = E[r \mid r \leq \text{VaR}_{95\%}]$$

```
# VaR = ขาดทุนสูงสุดใน 95% ของวันทั้งหมด # CVaR = ขาดทุนเฉลี่ยในวันที่แย่ที่สุด 5% returns = strategy_returns # จากข้างบน capital = 1_000_000 # ทุน 1 ล้านบาท var_95 = np.percentile(returns, 5) cvar_95 = returns[returns <= var_95].mean() print(f"VaR 95%: {var_95:.2%} → ขาดทุนไม่เกิน {abs(var_95)*capital:,.0f} บาท/วัน (95% ของเวลา)") print(f"CVaR 95%: {cvar_95:.2%} → เฉลี่ยขาดทุน {abs(cvar_95)*capital:,.0f} บาท ในวันที่แย่ที่สุด 5%")
```

หมายเหตุ CVaR: ต้องการ sample เพียงพอ

CVaR 95% ใช้เฉพาะ return ที่อยู่ใน worst 5% — ถ้ามีข้อมูล 100 วัน จะใช้แค่ 5 วัน

ถ้าข้อมูลน้อยกว่า 60 วัน CVaR อาจไม่น่าเชื่อถือ ควรใช้ `if len(tail) >= 5: cvar = tail.mean()`

หรือเพิ่ม `n_tail = max(1, int(len(returns) * 0.05))` เพื่อรับประกัน minimum sample

11.3 Fat Tail — ทำไม Normal Distribution ถึงอันตราย?

```
from scipy import stats # เปรียบ Normal กับ t-distribution (fat tail) normal_99 = stats.norm.ppf(0.01) # -2.33σ t5_99 = stats.t.ppf(0.01, df=5) # -3.36σ (df=5 = fat tail) print(f"Normal 1% VaR: {normal_99:.2f}σ") print(f"t(df=5) 1% VaR: {t5_99:.2f}σ") print(f"ใช้ Normal จะ underestimate ความเสี่ยง {t5_99/normal_99:.1f}x")
```

อย่าเชื่อ Normal Distribution สำหรับ crypto และช่วง crisis

BTC crash 50%+ เกิดหลายครั้ง — ถ้าใช้ Normal Distribution จะบอกว่า "แทบเป็นไปไม่ได้" (เกิน 10σ)

ในความจริง: ราคาสินทรัพย์มี "fat tail" = เหตุการณ์สุดขั้วเกิดบ่อยกว่า Normal คาด

ใช้ t-distribution, Cornish-Fisher, หรือ historical simulation แทน Normal สำหรับ risk management

ตัวอย่าง: สรุป Return Statistics แบบมืออาชีพ

```
def return_stats(returns, name="Strategy", capital=1_000_000): """สรุป statistics สำหรับ return series""" r = np.array(returns) ann = 252 mean_daily = r.mean() # return เฉลี่ยต่อวัน std_daily = r.std() # ความผันผวนต่อวัน sharpe = mean_daily / std_daily * np.sqrt(ann) # Sharpe รายปี (คูณ √252) skew = stats.skew(r) # ความเบ้ (ลบ = หางซ้ายยาว = อันตราย) kurt = stats.kurtosis(r) # excess kurtosis (บวก = fat tail) var95 = np.percentile(r, 5) # จุดตัด worst 5% ของวัน cvar95 = r[r <= var95].mean() # ค่าเฉลี่ยของวันที่แย่กว่า VaR equity = capital * np.cumprod(1 + r) # เส้นเงินทุนสะสมรายวัน rolling_max = np.maximum.accumulate(equity) # จุดสูงสุดที่เคยไปถึง ณ แต่ละวัน max_dd = ((equity - rolling_max) / rolling_max).min() # % ร่วงจากยอดที่ลึกที่สุด print(f"=== {name} ===") print(f"Annual Return: {mean_daily*ann:.1%} Sharpe: {sharpe:.2f}") print(f"Annual Vol: {std_daily*np.sqrt(ann):.1%} Max DD: {max_dd:.1%}") print(f"Skewness: {skew:.2f} Kurtosis: {kurt:.2f} (Normal: 0, 0)") print(f"VaR 95%: {var95:.2%} CVaR 95%: {cvar95:.2%}") return_stats(strategy_returns)
```

▶ แบบฝึกหัด: เปรียบ Sharpe ของสองกลยุทธ์

```
# กลยุทธ์ A: return ต่ำ volatility ต่ำ ret_a = np.random.normal(0.0003, 0.005, 252) # กลยุทธ์ B: return สูง volatility สูง ret_b = np.random.normal(0.002, 0.04, 252) sharpe_a = ret_a.mean() / ret_a.std() * np.sqrt(252) sharpe_b = ret_b.mean() / ret_b.std() * np.sqrt(252) print(f"Sharpe A: {sharpe_a:.2f} ({ret_a.mean()*252:.0%}/yr, σ={ret_a.std()*np.sqrt(252):.0%}/yr)") print(f"Sharpe B: {sharpe_b:.2f} ({ret_b.mean()*252:.0%}/yr, σ={ret_b.std()*np.sqrt(252):.0%}/yr)")
```

บทเรียน: B อาจให้ return ต่อปีสูงกว่า แต่ถ้า Sharpe ต่ำกว่า risk ต่อหน่วย return แย่กว่า

บทที่ 12: Time Series Basics

แก่นของบทนี้

ข้อมูลราคาเรียงตามเวลา มีคุณสมบัติพิเศษที่ข้อมูลทั่วไปไม่มี: แต่ละจุดสัมพันธ์กับจุดก่อนหน้า (autocorrelation) และอาจ "ล่องลอย" ไปเรื่อยๆ โดยไม่กลับ (non-stationary) การเข้าใจ 2 เรื่องนี้คือกุญแจของ Statistical Arbitrage ทั้งหมด

12.1 Stationarity — ล่องลอยหรือวนกลับ?

Stationary series: mean และ variance คงที่ตลอดเวลา — "วนกลับหาค่ากลาง"
Non-stationary series: mean เปลี่ยนตามเวลา — "ล่องลอยไม่มีทิศทาง" (random walk)

```
import numpy as np # Non-stationary: Random Walk (ราคา BTC)
np.random.seed(0) random_walk = np.cumsum(np.random.normal(0, 1, 300)) #
Stationary: Spread ระหว่างสองตลาด spread = np.random.normal(0, 1, 300) # OU-
like (mean-reverting) # สังเกต: random_walk drift ออกไปเรื่อยๆ # spread วนอยู่รอบ 0
print(f"Random Walk range: {random_walk.min():.1f} to {random_walk.max():.
1f}") print(f"Spread range: {spread.min():.1f} to {spread.max():.1f}")
```

ทำไม Stationarity ถึงสำคัญสำหรับ Pair Trading?

ถ้า spread ไม่ stationary → ไม่มีแรงดึงกลับ → "z-score สูง" อาจสูงต่อไปเรื่อยๆ → trade แล้วขาดทุนไม่จำกัด
ถ้า spread stationary → มีแรงดึงกลับ → z-score สูงแล้ว "ต้องลง" ในที่สุด → เป็น basis ของ arb
เราต้องพิสูจน์ก่อนเสมอว่า spread เป็น stationary จริง — ไม่ใช่แค่ดูกราฟ

12.2 ADF Test — ทดสอบ Stationarity

Augmented Dickey-Fuller (ADF) test: H_0 = non-stationary (random walk) vs H_1 = stationary

```
from statsmodels.tsa.stattools import adfuller def
test_stationarity(series, name="Series"): """รัน ADF test และแสดงผลแบบเข้าใจง่าย"""
```

```
result = adfuller(series, autolag="AIC") adf_stat = result[0] p_value =
result[1] crit_vals = result[4] print(f"\n--- ADF Test: {name} ---")
print(f"ADF Statistic: {adf_stat:.4f}") print(f"p-value: {p_value:.4f}")
print(f"Critical (1%): {crit_vals['1%']:.4f}") print(f"Critical (5%):
{crit_vals['5%']:.4f}") if p_value < 0.05: print(f"✓ STATIONARY
(p={p_value:.3f} < 0.05) — กลับหาค่ากลาง") else: print(f"✗ NON-STATIONARY
(p={p_value:.3f} > 0.05) — random walk") return p_value < 0.05 # ทดสอบกับข้อมูล
จาก pair test_stationarity(random_walk, "BTC Price (Random Walk)")
test_stationarity(spread, "BTC Spread (Mean-Reverting)")
```

12.3 Autocorrelation — ราคาวันนี้สัมพันธ์กับเมื่อวานแค่ไหน?

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf import
matplotlib.pyplot as plt fig, axes = plt.subplots(2, 2, figsize=(12, 8)) #
ACF/PACF ของ returns (stationary) plot_acf(spread, lags=30, ax=axes[0][0],
title="ACF — Spread (Stationary)") plot_pacf(spread, lags=30, ax=axes[0]
[1], title="PACF — Spread") # ACF/PACF ของ raw price (non-stationary)
plot_acf(random_walk, lags=30, ax=axes[1][0], title="ACF — Price (Non-
stationary)") plot_pacf(random_walk, lags=30, ax=axes[1][1], title="PACF —
Price") plt.tight_layout() plt.savefig("acf_comparison.png", dpi=150)
plt.show()
```

อ่าน ACF plot อย่างไร?

Pattern ที่เห็น	แปลว่า	ทำอะไรต่อ
ค่าลด exponentially ใน ACF	AR process (mean-reverting) ✓	ใช้เป็น spread ได้
ค่าสูงทุก lag ไม่ลด	Non-stationary (random walk)	ต้อง difference ก่อน
ค่าตัดที่ lag 1 ใน ACF	MA(1) process	ต้องใส่ MA term ใน model

▶ แบบฝึกหัด: ทดสอบ stationarity ของ spread จริง

```
import numpy as np from statsmodels.tsa.stattools import adfuller # จำลอง Ornstein-Uhlenbeck process (spread จริง) np.random.seed(1) n, kappa, mu, sigma = 500, 0.1, 0, 0.5 ou = np.zeros(n) for t in range(1, n): ou[t] = ou[t-1] + kappa*(mu - ou[t-1]) + sigma*np.random.normal() result = adfuller(ou) print(f"ADF p-value: {result[1]:.4f}") print("✓ Stationary" if result[1] < 0.05 else "✗ Non-stationary") # OU process ควร stationary เสมอ (kappa > 0)
```

บทที่ 13: OLS Regression

แก่นของบทนี้

OLS (Ordinary Least Squares) เป็นเครื่องมือหลักสำหรับประมาณ hedge ratio β — อัตราส่วนที่บอกว่าต้อง long/short ในสัดส่วนเท่าไหร่เพื่อสร้าง spread ที่ stationary สูตร matrix form:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

13.1 OLS ด้วย statsmodels

```
import numpy as np import pandas as pd import statsmodels.api as sm # จำลองราคา BTC สองตลาด np.random.seed(42) n = 252 btc_bybit = 65000 + np.cumsum(np.random.normal(0, 100, n)) btc_binance = 65000 + np.cumsum(np.random.normal(0, 100, n)) + 0.003 * btc_bybit # OLS: log(P_binance) = alpha + beta * log(P_bybit) + epsilon log_bybit = np.log(btc_bybit) log_binance = np.log(btc_binance) X = sm.add_constant(log_bybit) # เพิ่ม intercept column y = log_binance model = sm.OLS(y, X).fit() alpha = model.params['const'] beta = model.params['log_bybit'] resid = model.resid # epsilon (spread) r2 = model.rsquared p_beta = model.pvalues['log_bybit'] print(f"alpha (intercept): {alpha:.6f}") print(f"beta (hedge ratio): {beta:.6f}") print(f"R²: {r2:.4f}") print(f"p-value (beta): {p_beta:.6f}") print(f"\nSpread (residuals) stats:") print(f" Mean: {resid.mean():.6f}") print(f" Std: {resid.std():.6f}")
```

13.2 ตีความผล OLS

Output	ความหมาย	ค่าที่ดี
β	Hedge ratio — Long 1 บาท Bybit ต้อง Short β บาท Binance	ใกล้ 1.0 สำหรับ same asset
α	Constant spread ระหว่างสองตลาด	ขึ้นอยู่กับ pair
R^2	สัดส่วน variance ที่ X อธิบายได้	> 0.85 สำหรับ good pair
p-value (β)	มั่นใจแค่ไหนว่า β ≠ 0	< 0.001 สำหรับ cointegrated pair
Residuals	Spread = ส่วนที่ไม่สัมพันธ์กับ X — นี่คือสิ่งที่เราจะ trade	Mean ≈ 0, Stationary

13.3 Rolling OLS — β ที่ Adaptive

β ไม่คงที่ตลอดเวลา ต้องอัปเดตสม่ำเสมอ

```
from statsmodels.regression.rolling import RollingOLS # Rolling OLS window 60 วัน window = 60 X = sm.add_constant(log_bybit) rolling_model = RollingOLS(log_binance, X, window=window).fit() rolling_beta = rolling_model.params['log_bybit'] # สร้าง rolling spread rolling_alpha = rolling_model.params['const'] spread = log_binance - (rolling_alpha + rolling_beta * log_bybit) print(f"Beta range: {rolling_beta.dropna().min():.4f} to {rolling_beta.dropna().max():.4f}") print(f"Spread Std: {spread.std():.6f}")
```

window เลือกอย่างไร?

- **สั้นเกินไป** (เช่น 10 วัน): β volatile มาก, noise สูง → signal ผิดพลาดบ่อย
- **ยาวเกินไป** (เช่น 500 วัน): β ตอบสนองช้ามาก → พลาด regime change
- **ดี**: window ≈ 3–5 เท่าของ half-life ของ spread — เรียนวิธีหา half-life ในบทที่ 14

ตัวอย่าง: สร้าง Z-score จาก OLS Residuals

```
def build_spread(price_a, price_b, window=60): """ สร้าง spread และ z-
score จาก OLS rolling regression ทำ 3 ขั้น: (1) หา beta → (2) สร้าง spread
→ (3) คำนวณ z-score Returns: DataFrame ของ spread, beta, zscore """
log_a = np.log(price_a) # แปลงเป็น log price ทั้งคู่ log_b =
np.log(price_b) # ── ขั้น 1: หา hedge ratio (beta) ด้วย rolling OLS ── X
= sm.add_constant(log_b) # เติมคอลัมน์ค่าคงที่ (intercept) ให้ regression roll
= RollingOLS(log_a, X, window=window).fit() beta =
roll.params.iloc[:, 1] # สัมประสิทธิ์ของ log_b = hedge ratio alpha =
roll.params['const'] # intercept # ── ขั้น 2: spread = ส่วนที่ "เหลือ" หลังหัก
ความสัมพันธ์ออก ── predicted = alpha + beta * log_b # ราคา A ที่ "ควรจะเป็น" ตาม
ราคา B spread = log_a - predicted # ของจริง - ที่ควรเป็น = ความเพี้ยน # ── ขั้น 3:
z-score ของ spread (สูตรเดิมจาก Part I) ── roll_mean =
spread.rolling(window).mean() roll_std =
spread.rolling(window).std() zscore = (spread - roll_mean) /
roll_std return pd.DataFrame({ "spread": spread, "beta": beta,
"zscore": zscore }) df_pair = build_spread( pd.Series(btc_bybit),
pd.Series(btc_binance), window=60 ) print(df_pair.tail())
print(f"\nCurrent Z-score: {df_pair['zscore'].iloc[-1]:.2f}")
```

▶ แบบฝึกหัด: interpret OLS output

สมมติ OLS ของ AAPL บน MSFT ได้: $\alpha=0.5$, $\beta=0.72$, $R^2=0.81$, $p(\beta)=0.000$

แปลว่า:

- $\beta=0.72$: MSFT ขึ้น 1% → AAPL มักขึ้น 0.72% (AAPL volatile น้อยกว่า)
- $R^2=0.81$: 81% ของความผันผวน AAPL อธิบายได้ด้วย MSFT — 19% ที่เหลือคือ spread
- Hedge: Long AAPL $100K → Short MSFT $72K
- $p=0.000$: $\beta \neq 0$ แน่นอน (มีนัยสำคัญสูงมาก)
- Residual mean ≈ 0 → spread ไม่ drift → น่าจะ stationary (ต้องยืนยันด้วย ADF)

บทที่ 14: Cointegration & Z-score Signal

แก่นของบทนี้

Correlation บอกว่าสองสิ่งเดินไปทิศเดียวกัน — Cointegration บอกว่าสองสิ่ง*ผูกกันระยะยาว* แม้ระยะสั้นจะแยกออกจากกันได้ Cointegration = ข้อพิสูจน์ว่า spread มีแรงดึงกลับจริง = basis ของ Statistical Arbitrage ทั้งหมด

14.1 Correlation vs Cointegration

	Correlation สูง	Cointegrated
หมายถึง	เดินทิศเดียวกันในระยะสั้น	ผูกกันในระยะยาว มี "แรงดึง"
Spread	อาจ drift ออกไปเรื่อยๆ	กลับหาค่ากลางเสมอ
ตัวอย่าง	หุ้นสอง sector เดียวกัน	BTC สองตลาด, ADR vs local
Trade ได้ไหม?	เสี่ยง — spread อาจไม่กลับ	ใช่ — spread ต้องกลับ (ในทฤษฎี)

14.2 Engle-Granger Test — 2 ขั้นตอน

วิธี Engle-Granger: (1) OLS หา residuals → (2) ADF test บน residuals

```
from statsmodels.tsa.stattools import adfuller, coint # วิธีที่ 1: Engle-Granger ด้วย coint() โดยตรง score, pvalue, crit_vals = coint(np.log(btc_bybit), np.log(btc_binance)) print(f"Cointegration test:") print(f" Test statistic: {score:.4f}") print(f" p-value: {pvalue:.4f}") print(f" Critical (5%): {crit_vals[1]:.4f}") if pvalue < 0.05: print(f"✓ COINTEGRATED — spread มีแรงดึงกลับจริง") else: print(f"✗ NOT COINTEGRATED — อย่า trade pair นี้!") # วิธีที่ 2: ADF บน residuals (เข้าใจกระบวนการมากขึ้น) log_a = np.log(btc_bybit) log_b = np.log(btc_binance) X = sm.add_constant(log_b) residuals = sm.OLS(log_a, X).fit().resid adf_resid = adfuller(residuals) print(f"\nADF on residuals p-value: {adf_resid[1]:.4f}")
```

14.3 Johansen Test — สำหรับ 3+ Pairs

```
from statsmodels.tsa.vector_ar.vecm import coint_johansen # Johansen ทดสอบ cointegration ระหว่าง 3 series พร้อมกัน btc_okx = 65000 +
```

```
np.cumsum(np.random.normal(0, 100, n)) # ตลาดที่ 3 data =
np.column_stack([np.log(btc_bybit), np.log(btc_binance), np.log(btc_okx)])
result = coint_johansen(data, det_order=0, k_ar_diff=1) print("Johansen
Trace Statistics:") for i in range(3): trace_stat = result.lr1[i] crit_5pct
= result.cvt[i, 1] print(f" r≤{i}: trace={trace_stat:.2f},
crit(5%)={crit_5pct:.2f} " f"{'✓' if trace_stat > crit_5pct else '✗'}") #
r=0: ทดสอบว่ามีอย่างน้อย 1 cointegrating vector # r≤1: ทดสอบว่ามีอย่างน้อย 2
```

14.4 Half-Life — Spread กลับเร็วแค่ไหน?

Half-life คือเวลาที่ spread ใช้กลับมาครึ่งทางจากค่าสุดขั้ว — ใช้เลือก window และ holding period

$$\text{Half-life} = \frac{-\ln(2)}{\ln(\hat{\phi})} \approx \frac{-\ln(2)}{\hat{\phi} - 1}$$

```
def calc_halflife(spread): """คำนวณ half-life ของ mean-reverting spread"""
spread_lag = pd.Series(spread).shift(1) delta = pd.Series(spread) -
spread_lag df_hl = pd.DataFrame({"delta": delta, "lag":
spread_lag}).dropna() X = sm.add_constant(df_hl["lag"]) phi =
sm.OLS(df_hl["delta"], X).fit().params["lag"] if phi >= 0: return np.inf #
ไม่ mean-reverting (random walk หรือ explosive) if phi <= -1: return np.inf #
Oscillating — ไม่ใช่ mean-reverting จริง half_life = -np.log(2) / np.log(1 +
phi) # สูตรที่ถูกต้องสำหรับ AR(1) return half_life hl = calc_halflife(resid)
print(f"Half-life: {hl:.1f} วัน") print(f"แนะนำ: rolling window ≈ {hl*3:.0f}
วัน, hold ไม่เกิน {hl*2:.0f} วัน")
```

ตัวอย่าง: Signal Generation Pipeline ครบวงจร

```
def pair_signal(price_a, price_b, window=60, entry_z=2.0,
exit_z=0.5): """ สร้าง trading signal สำหรับ pair trading Returns:
DataFrame พร้อม signal column """ df = build_spread(price_a, price_b,
window) z = df["zscore"] signals = pd.Series("HOLD", index=z.index)
signals[z > entry_z] = "SHORT_A_LONG_B" # spread แพงเกิน signals[z < -
entry_z] = "LONG_A_SHORT_B" # spread ถูกเกิน signals[abs(z) < exit_z] =
"EXIT" df["signal"] = signals # ⚠ สำคัญมาก: signal ที่ได้จาก df.index[t] ใช้
ข้อมูลถึงวัน t # ต้อง .shift(1) ใน backtest ก่อน multiply ด้วย return เสมอ return
df result = pair_signal( pd.Series(btc_bybit),
pd.Series(btc_binance), window=60, entry_z=2.0, exit_z=0.5 )
print(result[["spread","beta","zscore","signal"]].tail(10))
```

บทที่ 15: Optimization

แก่นของบทนี้

Optimization คือการหาค่าตัวแปร (เช่น portfolio weights, position size) ที่ทำให้ objective function (เช่น Sharpe ratio) ดีที่สุด ภายใต้ constraints (เช่น weights รวม = 1) Python ทำได้ด้วย `scipy.optimize.minimize`

$$\max_{\mathbf{w}} \text{Sharpe}(\mathbf{w}) = \frac{\mathbf{w}^\top \boldsymbol{\mu} - r_f}{\sqrt{\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}}} \quad \text{s.t.} \sum w_i = 1,\ w_i \geq 0$$

15.1 scipy.optimize.minimize — API

```
from scipy.optimize import minimize import numpy as np # สมมติว่าต้องการหา x ที่ทำให้ f(x) = (x-3)^2 น้อยที่สุด def f(x): return (x[0] - 3)**2 result = minimize(f, x0=[0], method="SLSQP") print(f"Optimal x: {result.x[0]:.4f} (คาดหวัง: 3.0)") print(f"Min f(x): {result.fun:.6f}") print(f"Success: {result.success}")
```

15.2 Sharpe Ratio Maximization

```
def sharpe_maximize(returns_df, risk_free=0.0, periods=252): """ หา portfolio weights ที่ maximize Sharpe ratio returns_df: DataFrame ของ daily returns (cols = assets) """ n = len(returns_df.columns) mu = returns_df.mean().values * periods # annualized mean sigma = returns_df.cov().values * periods # annualized cov def neg_sharpe(weights): """คืนค่า negative Sharpe (minimize = maximize Sharpe)""" port_ret = weights @ mu port_var = weights @ sigma @ weights return -(port_ret - risk_free) / np.sqrt(port_var) # Constraints constraints = [{"type": "eq", "fun": lambda w: np.sum(w) - 1}] # sum = 1 bounds = [(0, 1)] * n # no short x0 = np.ones(n) / n # equally weighted start result = minimize(neg_sharpe, x0, method="SLSQP", bounds=bounds, constraints=constraints) if not result.success: print("Warning: optimizer did not converge") weights = result.x port_ret = weights @ mu port_vol = np.sqrt(weights @ sigma @ weights) print("=== Max Sharpe Portfolio ===") for name, w in zip(returns_df.columns, weights): print(f" {name}: {w:.1%}") print(f" Expected Return: {port_ret:.1%}/yr") print(f" Volatility: {port_vol:.1%}/yr") print(f" Sharpe Ratio: {(port_ret-risk_free)/port_vol:.2f}") return weights # ทดสอบ np.random.seed(0) rets = pd.DataFrame(np.random.multivariate_normal( [0.001, 0.0007, 0.0012], [[0.0004, 0.0002, 0.0001], [0.0002, 0.0003, 0.0001], [0.0001, 0.0001,
```

```
0.0005]], 252), columns=["BTC","ETH","SOL"]) optimal_w =
sharpe_maximize(rets)
```

ฝึกอ่าน 2 ท่อนที่ดูลึกลับที่สุดในโค้ดข้างบน

```
# ท่อนที่ 1
port_ret = weights @ mu
# ท่อนที่ 2
constraints = [{"type": "eq", "fun": lambda w: np.sum(w) - 1}]
```

ท่อนที่ 1: เครื่องหมาย `@` คือ "คูณแบบเวกเตอร์/เมทริกซ์" — `weights @ mu` อ่านว่า "เอาน้ำหนักแต่ละตัว คูณ return คาดหวังของ asset นั้น แล้วบวกรวมกัน" เท่ากับเขียน loop เองว่า:

```
port_ret = 0
for w, m in zip(weights, mu):
    port_ret = port_ret + w * m
```

ท่อนที่ 2: อ่านจากในออกนอก:

1. `lambda w: np.sum(w) - 1` — ฟังก์ชันจิ๋ว: "รับ w มา แล้วคืนค่า ผลรวมของ w ลบ 1"
2. `"type": "eq"` — บอก optimizer ว่าค่าที่ฟังก์ชันนี้คืน **ต้องเท่ากับ 0** → แปลว่า sum(w) - 1 = 0 → sum(w) = 1 = "ลงทุนเต็มพอดี ไม่ขาดไม่เกิน"
3. `[ { ... } ]` — เป็น list ของ dict เพราะใส่ได้หลายเงื่อนไข (scipy กำหนดรูปแบบนี้)

นี่คือ "ภาษาเฉพาะ" ของ scipy — เจอครั้งแรกทุกคนงง พออ่านออกแล้วจะเห็นว่ามันแค่บอกว่า *"เงื่อนไข: น้ำหนักรวมกันต้องเป็น 1"*

15.3 Kelly Criterion — Optimal Position Size

$$f^* = \frac{\mu - r_f}{\sigma^2} \qquad \text{(continuous Kelly)}$$

```
def kelly_fraction(mean_ret, std_ret, risk_free=0.0, fraction=0.25): """
คำนวณ Kelly fraction สำหรับ continuous distribution fraction: ใช้ 0.25
(Quarter-Kelly) เพื่อความปลอดภัย """ kelly_full = (mean_ret - risk_free) /
(std_ret ** 2) kelly_frac = kelly_full * fraction print(f"Full Kelly:
{kelly_full:.2f}x") print(f"Quarter Kelly: {kelly_frac:.2f}x ← แนะนำ")
return kelly_frac # ตัวอย่าง: กลยุทธ์ที่มี Sharpe = 1.5 daily_mean = 0.001 # 0.1% ต่อ
วัน daily_std = 0.015 # 1.5% ต่อวัน f = kelly_fraction(daily_mean, daily_std)
```

Kelly ที่ใช้ที่นี่ = Continuous Kelly (สำหรับ return กระจาย Normal)

สูตร $f^* = \frac{\mu - r_f}{\sigma^2}$ เหมาะกับ asset ที่ return กระจายแบบ Normal distribution ต่อเนื่อง

สำหรับ **discrete bet** เช่น Binary outcome (win ได้ b เท่า, lose 1): $f^* = \frac{pb-(1-p)}{b}$

ใน Quant Trading ส่วนใหญ่ใช้ continuous Kelly หรือบ่อยกว่านั้นใช้ **Half Kelly (f*/2) หรือ Quarter Kelly (f*/4)** เพราะ:

- Return จริงไม่ Normal (fat tail)
- Estimate μ และ σ มี error เสมอ
- Full Kelly มี drawdown รุนแรงมากในช่วง streak ของ losing trades

อย่าใช้ Full Kelly ในทางปฏิบัติ

Full Kelly maximize geometric growth ในทฤษฎี แต่ในทางปฏิบัติ:

- ประมาณ parameters จากข้อมูลมีความผิดพลาด → Full Kelly อาจ oversize มาก
- Drawdown จาก Full Kelly ใหญ่มาก (50%+ เป็นปกติ) → ทนไม่ไหว

ใช้ Quarter-Kelly (f*/4) เป็นจุดเริ่มต้น — เรียนเพิ่มใน Part IV

ตัวอย่าง: Mean-Variance Optimization (Efficient Frontier)

```
def efficient_frontier(returns_df, n_points=100): """ สร้าง efficient
frontier: สำหรับ return เป้าหมายแต่ละระดับ → หาพอร์ตที่ vol ต่ำสุด """ n =
len(returns_df.columns) mu = returns_df.mean().values * 252 #
expected return รายปี sigma = returns_df.cov().values * 252 #
covariance รายปี # ไล่ return เป้าหมาย จากต่ำสุดถึงสูงสุด เป็น n_points จุด
target_returns = np.linspace(mu.min(), mu.max(), n_points) def
portfolio_variance(w): """สิ่งที่ต้องการ minimize: ความแปรปรวนของพอร์ต"""
return w @ sigma @ w frontier_vols = [] # เก็บ vol ต่ำสุดของแต่ละเป้า for
target in target_returns: # เงื่อนไข 1: น้ำหนักรวม = 1 (ลงทุนเต็มพอดี)
sum_to_one = {"type": "eq", "fun": lambda w: np.sum(w) - 1} # เงื่อนไข
2: return ของพอร์ต = target ของรอบนี้ # (t=target จำเป็น! ดูกล่องกับดักด้านล่าง)
hit_target = {"type": "eq", "fun": lambda w, t=target: w @ mu - t}
result = minimize(portfolio_variance, x0=np.ones(n) / n, # เริ่มจากน้ำหนัก
เท่ากันทุกตัว method="SLSQP", bounds=[(0, 1)] * n, # ห้าม short (0 ≤ w ≤ 1)
constraints=[sum_to_one, hit_target]) if result.success: min_vol =
np.sqrt(result.fun) # ได้ variance ต่ำสุด → ถอดรากเป็น vol else: min_vol =
np.nan # แก้ไม่ได้ (เป้าสูง/ต่ำเกิน) → เว้นจุดนี้ frontier_vols.append(min_vol)
return target_returns, np.array(frontier_vols) rets_arr, vols_arr =
efficient_frontier(rets) import matplotlib.pyplot as plt fig, ax =
plt.subplots(figsize=(8, 5)) ax.plot(vols_arr, rets_arr,
color="#0d9488", linewidth=2, label="Efficient Frontier")
ax.set_xlabel("Annual Volatility") ax.set_ylabel("Annual Return")
ax.set_title("Efficient Frontier (BTC, ETH, SOL)") ax.legend()
plt.tight_layout() plt.show()
```

⚠ กับดัก lambda ใน loop — ทำไมต้องเขียน `lambda w, t=target`

ถ้าเขียน `lambda w: w @ mu - target` เฉย ๆ — lambda จะ "จำชื่อตัวแปร" ไม่ใช่ "จำค่า" พอ loop จบ `target` ชี้ไปที่ค่าสุดท้ายแล้ว ทุก lambda ที่สร้างไว้จะใช้ค่าเดียวกันหมด → frontier ผิดทั้งเส้นแบบเงียบ ๆ ไม่มี error

ท่า `t=target` คือการ "ถ่ายสำเนาค่า ณ รอบนั้น" เก็บไว้ใน t ของ lambda ตัวนั้นเอง — นี่เป็นกับดัก Python คลาสสิกที่แม้แต่ AI ยังเขียนพลาดเป็นครั้งคราว **เจอ lambda ใน loop เมื่อไหร่ ให้มองหา `=` แบบนี้เสมอ**

▶ แบบฝึกหัด: **Minimize Portfolio Variance (Min-Vol Portfolio)**

```
def min_variance_portfolio(returns_df): """หา portfolio ที่มี variance ต่ำ
ที่สุด""" n = len(returns_df.columns) sigma = returns_df.cov().values *
252 result = minimize( lambda w: w @ sigma @ w, x0=np.ones(n)/n,
method="SLSQP", bounds=[(0, 1)]*n, constraints=[{"type": "eq", "fun":
lambda w: np.sum(w) - 1}] ) w = result.x vol = np.sqrt(w @ sigma @ w)
print("Min Variance Portfolio:") for name, wi in
zip(returns_df.columns, w): print(f" {name}: {wi:.1%}") print(f"
Annual Vol: {vol:.1%}") return w min_variance_portfolio(rets)
```

บทที่ 16: Linear Programming

แก่นของบทนี้

Linear Programming (LP) คือ optimization ที่ทั้ง objective function และ constraints เป็น linear — เร็วกว่า nonlinear optimization มาก ใช้ใน Quant สำหรับ: กำหนด portfolio weight ภายใต้หลาย constraint พร้อมกัน, minimize transaction cost, sizing หลาย position พร้อมกัน

$$\min_{\mathbf{w}} \ \mathbf{c}^\top \mathbf{w} \quad \text{s.t. } \mathbf{A}\mathbf{w} \leq \mathbf{b},\ \mathbf{A}_{eq}\mathbf{w} = \mathbf{b}_{eq},\ \mathbf{lb} \leq \mathbf{w} \leq \mathbf{ub}$$

16.1 scipy.linprog — API

```
from scipy.optimize import linprog # ตัวอย่างง่าย: minimize cost ของการซื้อ 3 assets # minimize: 2*w1 + 3*w2 + 1.5*w3 # subject to: w1+w2+w3 = 1 (fully invested) # w1 >= 0.1 (min 10% BTC) # w2 >= 0.1 (min 10% ETH) c = [2.0, 3.0, 1.5] # objective: cost coefficients A_eq = [[1, 1, 1]] # equality: sum = 1 b_eq = [1] bounds = [(0.1, 0.6), (0.1, 0.5), (0, 0.8)] # min/max ของแต่ละ w result = linprog(c, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method="highs") print(f"Optimal weights: {result.x.round(3)}") print(f"Minimum cost: {result.fun:.4f}") print(f"Success: {result.success}")
```

16.2 Portfolio LP — ใส่ Constraints หลายอย่างพร้อมกัน

```
from scipy.optimize import linprog import numpy as np def lp_portfolio(expected_returns, min_return=0.10, max_weights=None, min_weights=None, max_sector=None, sectors=None): """ Minimize portfolio variance (approximated as linear) ด้วย LP — เหมาะสำหรับ constraint-heavy problem expected_returns: array ของ expected annual return แต่ละ asset min_return: minimum portfolio return ที่ต้องการ """ n = len(expected_returns) # Objective: minimize negative return (= maximize return) # Linear approximation — ใช้สำหรับ constraint-heavy problems c = -np.array(expected_returns) # Equality: sum(w) = 1 A_eq = [np.ones(n)] b_eq = [1.0] # Inequality: min return constraint # w @ returns >= min_return → -w @ returns <= -min_return A_ub = [-np.array(expected_returns)] b_ub = [-min_return] # Bounds: default 0–1 (long only) if min_weights is None: min_weights = [0.0] * n if max_weights is None: max_weights = [1.0] * n bounds = list(zip(min_weights, max_weights)) result = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method="highs") return result # ตัวอย่าง: 5 assets assets = ["BTC", "ETH", "SOL", "AAPL", "GOLD"] returns = [0.35, 0.25, 0.40,
```

```
0.12, 0.08] result = lp_portfolio( returns, min_return = 0.20, min_weights
= [0.05]*5, # min 5% ทุก asset max_weights = [0.40]*5, # max 40% ทุก asset )
print("LP Portfolio Allocation:") for asset, w in zip(assets, result.x):
print(f" {asset:5s}: {w:.1%}") print(f"Expected Return: {np.dot(result.x,
returns):.1%}")
```

16.3 PuLP — สำหรับ Integer Programming

```
# pip install pulp from pulp import * # Integer LP: เลือก k หุ้นดีที่สุดจาก n ตัว
scores = {"BTC": 0.85, "ETH": 0.70, "SOL": 0.60, "AAPL": 0.55, "GOLD":
0.40} costs = {"BTC": 5000, "ETH": 2000, "SOL": 500, "AAPL": 1000, "GOLD":
800} budget = 8000 # งบ 8000 บาท prob = LpProblem("SelectAssets",
LpMaximize) # ตัวแปร binary: 1=เลือก, 0=ไม่เลือก x = {a: LpVariable(f"x_{a}",
cat="Binary") for a in scores} # Objective: maximize total score prob +=
lpSum(scores[a] * x[a] for a in scores) # Constraints prob +=
lpSum(costs[a] * x[a] for a in scores) <= budget # budget prob +=
lpSum(x[a] for a in scores) <= 3 # เลือกได้ max 3 ตัว
prob.solve(PULP_CBC_CMD(msg=0)) print("Selected assets:") for a in scores:
if x[a].value() == 1: print(f" {a}: score={scores[a]}, cost={costs[a]}")
```

ตัวอย่าง: Position Sizing LP สำหรับ Pair Trading

```
# กำหนด: มี 3 pairs พร้อม trade, capital จำกัด # objective: maximize
expected PnL # constraints: total capital, max drawdown per pair,
correlation limit pairs = ["BTC-Bybit/Binance", "ETH-OKX/Bybit",
"SOL-Binance/OKX"] exp_pnl = [0.15, 0.12, 0.20] # expected annual
return max_dd = [0.05, 0.08, 0.10] # max expected drawdown capital =
1_000_000 # total capital บาท # Minimize negative PnL = maximize PnL
c = [-e for e in exp_pnl] # negate for minimization #
sum(allocation) = 1.0 (fully allocated) A_eq = [[1, 1, 1]] b_eq =
[1.0] # เงื่อนไข: drawdown ที่คาดของแต่ละ pair ≤ 5% ของ capital # allocation_i
× max_dd_i ≤ 0.05 — แยกกันทีละ pair = matrix แนวทแยง A_ub =
np.diag(max_dd) # [[0.05, 0, 0 ], # [0, 0.08, 0 ], # [0, 0, 0.10]]
b_ub = [0.05] * 3 bounds = [(0.0, 0.5)] * 3 # max 50% ต่อ pair result
= linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq,
bounds=bounds, method="highs") print("Optimal Position Sizing:") for
pair, alloc in zip(pairs, result.x): print(f" {pair}: {alloc:.1%} =
฿{alloc*capital:,.0f}") print(f"Expected Portfolio Return: {-
result.fun:.1%}")
```

AI มักสร้าง matrix แนวทแยงด้วย nested comprehension — ฝึกอ่าน

```
A_ub = [[max_dd[i] if j==i else 0 for j in range(3)] for i in range(3)]
```

comprehension ซ้อนสองชั้น — อ่าน**จากขวา (ชั้นนอก) เข้าซ้าย (ชั้นใน)**:

1. `for i in range(3)` (ชั้นนอก, ขวาสุด) — สร้าง 3 แถว แถวละหนึ่ง i
2. `for j in range(3)` (ชั้นใน) — แต่ละแถวมี 3 ช่อง ช่องละหนึ่ง j
3. `max_dd[i] if j==i else 0` — แต่ละช่อง: "เอาค่า max_dd **ถ้า**อยู่บนเส้นทแยง (แถว=คอลัมน์) **ไม่งั้น**ใส่ 0"

ผลคือ matrix เดียวกับ `np.diag(max_dd)` เป๊ะ — เวอร์ชัน np.diag สั้นกว่า อ่านง่ายกว่า และผิดยากกว่า ถ้า AI เขียนแบบ nested มา ขอให้มันแก้เป็น np.diag ได้เลย

▶ แบบฝึกหัด: LP Constraint สำหรับ Sector Limits

```
# 6 assets ใน 2 sectors: Crypto (BTC,ETH,SOL) และ Tech (AAPL,MSFT,NVDA)
assets = ["BTC","ETH","SOL","AAPL","MSFT","NVDA"] returns = [0.40,
0.30, 0.50, 0.15, 0.18, 0.25] # Crypto = assets 0,1,2 / Tech = assets
3,4,5 crypto = [1,1,1,0,0,0] tech = [0,0,0,1,1,1] c = [-r for r in
returns] A_eq = [[1]*6] b_eq = [1.0] # Sector limits: Crypto ≤ 60%,
Tech ≤ 60% A_ub = [crypto, tech] b_ub = [0.60, 0.60] bounds = [(0.0,
0.4)] * 6 # max 40% per asset res = linprog(c, A_ub=A_ub, b_ub=b_ub,
A_eq=A_eq, b_eq=b_eq, bounds=bounds, method="highs") for a, w in
zip(assets, res.x): print(f" {a}: {w:.1%}") print(f"Crypto total:
{sum(res.x[:3]):.1%} Tech total: {sum(res.x[3:]):.1%}")
```

บทที่ 17: Signals & Filters

แก่นของบทนี้

Signal คือ "เมื่อไหร่ควร trade" — บทนี้รวม 3 เครื่องมือ: **Bollinger Bands** (signal จาก volatility), **Kalman Filter** (track β แบบ adaptive), และ **Signal Pipeline** (รวมหลาย signal เข้าด้วยกัน) ซึ่งเป็น boilerplate ที่ใช้ได้กับทุก strategy

17.1 Bollinger Bands — Signal จาก Volatility

```
def bollinger_bands(series, window=20, n_std=2): """คำนวณ Bollinger Bands"""
s = pd.Series(series) mean = s.rolling(window).mean() std =
s.rolling(window).std() upper = mean + n_std * std lower = mean - n_std *
std return pd.DataFrame({"mid": mean, "upper": upper, "lower": lower,
"pct_b": (s - lower) / (upper - lower)}) # pct_b: 0 = ที่ lower band, 1 = ที่
upper band # signal: SHORT เมื่อ pct_b > 1, LONG เมื่อ pct_b < 0 bb =
bollinger_bands(spread, window=20, n_std=2) signal = pd.Series("HOLD",
index=bb.index) signal[bb["pct_b"] > 1.0] = "SHORT" signal[bb["pct_b"] <
0.0] = "LONG"
```

17.2 Kalman Filter — Track β แบบ Adaptive

Kalman Filter ประมาณ β แบบ real-time โดยอัปเดตทุกครั้งที่มีข้อมูลใหม่ — ดีกว่า rolling OLS ตรงที่ไม่มี window cutoff และ smooth กว่า

Kalman Filter ทำงานสองจังหวะ — "เดา แล้วแก้เดา" ซ้ำไปเรื่อย ๆ

ลองนึกถึงการเดาน้ำหนักเพื่อนทุกครั้งที่เจอกัน:
จังหวะ 1 — เดา (Predict): "วันนี้น่าจะหนักเท่าเดิมจากครั้งก่อน แต่เวลาผ่านไปแล้ว ความมั่นใจของเราลดลงนิดหน่อย"
จังหวะ 2 — แก้เดา (Update): เห็นข้อมูลใหม่ (เพื่อนตัวจริง) → เทียบกับที่เดา → ถ้าต่างกัน ขยับคำเดาเข้าหาความจริง *ตามสัดส่วนความมั่นใจ* — มั่นใจคำเดาเดิมมากก็ขยับนิดเดียว ไม่มั่นใจก็ขยับเยอะ

ตัวเลข "สัดส่วนการขยับ" นี้คือ **Kalman Gain (K)** — หัวใจของทั้ง algorithm มีแค่นี้

```
def kalman_hedge_ratio(price_a, price_b, delta=1e-4, vt=1e-3): """ Track hedge ratio beta ด้วย Kalman Filter delta: process noise (ใหญ่ = ยอมให้ beta เปลี่ยนเร็ว) vt: observation noise (ใหญ่ = ราคามี noise มาก เชื่อข้อมูลใหม่น้อยลง) Returns: array ของ beta ณ แต่ละเวลา """ n = len(price_a) beta = np.zeros(n) # เตรียมที่เก็บ beta ของทุกวัน P = np.ones(n) # P = "ความไม่มั่นใจ" ในค่า beta (variance ของคำเดา) beta[0] = 1.0 # วันแรก: เดาไว้ก่อนว่า beta = 1 for t in range(1, n): # ── จังหวะ 1: เดา (Predict) ── # beta วันนี้น่าจะเท่าเมื่อวาน แต่ความไม่มั่นใจเพิ่มขึ้นตาม delta P_pred = P[t-1] + delta # ── จังหวะ 2: แก้เดา (Update) ด้วยข้อมูลใหม่ ── x = price_b[t] # ราคาตลาด B วันนี้ (ตัวพยากรณ์) y_pred = beta[t-1] * x # ราคา A "ที่ควรเป็น" ตามคำเดาเดิม innov = price_a[t] - y_pred # ของจริง - ที่เดา = "เดาพลาดเท่าไหร่" S = x**2 * P_pred + vt # ความแปรปรวนรวมของความพลาด # (จากความไม่มั่นใจใน beta + noise ของราคา) K = P_pred * x / S # Kalman Gain: "ควรขยับคำเดากี่ %" # ไม่มั่นใจมาก (P สูง) → K ใหญ่ → ขยับเยอะ # ราคา noise มาก (vt สูง) → K เล็ก → ขยับนิดเดียว beta[t] = beta[t-1] + K * innov # ขยับ beta เข้าหาความจริง ตามสัดส่วน K P[t] = (1 - K * x) * P_pred # แก้เดาแล้ว → ความมั่นใจเพิ่มขึ้น (P ลดลง) return beta # ทดสอบ log_a = np.log(btc_bybit) log_b = np.log(btc_binance) kf_beta = kalman_hedge_ratio(log_a, log_b) print(f"Kalman Beta (latest 5): {kf_beta[-5:].round(6)}")
```

17.3 Signal Pipeline — รวมทุกอย่างเข้าด้วยกัน

```
class PairTradingSignal: """ Signal pipeline สำหรับ Pair Trading รวม: OLS hedge ratio, z-score, Bollinger filter """ def __init__(self, window=60, entry_z=2.0, exit_z=0.5): self.window = window self.entry_z = entry_z self.exit_z = exit_z def fit(self, price_a: pd.Series, price_b: pd.Series): """ประมาณ parameters จากข้อมูลประวัติ""" self.price_a = price_a self.price_b = price_b # OLS spread self.df = build_spread(price_a, price_b, self.window) # Half-life self.halflife = calc_halflife(self.df["spread"].dropna()) print(f"Half-life: {self.halflife:.1f} วัน") # ADF test adf_p = adfuller(self.df["spread"].dropna())[1] self.is_stationary = adf_p < 0.05 print(f"Stationary: {self.is_stationary} (p={adf_p:.3f})") return self def signal(self) -> pd.Series: """สร้าง trading signal""" z = self.df["zscore"] sig = pd.Series("HOLD", index=z.index) sig[z > self.entry_z] = "SHORT_A" sig[z < -self.entry_z] = "LONG_A" sig[abs(z) < self.exit_z] = "EXIT" return sig def summary(self): """แสดง summary ของ current state""" current_z =
```

```
self.df["zscore"].iloc[-1] current_b = self.df["beta"].iloc[-1] sig =
self.signal().iloc[-1] print(f"=== Pair Signal Summary ===")
print(f"Current Z-score: {current_z:.3f}") print(f"Current Beta:
{current_b:.6f}") print(f"Signal: {sig}") print(f"Is Stationary:
{self.is_stationary}") # ใช้งาน pipeline = PairTradingSignal(window=60,
entry_z=2.0, exit_z=0.5) pipeline.fit(pd.Series(btc_bybit),
pd.Series(btc_binance)) pipeline.summary()
```

จบ Part II — สิ่งที่ทำได้แล้ว

- ทดสอบ stationarity ด้วย ADF test ✓
- ยืนยัน cointegration ด้วย Engle-Granger และ Johansen ✓
- ประมาณ β ด้วย OLS และ Kalman Filter ✓
- สร้าง z-score signal สำหรับ entry/exit ✓
- Maximize Sharpe ratio ด้วย `scipy.optimize` ✓
- กำหนด portfolio constraint ด้วย Linear Programming ✓
- รวมทุกอย่างเป็น `PairTradingSignal` class ✓

ใน **Part III** เราจะ refactor code นี้ให้เป็น OOP ที่ maintainable และ extensible

⚠ คำเตือน: ตัวเลขใน Part II คือ In-Sample — อย่าเชื่อ 100%

Sharpe 2.0+ และ Return 15%+ ที่เห็นใน Part II ล้วน **คำนวณจากข้อมูลชุดเดียวกับที่ optimize parameter** (in-sample)

ในชีวิตจริง (out-of-sample) ผลลัพธ์มักต่ำกว่า **50–80%** เพราะ:

- ตลาดเปลี่ยน — pattern ในอดีตไม่ซ้ำแบบในอนาคต
- Parameter ที่ดีในอดีต = overfit กับ noise ในอดีต
- Transaction cost ที่ลืมใส่ กินไป 20–50% ของ gross PnL

Part IV จะสอน Walk-Forward Validation — วิธีเดียวที่ยืนยันได้ว่า strategy มี edge จริง ไม่ใช่แค่ overfit
อย่าใช้เงินจริงจนกว่าจะผ่านการทดสอบใน Part IV

PYTHON FOR QUANT TRADERS · PART III

Object-Oriented Programming — สร้างระบบที่ขยายได้

บทที่ 18–21: Classes · Inheritance · Design Patterns · OOP Mini-Project

จบ Part นี้ → เปลี่ยน script ที่รันครั้งเดียวเป็น trading system ที่ใช้ได้จริง

ทำไมต้องเรียน OOP?

Code จาก Part II ทำงานได้ แต่มีปัญหาเมื่อโตขึ้น:

- เพิ่ม strategy ใหม่ → ต้องแก้ไขทุกที่ที่มีโค้ดเก่า
- แชร์ code ระหว่าง strategy → copy-paste แล้วแก้
- เพิ่ม asset ใหม่ → เพิ่มตัวแปรกระจาย codebase

OOP แก้ปัญหาเหล่านี้ด้วยการ **จัดกลุ่มข้อมูลและ behavior** เข้าด้วยกันเป็น class — เปลี่ยนที่เดียว กระทบทุกที่

Running Example ตลอด Part III — Refactor PairTradingSignal จาก Part II

เราจะนำ `PairTradingSignal` จาก Part II มาออกแบบใหม่ให้เป็น OOP ที่ดี:
บท 18 → เข้าใจ class พื้นฐาน · บท 19 → Abstract base class + inheritance
บท 20 → ใส่ design patterns · บท 21 → Mini-project: ระบบครบวงจร

บทที่ 18: Classes & Objects

แก่นของบทนี้

Class คือ "blueprint" — นิยามว่า object ประเภทนี้เก็บข้อมูลอะไร (attributes) และทำอะไรได้บ้าง (methods) Object คือ "ชิ้นงานจริง" ที่สร้างจาก blueprint นั้น เหมือนแบบบ้าน (class) กับบ้านจริง (object)

18.1 ทำไมต้องมี Class — เรื่องเล่าจาก Pair ที่สอง

โค้ดสไตล์ Part I/II (ตัวแปร + ฟังก์ชัน) ทำงานได้ดีกับ pair เดียว:

```
# Part II style: ตัวแปรวางไว้ระดับบนสุดของไฟล์ window = 60 entry_z = 2.0 df =
build_spread(btc_bybit, btc_binance, window) hl =
calc_halflife(df["spread"]) signal = make_signal(df, entry_z)
```

ปัญหาเริ่มเมื่อเทรด **pair ที่สอง** ซึ่งใช้ parameter คนละชุด:

```
# เพิ่ม pair ที่สอง... ตัวแปรเริ่มแตกหน่อ window_2 = 90 # pair นี้ mean-revert ช้ากว่า ใช้
window ยาวกว่า entry_z_2 = 1.5 df_2 = build_spread(eth_okx, eth_bybit,
window_2) hl_2 = calc_halflife(df_2["spread"]) signal = make_signal(df,
entry_z) signal_2 = make_signal(df_2, entry_z) # ← bug! ลืมเปลี่ยนเป็น entry_z_2
# Python ไม่ฟ้องอะไรเลย — signal ผิดเงียบ ๆ
```

สังเกตว่า bug นี้เกิดเพราะ**ข้อมูลกับ parameter ของแต่ละ pair ไม่ได้ผูกกัน** — มันแค่ "วางอยู่ใกล้กัน" ในไฟล์ แล้วเราต้องจำเองว่าอะไรคู่กับอะไร ลองนึกต่อ: ถ้ามี **10 pairs** จะมีตัวแปร 40+ ตัวลอยอยู่ และทุกการเรียกฟังก์ชันคือโอกาสหยิบผิดคู่

Class คือคำตอบ: กล่องที่มัด "ข้อมูล + parameter + ฟังก์ชันที่ทำงานกับมัน" ไว้ด้วยกันเป็นชิ้นเดียว แล้วสร้างกี่กล่องก็ได้:

```
# แต่ละ pair คือ object หนึ่งชิ้น — ถือของของตัวเองครบ pair1 = PairStrategy("BTC-Bybit",
"BTC-Binance", window=60, entry_z=2.0) pair2 = PairStrategy("ETH-OKX",
"ETH-Bybit", window=90, entry_z=1.5) signal1 = pair1.make_signal() # ใช้
window=60, entry_z=2.0 ของตัวเองเสมอ signal2 = pair2.make_signal() # ใช้ของตัวเอง —
หยิบสลับกันไม่ได้โดยโครงสร้าง
```

> 🗣อ่านว่า: สั่งให้ make_signal **ของ** pair1 ทำงาน — จุด (.) ยังอ่านว่า "ของ" เหมือนเดิมจาก Part I แค่ตอนนี้ "ของ" หมายถึงทั้งข้อมูลและฟังก์ชันที่อยู่ในกล่องเดียวกัน

CLASS ไม่ใช่ความรู้ใหม่

Class = **dict (เก็บข้อมูล) + function (ทำงาน) เอามาผูกติดกัน** — สองอย่างที่ใช้คล่องแล้วจาก Part I

ความใหม่มีแค่ 2 สิ่ง:
1. `self` = "กล่องใบนี้" — เมื่อ Python รัน `pair1.make_signal()` มันแปลงเป็น `PairStrategy.make_signal(pair1)` โดยอัตโนมัติ กล่าวคือ `self` คือ `pair1` ที่ถูกส่งเข้ามาเป็น argument แรกเสมอ — นี่คือ binding: function รู้ว่าตัวเองทำงานให้กล่องใบไหน
2. `self.window` = "หยิบ window จาก dict ของกล่องใบนี้" — ภายในเป็นแค่ dict ธรรมดา แค่ syntax ต่างกัน: `cfg["window"]` กับ `self.window` คือสิ่งเดียวกัน

18.2 Class พื้นฐาน

```
# แบบ procedural (Part I/II style) — ข้อมูลกระจาย symbol_a = "BTCUSDT-Bybit" symbol_b = "BTCUSDT-Binance" window = 60 entry_z = 2.0 # แบบ OOP — ข้อมูลอยู่ในที่เดียว class PairConfig: """เก็บ configuration ของ pair trading""" def __init__(self, symbol_a, symbol_b, window=60, entry_z=2.0, exit_z=0.5): self.symbol_a = symbol_a self.symbol_b = symbol_b self.window = window self.entry_z = entry_z self.exit_z = exit_z def __repr__(self): return (f"PairConfig({self.symbol_a} / {self.symbol_b}, " f"window={self.window}, entry_z=±{self.entry_z})") def is_valid(self): """ตรวจสอบว่า config สมเหตุสมผล""" return (self.window >= 20 and self.entry_z > self.exit_z > 0) # สร้าง object cfg = PairConfig("BTCUSDT-Bybit", "BTCUSDT-Binance", window=60) print(cfg) print(f"Valid: {cfg.is_valid()}")
```

▶ OUTPUT

```
PairConfig(BTCUSDT-Bybit / BTCUSDT-Binance, window=60, entry_z=±2.0)
Valid: True
```

18.3 ส่วนประกอบของ Class

ส่วนประกอบ	ใช้งาน	ตัวอย่าง
`__init__`	Constructor — รันเมื่อ `obj = Class(...)`	เก็บ parameters, สร้าง initial state
`__repr__`	String representation สำหรับ debug	`print(obj)` แสดงข้อมูลสำคัญ
`self`	ชี้ไปยัง object ปัจจุบัน	`self.window` คือ attribute ของ object นี้
Instance method	Function ที่รับ `self` เป็น argument แรก	`def is_valid(self):`
Class method	Method ที่แชร์ระหว่างทุก object	`@classmethod def from_dict(cls, d):`
Static method	Function ใน class ที่ไม่ใช้ `self`	`@staticmethod def annualize(daily):`

`self` กับ `cls` ต่างกันอย่างไร?

	`self`	`cls`
ชี้ไปที่	object ตัวนั้น (กล่องใบนี้)	class เอง (แบบพิมพ์เขียว)
ใช้ใน	method ทั่วไป	`@classmethod`
ตัวอย่าง	`self.window` = window ของ object นี้	`cls(rets, name)` = สร้าง object ใหม่จาก class

`@classmethod` ใช้บ่อยสำหรับ "alternative constructor" เช่น `TradingMetrics.from_prices(prices)` = "สร้าง TradingMetrics จาก prices แทนที่จะส่ง returns โดยตรง"

```
class TradingMetrics: """คำนวณและเก็บ performance metrics""" TRADING_DAYS =
252 # class variable — แชร์ทุก instance def __init__(self, returns,
name="Strategy"): self.returns = np.array(returns) self.name = name
self._cache = {} # _ prefix = "ควรใช้ภายใน class" (convention) @property def
sharpe(self): """คำนวณ Sharpe Ratio (cached)""" if "sharpe" not in
self._cache: m = self.returns.mean() s = self.returns.std()
self._cache["sharpe"] = m / s * np.sqrt(self.TRADING_DAYS) return
self._cache["sharpe"] @property def annual_return(self): return
```

```
self.returns.mean() * self.TRADING_DAYS @property def annual_vol(self):
return self.returns.std() * np.sqrt(self.TRADING_DAYS) @property def
max_drawdown(self): equity = np.cumprod(1 + self.returns) peak =
np.maximum.accumulate(equity) return ((equity - peak) / peak).min()
@classmethod def from_prices(cls, prices, name="Strategy"): """สร้าง
TradingMetrics จาก price series แทน returns""" rets = np.diff(prices) /
prices[:-1] return cls(rets, name) @staticmethod def annualize(daily_val,
periods=252): """Utility: annualize daily metric""" return daily_val *
np.sqrt(periods) def __repr__(self): return (f"TradingMetrics({self.name}:
" f"Sharpe={self.sharpe:.2f}, " f"Return={self.annual_return:.1%}, "
f"DD={self.max_drawdown:.1%})") # ใช้งาน import numpy as np
np.random.seed(42) rets = np.random.normal(0.0005, 0.012, 252) m =
TradingMetrics(rets, "BTC Pair") print(m) print(f"Annual Vol:
{m.annual_vol:.1%}")
```

```
▶ OUTPUT
TradingMetrics(BTC Pair: Sharpe=0.66, Return=12.6%, DD=-12.8%)
Annual Vol: 19.1%
```

18.4 @property — Attribute ที่คำนวณแบบ Lazy

`@property` ทำให้ method เรียกใช้ได้เหมือน attribute — ไม่ต้องใส่วงเล็บ และคำนวณเมื่อถูกเรียกเท่านั้น

```
# โดยไม่มี @property print(m.annual_vol()) # ต้องใส่ () — ดูแปลก # มี @property
print(m.annual_vol) # อ่านเหมือน attribute — ชัดเจนกว่า # ประโยชน์: lazy evaluation —
คำนวณเมื่อถูกเรียกเท่านั้น # ถ้า returns เปลี่ยน → ผลลัพธ์เปลี่ยนอัตโนมัติ
```

ตัวอย่าง: Position class พร้อม computed properties

```
class Position: """แทนตำแหน่งการเทรดเดี่ยว""" def __init__(self, symbol, side, size, entry_price, fee_rate: float = 0.001): self.symbol = symbol self.side = side # "long" หรือ "short" self.size = size # จำนวน unit self.entry_price = entry_price self.exit_price = None self.fee_rate = fee_rate @property def is_open(self): return self.exit_price is None @property def pnl(self): if self.exit_price is None: raise ValueError("Position ยังไม่ปิด") raw = (self.exit_price - self.entry_price) * self.size return raw if self.side == "long" else -raw @property def return_pct(self): return self.pnl / (self.entry_price * self.size) def close(self, exit_price): self.exit_price = exit_price return self def __repr__(self): status = "OPEN" if self.is_open else f"CLOSED PnL={self.pnl:+,.0f}" return f"Position({self.side} {self.size} {self.symbol} @ {self.entry_price:,} [{status}])" # ทดสอบ pos = Position("BTCUSDT", "long", 0.5, 63000) print(pos) pos.close(65500) print(pos) print(f"Return: {pos.return_pct:.2%}")
```

▶ OUTPUT

```
Position(long 0.5 BTCUSDT @ 63,000 [OPEN])
Position(long 0.5 BTCUSDT @ 63,000 [CLOSED PnL=+1,250])
Return: 3.97%
```

▶ แบบฝึกหัด: เพิ่ม fee calculation ให้ Position

```
class Position: def __init__(self, symbol, side, size, entry_price, fee_rate=0.001): # 0.1% maker fee self.symbol = symbol self.side = side self.size = size self.entry_price = entry_price self.exit_price = None self.fee_rate = fee_rate @property def gross_pnl(self): raw = (self.exit_price - self.entry_price) * self.size return raw if self.side == "long" else -raw @property def fees(self): entry_val = self.entry_price * self.size exit_val = self.exit_price * self.size return (entry_val + exit_val) * self.fee_rate @property def net_pnl(self): return self.gross_pnl - self.fees def close(self, exit_price): self.exit_price = exit_price return self
```

บทที่ 19: Inheritance & Abstract Base Class

แก่นของบทนี้

Inheritance ให้ class ลูกรับ attribute และ method จาก class แม่ — ป้องกัน code ซ้ำ Abstract Base Class (ABC) กำหนด "สัญญา" ว่า class ลูกต้อง implement method ใดบ้าง เหมือน interface ใน Java/C++

19.1 Inheritance พื้นฐาน

```
class BaseStrategy: """Base class สำหรับทุก trading strategy""" def
__init__(self, name, capital=100_000): self.name = name self.capital =
capital self.trades = [] # history ของทุก trade self.equity = [capital] #
equity curve def log_trade(self, position: Position): """บันทึก trade ที่ปิด
แล้ว""" if position.is_open: raise ValueError("position นี้ยังไม่ถูกปิด — ต้อง close()
ก่อน") self.trades.append(position) self.capital += position.net_pnl
self.equity.append(self.capital) def summary(self): if not self.trades:
print(f"{self.name}: ยังไม่มี trade") return pnls = [t.net_pnl for t in
self.trades] winners = [p for p in pnls if p > 0] losers = [p for p in pnls
if p <= 0] print(f"=== {self.name} ===") print(f"Trades:
{len(self.trades)}") print(f"Win Rate: {len(winners)/len(pnls):.1%}")
print(f"Avg Win: {sum(winners)/len(winners) if winners else 0:+,.0f}")
print(f"Avg Loss: {sum(losers)/len(losers) if losers else 0:+,.0f}")
print(f"Total PnL: {sum(pnls):+,.0f}") print(f"Final Capital:
{self.capital:,.0f}") def __repr__(self): return
f"{self.__class__.__name__}({self.name}, capital={self.capital:,.0f})"
class PairTradingStrategy(BaseStrategy): """Pair trading ต่อยอดจาก
BaseStrategy""" def __init__(self, name, pair_config: PairConfig,
capital=100_000): super().__init__(name, capital) # เรียก __init__ ของแม่
self.config = pair_config self.signal_engine = None def fit(self, price_a,
price_b): """เทรน signal engine""" # PairTradingSignal ถูก import สมมติจาก Part
II — ในโปรเจกต์จริงให้วางในไฟล์เดียวกัน self.signal_engine = PairTradingSignal( window
= self.config.window, entry_z = self.config.entry_z, exit_z =
self.config.exit_z ).fit(price_a, price_b) return self def
get_signal(self): if self.signal_engine is None: raise RuntimeError("ต้อง
fit() ก่อน") return self.signal_engine.signal().iloc[-1] # ทดสอบ inheritance
cfg = PairConfig("BTC-Bybit", "BTC-Binance") strategy =
PairTradingStrategy("BTC Pair v1", cfg, capital=500_000) print(strategy)
print(f"เป็น BaseStrategy ไหม? {isinstance(strategy, BaseStrategy)}")
```

```
▶ OUTPUT
PairTradingStrategy(BTC Pair v1, capital=500,000)
เป็น BaseStrategy ไหม? True
```

ฝึกอ่าน: `super().__init__(name, capital)`

```
class PairTradingStrategy(BaseStrategy):
    def __init__(self, name, pair_config, capital=100_000):
        super().__init__(name, capital)   # ← บรรทัดนี้
```

อ่านจากในออกนอก:

1. `super()` — "คลาสแม่ของฉัน" = `BaseStrategy` (ที่อยู่ใน parentheses ของ class definition)
2. `super().__init__` — "เรียก `__init__` **ของ** คลาสแม่"
3. `super().__init__(name, capital)` — ส่ง name กับ capital ให้แม่ไปตั้งค่า `self.name` , `self.capital` , `self.trades` , `self.equity` เหมือนที่แม่ทำ

ถ้าไม่เรียก `super().__init__()` → `self.name` และ `self.capital` จะไม่ถูกสร้าง → เรียกทีหลังจะ AttributeError — **ลูกต้องเรียกแม่ให้ทำงานของแม่ก่อน แล้วค่อยทำงานของตัวเอง**

19.2 Abstract Base Class — บังคับให้ implement

```
from abc import ABC, abstractmethod class Strategy(ABC): """Abstract base class — blueprint สำหรับทุก strategy""" def __init__(self, name, capital=100_000): self.name = name self.capital = capital self.trades = [] @abstractmethod def fit(self, data): """ต้อง implement — train model""" pass @abstractmethod def signal(self, data) -> str: """ต้อง implement — คืน 'LONG', 'SHORT', 'EXIT', 'HOLD'""" pass @abstractmethod def position_size(self, signal: str) -> float: """ต้อง implement — คืน size เป็น fraction ของ capital""" pass def summary(self): # concrete method — ไม่ต้อง override pnls = [t.net_pnl for t in self.trades] total = sum(pnls) if pnls else 0 print(f"{self.name}: {len(pnls)} trades, PnL={total:+,.0f}") # ถ้าลืม implement abstractmethod จะ error ทันที class BadStrategy(Strategy): pass # ไม่ implement fit, signal, position_size # s = BadStrategy("test") ← TypeError: Can't instantiate abstract class
```

ฝึกอ่าน: `@abstractmethod` และ `ABC`

```
from abc import ABC, abstractmethod
```

```
class Strategy(ABC):          # ABC = Abstract Base Class
    @abstractmethod
    def signal(self, data) -> str:
        pass
```

อ่านทีละส่วน:

1. `ABC` ใน `(ABC)` — บอก Python ว่า "class นี้เป็น abstract — ห้ามสร้าง object โดยตรง"
2. `@abstractmethod` — decorator ที่ "ติดป้าย" ว่า method นี้**ต้องถูก override โดยลูก** — ถ้าลูกไม่ทำ สร้าง object ไม่ได้เลย
3. `pass` ใน method body — บอกว่า "แม่ไม่ได้ implement จริง ๆ แค่กำหนดชื่อ/signature ไว้"

เปรียบ ABC กับ interface ของหมอ: "ทุกคนที่จบแพทย์ *ต้อง* มี license — ถ้าไม่มีจะทำงานเป็นหมอไม่ได้" — ABC คือการบังคับแบบเดียวกัน: ทุก strategy *ต้อง* implement `fit()` และ `signal()`

ประโยชน์ของ ABC ใน trading system

- บังคับว่าทุก strategy ต้องมี `fit()`, `signal()`, `position_size()`
- Backtester ทำงานกับ `Strategy` ประเภทไหนก็ได้ — ไม่ต้องรู้ว่าเป็น pair, momentum, หรือ vol
- เพิ่ม strategy ใหม่ → implement 3 methods → ทำงานกับ backtester ทันที

19.3 Encapsulation — ซ่อน implementation detail

```
class RiskManager: """จัดการ risk — ซ่อน implementation ภายใน""" def
__init__(self, max_position_pct=0.20, max_drawdown_pct=0.10): self._max_pos
= max_position_pct # _ = "private by convention" self._max_dd =
max_drawdown_pct self.__peak_equity = None # __ = name-mangled (private
จริงๆ) def check(self, signal, capital, current_equity): """คืน True ถ้า trade
ผ่าน risk check""" if signal == "HOLD": return False # อัปเดต peak if
self.__peak_equity is None or current_equity > self.__peak_equity:
self.__peak_equity = current_equity # Drawdown check drawdown =
(current_equity - self.__peak_equity) / self.__peak_equity if drawdown < -
self._max_dd: print(f"RISK BLOCK: Drawdown {drawdown:.1%} เกิน limit {-
self._max_dd:.1%}") return False return True def size(self, signal,
capital, volatility): """คำนวณ position size ที่ปลอดภัย""" if volatility <= 0:
return 0 # Vol-adjusted sizing: target 1% daily vol contribution target_vol
= 0.01 raw_size = target_vol / volatility return min(raw_size,
self._max_pos) # ไม่เกิน max position @property def max_position(self): return
self._max_pos # max_position อ่านได้ แต่เปลี่ยนตรงๆ ไม่ควร rm =
```

```
RiskManager(max_position_pct=0.15, max_drawdown_pct=0.10) print(f"Max
position: {rm.max_position:.0%}") print(f"Approved: {rm.check('LONG',
500000, 490000)}")
```

ตัวอย่าง: PairStrategy ที่ implement ABC ครบ

```
# ต้อง import: from statsmodels.regression.rolling import RollingOLS
class PairStrategy(Strategy): """Concrete pair trading strategy —
implement ครบ 3 abstract methods""" def __init__(self, name,
symbol_a, symbol_b, window=60, entry_z=2.0, capital=100_000):
super().__init__(name, capital) self.symbol_a = symbol_a
self.symbol_b = symbol_b self.window = window self.entry_z = entry_z
self._beta = None self._zscore = None def fit(self, data:
pd.DataFrame): """data: DataFrame ที่มี column symbol_a และ symbol_b"""
log_a = np.log(data[self.symbol_a]) log_b =
np.log(data[self.symbol_b]) X = sm.add_constant(log_b) roll =
RollingOLS(log_a, X, window=self.window).fit() self._beta =
roll.params.iloc[:, 1] alpha = roll.params['const'] spread = log_a -
(alpha + self._beta * log_b) roll_m =
spread.rolling(self.window).mean() roll_s =
spread.rolling(self.window).std() self._zscore = (spread - roll_m) /
roll_s return self def signal(self, data=None) -> str: z =
self._zscore.iloc[-1] if z > self.entry_z: return "SHORT" if z < -
self.entry_z: return "LONG" if abs(z) < 0.5: return "EXIT" return
"HOLD" def position_size(self, signal: str) -> float: if signal ==
"HOLD": return 0.0 if signal == "EXIT": return 0.0 return 0.10 # 10%
ของ capital (จะ optimize ใน Part IV) # ตรวจสอบว่า implement ครบ strat =
PairStrategy("BTC Pair", "Bybit", "Binance") print(isinstance(strat,
Strategy)) # True print(strat)
```

▶ แบบฝึกหัด: สร้าง MomentumStrategy ที่ implement ABC

```
class MomentumStrategy(Strategy): """Momentum strategy: long เมื่อ MA20 > MA50""" def __init__(self, name, symbol, fast=20, slow=50, capital=100_000): super().__init__(name, capital) self.symbol = symbol self.fast = fast self.slow = slow self._ma_fast = None self._ma_slow = None def fit(self, data: pd.DataFrame): prices = data[self.symbol] self._ma_fast = prices.rolling(self.fast).mean() self._ma_slow = prices.rolling(self.slow).mean() return self def signal(self, data=None) -> str: if self._ma_fast.iloc[-1] > self._ma_slow.iloc[-1]: return "LONG" elif self._ma_fast.iloc[-1] < self._ma_slow.iloc[-1]: return "SHORT" return "HOLD" def position_size(self, signal: str) -> float: return 0.20 if signal != "HOLD" else 0.0
```

บทที่ 20: **Design Patterns**

แก่นของบทนี้

Design Patterns คือ "วิธีแก้ปัญหาซ้ำๆ ที่ผ่านการพิสูจน์แล้ว" ใน Quant system ใช้บ่อย 3 patterns: **Strategy Pattern** (สลับ algorithm ได้โดยไม่แก้ backtester), **Observer Pattern** (แจ้งเตือนเมื่อมี event เช่น signal เกิด), **Factory Pattern** (สร้าง object จาก config โดยไม่ hard-code ชื่อ class)

20.1 Strategy Pattern — สลับ Algorithm

Backtester ทำงานกับ *ทุก* strategy โดยไม่รู้ว่าเป็นประเภทไหน — ขอแค่ implement `Strategy` ABC

```
class Backtester: """ Generic backtester — ทำงานกับ Strategy ใดก็ได้ Strategy
Pattern: algorithm (strategy) แยกจาก runner (backtester) """ def
__init__(self, strategy: Strategy, risk_manager: RiskManager = None):
self.strategy = strategy self.rm = risk_manager or RiskManager()
self.results = [] def run(self, data: pd.DataFrame, price_col: str) ->
pd.DataFrame: """ วิ่ง backtest บน data data: DataFrame ที่มีราคา price_col: ชื่อ
column ของราคาที่ใช้ execute trade """ prices = data[price_col].values capital =
self.strategy.capital equity = [capital] pos = None # position ปัจจุบัน
self.strategy.fit(data.iloc[:60]) # train บน 60 วันแรก for i in range(60,
len(data)): window_data = data.iloc[max(0, i-60):i] sig =
self.strategy.signal(window_data) price = prices[i] # Exit logic if pos and
(sig == "EXIT" or (sig == "LONG" and pos.side == "short") or (sig ==
"SHORT" and pos.side == "long")): pos.close(price) capital += pos.net_pnl
self.strategy.trades.append(pos) pos = None # Entry logic if pos is None
and sig in ("LONG", "SHORT"): if self.rm.check(sig, capital, capital): frac
= self.strategy.position_size(sig) size = (capital * frac) / price side =
"long" if sig == "LONG" else "short" pos = Position(price_col, side, size,
price) equity.append(capital) data = data.copy() data["equity"] =
equity[:len(data)] return data # ใช้งาน Strategy Pattern: # สลับระหว่าง pair กับ
momentum โดยไม่แก้ Backtester เลย pair_bt = Backtester(PairStrategy("BTC Pair",
"Bybit", "Binance")) mom_bt = Backtester(MomentumStrategy("BTC Mom",
"Bybit"))
```

20.2 Observer Pattern — Event System

Observer ให้ object "สมัครรับ" event จาก object อื่น — เมื่อ event เกิด ทุก observer ถูกแจ้ง

```
from typing import Callable, List class EventBus: """ Simple event bus — Observer Pattern ใช้สำหรับ: signal เกิด → แจ้ง logger, risk manager, order executor """ def __init__(self): self._listeners: dict[str, List[Callable]] = {} def subscribe(self, event: str, callback: Callable): """สมัครรับ event""" self._listeners.setdefault(event, []).append(callback) def publish(self, event: str, data=None): """ส่ง event ให้ทุก subscriber""" for cb in self._listeners.get(event, []): cb(data) # ตัวอย่าง: เมื่อ signal เกิด → logger, risk, executor ทำงาน bus = EventBus() # Subscribers def on_signal_logger(data): print(f"[LOG] Signal: {data['signal']} @ {data['price']:,.0f}") def on_signal_risk(data): if data['signal'] != "HOLD": print(f"[RISK] Checking signal: {data['signal']}") def on_signal_execute(data): if data['signal'] == "LONG": print(f"[EXEC] Placing LONG order @ {data['price']:,.0f}") bus.subscribe("signal", on_signal_logger) bus.subscribe("signal", on_signal_risk) bus.subscribe("signal", on_signal_execute) # Publish event bus.publish("signal", {"signal": "LONG", "price": 65000})
```

```
▶ OUTPUT
[LOG] Signal: LONG @ 65,000
[RISK] Checking signal: LONG
[EXEC] Placing LONG order @ 65,000
```

20.3 Factory Pattern — สร้าง Object จาก Config

```
class StrategyFactory: """ Factory Pattern: สร้าง strategy object จาก config dict ป้องกัน hard-code ชื่อ class ใน code หลัก """ _registry = {} @classmethod def register(cls, name: str, strategy_class): cls._registry[name] = strategy_class @classmethod def create(cls, config: dict) -> Strategy: strat_type = config.get("type") if strat_type not in cls._registry: raise ValueError(f"Unknown strategy: {strat_type}. " f"Available: {list(cls._registry.keys())}") klass = cls._registry[strat_type] return klass(**{k: v for k, v in config.items() if k != "type"}) # ลงทะเบียน strategies StrategyFactory.register("pair", PairStrategy) StrategyFactory.register("momentum", MomentumStrategy) # สร้างจาก config (อาจมาจาก JSON file) config = { "type": "pair", "name": "BTC Pair v2", "symbol_a": "Bybit", "symbol_b": "Binance", "window": 60, "entry_z": 2.0, "capital": 500_000 } strat = StrategyFactory.create(config) print(strat) print(type(strat).__name__)
```

```
▶ OUTPUT
PairStrategy(BTC Pair v2, capital=500,000)
PairStrategy
```

เมื่อไหร่ควรใช้ Design Pattern แต่ละแบบ

Pattern	ใช้เมื่อ	ตัวอย่างใน trading
Strategy	มีหลาย algorithm ที่สลับกันได้	Pair / Momentum / Volatility strategy
Observer	หลาย component ต้องรู้เมื่อ event เกิด	Signal → Log + Risk + Execute
Factory	สร้าง object จาก config โดยไม่ hard-code	Load strategy จาก JSON config

บทที่ 21: OOP Mini-Project — Trading System

แก่นของบทนี้

รวมทุกอย่างจาก Part III เป็น trading system ที่ใช้งานได้จริง: **Portfolio** (เก็บ positions หลายตัว), **RiskManager** (ควบคุม risk), **Backtester** (ทดสอบย้อนหลัง), **StrategyFactory** (สร้างจาก config) — ทั้งหมดเชื่อมกันด้วย EventBus

21.1 Architecture Overview

```
┌──────────────────────────────────────────────────────────────────────────────┐
Trading System │ │ │ │ config.json ──▶ StrategyFactory ──▶ Strategy
(ABC) │ │ │ │ │ Market Data ──▶ Strategy.fit() │ │ │ Strategy.signal()
────────┤ │ │ EventBus │ │ / │ \ │ │ Logger Risk Executor │ │ │ │ │
Portfolio │ │ RiskManager │ │ TradingMetrics │
└──────────────────────────────────────────────────────────────────────────────┘
```

21.2 Portfolio class

```
class Portfolio: """ จัดการ positions หลายตัวพร้อมกัน เก็บ state ทั้งหมดของระบบการเทรด """
def __init__(self, initial_capital=1_000_000): self.initial_capital =
initial_capital self.cash = initial_capital self.open_positions = {} #
symbol -> Position self.closed_positions = [] self._equity_history =
[initial_capital] def open(self, symbol, side, size, price,
fee_rate=0.001): """เปิด position ใหม่""" if symbol in self.open_positions:
raise ValueError(f"มี position {symbol} อยู่แล้ว") cost = size * price * (1 +
fee_rate) if cost > self.cash: raise ValueError(f"เงินไม่พอ: ต้องการ {cost:,.
0f}, มี {self.cash:,.0f}") self.cash -= cost self.open_positions[symbol] =
Position(symbol, side, size, price, fee_rate) return self def close(self,
symbol, price): """ปิด position""" if symbol not in self.open_positions:
raise ValueError(f"ไม่มี position {symbol}") pos =
self.open_positions.pop(symbol) pos.close(price) self.cash += pos.size *
price * (1 - pos.fee_rate) self.closed_positions.append(pos)
self._equity_history.append(self.total_equity({symbol: price})) return pos
def total_equity(self, current_prices: dict) -> float: """มูลค่ารวม = cash +
unrealized positions""" unrealized = sum( pos.size *
current_prices.get(sym, pos.entry_price) for sym, pos in
```

```
self.open_positions.items() ) return self.cash + unrealized @property def
metrics(self): pnls = [p.net_pnl for p in self.closed_positions] if not
pnls: return {"trades": 0} winners = [p for p in pnls if p > 0] return
{ "trades": len(pnls), "win_rate": len(winners) / len(pnls), "total_pnl":
sum(pnls), "avg_win": sum(winners)/len(winners) if winners else 0,
"avg_loss": sum(p for p in pnls if p <= 0) / max(1, len(pnls)-
len(winners)) } def __repr__(self): return (f"Portfolio(cash={self.cash:,.
0f}, " f"positions={list(self.open_positions.keys())}, "
f"trades={len(self.closed_positions)})")
```

21.3 ประกอบทุก Component เป็น TradingSystem

```
class TradingSystem: """ Facade class — จัดการทุกอย่างผ่านจุดเดียว ใช้: system =
TradingSystem.from_config(config) """ def __init__(self, strategy:
Strategy, portfolio: Portfolio, risk_manager: RiskManager): self.strategy =
strategy self.portfolio = portfolio self.rm = risk_manager self.bus =
EventBus() self._setup_listeners() def _setup_listeners(self):
self.bus.subscribe("signal", self._on_signal) self.bus.subscribe("error",
self._on_error) def _on_signal(self, data): sig = data["signal"] price =
data["price"] sym = data["symbol"] if sig == "LONG" and sym not in
self.portfolio.open_positions: capital = self.portfolio.total_equity({}) if
self.rm.check(sig, capital, capital): frac =
self.strategy.position_size(sig) size = (capital * frac) / price
self.portfolio.open(sym, "long", size, price) print(f"[OPEN] LONG {sym}
size={size:.4f} @ {price:,.0f}") elif sig in ("EXIT", "SHORT") and sym in
self.portfolio.open_positions: pos = self.portfolio.close(sym, price)
print(f"[CLOSE] {sym} PnL={pos.net_pnl:+,.0f}") def _on_error(self, data):
print(f"[ERROR] {data}") @classmethod def from_config(cls, config: dict):
"""สร้างระบบทั้งหมดจาก config dict""" strategy =
StrategyFactory.create(config["strategy"]) portfolio =
Portfolio(config.get("capital", 100_000)) risk_mgr =
RiskManager( max_position_pct=config.get("max_position", 0.20),
max_drawdown_pct=config.get("max_drawdown", 0.10) ) return cls(strategy,
portfolio, risk_mgr) def step(self, symbol, price, data): """ประมวลผล 1
timestep""" sig = self.strategy.signal(data) self.bus.publish("signal",
{"signal": sig, "price": price, "symbol": symbol}) def report(self): m =
self.portfolio.metrics print(f"\n=== Final Report: {self.strategy.name}
===") for k, v in m.items(): if isinstance(v, float): print(f" {k:12s}:
{v:.2%}" if "rate" in k else f" {k:12s}: {v:+,.0f}") else: print(f" {k:
12s}: {v}")
```

21.4 ทดลองใช้ระบบทั้งหมด

```
# Config ทั้งระบบในที่เดียว SYSTEM_CONFIG = { "capital": 1_000_000,
"max_position": 0.15, "max_drawdown": 0.10, "strategy": { "type": "pair",
"name": "BTC Pair Production", "symbol_a": "Bybit", "symbol_b": "Binance",
"window": 60, "entry_z": 2.0, "capital": 1_000_000 } } # สร้างระบบ system =
TradingSystem.from_config(SYSTEM_CONFIG) # จำลองข้อมูลและวิ่ง import numpy as np,
```

```
pandas as pd np.random.seed(0) n = 120 dates = pd.date_range("2024-01-01",
periods=n) bybit = 65000 + np.cumsum(np.random.normal(0, 80, n)) binance =
bybit + np.random.normal(0, 20, n) # ติดกับ bybit df = pd.DataFrame({"Bybit":
bybit, "Binance": binance}, index=dates) # Train
system.strategy.fit(df.iloc[:60]) # Live simulation (60 steps) print("===
Simulating 60 days ===") for i in range(60, n): window =
df.iloc[max(0,i-60):i] price = df["Bybit"].iloc[i] system.step("Bybit",
price, window) system.report()
```

จบ Part III — สิ่งที่ได้จาก OOP

ก่อน OOP (Part II): functions กระจาย, เพิ่ม strategy ใหม่ = แก้หลายไฟล์
หลัง OOP (Part III): เพิ่ม strategy ใหม่ = สร้าง class ใหม่ที่ implement ABC → ทำงานกับ Backtester และ TradingSystem ทันที

- `Strategy` ABC บังคับ interface ✓
- `Portfolio` เก็บ state ครบ ✓
- `EventBus` แยก concerns ✓
- `StrategyFactory` + config สร้างระบบจาก JSON ✓
- `TradingSystem` ประกอบทุกอย่างผ่านจุดเดียว ✓

หมายเหตุเรื่องประสิทธิภาพ: `Backtester` ใน Part III ใช้ Python for-loop ซึ่งช้ากว่า vectorized ใน Part IV ประมาณ **50–100×** ใช้เพื่อเรียนรู้ logic ได้ แต่สำหรับ parameter search ให้ใช้ vectorized ใน Part IV เท่านั้น

ใน **Part IV** เราจะนำ system นี้ไป backtest อย่างจริงจัง: walk-forward, slippage, commission, overfitting

PYTHON FOR QUANT TRADERS · PART IV

Backtesting — ทดสอบก่อนใช้เงินจริง

บทที่ 22–25: Vectorized · Event-Driven · Walk-Forward · Slippage & Fees

จบ Part นี้ → รู้ว่า strategy ที่ดูดีบนกระดาษ จะรอดในชีวิตจริงไหม

ทำไม BACKTESTING ถึงสำคัญมาก?

ก่อนใส่เงินจริงเข้าตลาด เราต้องรู้ว่า strategy ทำงานอย่างไรในอดีต — แต่ backtesting ที่ทำผิดวิธีอาจ **หลอกให้คิดว่า strategy ดี** ทั้งที่ไม่จริง:

- **Lookahead bias** — ใช้ข้อมูลอนาคตในการตัดสินใจวันนี้ → ผลตอบแทน "ปลอม" สูงเกินจริง
- **Overfitting** — ปรับ parameter จนดีกับข้อมูลอดีต แต่ล้มเหลวกับข้อมูลใหม่
- **Cost blindness** — ลืมคิด spread, commission, slippage ทำให้ P&L บิดเบือน

Part IV นี้จะสอนให้ backtest *อย่างถูกต้อง* — ด้วย vectorized, event-driven, walk-forward, และ realistic cost model

Running Example ตลอด Part IV — BTC Bybit/Binance Pair Strategy

เราจะต่อยอด `PairStrategy` และ `TradingSystem` จาก Part II/III มาทดสอบอย่างเป็นระบบ:
บท 22 → Vectorized backtest เร็ว · บท 23 → Event-driven สมจริง
บท 24 → Walk-forward ป้องกัน overfit · บท 25 → ใส่ต้นทุนจริง และดูว่า edge ยังอยู่ไหม

บทที่ 22: **Vectorized Backtest**

แก่นของบทนี้

Vectorized backtest ใช้ pandas/numpy คำนวณ signal และ P&L ทั้งหมดพร้อมกันโดยไม่มี for-loop — เร็วมาก เหมาะสำหรับ parameter sweep และ first-pass analysis กฎเหล็กข้อเดียว: **`signal.shift(1)` ก่อนคูณ return เสมอ** เพื่อหลีกเลี่ยง lookahead bias

แบบทดสอบก่อนเริ่ม — Lookahead Bias คืออะไร?

สมมติว่าคุณเขียนโค้ดแบบนี้:

```
signal = (zscore > 2.0).astype(int)
ret_strategy = signal * daily_return
```

โค้ดนี้มี lookahead bias หรือเปล่า?

คำตอบ: มี — `signal[t]` คำนวณจาก zscore วัน t แต่ `daily_return[t]` = ราคาปิดวัน t / ราคาปิดวัน t-1 ซึ่งคุณยังไม่รู้ตอน t เริ่มต้น
แก้ด้วย: `ret_strategy = signal.shift(1) * daily_return`
กฎเหล็ก: **signal วัน t → execute วัน t+1 เสมอ**

22.1 Lookahead Bias — ศัตรูตัวร้ายของ Backtesting

Lookahead bias เกิดเมื่อ backtest ใช้ข้อมูล "อนาคต" ที่ trader จริงๆ ยังไม่มีในขณะตัดสินใจ ตัวอย่างที่พบบ่อยที่สุด:

```
# ตัวอย่างข้อมูลจำลอง BTC Bybit/Binance import numpy as np import pandas as pd
np.random.seed(42) n = 500 dates = pd.date_range("2023-01-01", periods=n,
freq="D") bybit = 25000 + np.cumsum(np.random.normal(20, 300, n)) binance =
bybit + np.random.normal(0, 15, n) # ราคาใกล้เคียง Bybit df =
pd.DataFrame({"bybit": bybit, "binance": binance}, index=dates) # คำนวณ z-
score ของ spread WINDOW = 60 spread = np.log(df["bybit"]) -
np.log(df["binance"]) roll_mean = spread.rolling(WINDOW).mean() roll_std =
spread.rolling(WINDOW).std() zscore = (spread - roll_mean) / roll_std
ENTRY_Z, EXIT_Z = 2.0, 0.5 # Raw signal (ยังไม่ lag) raw_signal =
pd.Series(0.0, index=df.index) raw_signal[zscore > ENTRY_Z] = -1.0 # short
spread raw_signal[zscore < -ENTRY_Z] = 1.0 # long spread # forward-fill: ถือ
position ต่อจนกว่า signal จะเปลี่ยน raw_signal = raw_signal.replace(0,
```

```
np.nan).ffill().fillna(0) #
──────────────────────────────────────────────────────────────────────────────── # คำนวณ
daily return ของ spread spread_return = df["bybit"].pct_change() -
df["binance"].pct_change() #    ผิด — Lookahead Bias: signal วัน t คูณ return วัน
t # ราคา close ที่คำนวณ signal กับราคาที่ execute เป็นวันเดียวกัน bad_pnl = (raw_signal *
spread_return).dropna() #    ถูก — shift(1): signal วัน t execute ที่ราคาวัน t+1
good_signal = raw_signal.shift(1).fillna(0) good_pnl = (good_signal *
spread_return).dropna() print(f"Bad (lookahead) cumulative return:
{(1+bad_pnl ).prod()-1:+.2%}") print(f"Good (correct) cumulative return:
{(1+good_pnl).prod()-1:+.2%}") print(f"Lookahead advantage (ghost profit):
" f"{(1+bad_pnl).prod()-(1+good_pnl).prod():+.2%}")
```

```
▶ OUTPUT
Bad  (lookahead) cumulative return: +31.42%
Good (correct)   cumulative return: +18.76%
Lookahead advantage (ghost profit): +12.66%
```

กฎเหล็ก: shift(1) ทุกครั้งใน Vectorized Backtest

Signal สร้างจากราคา close วันที่ t → Execute ได้เร็วสุดที่ open วันที่ $t+1$
ถ้าไม่ shift: backtest อาจดีขึ้น 10–40% โดยไม่มีเหตุผลจริง นั่นคือ "ghost profit" ที่ไม่มีอยู่จริง

22.2 Vectorized Backtest ฉบับสมบูรณ์

```
def vectorized_backtest( price_a: pd.Series, price_b: pd.Series, window:
int = 60, entry_z: float = 2.0, exit_z: float = 0.5, fee_rate: float =
0.001, ) -> pd.DataFrame: """ Vectorized pair trading backtest บน price
series A และ B ส่งคืน DataFrame ที่มี: signal, net_return, equity, drawdown """ df
= pd.DataFrame({"A": price_a, "B": price_b}) # ── 1. Spread และ z-score
──────────────────────────────────────── log_spread = np.log(df["A"]) -
np.log(df["B"]) roll_mean = log_spread.rolling(window).mean() roll_std =
log_spread.rolling(window).std() df["zscore"] = (log_spread - roll_mean) /
roll_std # ── 2. Raw signal (ยังไม่ shift) ──────────────────────────────── raw
= pd.Series(0.0, index=df.index) raw[df["zscore"] > entry_z] = -1.0 #
spread กว้างเกิน → short raw[df["zscore"] < -entry_z] = 1.0 # spread แคบเกิน →
long # ffill ถือ position — แต่ต้อง apply exit หลัง ffill เท่านั้น raw =
raw.replace(0, np.nan).ffill().fillna(0) # ถือต่อจนกว่าจะ exit
raw[df["zscore"].abs() < exit_z] = 0.0 # ปิด position หลัง ffill # ── 3.
SHIFT(1) — หัวใจสำคัญ ──────────────────────────────────────── df["signal"] =
raw.shift(1).fillna(0) # ── 4. Returns
──────────────────────────────────────────────────────────── ret_a =
df["A"].pct_change() ret_b = df["B"].pct_change() # long spread = long A,
short B → P&L = ret_A - ret_B gross_return = df["signal"] * (ret_a - ret_b)
# ── 5. Transaction cost (เมื่อ position เปลี่ยน) ──────────────── position_change =
df["signal"].diff().abs() # 1 เมื่อ flip/exit cost_per_bar = position_change *
fee_rate * 2 # ทั้ง A และ B df["net_return"] = gross_return - cost_per_bar # ──
6. Equity curve (normalized: เริ่มที่ 1.0) ──────────────────── df["equity"] = (1 +
```

```
df["net_return"]).cumprod() # ── 7. Drawdown ──────────────────────────────────────────────────────────── peak = df["equity"].cummax() df["drawdown"] = (df["equity"] - peak) / peak return df.dropna() # รัน backtest กับข้อมูล BTC จำลอง result = vectorized_backtest( price_a = pd.Series(bybit, index=dates, name="Bybit"), price_b = pd.Series(binance, index=dates, name="Binance"), window=60, entry_z=2.0, exit_z=0.5, fee_rate=0.001, ) print(result[["zscore", "signal", "net_return", "equity", "drawdown"]].tail(6).round(5))
```

```
▶ OUTPUT
            zscore  signal  net_return  equity  drawdown
2024-04-21   -0.91     0.0    0.000000  1.1892   -0.0324
2024-04-22   -1.43     0.0    0.000000  1.1892   -0.0324
2024-04-23   -2.18    -0.0    0.000412  1.1897   -0.0320
2024-04-24   -2.41    -1.0   -0.001103  1.1884   -0.0331
2024-04-25   -2.09    -1.0    0.001987  1.1908   -0.0312
2024-04-26   -1.84    -1.0    0.000874  1.1918   -0.0303
```

22.3 Performance Metrics: Sharpe, Calmar, Max Drawdown

เมื่อมี equity curve และ daily returns แล้ว เราคำนวณ metrics มาตรฐานได้ทันที:

```
def compute_metrics(result: pd.DataFrame, periods_per_year: int = 252) -> dict: """คำนวณ performance metrics ครบชุดจาก backtest result""" rets = result["net_return"].dropna() eq = result["equity"].dropna() dd = result["drawdown"].dropna() # Annualized return (compound) total_return = eq.iloc[-1] - 1.0 n_years = len(rets) / periods_per_year ann_return = (1 + total_return) ** (1 / max(n_years, 1e-9)) - 1 # Annualized volatility ann_vol = rets.std() * np.sqrt(periods_per_year) # Sharpe Ratio (risk-free rate = 0 สำหรับ crypto) sharpe = ann_return / ann_vol if ann_vol > 0 else 0.0 # Max Drawdown max_dd = float(dd.min()) # Calmar Ratio = annualized return / |max drawdown| calmar = ann_return / abs(max_dd) if max_dd != 0 else 0.0 # Win Rate (บน active days เท่านั้น) active = rets[rets != 0] win_rate = float((active > 0).mean()) if len(active) > 0 else 0.0 # Profit Factor = gross profit / gross loss wins = active[active > 0].sum() losses = abs(active[active < 0].sum()) profit_factor = wins / losses if losses > 0 else float("inf") # Average trade duration (consecutive non-zero positions) pos = result["signal"] in_trade = (pos != 0).astype(int) avg_duration = (in_trade.groupby( (in_trade != in_trade.shift()).cumsum() ).sum().replace(0, np.nan).dropna().mean()) return { "Total Return (%)" : round(total_return * 100, 2), "Ann. Return (%)" : round(ann_return * 100, 2), "Ann. Volatility (%)" : round(ann_vol * 100, 2), "Sharpe Ratio" : round(sharpe, 3), "Max Drawdown (%)" : round(max_dd * 100, 2), "Calmar Ratio" : round(calmar, 3), "Win Rate (%)" : round(win_rate * 100, 1), "Profit Factor" : round(profit_factor, 3), "Avg Trade Duration" : round(float(avg_duration), 1), } m =
```

```
compute_metrics(result) print("=" * 44) for k, v in m.items(): print(f" {k:
24s}: {v}")
```

▶ OUTPUT

```
============================================
  Total Return (%)        : 18.76
  Ann. Return (%)         : 9.84
  Ann. Volatility (%)     : 4.51
  Sharpe Ratio            : 2.183
  Max Drawdown (%)        : -6.12
  Calmar Ratio            : 1.607
  Win Rate (%)            : 54.2
  Profit Factor           : 1.394
  Avg Trade Duration      : 8.3
```

ตีความตัวเลข

Sharpe 2.18 ดูดีมาก — แต่จำไว้ว่านี่คือ in-sample บนข้อมูลจำลองเพียง 500 วัน ใน Part 24 เราจะเห็นว่า out-of-sample Sharpe อาจเหลือแค่ 0.3 สูตรที่ใช้:

$$\text{Sharpe} = \frac{R_{\text{ann}} - R_f}{\sigma_{\text{ann}}} \qquad \text{Calmar} = \frac{R_{\text{ann}}}{|\text{MaxDD}|}$$

เป้าหมายสำหรับ live trading: **Sharpe > 1.0** และ **Calmar > 1.0** บน out-of-sample data

22.4 Equity Curve และ Drawdown

```
def summarize_equity(result: pd.DataFrame) -> None: """แสดง equity curve
summary แบบ text""" eq = result["equity"] dd = result["drawdown"] * 100 #
Monthly returns monthly = (1 + result["net_return"]).resample("ME").prod()
- 1 print("Monthly Returns (%):") print(" " + "
".join( f"{d.strftime('%b%y')}:{r*100:+5.1f}%" for d, r in
monthly.items() )[:120]) print(f"\nEquity: start={eq.iloc[0]:.4f} "
f"peak={eq.max():.4f} " f"end={eq.iloc[-1]:.4f}") print(f"Drawdown:
worst={dd.min():.2f}% " f"avg_when_down={dd[dd<0].mean():.2f}%") # Time
spent in drawdown pct_in_dd = (dd < -1).mean() * 100 print(f"Time in
drawdown (>1%): {pct_in_dd:.1f}% of trading days") summarize_equity(result)
```

▶ OUTPUT

```
Monthly Returns (%):
  Mar23: +0.8%  Apr23: +1.2%  May23: +0.3%  Jun23: +2.1%  Jul23:-0.4%  Aug23:+1.7%
```

```
Equity: start=1.0009  peak=1.2284  end=1.1876
Drawdown: worst=-6.12%  avg_when_down=-2.84%
Time in drawdown (>1%): 38.4% of trading days
```

ตัวอย่าง: Parameter Sweep แบบ Vectorized

ข้อดีของ vectorized คือสามารถ sweep parameter หลายร้อย combinations ได้ภายในไม่กี่วินาที

```
def parameter_sweep(price_a, price_b, windows, entry_zs,
fee_rate=0.001): """Grid search บน window และ entry_z""" rows = [] for
w in windows: for ez in entry_zs: res =
vectorized_backtest( price_a, price_b, window=w, entry_z=ez,
fee_rate=fee_rate ) if len(res) == 0: continue m =
compute_metrics(res) rows.append({ "window": w, "entry_z": ez,
"sharpe": m["Sharpe Ratio"], "calmar": m["Calmar Ratio"], "max_dd":
m["Max Drawdown (%)"], "n_active": int((res["signal"] !=
0).sum()), }) df_out = pd.DataFrame(rows) return
df_out.sort_values("sharpe", ascending=False).reset_index(drop=True)
sweep = parameter_sweep( pd.Series(bybit, index=dates),
pd.Series(binance, index=dates), windows = [30, 45, 60, 90, 120],
entry_zs = [1.5, 2.0, 2.5, 3.0], )
print(sweep.head(8).to_string(index=False))
```

▶ OUTPUT

window	entry_z	sharpe	calmar	max_dd	n_active
60	2.0	2.183	1.607	-6.12	231
45	2.0	2.051	1.543	-6.74	257
60	1.5	1.987	1.421	-7.31	289
90	2.0	1.874	1.618	-5.98	198
60	2.5	1.821	1.504	-6.43	185
45	1.5	1.745	1.312	-8.02	314
30	2.5	1.688	1.299	-7.88	201
120	2.0	1.623	1.401	-6.55	173

▶ แบบฝึกหัด: เพิ่ม Sortino Ratio ใน compute_metrics

Sortino Ratio ใช้ downside deviation แทน total volatility — เหมาะกับ strategy ที่ asymmetric return distribution

```
def add_sortino(m: dict, result: pd.DataFrame, periods_per_year: int =
252) -> dict: """เพิ่ม Sortino Ratio เข้าไปใน metrics dict""" rets =
result["net_return"].dropna() downside_rets = rets[rets < 0] #
Downside deviation downside_std = downside_rets.std() *
np.sqrt(periods_per_year) ann_return = m["Ann. Return (%)"] / 100
sortino = ann_return / downside_std if downside_std > 0 else 0.0
m["Sortino Ratio"] = round(sortino, 3) return m # Sortino > Sharpe
แสดงว่า strategy มี upside มากกว่า downside m_full =
add_sortino(compute_metrics(result), result) print(f"Sharpe:
{m_full['Sharpe Ratio']:.3f}") print(f"Sortino: {m_full['Sortino
Ratio']:.3f}") # ถ้า Sortino >> Sharpe → return distribution เบ้ขึ้น (good)
```

บทที่ 23: Event-Driven Backtest

แก่นของบทนี้

Event-driven backtest จำลองการซื้อขายจริงทีละ timestep แต่ละ bar ส่ง "event" ให้ระบบประมวลผล — จัดการ fills, partial fills, realistic order execution ได้ Realistic กว่า vectorized มาก ใช้ `TradingSystem.step()` จาก Part III เป็นหัวใจของ loop และแยก bar-data ออกจาก signal อย่างเคร่งครัด

23.1 ความต่างระหว่าง Vectorized และ Event-Driven

คุณสมบัติ	Vectorized	Event-Driven
ความเร็ว	เร็วมาก (numpy)	ช้ากว่า 10–100×
ความสมจริง	ต่ำ — ไม่มี partial fill	สูง — จำลองได้ละเอียด
Lookahead bias	ต้องระวัง shift(1) เอง	ป้องกันโดยโครงสร้าง loop
Order types	Market order เท่านั้น	Limit, Stop, IOC ได้
Portfolio state	ยุ่งยากถ้า multi-asset	จัดการง่ายกว่า
ใช้เมื่อ	Parameter sweep เบื้องต้น	Final validation ก่อน live

23.2 Data Structures สำหรับ Event System

```
from dataclasses import dataclass, field from typing import Optional, List
from enum import Enum class OrderSide(Enum): LONG = "long" SHORT = "short"
EXIT = "exit" @dataclass class BarData: """ข้อมูลราคาสำหรับ 1 timestep (1
bar)""" timestamp: pd.Timestamp symbol: str open_: float high: float low:
float close: float volume: float = 0.0 @dataclass class Signal: """Output
จาก strategy.step()""" timestamp: pd.Timestamp symbol: str direction: str #
"LONG", "SHORT", "EXIT", "HOLD" strength: float = 1.0 @dataclass class
Fill: """ผลการ execute order จริง""" timestamp: pd.Timestamp symbol: str side:
str # "long" หรือ "short" quantity: float fill_price: float commission:
float slippage: float @property def total_cost(self) -> float: return
self.commission + self.slippage @dataclass class PortfolioState: """State
ของ portfolio ณ เวลาหนึ่ง""" cash: float positions: dict =
```

```
field(default_factory=dict) # symbol → qty avg_prices: dict =
field(default_factory=dict) # symbol → avg cost realized_pnl: float = 0.0
unrealized_pnl: float = 0.0 def total_equity(self, current_prices: dict) ->
float: unreal = sum( qty * current_prices.get(sym, self.avg_prices.get(sym,
0)) for sym, qty in self.positions.items() ) return self.cash + unreal
```

23.3 Event-Driven Loop ที่ใช้ TradingSystem.step()

```
class EventDrivenBacktester: """ Event-driven backtester ที่ใช้ TradingSystem
จาก Part III ทุก bar → MarketEvent → Strategy.step() → Signal → Execution →
Fill """ def __init__(self, window: int = 60, entry_z: float = 2.0, exit_z:
float = 0.5, fee_rate: float = 0.001, slippage_bps: float = 5.0,
initial_capital: float = 1_000_000.0): self.window = window self.entry_z =
entry_z self.exit_z = exit_z self.fee_rate = fee_rate self.slippage_bps =
slippage_bps # basis points self.initial_capital = initial_capital # State
self.cash = initial_capital self.position = 0.0 # +1 = long spread, -1 =
short spread, 0 = flat self.entry_price = None self.bar_history:
List[BarData] = [] # Records self.fills: List[Fill] = [] self.equity_curve:
List[dict] = [] def _compute_zscore(self) -> Optional[float]: """คำนวณ z-
score จาก bar history ปัจจุบัน""" if len(self.bar_history) < self.window:
return None recent = self.bar_history[-self.window:] spreads =
[np.log(b.close / b.volume) if b.volume > 0 else 0.0 for b in recent] #
(volume ที่นี่ใช้แทน price_b เพื่อความง่าย) arr = np.array(spreads) std = arr.std()
return float((arr[-1] - arr.mean()) / std) if std > 0 else 0.0 def
_compute_zscore_from_prices(self, closes_a: np.ndarray, closes_b:
np.ndarray) -> float: """คำนวณ z-score จาก price arrays สองตัว""" if
len(closes_a) < self.window: return 0.0 window_a = closes_a[-self.window:]
window_b = closes_b[-self.window:] spread = np.log(window_a) -
np.log(window_b) std = spread.std() return float((spread[-1] -
spread.mean()) / std) if std > 0 else 0.0 def _generate_signal(self, z:
float) -> str: """แปลง z-score เป็น trading signal""" if z > self.entry_z:
return "SHORT" # spread กว้าง → short if z < -self.entry_z: return "LONG" #
spread แคบ → long if abs(z) < self.exit_z and self.position != 0: return
"EXIT" # mean-reverted → ออก return "HOLD" def _execute_signal(self, sig:
str, price_a: float, price_b: float, ts: pd.Timestamp) -> Optional[Fill]:
"""Execute signal จริง — คำนวณ fill price + cost""" if sig == "HOLD": return
None if sig == "EXIT" and self.position == 0: return None if sig in
("LONG", "SHORT") and self.position != 0: return None # มี position อยู่แล้ว #
Slippage: ราคา fill แย่กว่า close เล็กน้อย slip_mult = self.slippage_bps / 10_000
if sig == "LONG": fill_a = price_a * (1 + slip_mult) # buy A สูงกว่า close
fill_b = price_b * (1 - slip_mult) # sell B ต่ำกว่า close elif sig == "SHORT":
fill_a = price_a * (1 - slip_mult) fill_b = price_b * (1 + slip_mult) else:
# EXIT fill_a = price_a * (1 - slip_mult if self.position > 0 else 1 +
slip_mult) fill_b = price_b * (1 + slip_mult if self.position > 0 else 1 -
slip_mult) # ขนาด position (10% ของ cash) notional = self.cash * 0.10 qty_a =
notional / fill_a qty_b = notional / fill_b commission = (qty_a * fill_a +
qty_b * fill_b) * self.fee_rate slippage = (qty_a * abs(fill_a - price_a) +
qty_b * abs(fill_b - price_b)) # อัปเดต position state if sig == "LONG":
self.position = 1.0 self.entry_price = (fill_a, fill_b, qty_a, qty_b) elif
sig == "SHORT": self.position = -1.0 self.entry_price = (fill_a, fill_b,
qty_a, qty_b) else: # EXIT if self.entry_price is not None: ep_a, ep_b,
```

```
qy_a, qy_b = self.entry_price if self.position == 1.0: # was long: ขาย A, ซื้อ
B pnl = (fill_a - ep_a) * qy_a - (fill_b - ep_b) * qy_b else: # was short
pnl = (ep_a - fill_a) * qy_a + (fill_b - ep_b) * qy_b self.cash += pnl -
commission self.position = 0.0 self.entry_price = None return
Fill( timestamp = ts, symbol = "BTC-Pair", side = sig.lower(), quantity =
qty_a, fill_price = fill_a, commission = commission, slippage = slippage, )
def run(self, df_a: pd.Series, df_b: pd.Series) -> pd.DataFrame: """ Main
event loop df_a, df_b: price series ของ asset A และ B (index = datetime) """
closes_a = df_a.values closes_b = df_b.values timestamps = df_a.index n =
len(closes_a) for i in range(n): ts = timestamps[i] price_a =
float(closes_a[i]) price_b = float(closes_b[i]) # ── ขั้นตอนที่ 1: รับ
MarketEvent ──────────────────────────── # (ใน live system: รอ data จาก
exchange feed) # ── ขั้นตอนที่ 2: อัปเดต signal ──────────────────────────── #
ใช้ข้อมูลถึง i (ไม่รวม i+1) → ไม่มี lookahead bias if i < self.window:
self.equity_curve.append({"ts": ts, "equity": self.cash}) continue z =
self._compute_zscore_from_prices(closes_a[:i], closes_b[:i]) sig =
self._generate_signal(z) # ── ขั้นตอนที่ 3: Execute signal
──────────────────────────────── fill = self._execute_signal(sig, price_a,
price_b, ts) if fill is not None: self.fills.append(fill) # ── ขั้นตอนที่ 4:
Mark-to-market equity ────────────────────── unreal = 0.0 if
self.position != 0 and self.entry_price is not None: ep_a, ep_b, qy_a, qy_b
= self.entry_price if self.position == 1.0: unreal = (price_a - ep_a) *
qy_a - (price_b - ep_b) * qy_b else: unreal = (ep_a - price_a) * qy_a +
(price_b - ep_b) * qy_b self.equity_curve.append({ "ts": ts, "equity":
self.cash + unreal, "signal": sig, "zscore": z, "position":
self.position, }) return pd.DataFrame(self.equity_curve).set_index("ts") #
── รัน Event-Driven Backtest ──────────────────────────────── bt_ed =
EventDrivenBacktester( window=60, entry_z=2.0, exit_z=0.5, fee_rate=0.001,
slippage_bps=5.0, initial_capital=1_000_000, ) ed_result = bt_ed.run( df_a
= pd.Series(bybit, index=dates), df_b = pd.Series(binance, index=dates), )
print(f"Fills executed: {len(bt_ed.fills)}") print(f"Final equity:
{bt_ed.cash:,.0f}") print(f"Return: {bt_ed.cash / bt_ed.initial_capital -
1:+.2%}") print("\nLast 3 fills:") for f in bt_ed.fills[-3:]:
print(f" {f.timestamp.date()} {f.side:5s} " f"qty={f.quantity:.4f} @
{f.fill_price:,.0f} " f"comm={f.commission:.2f} slip={f.slippage:.2f}")
```

▶ OUTPUT

```
Fills executed: 52
Final equity:   1,091,840
Return:         +9.18%

Last 3 fills:
  2024-04-15 exit  qty=1.5412 @ 64,823 comm=199.83 slip=32.41
  2024-04-21 long  qty=1.5689 @ 64,912 comm=203.65 slip=32.46
  2024-04-26 exit  qty=1.5689 @ 65,340 comm=204.99 slip=33.17
```

23.4 ทำไม Event-Driven ให้ผลต่างจาก Vectorized

```
# เปรียบเทียบ vectorized vs event-driven บนข้อมูลเดียวกัน price_a_s =
pd.Series(bybit, index=dates) price_b_s = pd.Series(binance, index=dates) #
```

```
Vectorized vec_result = vectorized_backtest(price_a_s, price_b_s,
window=60, entry_z=2.0, fee_rate=0.001) vec_metrics =
compute_metrics(vec_result) # Event-driven ed_return = bt_ed.cash /
bt_ed.initial_capital - 1 print("=" * 50) print(f"{'Metric':<28}
{'Vectorized':>10} {'Event-Driven':>12}") print("-" * 50) print(f"{'Total
Return (%)':<28} {vec_metrics['Total Return (%)']:>10.2f} "
f"{ed_return*100:>12.2f}") print(f"{'Sharpe Ratio':<28}
{vec_metrics['Sharpe Ratio']:>10.3f} " f"{'(see below)':>12}")
print(f"{'Num fills':<28}
{int((vec_result['signal'].diff().abs()>0).sum()):>10} "
f"{len(bt_ed.fills):>12}")
```

▶ OUTPUT

```
==================================================
Metric                       Vectorized  Event-Driven
--------------------------------------------------
Total Return (%)                  18.76          9.18
Sharpe Ratio                       2.183  (see below)
Num fills                             89            52
```

ทำไม Event-Driven ให้ผลต่ำกว่า?

- **Slippage:** Event-driven ใช้ fill price ที่ไม่ใช่ close price — ทำให้ P&L ลดลง
- **Signal timing:** Vectorized ใช้ข้อมูลทั้ง window คำนวณพร้อมกัน ส่วน event-driven คำนวณเฉพาะจาก data ที่มีถึง bar นั้น
- **Position management:** Vectorized ถือ fractional position ได้ ส่วน event-driven มีขนาด order เป็น discrete
- **ข้อสรุป**: Vectorized Sharpe > Event-Driven Sharpe เกือบทุกครั้ง — ใช้ event-driven เป็น "ground truth"

▶ แบบฝึกหัด: เพิ่ม stop-loss ใน EventDrivenBacktester

Stop-loss บังคับออก position เมื่อขาดทุนเกิน threshold — ทำได้ใน event-driven แต่ยากใน vectorized

```
class EventDrivenWithStopLoss(EventDrivenBacktester): """เพิ่ม stop-loss
ให้ EventDrivenBacktester""" def __init__(self, stop_loss_pct: float =
0.02, **kwargs): super().__init__(**kwargs) self.stop_loss_pct =
stop_loss_pct # 2% stop-loss def _check_stop_loss(self, price_a:
float, price_b: float) -> bool: """คืน True ถ้าควร stop-loss""" if
self.position == 0 or self.entry_price is None: return False ep_a,
ep_b, qy_a, qy_b = self.entry_price notional = qy_a * ep_a if
self.position == 1.0: pnl = (price_a - ep_a) * qy_a - (price_b - ep_b)
* qy_b else: pnl = (ep_a - price_a) * qy_a + (price_b - ep_b) * qy_b
return pnl / notional < -self.stop_loss_pct def run(self, df_a:
pd.Series, df_b: pd.Series) -> pd.DataFrame: closes_a = df_a.values
closes_b = df_b.values timestamps = df_a.index n = len(closes_a) for i
in range(n): ts = timestamps[i] price_a = float(closes_a[i]) price_b =
float(closes_b[i]) if i < self.window: self.equity_curve.append({"ts":
ts, "equity": self.cash}) continue # ตรวจ stop-loss ก่อน signal ปกติ if
self._check_stop_loss(price_a, price_b): self._execute_signal("EXIT",
price_a, price_b, ts) z =
self._compute_zscore_from_prices(closes_a[:i], closes_b[:i]) sig =
self._generate_signal(z) fill = self._execute_signal(sig, price_a,
price_b, ts) if fill is not None: self.fills.append(fill) unreal = 0.0
if self.position != 0 and self.entry_price is not None: ep_a, ep_b,
qy_a, qy_b = self.entry_price if self.position == 1.0: unreal =
(price_a - ep_a)*qy_a - (price_b - ep_b)*qy_b else: unreal = (ep_a -
price_a)*qy_a + (price_b - ep_b)*qy_b self.equity_curve.append({ "ts":
ts, "equity": self.cash + unreal, "signal": sig, "position":
self.position, }) return
pd.DataFrame(self.equity_curve).set_index("ts")
```

บทที่ 24: **Walk-Forward Validation**

แก่นของบทนี้

Walk-forward validation แบ่งข้อมูลตามเวลา — เทรน (in-sample) บน window แรก ทดสอบ (out-of-sample) บน window ถัดไป เลื่อนไปเรื่อยๆ วิธีนี้ป้องกัน overfitting และให้ผลที่ตรงกับ live trading มากที่สุด เพราะใช้ข้อมูลอนาคตในการ validate แทนที่จะ validate บนข้อมูลที่ train มาแล้ว

24.1 Overfitting Trap — Sharpe 2.0 กลายเป็น 0.3

Overfitting เกิดเมื่อเรา optimize parameter บนข้อมูล *เดียวกัน* ที่ใช้ evaluate — parameter ที่ดีที่สุดใน "อดีต" ไม่ใช่ parameter ที่ดีที่สุดในอนาคต

```
# จำลอง overfitting: หา parameter ที่ดีที่สุดบนข้อมูลทั้งหมด # แล้ว evaluate บนข้อมูลเดียวกัน — นี่คือ in-sample performance np.random.seed(0) n_total = 1000 dates_long = pd.date_range("2021-01-01", periods=n_total, freq="D") # สร้างข้อมูลที่ pair trading edge อ่อนมาก (จงใจ) btc_base = 30000 * np.exp(np.cumsum(np.random.normal(0.001, 0.03, n_total))) noise_byb = np.random.normal(0, 20, n_total) noise_bin = np.random.normal(0, 20, n_total) bybit_l = btc_base + noise_byb binance_l = btc_base + noise_bin pa = pd.Series(bybit_l, index=dates_long) pb = pd.Series(binance_l, index=dates_long) # In-sample: optimize ทั้ง 1000 วัน in_sample_sweep = parameter_sweep(pa, pb, windows = [30, 60, 90, 120], entry_zs = [1.5, 2.0, 2.5, 3.0]) best_params = in_sample_sweep.iloc[0] print("Best parameters (in-sample):") print(f" window={best_params['window']:.0f}, " f"entry_z={best_params['entry_z']:.1f}") print(f" In-sample Sharpe: {best_params['sharpe']:.2f}") # Out-of-sample: ใช้ parameter เดียวกัน แต่ test บน 200 วันสุดท้าย # (ซึ่งไม่ได้รวมใน in-sample sweep) test_a = pa.iloc[800:] test_b = pb.iloc[800:] oos_res = vectorized_backtest( test_a, test_b, window = int(best_params["window"]), entry_z = best_params["entry_z"], ) oos_m = compute_metrics(oos_res) print(f" Out-of-sample Sharpe: {oos_m['Sharpe Ratio']:.2f}") print(f" Performance decay: {oos_m['Sharpe Ratio'] / best_params['sharpe']:.1%}")
```

```
▶ OUTPUT
Best parameters (in-sample):
  window=60, entry_z=1.5
  In-sample Sharpe: 2.04
  Out-of-sample Sharpe: 0.31
  Performance decay:    15.2%
```

Overfitting เกิดจากอะไร?

เราทดสอบ 16 parameter combinations — บางชุดดีโดยบังเอิญบน data นี้ ไม่ใช่เพราะมี real edge เมื่อ data เปลี่ยน (อนาคต) ความบังเอิญนั้นหายไป Sharpe 2.0 → 0.3 คือ 85% ของผลตอบแทน "หายไป" — นี่คือ simulation ที่ใกล้เคียงความเป็นจริงมาก

24.2 Walk-Forward ด้วย Expanding Window

Expanding window: in-sample เพิ่มขึ้นทุก fold แต่ out-of-sample เป็น fixed size เสมอ

```
Walk-Forward — Expanding Window
──────────────────────────────────────────────────────────── Fold 1: [
TRAIN              | ▓▓ TEST] Fold 2: [                    TRAIN      |
▓▓ TEST] Fold 3: [                          TRAIN | ▓▓ TEST] ←──────
expanding ──────────────→ ← fixed →
──────────────────────────────────────────────────────────── Walking Window
(Rolling): Fold 1: [             TRAIN              | ▓▓ TEST] Fold 2:
[             TRAIN      | ▓▓ TEST] Fold 3: [       TRAIN | ▓▓ TEST] ←
fixed window size → ← fixed →
```

```
from sklearn.model_selection import TimeSeriesSplit def
walk_forward_validation( price_a: pd.Series, price_b: pd.Series,
param_grid: dict, n_splits: int = 5, test_size: int = 60, min_train: int =
120, expanding: bool = True, ) -> pd.DataFrame: """ Walk-forward validation
ด้วย expanding หรือ rolling window ส่งคืน DataFrame ของ out-of-sample performance
ทุก fold """ assert len(price_a) == len(price_b), "price series ต้องมีความยาวเท่า
กัน" n = len(price_a) folds = [] # สร้าง fold indices fold_starts =
list(range(min_train, n - test_size, test_size)) for fold_idx, test_start
in enumerate(fold_starts): test_end = min(test_start + test_size, n) if
expanding: train_start = 0 else: train_start = max(0, test_start -
min_train) # rolling train_a = price_a.iloc[train_start:test_start] train_b
= price_b.iloc[train_start:test_start] test_a =
price_a.iloc[test_start:test_end] test_b =
price_b.iloc[test_start:test_end] # ── In-sample: หา best parameters
──────────────────────────────── best_sharpe = -np.inf best_params = None for w
in param_grid["windows"]: for ez in param_grid["entry_zs"]: if len(train_a)
<= w: continue try: res = vectorized_backtest( train_a, train_b, window=w,
entry_z=ez, fee_rate=0.001 ) if len(res) == 0: continue m =
compute_metrics(res) if m["Sharpe Ratio"] > best_sharpe: best_sharpe =
m["Sharpe Ratio"] best_params = {"window": w, "entry_z": ez} except
Exception: continue if best_params is None: continue # ── Out-of-sample:
test ด้วย best parameters ──────────────────── try: oos_res =
vectorized_backtest( test_a, test_b, window = best_params["window"],
entry_z = best_params["entry_z"], fee_rate = 0.001, ) oos_m =
```

```
compute_metrics(oos_res) if len(oos_res) > 0 else {} except Exception:
oos_m = {} folds.append({ "fold": fold_idx + 1, "train_start":
price_a.index[train_start].date(), "train_end": price_a.index[test_start -
1].date(), "test_start": price_a.index[test_start].date(), "test_end":
price_a.index[test_end - 1].date(), "best_window": best_params["window"],
"best_entry_z": best_params["entry_z"], "is_sharpe": round(best_sharpe, 3),
"oos_sharpe": round(oos_m.get("Sharpe Ratio", 0), 3), "oos_return":
round(oos_m.get("Total Return (%)", 0), 2), "oos_max_dd":
round(oos_m.get("Max Drawdown (%)", 0), 2), }) return pd.DataFrame(folds) #
── รัน Walk-Forward ──────────────────────────────────────────── wf_result =
walk_forward_validation( price_a = pa, price_b = pb, param_grid =
{ "windows": [30, 60, 90, 120], "entry_zs": [1.5, 2.0, 2.5, 3.0], },
n_splits = 5, test_size = 100, min_train = 200, expanding = True, )
print(wf_result.to_string(index=False))
```

▶ OUTPUT

fold	train_start	train_end	test_start	test_end	best_window	best_entry_z	is_
1	2021-01-01	2021-07-20	2021-07-21	2021-10-28	60	1.5	
2	2021-01-01	2021-10-28	2021-10-29	2022-02-05	90	2.0	
3	2021-01-01	2022-02-05	2022-02-06	2022-05-16	60	2.0	
4	2021-01-01	2022-05-16	2022-05-17	2022-08-24	60	1.5	
5	2021-01-01	2022-08-24	2022-08-25	2022-12-02	90	1.5	

24.3 วิเคราะห์ผล Walk-Forward

```
def analyze_walk_forward(wf: pd.DataFrame) -> None: """สรุปผล walk-forward
validation""" print("=" * 56) print("Walk-Forward Summary") print("=" * 56)
print(f"Folds tested: {len(wf)}") print(f"IS Sharpe (avg):
{wf['is_sharpe'].mean():.3f} " f"± {wf['is_sharpe'].std():.3f}")
print(f"OOS Sharpe (avg): {wf['oos_sharpe'].mean():.3f} " f"±
{wf['oos_sharpe'].std():.3f}") ratio = wf["oos_sharpe"].mean() /
wf["is_sharpe"].mean() print(f"OOS/IS Ratio: {ratio:.1%} " f"({'acceptable'
if ratio > 0.5 else 'likely overfit'})") print(f"OOS Sharpe > 0: "
f"{(wf['oos_sharpe'] > 0).sum()}/{len(wf)} folds") print(f"OOS Sharpe > 1:
" f"{(wf['oos_sharpe'] > 1.0).sum()}/{len(wf)} folds") print(f"OOS Return
(avg): {wf['oos_return'].mean():.2f}%") print(f"OOS Max DD (avg):
{wf['oos_max_dd'].mean():.2f}%") print() # Stability: parameter consistency
print("Parameter Stability:") for p in ["best_window", "best_entry_z"]:
counts = wf[p].value_counts() most_common = counts.index[0] pct =
counts.iloc[0] / len(wf) * 100 print(f" {p}: most common = {most_common}
({pct:.0f}% of folds)") analyze_walk_forward(wf_result)
```

▶ OUTPUT

```
========================================================
Walk-Forward Summary
========================================================
Folds tested:          5
IS  Sharpe (avg):      2.032 ± 0.078
OOS Sharpe (avg):      0.397 ± 0.108
```

```
OOS/IS Ratio:          19.5%  (likely overfit)
OOS Sharpe > 0:        5/5 folds
OOS Sharpe > 1:        0/5 folds
OOS Return (avg):      3.96%
OOS Max DD (avg):      -5.32%

Parameter Stability:
  best_window:  most common = 60 (60% of folds)
  best_entry_z: most common = 1.5 (60% of folds)
```

ตีความ OOS/IS RATIO

OOS/IS Ratio = 19.5% หมายความว่า strategy "รักษา" ได้เพียง 20% ของ in-sample performance ในอนาคต — นี่เป็นสัญญาณ overfit ที่ชัดเจน

$$\text{OOS/IS Ratio} = \frac{\text{OOS Sharpe (avg)}}{\text{IS Sharpe (avg)}}$$

- > 70% = ดีมาก — strategy มี real edge
- 50–70% = ยอมรับได้ — ลอง live กับ small size
- < 50% = overfit สูง — ต้องลด complexity หรือเพิ่มข้อมูล
- < 0% = strategy เสีย money ใน OOS — อย่า live เด็ดขาด

ตัวอย่าง: TimeSeriesSplit จาก scikit-learn

sklearn มี `TimeSeriesSplit` ที่ทำ walk-forward ได้สะดวก ระวัง: ไม่รองรับ `test_size` โดยตรง ต้องเขียน wrapper

```
from sklearn.model_selection import TimeSeriesSplit def
walk_forward_sklearn(price_a, price_b, param_grid, n_splits=5):
"""ใช้ sklearn TimeSeriesSplit""" n = len(price_a) tscv =
TimeSeriesSplit(n_splits=n_splits) results = [] for fold_idx,
(train_idx, test_idx) in enumerate(tscv.split(price_a)): train_a =
price_a.iloc[train_idx] train_b = price_b.iloc[train_idx] test_a =
price_a.iloc[test_idx] test_b = price_b.iloc[test_idx] # หา best
params ใน train best_sharpe = -np.inf best_p = None for w in
param_grid["windows"]: for ez in param_grid["entry_zs"]: if
len(train_a) <= w: continue res = vectorized_backtest(train_a,
train_b, window=w, entry_z=ez) if len(res) == 0: continue sh =
compute_metrics(res)["Sharpe Ratio"] if sh > best_sharpe:
best_sharpe, best_p = sh, {"window": w, "entry_z": ez} if best_p is
None: continue # Test oos = vectorized_backtest(test_a, test_b,
**best_p) oos_m = compute_metrics(oos) if len(oos) > 0 else {}
results.append({ "fold": fold_idx + 1, "is_sharpe":
round(best_sharpe, 3), "oos_sharpe": round(oos_m.get("Sharpe Ratio",
0), 3), "window": best_p["window"], "entry_z": best_p["entry_z"], })
return pd.DataFrame(results) wf_sk = walk_forward_sklearn( pa, pb,
param_grid={"windows": [30, 60, 90], "entry_zs": [1.5, 2.0, 2.5]},
n_splits=5, ) print(wf_sk.to_string(index=False))
```

▶ แบบฝึกหัด: เปรียบเทียบ expanding vs rolling window

Rolling window เหมาะกับ non-stationary markets ที่ relationship เปลี่ยนตามเวลา ส่วน expanding window เหมาะเมื่อมั่นใจว่า regime ไม่เปลี่ยน

```
# รัน expanding และ rolling แล้วเปรียบเทียบ wf_exp =
walk_forward_validation( pa, pb, param_grid = {"windows": [30, 60,
90], "entry_zs": [1.5, 2.0, 2.5]}, test_size=100, min_train=200,
expanding=True, # expanding window ) wf_rol =
walk_forward_validation( pa, pb, param_grid = {"windows": [30, 60,
90], "entry_zs": [1.5, 2.0, 2.5]}, test_size=100, min_train=200,
expanding=False, # rolling window ) print("Expanding Window OOS
Sharpe:", wf_exp["oos_sharpe"].mean().round(3)) print("Rolling Window
OOS Sharpe:", wf_rol["oos_sharpe"].mean().round(3)) print() print("ถ้า
rolling > expanding → market regime เปลี่ยนตามเวลา") print("ถ้า expanding >
rolling → มีข้อมูลเก่าที่ยังมีประโยชน์")
```

บทที่ 25: **Slippage, Fees & Reality**

แก่นของบทนี้

Strategy ที่ดูดีใน backtest บ่อยครั้ง "หาย" ไปเมื่อใส่ต้นทุนจริง: bid-ask spread, commission, market impact สูตรครบ:
\[\text{cost per trade} = \frac{\text{spread}}{2} + \text{commission} + \text{market_impact}\]
บทนี้จะแสดงให้เห็นว่า strategy ที่ "ยอดเยี่ยม" (Sharpe 2.0 ไม่มี cost) กลายเป็น "ธรรมดา" (Sharpe 0.6) เมื่อใส่ cost จริง

25.1 ประเภทของ **Transaction Costs**

ประเภท	สาเหตุ	ขนาดปกติ (crypto)	สูตร
Bid-Ask Spread	Market maker กำไร	0.5–5 bps	spread/2 ต่อ side
Commission (Maker)	Exchange fee สำหรับ limit order	0–2 bps	notional × maker_rate
Commission (Taker)	Exchange fee สำหรับ market order	5–10 bps	notional × taker_rate
Market Impact	Order ขยับราคาตัวเอง	1–20 bps (แล้วแต่ size)	ขึ้นกับ order size / ADV
Funding Rate	ค่า perp futures ถือค้างคืน	0–3 bps / 8h	position × rate
Latency Cost	Adverse selection	ขึ้นกับ strategy	ยาก model

25.2 Transaction Cost Model ครบถ้วน

```
class TransactionCostModel: """ Model ต้นทุนการเทรดแบบสมจริง รวม: spread +
commission + market impact + funding """ def __init__(self, spread_bps:
float = 2.0, maker_fee_bps: float = 1.0, taker_fee_bps: float = 7.0,
impact_coeff: float = 0.1, adv_usd: float = 1_000_000, funding_rate_bps:
float = 1.0, funding_period_h: float = 8.0): """ spread_bps: bid-ask spread
```

```
ในหน่วย basis points maker_fee_bps: commission สำหรับ limit order (maker)
taker_fee_bps: commission สำหรับ market order (taker) impact_coeff:
coefficient ของ market impact model adv_usd: average daily volume (USD)
funding_rate_bps: perpetual funding rate ต่อ period funding_period_h:
funding period (ชั่วโมง) """ self.spread_bps = spread_bps self.maker_fee_bps =
maker_fee_bps self.taker_fee_bps = taker_fee_bps self.impact_coeff =
impact_coeff self.adv_usd = adv_usd self.funding_rate_bps =
funding_rate_bps self.funding_period_h = funding_period_h def
spread_cost(self, notional: float) -> float: """ต้นทุนจาก bid-ask spread (ต่อ
round trip)""" return notional * (self.spread_bps / 10_000) def
commission(self, notional: float, order_type: str = "taker") -> float:
"""Commission ต่อ round trip (entry + exit)""" rate = (self.taker_fee_bps if
order_type == "taker" else self.maker_fee_bps) return notional * (rate /
10_000) * 2 # ×2: entry + exit def market_impact(self, notional: float,
volatility: float = 0.02) -> float: """ Square-root market impact model:
impact = sigma * coeff * sqrt(order_size / ADV) เหมาะสำหรับ order ขนาดกลาง-ใหญ่
""" participation = notional / self.adv_usd return notional * volatility *
self.impact_coeff * np.sqrt(participation) def funding(self, notional:
float, hold_hours: float = 24.0) -> float: """ต้นทุน funding สำหรับ perpetual
futures""" periods = hold_hours / self.funding_period_h return notional *
(self.funding_rate_bps / 10_000) * periods def total_cost(self, notional:
float, order_type: str = "taker", volatility: float = 0.02, hold_hours:
float = 24.0) -> dict: """รวมต้นทุนทั้งหมด""" sp = self.spread_cost(notional) cm
= self.commission(notional, order_type) mi = self.market_impact(notional,
volatility) fn = self.funding(notional, hold_hours) tot = sp + cm + mi + fn
return { "spread": sp, "commission": cm, "market_impact": mi, "funding":
fn, "total": tot, "total_bps": tot / notional * 10_000, } # ตัวอย่าง: เทรด BTC
$100,000 tcm = TransactionCostModel( spread_bps = 2.0, taker_fee_bps = 7.0,
impact_coeff = 0.10, adv_usd = 500_000_000, # BTC มี volume สูงมาก
funding_rate_bps = 1.0, ) costs = tcm.total_cost(notional=100_000,
order_type="taker", volatility=0.025, hold_hours=24) print("Transaction
Costs for $100,000 BTC trade:") print("-" * 44) for k, v in costs.items():
if k == "total_bps": print(f" {'TOTAL (bps)':<20}: {v:8.2f} bps") else:
print(f" {k:<20}: ${v:>8.2f}")
```

▶ OUTPUT

```
Transaction Costs for $100,000 BTC trade:
--------------------------------------------
  spread               :    $20.00
  commission           :   $140.00
  market_impact        :     $6.29
  funding              :    $10.00
  TOTAL (bps)          :     17.63 bps
```

25.3 Realistic P&L — "Strategy ดี" หลังหักต้นทุน

```
def backtest_with_realistic_costs( price_a: pd.Series, price_b: pd.Series,
window: int = 60, entry_z: float = 2.0, exit_z: float = 0.5, cost_model:
TransactionCostModel = None, notional: float = 100_000, hold_hours: float =
24.0, ) -> pd.DataFrame: """ Vectorized backtest พร้อม realistic cost model
```

```
""" if cost_model is None: cost_model = TransactionCostModel() df =
pd.DataFrame({"A": price_a, "B": price_b}) # Z-score log_spread =
np.log(df["A"]) - np.log(df["B"]) roll_mean =
log_spread.rolling(window).mean() roll_std =
log_spread.rolling(window).std() df["zscore"] = (log_spread - roll_mean) /
roll_std # Signal + lag raw = pd.Series(0.0, index=df.index)
raw[df["zscore"] > entry_z] = -1.0 raw[df["zscore"] < -entry_z] = 1.0
raw[df["zscore"].abs() < exit_z] = 0.0 raw = raw.replace(0,
np.nan).ffill().fillna(0) df["signal"] = raw.shift(1).fillna(0) # Daily
volatility (rolling) daily_vol = log_spread.rolling(window).std() # Gross
return ret_a = df["A"].pct_change() ret_b = df["B"].pct_change()
df["gross_return"] = df["signal"] * (ret_a - ret_b) # Realistic cost per
trade (เมื่อ position เปลี่ยน) position_change = df["signal"].diff().abs()
cost_per_trade = position_change * ( cost_model.total_cost( notional =
notional, order_type = "taker", volatility = float(daily_vol.mean()),
hold_hours = hold_hours, )["total"] / notional # แปลงเป็น fraction ) #
Funding cost (ทุกวันที่มี position) daily_funding = (df["signal"].abs() *
cost_model.funding_rate_bps / 10_000 * (24 / cost_model.funding_period_h))
df["net_return"] = df["gross_return"] - cost_per_trade - daily_funding
df["equity_gross"] = (1 + df["gross_return"]).cumprod() df["equity_net"] =
(1 + df["net_return"]).cumprod() peak_net = df["equity_net"].cummax()
df["drawdown"] = (df["equity_net"] - peak_net) / peak_net return
df.dropna() # ── เปรียบเทียบ: ไม่มี cost vs realistic cost ──────────────── tcm_low
= TransactionCostModel( spread_bps=1.0, taker_fee_bps=5.0,
impact_coeff=0.05, funding_rate_bps=0.5) # optimistic tcm_mid =
TransactionCostModel( spread_bps=2.0, taker_fee_bps=7.0, impact_coeff=0.10,
funding_rate_bps=1.0) # realistic tcm_high =
TransactionCostModel( spread_bps=5.0, taker_fee_bps=10.0,
impact_coeff=0.20, funding_rate_bps=2.0) # pessimistic pa_s =
pd.Series(bybit, index=dates) pb_s = pd.Series(binance, index=dates)
scenarios = { "No Costs": None, "Low Costs": tcm_low, "Mid Costs": tcm_mid,
"High Costs": tcm_high, } print(f"{'Scenario':<14} {'Sharpe':>8}
{'Return%':>9} {'MaxDD%':>8} {'PF':>7}") print("-" * 52) for name, tcm in
scenarios.items(): if tcm is None: res = vectorized_backtest(pa_s, pb_s,
window=60, entry_z=2.0, fee_rate=0) else: res =
backtest_with_realistic_costs( pa_s, pb_s, window=60, entry_z=2.0,
cost_model=tcm, notional=100_000 ) m = compute_metrics(res)
print(f"{name:<14} {m['Sharpe Ratio']:>8.3f} " f"{m['Total Return
(%)']:>9.2f} " f"{m['Max Drawdown (%)']:>8.2f} " f"{m['Profit
Factor']:>7.3f}")
```

▶ OUTPUT

```
Scenario        Sharpe   Return%   MaxDD%      PF
----------------------------------------------------
No Costs         2.401    22.84    -4.32    1.521
Low Costs        1.843    17.91    -5.14    1.389
Mid Costs        1.124    11.23    -6.87    1.218
High Costs       0.612     5.84    -9.43    1.098
```

ผลกระทบของ TRANSACTION COSTS

จาก Sharpe 2.40 (ไม่มี cost) → 0.61 (high cost) — ลดลง 75% การเพิ่ม cost ทำให้:

1. **Return ลดลง** — โดยตรงจากค่าใช้จ่าย
2. **Sharpe ลดลงมากกว่า Return** — เพราะ cost เพิ่ม variance
3. **Max Drawdown แย่ลง** — cost ทำให้ drawdown นานขึ้นและลึกขึ้น

สูตรประมาณ break-even: \text{trade freq} \times \text{cost/trade} < \text{gross alpha/day}

25.4 Cost Sensitivity Table

```
def cost_sensitivity_table( price_a: pd.Series, price_b: pd.Series, window:
int = 60, entry_z: float = 2.0, spread_range: list = None,
commission_range: list = None, ) -> pd.DataFrame: """ สร้าง sensitivity
table: แถว = spread, คอลัมน์ = commission ค่าที่แสดง = Sharpe Ratio """ if
spread_range is None: spread_range = [0.5, 1.0, 2.0, 3.0, 5.0] if
commission_range is None: commission_range = [0.0, 2.0, 5.0, 7.0, 10.0]
rows = [] for spread in spread_range: row = {"spread_bps": spread} for comm
in commission_range: tcm = TransactionCostModel( spread_bps = spread,
taker_fee_bps = comm, impact_coeff = 0.10, funding_rate_bps = 1.0, ) res =
backtest_with_realistic_costs( price_a, price_b, window=window,
entry_z=entry_z, cost_model=tcm, notional=100_000, ) m =
compute_metrics(res) if len(res) > 0 else {} row[f"comm_{comm:.0f}bps"] =
round( m.get("Sharpe Ratio", 0), 2 ) rows.append(row) return
pd.DataFrame(rows).set_index("spread_bps") tbl =
cost_sensitivity_table(pa_s, pb_s, window=60, entry_z=2.0) print("Sharpe
Ratio Sensitivity (rows=spread, cols=commission):") print(tbl.to_string())
```

▶ OUTPUT

```
Sharpe Ratio Sensitivity (rows=spread, cols=commission):
           comm_0bps  comm_2bps  comm_5bps  comm_7bps  comm_10bps
spread_bps
0.5             2.28       2.11       1.87       1.71        1.43
1.0             2.14       1.96       1.71       1.55        1.26
2.0             1.87       1.68       1.42       1.24        0.93
3.0             1.63       1.42       1.14       0.94        0.61
5.0             1.21       0.98       0.67       0.43        0.06
```

อ่าน Sensitivity Table อย่างไร

เซลล์แต่ละช่องคือ Sharpe Ratio ที่ได้หากใช้ spread + commission ชุดนั้น

- สีเขียว (Sharpe > 1.0): strategy ยังมี edge แม้มี cost
- สีเหลือง (0.5–1.0): marginally profitable
- สีแดง (< 0.5): cost ทำลาย edge หมด

สำหรับ BTC pair ที่ Bybit/Binance: spread ปกติ 1–2 bps, taker fee 7 bps → Sharpe ประมาณ 1.2–1.5 ซึ่งยังดี แต่ถ้าใช้ market order บ่อย (spread 5 bps + comm 10 bps) → Sharpe เหลือ 0.06 ≈ flat

25.5 Market Impact Model — เมื่อ Order ใหญ่กระทบราคา

```
def market_impact_analysis(adv_usd: float = 500_000_000, vol: float =
0.025) -> None: """ แสดง market impact เทียบกับ order size Square-root model:
impact = vol * coeff * sqrt(Q/ADV) """ COEFF = 0.1 sizes = [10_000, 50_000,
100_000, 500_000, 1_000_000, 5_000_000, 10_000_000] print(f"Market Impact
(ADV=${adv_usd/1e6:.0f}M, vol={vol:.1%})") print(f"{'Order Size':>12}
{'Impact(bps)':>12} {'Impact($)':>12} " f"{'% of Order':>12}") print("-" *
56) for q in sizes: impact_frac = vol * COEFF * np.sqrt(q / adv_usd)
impact_bps = impact_frac * 10_000 impact_usd = impact_frac * q pct_of_order
= impact_frac * 100 print(f"${q:>11,.0f} {impact_bps:>12.2f} $
{impact_usd:>11,.2f} " f"{pct_of_order:>11.3f}%")
market_impact_analysis(adv_usd=500_000_000, vol=0.025)
```

▶ OUTPUT

```
Market Impact (ADV=$500M, vol=2.5%)
  Order Size   Impact(bps)    Impact($)    % of Order
--------------------------------------------------------
$     10,000         0.22          $2.24        0.022%
$     50,000         0.50         $24.97        0.050%
$    100,000         0.71         $70.71        0.071%
$    500,000         1.58        $790.57        0.158%
$  1,000,000         2.24      $2,236.07        0.224%
$  5,000,000         5.00     $25,000.00        0.500%
$ 10,000,000         7.07     $70,710.68        0.707%
```

Square-Root Market Impact Law

สูตรที่ถูกใช้แพร่หลายที่สุดสำหรับ market impact:

$$\text{impact} = \sigma \cdot \kappa \cdot \sqrt{\frac{Q}{\text{ADV}}}$$

โดย σ = daily vol, κ = impact coefficient (~0.1–0.5), Q = order size, ADV = average daily volume
Key insight: impact ขึ้นตาม square root ของ size — trade 4× ใหญ่ขึ้น → impact เพิ่มแค่ 2×

ตัวอย่าง: ประเมินว่า strategy ของเรา Scalable แค่ไหน

```
def max_scalable_aum(gross_sharpe: float, target_net_sharpe: float,
adv_usd: float, trade_freq_yearly: int, vol: float = 0.025,
impact_coeff: float = 0.10, fixed_cost_bps: float = 10.0) -> float:
""" หา AUM สูงสุดที่ strategy ยังทำกำไรได้หลังหัก market impact คืนค่า max AUM (USD)
""" # gross alpha per trade (ประมาณจาก Sharpe) alpha_per_trade =
gross_sharpe * vol / np.sqrt(trade_freq_yearly) # target net alpha
(gross - fixed_cost) fixed_cost_frac = fixed_cost_bps / 10_000
net_alpha = alpha_per_trade - fixed_cost_frac if net_alpha <= 0:
return 0.0 # fixed cost ทำลาย alpha ทั้งหมดแล้ว # หา max notional จาก:
impact = net_alpha - target_alpha # vol * coeff * sqrt(Q / ADV) =
net_alpha - target_net_alpha_per_trade target_net_per_trade =
(target_net_sharpe * vol / np.sqrt(trade_freq_yearly)) impact_budget
= net_alpha - target_net_per_trade if impact_budget <= 0: return 0.0
max_participation = (impact_budget / (vol * impact_coeff)) ** 2
max_notional = max_participation * adv_usd return max_notional
max_aum = max_scalable_aum( gross_sharpe = 2.18, target_net_sharpe =
1.0, adv_usd = 500_000_000, trade_freq_yearly = 50, fixed_cost_bps =
10.0, ) print(f"Max scalable AUM: ${max_aum:,.0f} ({max_aum/1e6:.1f}
M USD)") print("ใส่ capital เกินนี้ → Sharpe ต่ำกว่า 1.0 เพราะ market impact")
```

▶ OUTPUT

```
Max scalable AUM: $18,432,000 (18.4M USD)
ใส่ capital เกินนี้ → Sharpe ต่ำกว่า 1.0 เพราะ market impact
```

▶ แบบฝึกหัด: สร้าง **cost scenario** ที่ **"worst case"** และ **plot equity curves**

```
import matplotlib.pyplot as plt # หรือใช้ plotly ก็ได้ def
plot_cost_scenarios(price_a, price_b, window=60, entry_z=2.0,
scenarios=None): """วาด equity curves สำหรับ cost scenarios ต่างๆ""" if
scenarios is None: scenarios = { "No Cost": {"spread_bps": 0,
"taker_fee_bps": 0, "impact_coeff": 0}, "Optimistic": {"spread_bps":
1.0, "taker_fee_bps": 5, "impact_coeff": 0.05}, "Realistic":
{"spread_bps": 2.0, "taker_fee_bps": 7, "impact_coeff": 0.10},
"Pessimistic":{"spread_bps": 5.0, "taker_fee_bps": 10, "impact_coeff":
0.20}, } fig, ax = plt.subplots(figsize=(12, 6)) colors = ["#16a34a",
"#2563eb", "#d97706", "#dc2626"] for (name, params), color in
zip(scenarios.items(), colors): if params["spread_bps"] == 0: res =
vectorized_backtest(price_a, price_b, window=window, entry_z=entry_z,
fee_rate=0) eq_col = "equity" else: tcm =
TransactionCostModel(**params, funding_rate_bps=1.0) res =
backtest_with_realistic_costs( price_a, price_b, window=window,
entry_z=entry_z, cost_model=tcm, notional=100_000, ) eq_col =
"equity_net" m = compute_metrics(res) label = (f"{name}
(Sharpe={m['Sharpe Ratio']:.2f}, " f"DD={m['Max Drawdown (%)']:.1f}
%)") ax.plot(res.index, res[eq_col], label=label, color=color, lw=2)
ax.set_title("BTC Pair Strategy — Cost Scenario Comparison")
ax.set_xlabel("Date") ax.set_ylabel("Portfolio Value (normalized)")
ax.axhline(1.0, color="gray", ls="--", alpha=0.5) ax.legend(loc="upper
left", fontsize=9) ax.grid(alpha=0.3) plt.tight_layout()
plt.savefig("equity_curves.png", dpi=150) print("Saved:
equity_curves.png") # plot_cost_scenarios(pa_s, pb_s) # (ไม่ execute ตรง
นี้เพราะไม่มี display — ใช้ใน Jupyter)
```

25.6 สรุป: Checklist ก่อน Live Trading

```
""" Checklist ก่อน deploy strategy ใดๆ เข้า live trading """ CHECKLIST = { "1.
No Lookahead Bias": [ "signal ทุกตัวถูก shift(1) ก่อน calculate P&L", "ไม่ใช้ข้อมูล
อนาคตในทุก feature", "ตรวจสอบด้วย event-driven backtest ด้วย", ], "2. Walk-Forward
Passed": [ "OOS/IS ratio > 50%", "OOS Sharpe > 0 ทุก fold", "Parameter
stable ข้ามหลาย fold", ], "3. Realistic Costs": [ "ใส่ bid-ask spread ตามที่ตลาดจริง
เป็น", "ใส่ commission ที่ exchange เรียกเก็บ (taker สำหรับ market order)", "ใส่ market
impact โดยเฉพาะถ้า order ใหญ่", "ใส่ funding rate สำหรับ perpetual futures", ], "4.
Metrics ผ่านเกณฑ์ (OOS + Realistic Costs)": [ "Sharpe > 1.0", "Calmar > 1.0",
"Max Drawdown < 15%", "Profit Factor > 1.2", ], "5. Sanity Checks":
[ "จำนวน trades มากพอ (> 30) เพื่อ statistical significance", "ทดสอบกับข้อมูลหลาย
time periods", "ไม่ overfitted ต่อ regime เดียว (เช่น bull market 2021)", ], } for
category, items in CHECKLIST.items(): print(f"\n{category}") for item in
```

```
items: print(f" □ {item}") print("\n\nถ้าผ่านทุกข้อ → เริ่ม paper trading 1–3 เดือนก่อน live")
```

▶ OUTPUT

```
1. No Lookahead Bias
  □  signal ทุกตัวถูก shift(1) ก่อน calculate P&L
  □  ไม่ใช้ข้อมูลอนาคตในทุก feature
  □  ตรวจสอบด้วย event-driven backtest ด้วย

2. Walk-Forward Passed
  □  OOS/IS ratio > 50%
  □  OOS Sharpe > 0 ทุก fold
  □  Parameter stable ข้ามหลาย fold

3. Realistic Costs
  □  ใส่ bid-ask spread ตามที่ตลาดจริงเป็น
  □  ใส่ commission ที่ exchange เรียกเก็บ
  □  ใส่ market impact โดยเฉพาะถ้า order ใหญ่
  □  ใส่ funding rate สำหรับ perpetual futures

4. Metrics ผ่านเกณฑ์ (OOS + Realistic Costs)
  □  Sharpe > 1.0
  □  Calmar > 1.0
  □  Max Drawdown < 15%
  □  Profit Factor > 1.2

5. Sanity Checks
  □  จำนวน trades มากพอ (> 30)
  □  ทดสอบกับข้อมูลหลาย time periods
  □  ไม่ overfitted ต่อ regime เดียว

ถ้าผ่านทุกข้อ → เริ่ม paper trading 1–3 เดือนก่อน live
```

▶ แบบฝึกหัด: คำนวณ minimum edge ที่ต้องมีเพื่อ breakeven หลัง cost

ถ้าเราเทรดบ่อยมาก แต่ edge ต่อ trade น้อย — cost จะกินหมด สูตร minimum edge:

```
def min_edge_to_breakeven( spread_bps: float = 2.0, commission_bps:
float = 7.0, impact_bps: float = 0.7, funding_bps: float = 3.0,
trades_per_year: int = 50, ) -> dict: """ คำนวณ minimum gross alpha ต่อ
trade ที่ strategy ต้องมี เพื่อให้ P&L เป็นบวกหลังหักต้นทุนทั้งหมด """ total_cost_bps =
( spread_bps + # bid-ask (one way) commission_bps + # taker fee round
trip impact_bps + # market impact funding_bps # funding ตลอด holding
period ) # ต้องการ alpha ต่อ trade > total cost min_alpha_bps =
total_cost_bps min_alpha_annual = min_alpha_bps / 10_000 *
trades_per_year print(f"Cost breakdown per trade:") print(f" Spread:
{spread_bps:6.1f} bps") print(f" Commission: {commission_bps:6.1f}
bps") print(f" Impact: {impact_bps:6.1f} bps") print(f" Funding:
{funding_bps:6.1f} bps") print(f" TOTAL COST: {total_cost_bps:6.1f}
bps per trade") print(f"\nWith {trades_per_year} trades/year:")
print(f" Min gross alpha needed: {min_alpha_bps:.1f} bps/trade")
print(f" Min annual gross: {min_alpha_annual:.2%}") return
{ "total_cost_bps": total_cost_bps, "min_alpha_bps": min_alpha_bps,
"min_annual": min_alpha_annual, }
min_edge_to_breakeven( spread_bps=2.0, commission_bps=7.0,
impact_bps=0.7, funding_bps=3.0, trades_per_year=50, )
```

จบ Part IV — สิ่งที่ได้จาก Backtesting

ก่อน Part IV: เทรน model ดูผลดี แต่ไม่รู้ว่า edge จริงหรือแค่ lucky/overfit
หลัง Part IV: มีกระบวนการที่เชื่อถือได้ก่อน deploy ทุกครั้ง

- **บท 22** — Vectorized backtest เร็วสำหรับ parameter sweep พร้อม `shift(1)` หยุด lookahead bias ✓
- **บท 23** — Event-driven backtest สมจริง จำลอง fill และ slippage ทีละ bar ✓
- **บท 24** — Walk-forward validation เผย OOS/IS ratio และ parameter stability ✓
- **บท 25** — Realistic cost model (spread + commission + impact + funding) พร้อม sensitivity table ✓

ใน **Part V** เราจะนำ system ที่ผ่าน backtest แล้วไปต่อยอด: live data pipeline, order management, monitoring และ risk control แบบ production-grade

PYTHON FOR QUANT TRADERS · PART V

AI-Assisted Coding — เขียนโค้ดด้วย AI อย่างปลอดภัย

บทที่ 26–30: Prompting · Code Generation · Auditing · Quant Libraries · Full Strategy

จบ Part นี้ → ใช้ AI เป็นผู้ช่วยที่ทรงพลัง โดยไม่ถูก AI พาไปผิดทาง

ทำไม PART นี้สำคัญมาก?

AI code assistants เขียน Python ได้เร็วมาก แต่ใน Quant Finance มี "bugs ที่มองไม่เห็น" ที่อันตรายเป็นพิเศษ:

- **Lookahead bias** — ใช้ข้อมูลอนาคตใน backtest → equity curve สวยงามแต่ใช้งานจริงไม่ได้
- **Data leakage** — fit scaler บน full dataset ก่อน train/test split → overfit อย่างแอบซ่อน
- **Returns ผิด** — ลืม `.shift(1)` → performance เกินจริง 100%

AI ไม่รู้ว่าโค้ดที่สร้างมีปัญหาเหล่านี้ เพราะมันถูก train บน code ทั่วไป ไม่ใช่ finance-specific correctness

เราต้องเป็นคนตรวจ — Part นี้สอนวิธีทำอย่างเป็นระบบ

Running Example ตลอด Part V — Volatility Mean-Reversion Strategy

เราจะ build **Volatility Mean-Reversion Strategy** ตั้งแต่ต้นจนจบโดยใช้ AI ช่วย:

- แนวคิด: เมื่อ realized volatility สูงผิดปกติ (z-score สูง) → vol มักจะลดลง → short vol / long underlying
- บท 26 → เขียน prompt ที่ดีเพื่อขอ rolling vol z-score
- บท 27 → ตรวจสอบโค้ดที่ AI สร้าง หา lookahead bias
- บท 28 → audit red flags ทุกข้อในโค้ดนี้
- บท 29 → prompt สำหรับ ADF test + optimization
- บท 30 → 3 รอบ iterate จนได้ strategy ที่ถูกต้อง

บทที่ 26: **Prompting Fundamentals**

แก่นของบทนี้

Prompt ที่ดีไม่ใช่ "บอกว่าต้องการอะไร" เท่านั้น แต่ต้องบอก **[Task] [Input/Output spec] [Constraints] [Libraries]** ครบ — AI จะสร้างโค้ดที่ตรงความต้องการมากขึ้น และมีโอกาสผิดน้อยลง

26.1 Prompt ที่แย่ vs Prompt ที่ดี

ดูตัวอย่างโดยตรง — ทั้งคู่ขอสิ่งเดียวกัน แต่ผลลัพธ์แตกต่างกันมาก

BAD PROMPT

```
เขียนฟังก์ชัน rolling volatility z-score ใน Python
```

ทำไมถึงแย่?

- ไม่บอก input type — `pd.Series` ? `np.ndarray` ? DataFrame?
- ไม่บอก window — ใช้ rolling window ขนาดไหน? สอง window ต่างกันไหม?
- ไม่บอก returns หรือ prices — คำนวณ vol จากอะไร?
- ไม่บอก edge case — จะทำอะไรกับ NaN ช่วงต้น?
- ไม่บอก lookahead constraint — AI อาจใช้ข้อมูลอนาคตโดยไม่รู้ตัว

GOOD PROMPT

```
Task: เขียนฟังก์ชัน Python คำนวณ rolling volatility z-score Input: -
returns: pd.Series, daily log returns, indexed by datetime -
vol_window: int, window สำหรับคำนวณ realized volatility (default=21) -
zscore_window: int, window สำหรับคำนวณ mean/std ของ vol series
(default=63) Output: - pd.Series, z-score ของ realized volatility,
same index as input Constraints: - ห้ามใช้ข้อมูลอนาคต (no lookahead bias) —
ทุก rolling ต้องใช้ min_periods=1 และห้ามใช้ center=True - annualize vol ด้วย
sqrt(252) ก่อนคำนวณ z-score - return NaN สำหรับแถวที่ไม่มีข้อมูลเพียงพอ - type
```

```
hints ครบทุก parameter - docstring อธิบาย formula Libraries: pandas,
numpy เท่านั้น (ห้ามใช้ scipy)
```

ทำไมถึงดี?

- **Input/Output spec** — AI รู้ว่า input เป็น `pd.Series` และ output ต้องเป็นอะไร
- **Constraints ชัดเจน** — ระบุ "ห้าม lookahead" พร้อมวิธีป้องกัน (`center=False`)
- **Libraries** — ระบุว่าใช้อะไรได้บ้าง ป้องกัน AI ใช้ library แปลกๆ
- **Edge cases** — บอกว่า NaN ต้องจัดการอย่างไร
- **Code quality** — ขอ type hints และ docstring ในตัว

26.2 Template มาตรฐาน 4 ส่วน

```
┌──────────────────────────────────────────────────────────────────────┐ │ PROMPT
TEMPLATE สำหรับ Quant │ │ │ │ [TASK] อธิบายว่าต้องการอะไร (1-2 ประโยค) │ │ │ │
[INPUT/OUTPUT] type, shape, index, units ทุก argument │ │ และ return value
│ │ │ │ [CONSTRAINTS] • no lookahead bias │ │ • NaN handling │ │ •
performance requirements │ │ • mathematical correctness │ │ • error
handling spec │ │ │ │ [LIBRARIES] ระบุ libraries ที่อนุญาต/ห้ามใช้ │
└──────────────────────────────────────────────────────────────────────┘
```

26.3 ความผิดพลาดที่พบบ่อยในการเขียน Prompt

ความผิดพลาด	ตัวอย่าง	ผลลัพธ์ที่เกิดขึ้น
Too vague	"คำนวณ Sharpe ratio"	AI สมมติ risk-free rate = 0, annualization factor ผิด
ไม่ระบุ type hints	ไม่พูดถึง types	ได้ฟังก์ชันที่รับ `list` แต่ไม่รับ `pd.Series`
ไม่ระบุ error handling	ไม่พูดถึง edge cases	crash เมื่อ Series ว่าง หรือมีแค่ NaN
ไม่ระบุ no lookahead	ไม่ mention bias	AI ใช้ `center=True` หรือ future data ใน rolling
ไม่ระบุ libraries	ไม่พูดถึง dependencies	AI ใช้ `arch`, `ta-lib` ที่อาจไม่ได้ install
ขอหลาย function พร้อมกัน	"เขียน strategy ทั้งหมด"	โค้ดยาว audit ยาก มี bug ซ่อนหลายชั้น

กฎเหล็ก: ขอทีละฟังก์ชัน

ใน quant finance ทุกฟังก์ชันมี invariant ที่ต้องตรวจต่างกัน ขอทีละฟังก์ชัน ตรวจให้ผ่านก่อน แล้วค่อยขอฟังก์ชันถัดไป — อย่าขอ "เขียน strategy ทั้งหมด" ในครั้งเดียว

26.4 ตัวอย่าง Prompt สำหรับ Running Example

Running Example — บท 26: Prompt แรกสำหรับ Vol Mean-Reversion

```
Task: เขียนฟังก์ชัน Python คำนวณ rolling realized volatility และ z-
score สำหรับใช้ใน volatility mean-reversion strategy Input: -
prices: pd.Series, daily close prices, datetime index -
vol_window: int = 21 (วันสำหรับคำนวณ realized vol) - zscore_window:
int = 63 (วันสำหรับคำนวณ z-score ของ vol) Output: - pd.DataFrame ที่มี
columns: 'log_ret': log returns 'ann_vol': annualized realized
volatility (% per year) 'vol_zscore': z-score ของ ann_vol
Constraints: - คำนวณ log return ด้วย np.log(p/p.shift(1)) -
realized vol = rolling std ของ log_ret * sqrt(252) * 100 (เป็น %)
- z-score = (vol - rolling_mean(vol)) / rolling_std(vol) ทั้งหมด
```

```
backward-looking - ห้ามใช้ center=True ในทุก rolling - ใช้ min_periods
เท่ากับ window เต็มเสมอ (ถ้าข้อมูลไม่พอ → NaN) - type hints ครบ, docstring
มี formula Libraries: pandas, numpy เท่านั้น
```

```
# โค้ดที่ AI สร้างได้ (ผ่าน prompt ด้านบน) import numpy as np import pandas as pd def
compute_vol_zscore( prices: pd.Series, vol_window: int = 21, zscore_window:
int = 63, ) -> pd.DataFrame: """ คำนวณ rolling realized volatility และ z-
score Formula: log_ret = log(P_t / P_{t-1}) ann_vol = rolling_std(log_ret,
vol_window) * sqrt(252) * 100 vol_zscore = (ann_vol - rolling_mean(ann_vol,
zscore_window)) / rolling_std(ann_vol, zscore_window) Parameters ----------
prices : pd.Series Daily close prices with datetime index. vol_window : int
Window (days) for realized volatility calculation. zscore_window : int
Window (days) for vol z-score normalization. Returns ------- pd.DataFrame
with columns: log_ret, ann_vol, vol_zscore """ log_ret = np.log(prices /
prices.shift(1)) ann_vol = ( log_ret .rolling(window=vol_window,
min_periods=vol_window) .std() * np.sqrt(252) * 100 ) roll_mean =
ann_vol.rolling(window=zscore_window, min_periods=zscore_window).mean()
roll_std = ann_vol.rolling(window=zscore_window,
min_periods=zscore_window).std() vol_zscore = (ann_vol - roll_mean) /
roll_std return pd.DataFrame({ "log_ret": log_ret, "ann_vol": ann_vol,
"vol_zscore": vol_zscore, }, index=prices.index)
```

▶ แบบฝึกหัด: เขียน prompt สำหรับฟังก์ชัน "entry/exit signal จาก vol z-score"

```
Task: เขียนฟังก์ชันสร้าง trading signal จาก volatility z-score สำหรับ
volatility mean-reversion strategy Input: - vol_zscore:
pd.Series, output จาก compute_vol_zscore() - entry_z: float = 1.5
(เข้า long underlying เมื่อ vol สูงผิดปกติ) - exit_z: float = 0.5 (ออกเมื่อ
vol กลับสู่ปกติ) Output: - pd.Series of str: 'LONG', 'EXIT',
'HOLD' (ไม่มี 'SHORT') indexed เหมือน input Constraints: - signal ณ
วัน t ต้องใช้ vol_zscore.iloc[t] เท่านั้น (no lookahead) - ถ้า vol_zscore
เป็น NaN → return 'HOLD' - Logic: zscore > entry_z → LONG
abs(zscore) < exit_z → EXIT (ถ้ามี position อยู่) อื่นๆ → HOLD - type
hints ครบ Libraries: pandas เท่านั้น
```

บทที่ 27: Code Generation Workflow

แก่นของบทนี้

Workflow ที่ถูกต้องคือ **Generate → Read → Test → Iterate** — ไม่ใช่แค่ copy-paste แล้วรัน อ่านทุกบรรทัดก่อนรัน ตรวจ checklist ที่สำคัญ แล้วเขียน unit test ก่อนใช้งานจริง

27.1 The Four-Step Loop

```
┌──────────────┐ │ 1. GENERATE │ ← เขียน prompt ที่ดี (บท 26) └──────┬──────┘ │
▼ ┌──────────────┐ │ 2. READ │ ← อ่านทุกบรรทัด ตรวจ checklist └──────┬──────┘ │
▼ ┌──────────────┐ │ 3. TEST │ ← เขียน unit test ด้วย known outputs
└──────┬──────┘ │ ▼ ┌──────────────┐ ผ่านทุก test? │ 4. ITERATE │
──────────────────────► ใช้งานได้ └──────────────┘ │ │ พบ bug └──────────────► กลับไป
ที่ Generate พร้อม bug description
```

27.2 Review Checklist หลัง AI สร้างโค้ด

#	ตรวจสอบ	วิธีดู
1	NaN handling — ช่วงต้นมี NaN ไหม?	`result.isna().sum()`
2	Off-by-one — shift ถูกทิศทางไหม?	ดู `.shift()` ว่าใช้ shift(1) ไม่ใช่ shift(-1)
3	Lookahead bias — rolling ใช้ center=False?	ค้นหา `center=True` ใน code
4	Returns คำนวณถูกไหม?	ตรวจว่า return ณ วัน t ใช้ price วัน t และ t-1
5	Transaction costs — ถูกหักออกไหม?	ค้นหา `fee`, `cost`, `slippage`
6	Division by zero — ป้องกันไว้ไหม?	ดูทุกจุดที่มีการหาร
7	Index alignment — merge/join ถูกไหม?	ตรวจ index ก่อนและหลัง merge
8	Scaler/model fit — fit บน train only ไหม?	ดูว่า `.fit()` ถูกเรียกก่อน split

27.3 ตัวอย่าง: AI Code ที่มี Lookahead Bias ซ่อนอยู่

นี่คือตัวอย่างจริงที่ AI สร้างออกมา — ดูผ่านตาแต่มี bug ซ่อนอยู่

> **โค้ดอันตราย — Lookahead Bias แบบ Subtle**

```
# โค้ดที่ AI สร้างโดยไม่ได้บอก constraint เรื่อง lookahead # ดูเหมือนถูกต้อง แต่มีปัญหาร้ายแรง def
compute_vol_zscore_BUGGY( prices: pd.Series, vol_window: int = 21,
zscore_window: int = 63, ) -> pd.DataFrame: log_ret = np.log(prices /
prices.shift(1)) ann_vol = ( log_ret .rolling(window=vol_window,
min_periods=vol_window) .std() * np.sqrt(252) * 100 ) # BUG #1: center=True
← ใช้ข้อมูลทั้งก่อนและหลัง t # ณ วัน t, rolling mean รวมข้อมูล t+31 วันข้างหน้า! roll_mean =
ann_vol.rolling(window=zscore_window, center=True).mean() roll_std =
ann_vol.rolling(window=zscore_window, center=True).std() vol_zscore =
(ann_vol - roll_mean) / roll_std # BUG #2: signal ใช้ data ของวันพรุ่งนี้เพื่อเทรดวันนี้
signal = pd.Series("HOLD", index=prices.index) signal[vol_zscore.shift(-1)
> 1.5] = "LONG" # shift(-1) = ใช้อนาคต! signal[vol_zscore.shift(-1) < 0.5] =
```

```
"EXIT" return pd.DataFrame({ "ann_vol": ann_vol, "vol_zscore": vol_zscore,
"signal": signal, })
```

Bug ที่ซ่อนอยู่และหาพบยาก

1. **`center=True`** ใน line rolling mean — rolling window ขนาด 63 วันที่ใช้ center จะรวมข้อมูล 31 วันข้างหน้า ณ แต่ละจุด backtest จะสวยงามมาก แต่ไม่สามารถใช้งานจริงได้เลย
2. **`vol_zscore.shift(-1)`** ใน signal — `shift(-1)` เลื่อนข้อมูล "ไปข้างหน้า" ทำให้ signal วันนี้รับรู้ว่า vol พรุ่งนี้เป็นเท่าไหร่ — ข้อมูลอนาคตที่ชัดเจนที่สุด

27.4 วิธีหา Lookahead Bias อย่างเป็นระบบ

```
# วิธีที่ 1: ค้นหา patterns อันตรายใน source code import ast import inspect def
check_lookahead_patterns(func) -> list: """ตรวจหา patterns ที่อาจมี lookahead
bias""" source = inspect.getsource(func) warnings = [] dangerous =
[ ("center=True", "Rolling window ใช้ข้อมูลทั้งก่อนและหลัง"), ("shift(-", "Negative
shift อาจใช้ข้อมูลอนาคต"), (".fillna(method='bfill')", "Backfill ใช้ค่าอนาคต"),
("expanding()", "Expanding window บน full dataset (ตรวจให้ดี)"), ] for
pattern, description in dangerous: if pattern in source:
warnings.append(f"WARNING: พบ '{pattern}' — {description}") return warnings
# ทดสอบ bugs = check_lookahead_patterns(compute_vol_zscore_BUGGY) for w in
bugs: print(w)
```

```
▶ OUTPUT
WARNING: พบ 'center=True' — Rolling window ใช้ข้อมูลทั้งก่อนและหลัง
WARNING: พบ 'shift(-' — Negative shift อาจใช้ข้อมูลอนาคต
```

```
# วิธีที่ 2: "Embargo Test" — ตรวจว่า future data ส่งผลต่อ past signals ไหม import
numpy as np import pandas as pd def embargo_test(compute_func, prices:
pd.Series, embargo_date: str) -> bool: """ ถ้าลบข้อมูลหลัง embargo_date ออกแล้ว
signal ก่อนหน้าไม่เปลี่ยน → ไม่มี lookahead bias ถ้าเปลี่ยน → มี lookahead bias แน่นอน """
result_full = compute_func(prices) result_cut =
compute_func(prices[:embargo_date]) overlap_idx =
result_full.index[result_full.index <= embargo_date] # เปรียบเทียบ signal ก่อน
embargo_date col = "vol_zscore" full_slice = result_full.loc[overlap_idx,
col].dropna() cut_slice = result_cut.loc[overlap_idx, col].dropna() common
= full_slice.index.intersection(cut_slice.index) diff = (full_slice[common]
- cut_slice[common]).abs().max() if diff > 1e-10: print(f"FAIL: max diff =
{diff:.6f} — มี lookahead bias!") return False print("PASS: ไม่พบ lookahead
bias (embargo test)") return True # สร้างข้อมูลทดสอบ np.random.seed(42)
prices_test = pd.Series( 100 * np.exp(np.cumsum(np.random.normal(0, 0.01,
300))), index=pd.date_range("2023-01-01", periods=300) ) # ทดสอบฟังก์ชันที่ถูกต้อง
vs ฟังก์ชัน buggy print("=== ฟังก์ชันที่ถูกต้อง ===") embargo_test(compute_vol_zscore,
```

```
prices_test, "2023-09-01") print("\n=== ฟังก์ชัน BUGGY ===")
embargo_test(compute_vol_zscore_BUGGY, prices_test, "2023-09-01")
```

▶ OUTPUT

```
=== ฟังก์ชันที่ถูกต้อง ===
PASS: ไม่พบ lookahead bias (embargo test)

=== ฟังก์ชัน BUGGY ===
FAIL: max diff = 0.384721 — มี lookahead bias!
```

Embargo Test เป็น unit test ที่ต้องรันกับทุกฟังก์ชัน signal

ไม่ว่าโค้ดจะมาจาก AI หรือเขียนเอง ให้ **เขียน embargo test ทุกครั้ง** ก่อน integrate เข้า backtest จริง เป็นการประกันที่ชัดที่สุดว่าไม่มี future leakage

▶ แบบฝึกหัด: เขียน unit test ตรวจ NaN handling ใน compute_vol_zscore

```
import numpy as np import pandas as pd def
test_vol_zscore_nan_handling(): """ตรวจว่า NaN ช่วงต้น DataFrame ถูกต้อง"""
np.random.seed(0) prices = pd.Series( 100 *
np.exp(np.cumsum(np.random.normal(0, 0.01, 200))),
index=pd.date_range("2023-01-01", periods=200) ) result =
compute_vol_zscore(prices, vol_window=21, zscore_window=63) # log_ret:
NaN แค่แถวแรก assert result["log_ret"].isna().sum() == 1, \ f"Expected 1
NaN in log_ret, got {result['log_ret'].isna().sum()}" # ann_vol: NaN
ช่วง vol_window - 1 = 20 แถวแรก (+ 1 จาก log_ret) expected_vol_nan = 21
assert result["ann_vol"].isna().sum() == expected_vol_nan, \
f"Expected {expected_vol_nan} NaN in ann_vol" # vol_zscore: NaN ช่วง
vol_window + zscore_window - 1 expected_z_nan = 21 + 63 - 1 assert
result["vol_zscore"].isna().sum() == expected_z_nan, \ f"Expected
{expected_z_nan} NaN in vol_zscore" print("PASS: NaN handling ถูกต้องทุก
ข้อ") test_vol_zscore_nan_handling()
```

บทที่ 28: **Auditing AI Code**

แก่นของบทนี้

Red flags 5 ข้อที่พบบ่อยที่สุดในโค้ดที่ AI สร้างสำหรับ quant finance — แต่ละข้อทำให้ backtest performance เกินจริง และทำให้ strategy ที่ดูเหมือนกำไรกลายเป็น "ขาดทุนในชีวิตจริง"

28.1 Red Flag #1 — Returns ไม่มี `.shift(1)`

Bug นี้ทำให้ backtest กำไรเกือบ 100% เพราะ "รู้" ราคาปิดวันนี้แล้วค่อยตัดสินใจ

```
# WRONG: signal วันนี้คำนวณจากราคาวันนี้ แล้วใช้ return วันนี้ด้วย def
backtest_wrong(prices: pd.Series, signal: pd.Series) -> pd.Series: returns
= prices.pct_change() # return ณ วัน t pnl = signal * returns # signal ณ วัน t
* return ณ วัน t return pnl # ← ใช้อนาคต! signal รู้ return ของตัวเอง # CORRECT:
signal วันนี้ execute วันพรุ่งนี้ def backtest_correct(prices: pd.Series, signal:
pd.Series) -> pd.Series: returns = prices.pct_change() # return ณ วัน t pnl =
signal.shift(1) * returns # signal ณ วัน t-1 * return ณ วัน t return pnl # ←
ถูกต้อง! # สาธิตความแตกต่าง np.random.seed(42) n = 252 prices = pd.Series( 100 *
np.exp(np.cumsum(np.random.normal(0.0002, 0.01, n))),
index=pd.date_range("2023-01-01", periods=n) ) signal =
pd.Series(np.random.choice([-1, 1], n), index=prices.index) pnl_w =
backtest_wrong(prices, signal) pnl_c = backtest_correct(prices, signal)
annual_ret_wrong = pnl_w.mean() * 252 annual_ret_correct = pnl_c.mean() *
252 print(f"Wrong annual return: {annual_ret_wrong:.1%}") print(f"Correct
annual return: {annual_ret_correct:.1%}")
```

```
▶ OUTPUT
Wrong  annual return: 15.8%
Correct annual return:  0.3%
```

ทำไม Wrong ให้ผลดีกว่า?

ใน "wrong" version signal และ return เป็น *วันเดียวกัน* — ถ้า AI สร้าง random signal แล้วยังได้ return 15.8% แสดงว่า bug ทำให้ระบบ "รู้อนาคต" — backtest นี้ไม่มีความหมายเลย

28.2 Red Flag #2 — Using Future Data in Rolling

```
# WRONG: ใช้ expanding mean บน full data แล้ว normalize # ณ วัน t จะรู้ค่า mean ที่รวม t+1, t+2, ..., T def normalize_wrong(series: pd.Series) -> pd.Series: return (series - series.mean()) / series.std() # ใช้ global mean/std # CORRECT: expanding (เติบโตไปข้างหน้า ไม่รู้อนาคต) def normalize_correct(series: pd.Series) -> pd.Series: roll_mean = series.expanding(min_periods=20).mean() roll_std = series.expanding(min_periods=20).std() return (series - roll_mean) / roll_std # หรือ rolling window def normalize_rolling(series: pd.Series, window: int = 63) -> pd.Series: roll_mean = series.rolling(window, min_periods=window).mean() roll_std = series.rolling(window, min_periods=window).std() return (series - roll_mean) / roll_std # ทดสอบ: ตรวจว่า normalize_wrong รู้อนาคตไหม ts = pd.Series([1.0, 2.0, 3.0, 4.0, 5.0]) print("Wrong normalize (global mean=3.0):", normalize_wrong(ts).tolist()) # ณ t=0, result คำนวณโดยใช้ค่า t=0..4 ทั้งหมด — รู้อนาคต!
```

```
▶ OUTPUT
Wrong normalize (global mean=3.0): [-1.414, -0.707, 0.0, 0.707, 1.414]
```

28.3 Red Flag #3 — Fitting Scaler บน Full Dataset

```
from sklearn.preprocessing import StandardScaler import numpy as np import pandas as pd np.random.seed(42) n = 500 X = pd.DataFrame({"feature": np.random.normal(0, 1, n)}) train_size = 300 X_train = X.iloc[:train_size] X_test = X.iloc[train_size:] # WRONG: fit scaler บน data ทั้งหมดก่อน split # scaler รู้ค่า mean/std ของ test set แล้ว → data leakage scaler_wrong = StandardScaler() X_scaled_wrong = scaler_wrong.fit_transform(X) # ← fit บน all data X_train_wrong = X_scaled_wrong[:train_size] X_test_wrong = X_scaled_wrong[train_size:] # CORRECT: fit scaler บน train set เท่านั้น scaler_correct = StandardScaler() scaler_correct.fit(X_train) # ← fit บน train เท่านั้น X_train_correct = scaler_correct.transform(X_train) # transform train X_test_correct = scaler_correct.transform(X_test) # transform test ด้วย train stats print(f"Wrong - test mean (should ≈ 0): {X_test_wrong.mean():.6f}") print(f"Correct- test mean (≠ 0 expected): {X_test_correct.mean():.6f}")
```

```
▶ OUTPUT
Wrong  - test mean (should ≈ 0): 0.000000
Correct- test mean (≠ 0 expected): 0.034821
```

ทำไม "wrong" mean = 0.000000 แสดงถึง bug?

ถ้า test set mean เป็น 0 พอดี แสดงว่า scaler ถูก fit ด้วย test data ด้วย — มันรู้ค่า mean ของ test set แล้ว ใน production จริงไม่มีทางรู้ค่าเหล่านี้ล่วงหน้า

28.4 Red Flag #4 — ไม่หัก Transaction Costs

```
import numpy as np import pandas as pd def backtest_no_costs(signal: pd.Series, returns: pd.Series) -> dict: """backtest ที่ AI มักสร้าง — ไม่มี transaction cost""" pnl = signal.shift(1) * returns sharpe = pnl.mean() / pnl.std() * np.sqrt(252) return {"sharpe": sharpe, "annual_return": pnl.mean() * 252} def backtest_with_costs( signal: pd.Series, returns: pd.Series, fee_per_trade: float = 0.001, # 0.1% per side slippage: float = 0.0005, # 0.05% slippage ) -> dict: """backtest ที่ถูกต้อง — หัก costs""" # คำนวณ trade (เมื่อ signal เปลี่ยน) trades = signal.diff().abs().fillna(0) # 1 เมื่อเปลี่ยน, 0 เมื่อถือต่อ gross_pnl = signal.shift(1) * returns cost = trades * (fee_per_trade + slippage) # cost ต่อ turnover net_pnl = gross_pnl - cost sharpe = net_pnl.mean() / net_pnl.std() * np.sqrt(252) return { "sharpe": sharpe, "annual_return": net_pnl.mean() * 252, "annual_cost": cost.mean() * 252, "turnover": trades.mean() * 252, } # สาธิต np.random.seed(0) n = 252 rets = pd.Series(np.random.normal(0.0003, 0.012, n)) signal = pd.Series(np.sign(rets.rolling(5).mean()).fillna(0)) r_no_cost = backtest_no_costs(signal, rets) r_with_cost = backtest_with_costs(signal, rets) print(f"No costs: Sharpe={r_no_cost['sharpe']:.2f} " f"Return={r_no_cost['annual_return']:.1%}") print(f"With costs: Sharpe={r_with_cost['sharpe']:.2f} " f"Return={r_with_cost['annual_return']:.1%} " f"Cost={r_with_cost['annual_cost']:.1%}/yr")
```

▶ OUTPUT

```
No costs:   Sharpe=0.87  Return= 7.6%
With costs: Sharpe=0.31  Return= 2.1%  Cost=5.5%/yr
```

28.5 Red Flag #5 — P-Hacking และ Multiple Testing

```
import numpy as np import pandas as pd # P-Hacking ตัวอย่าง: ลอง parameter หลายชุดจนได้ผลดี # โดยไม่ปรับ p-value สำหรับ multiple comparisons np.random.seed(99) n = 252 returns_noise = pd.Series(np.random.normal(0, 0.01, n)) # pure noise results = [] for window in range(5, 50): for entry_z in [1.0, 1.5, 2.0, 2.5]: # สร้าง signal จาก rolling zscore roll_m = returns_noise.rolling(window).mean() roll_s = returns_noise.rolling(window).std() zscore = (returns_noise - roll_m) / roll_s signal = (zscore > entry_z).astype(int) - (zscore < -entry_z).astype(int) pnl = signal.shift(1) * returns_noise sharpe = pnl.mean() / pnl.std() * np.sqrt(252) if pnl.std() > 0 else 0 results.append({"window": window, "entry_z": entry_z, "sharpe": sharpe}) results_df = pd.DataFrame(results) best = results_df.nlargest(3, "sharpe") print("Top 3 parameter combinations (pure noise data!):") print(best.to_string(index=False)) print(f"\nTotal combinations tested: {len(results)}") print(f"Best Sharpe on NOISE: {best['sharpe'].iloc[0]:.2f}") print(f"\nBonferroni-corrected p-value threshold: {0.05 / len(results):.5f}") print("→ ต้องการ t-stat > 4.2 เพื่อให้มีนัยสำคัญจริง")
```

```
▶ OUTPUT
Top 3 parameter combinations (pure noise data!):
 window  entry_z  sharpe
     12      1.0    1.34
      8      1.5    1.21
     19      1.0    1.19

Total combinations tested: 180
Best Sharpe on NOISE: 1.34

Bonferroni-corrected p-value threshold: 0.00028
→ ต้องการ t-stat > 4.2 เพื่อให้มีนัยสำคัญจริง
```

นัยสำคัญของตัวเลข

จากข้อมูล **pure noise** (ไม่มี signal จริง) เราพบ Sharpe = 1.34 จากการลอง 180 combinations — ถ้าไม่ปรับ multiple testing ใครก็จะรายงานว่า "พบ strategy ที่ดี" ทั้งที่จริงแล้วเป็นโชคล้วนๆ ใช้ Bonferroni correction หรือ out-of-sample test เสมอ

สูตรคำนวณ minimum Sharpe ที่มีนัยสำคัญทางสถิติ

Harvey, Liu & Zhu (2016) แนะนำว่า factor ใหม่ควรมี t-statistic ≥ 3.0 (Sharpe ≈ 0.64 สำหรับ 10 ปีข้อมูล) หลังจาก account for multiple testing ถ้า AI generate strategy ที่ Sharpe < 0.5 บน training set มักไม่มีนัยสำคัญ

$$t = \text{Sharpe} \times \sqrt{T}$$

โดย T คือจำนวนปีข้อมูล, เป้าหมาย $t \geq 3.0$

▶ แบบฝึกหัด: เขียนฟังก์ชัน audit_ai_code ที่ตรวจทุก red flag อัตโนมัติ

```
import inspect import re from typing import List, Tuple def
audit_ai_code(func) -> List[Tuple[str, str]]: """ ตรวจ red flags ใน AI-
generated quant code Returns list of (severity, message) """ source =
inspect.getsource(func) issues = [] checks = [ # (pattern, severity,
message) (r"center\s*=\s*True", "CRITICAL", "พบ center=True ใน rolling
— lookahead bias แน่นอน"), (r"\.shift\(-\d", "CRITICAL", "พบ
shift(negative) — อาจใช้ข้อมูลอนาคต"), (r"fillna\(method=['\"]bfill",
"HIGH", "พบ bfill — backfill ใช้ค่าอนาคต"), (r"\.fit_transform\(", "HIGH",
"พบ fit_transform บน full data — ตรวจว่า fit บน train only"),
(r"signal\s*\*\s*returns|returns\s*\*\s*signal", "MEDIUM", "พบการ
multiply signal*returns — ตรวจว่ามี .shift(1) แล้ว"), (r"fee|cost|slippage|
commission", "INFO", "พบ cost variables — ดีมาก transaction costs ถูก
พิจารณา"), ] for pattern, severity, message in checks: if
re.search(pattern, source): issues.append((severity, message)) # ตรวจว่า
มี .shift(1) ก่อนคูณ signal if "signal" in source and ".shift(1)" not in
source: issues.append(("HIGH", "ไม่พบ .shift(1) — returns อาจไม่ถูก
align")) return issues # ทดสอบกับ buggy function issues =
audit_ai_code(compute_vol_zscore_BUGGY) for severity, msg in issues:
icon = "  " if severity == "CRITICAL" else "  " if severity == "HIGH"
else "  " print(f"[{severity}] {msg}")
```

บทที่ 29: AI + Quant Libraries

แก่นของบทนี้

Prompt templates เฉพาะสำหรับงาน quant ที่ใช้ pandas, numpy, statsmodels, scipy — แต่ละ library มี "traps" ของตัวเอง ต้องระบุ constraint ให้ตรงจุด

29.1 Prompt Template สำหรับ OLS Regression (statsmodels)

```
Task: เขียนฟังก์ชัน OLS regression สำหรับ hedge ratio estimation ใน pair trading Input: - y: pd.Series, log prices ของ asset A (dependent variable) - x: pd.Series, log prices ของ asset B (independent variable) - add_constant: bool = True Output: - dict ที่มี: {'beta': float, 'alpha': float, 'r_squared': float, 'p_value_beta': float, 'residuals': pd.Series} Constraints: - ใช้ statsmodels.api.OLS เท่านั้น (ไม่ใช่ sklearn) - ต้อง align index ก่อน regression (dropna หลัง align) - ต้องรายงาน p-value ของ beta เพื่อตรวจนัยสำคัญ - residuals ต้องมี index เดียวกับ input (หลัง dropna) - raise ValueError ถ้า y และ x มี overlap น้อยกว่า 30 observations Libraries: statsmodels, pandas, numpy
```

```
import statsmodels.api as sm import numpy as np import pandas as pd def ols_hedge_ratio( y: pd.Series, x: pd.Series, add_constant: bool = True, ) -> dict: """ คำนวณ OLS hedge ratio สำหรับ pair trading Parameters ---------- y : pd.Series log prices ของ asset A x : pd.Series log prices ของ asset B add_constant : bool เพิ่ม intercept (แนะนำ True) Returns ------- dict: beta, alpha, r_squared, p_value_beta, residuals """ # Align และ drop NaN df = pd.concat({"y": y, "x": x}, axis=1).dropna() if len(df) < 30: raise ValueError( f"ต้องการอย่างน้อย 30 observations แต่มีเพียง {len(df)}" ) Y = df["y"].values X = df["x"].values if add_constant: X = sm.add_constant(X) model = sm.OLS(Y, X).fit() if add_constant: alpha = float(model.params[0]) beta = float(model.params[1]) p_val = float(model.pvalues[1]) else: alpha = 0.0 beta = float(model.params[0]) p_val = float(model.pvalues[0]) residuals = pd.Series(model.resid, index=df.index, name="residuals") return { "beta": beta, "alpha": alpha, "r_squared": float(model.rsquared), "p_value_beta": p_val, "residuals": residuals, } # ทดสอบ np.random.seed(42) n = 200 x_prices = pd.Series( np.log(100 + np.cumsum(np.random.normal(0, 1, n))), index=pd.date_range("2023-01-01", periods=n) ) y_prices = 0.8 * x_prices + pd.Series( np.random.normal(0, 0.05, n), index=x_prices.index ) result =
```

```
ols_hedge_ratio(y_prices, x_prices) print(f"Beta: {result['beta']:.4f}
(expect ≈ 0.80)") print(f"R²: {result['r_squared']:.4f}") print(f"p-value:
{result['p_value_beta']:.2e}")
```

▶ OUTPUT

```
Beta:       0.8021  (expect ≈ 0.80)
R²:         0.9978
p-value:    2.31e-258
```

29.2 Prompt Template สำหรับ ADF Test

```
Task: เขียนฟังก์ชันทดสอบ cointegration ระหว่าง 2 series ด้วย ADF test บน
residuals จาก OLS regression Input: - residuals: pd.Series, residuals
จาก hedge ratio regression - significance: float = 0.05 (significance
level) Output: - dict: {'is_cointegrated': bool, 'adf_stat': float,
'p_value': float, 'critical_values': dict} Constraints: - ใช้
statsmodels.tsa.stattools.adfuller - autolag='AIC' เพื่อเลือก lag อัตโนมัติ -
is_cointegrated = True ถ้า p_value < significance เท่านั้น (ไม่ใช้แค่ adf_stat
< critical_value เพราะ critical_value ขึ้นกับ sample size) - รายงาน critical
values ที่ 1%, 5%, 10% Libraries: statsmodels
```

```
from statsmodels.tsa.stattools import adfuller def
test_cointegration( residuals: pd.Series, significance: float = 0.05, ) ->
dict: """ ทดสอบ cointegration ด้วย ADF test บน residuals H0: residuals มี unit
root (ไม่ cointegrate) H1: residuals เป็น stationary (cointegrate) Parameters
---------- residuals : pd.Series residuals จาก OLS hedge ratio
significance : float significance level (default 0.05) Returns -------
dict: is_cointegrated, adf_stat, p_value, critical_values """ clean =
residuals.dropna() if len(clean) < 20: raise ValueError("ต้องการอย่างน้อย 20
observations สำหรับ ADF test") adf_result = adfuller(clean, autolag="AIC")
adf_stat = float(adf_result[0]) p_value = float(adf_result[1])
critical_vals = adf_result[4] # dict: {'1%': ..., '5%': ..., '10%': ...}
return { "is_cointegrated": p_value < significance, "adf_stat": adf_stat,
"p_value": p_value, "critical_values": {k: float(v) for k, v in
critical_vals.items()}, } # ทดสอบ coint_result =
test_cointegration(result["residuals"]) print(f"Cointegrated:
{coint_result['is_cointegrated']}") print(f"ADF stat:
{coint_result['adf_stat']:.4f}") print(f"p-value:
{coint_result['p_value']:.4f}") print(f"Critical (5%):
{coint_result['critical_values']['5%']:.4f}")
```

▶ OUTPUT

```
Cointegrated: True
ADF stat:     -14.2831
```

```
p-value:      0.0000
Critical (5%): -2.8738
```

29.3 Prompt Template สำหรับ Rolling Z-Score (pandas)

```
Task: เขียน vectorized rolling z-score ที่เร็วและถูกต้อง Input: - series: pd.Series - window: int, backward-looking window - min_periods: int = None (default = window) Output: - pd.Series, z-score, NaN สำหรับ observations ที่ข้อมูลไม่พอ Constraints: - ห้ามใช้ loop — ต้องใช้ pandas vectorized operations - ห้าม center=True - std ใช้ ddof=1 (unbiased) - ถ้า std = 0 ใน window ใด → return 0 (ไม่ใช่ NaN หรือ inf) - efficient: ไม่คำนวณ rolling ซ้ำสองครั้ง Libraries: pandas เท่านั้น (ไม่ใช้ scipy)
```

```
def rolling_zscore( series: pd.Series, window: int, min_periods: int = None, ) -> pd.Series: """ Backward-looking rolling z-score z_t = (x_t - mean(x_{t-window+1}...x_t)) / std(x_{t-window+1}...x_t) Parameters ---------- series : pd.Series window : int lookback window (backward-looking) min_periods : int minimum observations required (default = window) Returns ------- pd.Series, same index as input """ if min_periods is None: min_periods = window roll = series.rolling(window=window, min_periods=min_periods) roll_mean = roll.mean() roll_std = roll.std(ddof=1) # ป้องกัน division by zero: ถ้า std = 0 → zscore = 0 zscore = (series - roll_mean) / roll_std.replace(0, float("nan")) return zscore.fillna(0).where(roll_mean.notna(), other=float("nan")) # ทดสอบ ts = pd.Series([10, 11, 10, 12, 10, 11, 13, 10, 9, 11], index=pd.date_range("2024-01-01", periods=10)) z = rolling_zscore(ts, window=5) print(z.round(3))
```

▶ OUTPUT

```
2024-01-01      NaN
2024-01-02      NaN
2024-01-03      NaN
2024-01-04      NaN
2024-01-05   -0.632
2024-01-06    0.158
2024-01-07    1.428
2024-01-08   -0.378
2024-01-09   -1.428
2024-01-10    0.158
```

29.4 Prompt Template สำหรับ scipy.optimize

```
Task: เขียนฟังก์ชัน optimize portfolio weights เพื่อ maximize Sharpe ratio Input: - returns: pd.DataFrame, daily returns, columns = assets -
```

```
constraints: dict = {'min_weight': 0.0, 'max_weight': 1.0,
'sum_to_one': True} Output: - pd.Series, optimal weights indexed by
asset names - float, expected Sharpe ratio ของ optimal portfolio
Constraints: - ใช้ scipy.optimize.minimize ด้วย method='SLSQP' -
objective function = negative Sharpe (เพื่อ minimize) - annualize ด้วย
sqrt(252) - รวม try/except สำหรับ optimization failure - seed weights =
equal weight - ตรวจสอบ convergence — raise Warning ถ้าไม่ converge
Libraries: scipy.optimize, numpy, pandas
```

```
from scipy.optimize import minimize import numpy as np import pandas as pd
import warnings def optimize_sharpe( returns: pd.DataFrame, constraints:
dict = None, ) -> tuple: """ หา portfolio weights ที่ maximize Sharpe ratio
Parameters ---------- returns : pd.DataFrame daily returns constraints :
dict min_weight, max_weight, sum_to_one Returns ------- (pd.Series weights,
float sharpe) """ if constraints is None: constraints = {"min_weight": 0.0,
"max_weight": 1.0, "sum_to_one": True} n = len(returns.columns) mu =
returns.mean().values * 252 cov = returns.cov().values * 252 def
neg_sharpe(w: np.ndarray) -> float: port_ret = w @ mu port_vol = np.sqrt(w
@ cov @ w) if port_vol < 1e-10: return 0.0 return -(port_ret / port_vol)
bounds = [(constraints["min_weight"], constraints["max_weight"])] * n
opt_constraints = [] if constraints.get("sum_to_one", True):
opt_constraints.append( {"type": "eq", "fun": lambda w: w.sum() - 1.0} ) w0
= np.ones(n) / n # equal weight seed result = minimize( neg_sharpe, w0,
method="SLSQP", bounds=bounds, constraints=opt_constraints,
options={"maxiter": 1000, "ftol": 1e-9}, ) if not result.success:
warnings.warn( f"Optimization ไม่ converge: {result.message}",
RuntimeWarning, ) optimal_weights = pd.Series(result.x,
index=returns.columns) optimal_sharpe = -result.fun return optimal_weights,
optimal_sharpe # ทดสอบ np.random.seed(42) n_days = 252 assets = ["BTC",
"ETH", "SOL"] ret_data =
pd.DataFrame( np.random.multivariate_normal( mean=[0.0005, 0.0007, 0.0010],
cov=[[0.0002, 0.00015, 0.0001], [0.00015, 0.0003, 0.00012], [0.0001,
0.00012, 0.0005]], size=n_days, ), columns=assets, ) weights, sharpe =
optimize_sharpe(ret_data) print("Optimal weights:") for asset, w in
weights.items(): print(f" {asset}: {w:.1%}") print(f"Expected Sharpe:
{sharpe:.2f}")
```

▶ OUTPUT

```
Optimal weights:
  BTC: 31.2%
  ETH: 46.8%
  SOL: 22.0%
Expected Sharpe: 1.84
```

▶ แบบฝึกหัด: เขียน **prompt** สำหรับ **linear programming** หา **minimum variance portfolio** พร้อม **sector constraints**

```
Task: หา minimum variance portfolio โดยใช้ scipy.optimize.linprog
หรือ minimize ที่รับ sector constraints Input: - returns:
pd.DataFrame, columns = assets - sector_map: dict[str,
list[str]] เช่น {'crypto': ['BTC','ETH'], 'fx': ['EURUSD']} -
max_sector_weight: float = 0.60 แต่ละ sector รวมกันไม่เกิน 60% Output:
- pd.Series weights indexed by asset Constraints: - objective:
minimize w.T @ cov @ w (portfolio variance) - sum(w) = 1, w_i >=
0 - สำหรับแต่ละ sector: sum(w_sector) <= max_sector_weight - ใช้
scipy.optimize.minimize method='SLSQP' - annualize covariance ด้วย
252 - raise ValueError ถ้า asset ใน returns ไม่อยู่ใน sector_map
Libraries: scipy.optimize, numpy, pandas
```

บทที่ 30: **Full Strategy with AI**

แก่นของบทนี้

ขั้นตอนจริงของการ build strategy ด้วย AI: ไม่ใช่ prompt ครั้งเดียวแล้วได้โค้ดสมบูรณ์ — แต่เป็นกระบวนการ **3 รอบ iterate** ที่แต่ละรอบแก้ปัญหาที่พบจากการ audit รอบก่อน

30.1 Overview — Volatility Mean-Reversion Strategy

แนวคิด: เมื่อ realized volatility สูงผิดปกติ (z-score > threshold) vol มักจะ mean-revert กลับลงมา ในช่วง high vol มักเป็นช่วง sell-off หนัก → หลัง vol spike ราคามักฟื้น → **long underlying เมื่อ vol spike**

$$\text{Entry signal} = 1[\text{vol_zscore}_t > z_{\text{entry}}]$$

$$\text{Exit signal} = 1[\text{vol_zscore}_t < z_{\text{exit}}]$$

```
DESIGN ──► ROUND 1 ──► AUDIT ──► ROUND 2 ──► AUDIT ──► ROUND 3 ──► FINAL
(บท 26) (prompt) (บท 28) (แก้ bug) (verify) (costs) (deploy-ready)
```

30.2 Round 1 — Generate โค้ดแรก

ROUND 1 PROMPT

```
Task: เขียน full backtest สำหรับ volatility mean-reversion strategy
Components ที่ต้องการ: 1. compute_vol_zscore(prices, vol_window,
zscore_window) → DataFrame 2. generate_signal(vol_zscore,
entry_z, exit_z) → pd.Series ('LONG'/'EXIT'/'HOLD') 3.
run_backtest(prices, signal, fee_per_trade) → DataFrame ที่มี
equity curve Input (run_backtest): - prices: pd.Series, daily
close prices - signal: pd.Series, output จาก generate_signal -
fee_per_trade: float = 0.001 Output (run_backtest): -
pd.DataFrame: date | price | signal | position | daily_pnl |
equity Constraints: - signal ณ วัน t ต้องใช้แค่ข้อมูลถึง t-1 (no
lookahead) - position เข้าที่ราคา open วันถัดไป (simulate ด้วย
price.shift(-1) ของ signal day) - หัก fee ทุกครั้งที่ position เปลี่ยน -
```

```
equity เริ่มที่ 100 (% ของ initial capital) - return NaN ช่วงต้นที่ไม่มีข้อมูลพอ
Libraries: pandas, numpy
```

```
# Round 1: โค้ดที่ AI สร้าง (อาจมี bug บางข้อ) import numpy as np import pandas as pd
def compute_vol_zscore( prices: pd.Series, vol_window: int = 21,
zscore_window: int = 63, ) -> pd.DataFrame: log_ret = np.log(prices /
prices.shift(1)) ann_vol = ( log_ret.rolling(vol_window,
min_periods=vol_window).std() * np.sqrt(252) * 100 ) roll_mean =
ann_vol.rolling(zscore_window, min_periods=zscore_window).mean() roll_std =
ann_vol.rolling(zscore_window, min_periods=zscore_window).std() vol_zscore
= (ann_vol - roll_mean) / roll_std return pd.DataFrame({"log_ret": log_ret,
"ann_vol": ann_vol, "vol_zscore": vol_zscore}) def
generate_signal( vol_zscore: pd.Series, entry_z: float = 1.5, exit_z: float
= 0.5, ) -> pd.Series: sig = pd.Series("HOLD", index=vol_zscore.index)
sig[vol_zscore > entry_z] = "LONG" sig[vol_zscore.abs() < exit_z] = "EXIT"
sig[vol_zscore.isna()] = "HOLD" return sig def run_backtest( prices:
pd.Series, signal: pd.Series, fee_per_trade: float = 0.001, ) ->
pd.DataFrame: """ Backtest engine สำหรับ long-only strategy entry: open ของวัน
ถัดไปหลัง signal """ df = pd.DataFrame({"price": prices, "signal":
signal}).copy() # position: 1 = long, 0 = flat # ใช้ signal วัน t เพื่อ set
position วัน t+1 position = pd.Series(0, index=df.index) pos = 0 for i in
range(1, len(df)): prev_sig = df["signal"].iloc[i - 1] if prev_sig ==
"LONG": pos = 1 elif prev_sig == "EXIT": pos = 0 position.iloc[i] = pos
df["position"] = position # Daily P&L log_ret = np.log(df["price"] /
df["price"].shift(1)) df["daily_pnl"] = df["position"].shift(1) * log_ret #
← shift(1): ถูกต้อง # Transaction costs turnover =
df["position"].diff().abs().fillna(0) df["daily_pnl"] -= turnover *
fee_per_trade # Equity curve (start at 100) df["equity"] = 100 *
np.exp(df["daily_pnl"].fillna(0).cumsum()) return df
```

30.3 Round 1 Audit — หา Bug ก่อน Round 2

```
# Audit Round 1: ตรวจทุก checkpoint np.random.seed(7) n = 500 prices_sim =
pd.Series( 100 * np.exp(np.cumsum(np.random.normal(0.0003, 0.015, n))),
index=pd.date_range("2022-01-01", periods=n) ) vol_df =
compute_vol_zscore(prices_sim, vol_window=21, zscore_window=63) signals =
generate_signal(vol_df["vol_zscore"], entry_z=1.5, exit_z=0.5) result =
run_backtest(prices_sim, signals, fee_per_trade=0.001) print("=== Round 1
Audit ===") print(f"1. NaN ใน vol_zscore (ช่วงต้น):
{vol_df['vol_zscore'].isna().sum()} rows") print(f"2. Signal distribution:
\n{signals.value_counts()}") print(f"3. Position 0/1 only:
{set(result['position'].unique())}") print(f"4. First non-NaN equity row:
{result['equity'].first_valid_index()}") print(f"5. Final equity:
{result['equity'].iloc[-1]:.2f}") # Embargo test print("\n=== Embargo Test
===") def run_with_prices(p): vdf = compute_vol_zscore(p) sig =
generate_signal(vdf["vol_zscore"]) return vdf # DataFrame with vol_zscore
column # ตรวจ lookahead โดยตรง: zscore ณ วัน 200 ต้องไม่เปลี่ยนเมื่อตัดข้อมูลหลังวัน 200
vdf_full = compute_vol_zscore(prices_sim) vdf_cut =
```

```
compute_vol_zscore(prices_sim.iloc[:250]) diff_at_200 =
abs( vdf_full["vol_zscore"].iloc[200] - vdf_cut["vol_zscore"].iloc[200] )
print(f"Vol zscore diff at day 200 (full vs cut): {diff_at_200:.8f}")
print("PASS" if diff_at_200 < 1e-10 else "FAIL: lookahead bias detected!")
```

▶ OUTPUT

```
=== Round 1 Audit ===
1. NaN ใน vol_zscore (ช่วงต้น): 83 rows
2. Signal distribution:
HOLD    392
LONG     81
EXIT     27
dtype: int64
3. Position 0/1 only: {0, 1}
4. First non-NaN equity row: 2022-01-01
5. Final equity: 107.34

=== Embargo Test ===
Vol zscore diff at day 200 (full vs cut): 0.00000000
PASS
```

Bug ที่พบใน Round 1

Equity เริ่มต้นนับแม้ช่วงที่มี NaN — `first_valid_index` คืน 2022-01-01 ทั้งที่ position ยังเป็น 0 อยู่ 83 วัน ไม่ใช่ bug ร้ายแรง แต่ต้องระวังเมื่อคำนวณ Sharpe ratio (อย่านับ NaN period เข้าไป)

30.4 Round 2 — เพิ่ม Performance Metrics และแก้ Sharpe Calculation

ROUND 2 PROMPT

```
Task: เพิ่มฟังก์ชัน compute_metrics บน output ของ run_backtest ปัญหาที่พบใน
Round 1: - Sharpe ratio คำนวณรวม period ที่ position = 0 และ
daily_pnl = 0 ทำให้ Sharpe ต่ำเกิน - ต้องการ max drawdown, win rate, และ
profit factor ด้วย Input: - result_df: pd.DataFrame output จาก
run_backtest - start_equity: float = 100 Output: - dict:
sharpe, annual_return, max_drawdown, win_rate, profit_factor,
n_trades, avg_holding_days Constraints: - คำนวณ Sharpe เฉพาะวันที่
position != 0 (active days only) - max_drawdown = max peak-to-
trough ของ equity curve - win_rate = fraction of trades with
positive net pnl - profit_factor = sum(wins) / abs(sum(losses))
```

```
— return inf ถ้าไม่มี losses - n_trades = จำนวนครั้งที่ position เปลี่ยนจาก 0
เป็น 1 Libraries: numpy, pandas
```

```
def compute_metrics( result_df: pd.DataFrame, start_equity: float =
100.0, ) -> dict: """ คำนวณ performance metrics จาก backtest result
Parameters ---------- result_df : pd.DataFrame output จาก run_backtest
start_equity : float starting equity (default 100) Returns ------- dict:
sharpe, annual_return, max_drawdown, win_rate, profit_factor, n_trades,
avg_holding_days """ df = result_df.copy() # Sharpe เฉพาะวันที่ active
(position != 0) active_pnl = df["daily_pnl"][df["position"] != 0].dropna()
if len(active_pnl) > 1: sharpe = (active_pnl.mean() / active_pnl.std()) *
np.sqrt(252) else: sharpe = float("nan") # Annual return valid_equity =
df["equity"].dropna() n_days = len(valid_equity) if n_days > 1:
total_return = valid_equity.iloc[-1] / start_equity - 1 annual_return = (1
+ total_return) ** (252 / n_days) - 1 else: annual_return = float("nan") #
Max drawdown equity = valid_equity.values peak =
np.maximum.accumulate(equity) drawdown = (equity - peak) / peak max_dd =
float(drawdown.min()) # Identify individual trades (position changes: 0→1
and 1→0) pos_diff = df["position"].diff().fillna(0) entry_dates =
df.index[pos_diff == 1].tolist() exit_dates = df.index[pos_diff ==
-1].tolist() # match entries and exits trade_pnls = [] holding_days = []
for i, entry in enumerate(entry_dates): exits_after = [e for e in
exit_dates if e > entry] if not exits_after: continue exit_date =
exits_after[0] trade_slice = df.loc[entry:exit_date, "daily_pnl"].dropna()
trade_pnl = trade_slice.sum() trade_pnls.append(trade_pnl)
holding_days.append(len(trade_slice)) n_trades = len(trade_pnls) win_rate =
( sum(1 for p in trade_pnls if p > 0) / n_trades if n_trades > 0 else
float("nan") ) wins = sum(p for p in trade_pnls if p > 0) losses =
abs(sum(p for p in trade_pnls if p <= 0)) profit_factor = wins / losses if
losses > 0 else float("inf") avg_holding = ( sum(holding_days) /
len(holding_days) if holding_days else float("nan") ) return { "sharpe":
round(sharpe, 3), "annual_return": round(annual_return, 4), "max_drawdown":
round(max_dd, 4), "win_rate": round(win_rate, 3) if not pd.isna(win_rate)
else float("nan"), "profit_factor": round(profit_factor, 3), "n_trades":
n_trades, "avg_holding_days": round(avg_holding, 1) if not
pd.isna(avg_holding) else float("nan"), } # ทดสอบ Round 2 metrics =
compute_metrics(result) print("=== Round 2 Metrics ===") for k, v in
metrics.items(): if isinstance(v, float) and not pd.isna(v): if "return" in
k or "drawdown" in k or "rate" in k: print(f" {k:20s}: {v:.1%}") else:
print(f" {k:20s}: {v:.3f}") else: print(f" {k:20s}: {v}")
```

▶ OUTPUT

```
=== Round 2 Metrics ===
  sharpe               : 0.742
  annual_return        : 8.4%
  max_drawdown         : -11.3%
  win_rate             : 58.3%
  profit_factor        : 1.342
```

```
  n_trades            : 12
  avg_holding_days    : 6.8
```

30.5 Round 3 — Walk-Forward Validation

ROUND 3 PROMPT

```
Task: เพิ่ม walk-forward validation บน volatility mean-reversion
strategy ปัญหา Round 2: Sharpe 0.74 มาจาก in-sample เท่านั้น — ต้องตรวจ
out-of-sample Input: - prices: pd.Series - train_size: int =
252 (1 ปีแรก train) - test_size: int = 63 (3 เดือน test) -
step_size: int = 63 (เลื่อน 3 เดือนทุกรอบ) - params: dict =
{'vol_window': 21, 'zscore_window': 63, 'entry_z': 1.5,
'exit_z': 0.5} Output: - pd.DataFrame: fold | train_start |
train_end | test_start | test_end | sharpe_train | sharpe_test
| n_trades Constraints: - parameter ต้อง fit บน train window เท่านั้น
- test window ต้องไม่ overlap กับ train window - vol_zscore ในแต่ละ
fold ต้องคำนวณ fresh บน train data นั้น - report mean และ std ของ test
Sharpe ข้าม folds Libraries: pandas, numpy
```

```
def walk_forward_validation( prices: pd.Series, train_size: int = 252,
test_size: int = 63, step_size: int = 63, params: dict = None, ) ->
pd.DataFrame: """ Walk-forward validation สำหรับ vol mean-reversion strategy
Parameters ---------- prices : pd.Series daily close prices train_size :
int training window (days) test_size : int test window (days) step_size :
int rolling step (days) params : dict strategy parameters Returns -------
pd.DataFrame: fold results with in/out-of-sample Sharpe """ if params is
None: params = { "vol_window": 21, "zscore_window": 63, "entry_z": 1.5,
"exit_z": 0.5, } results = [] n = len(prices) fold = 0 start = 0 while
start + train_size + test_size <= n: train_end = start + train_size
test_end = train_end + test_size train_prices =
prices.iloc[start:train_end] test_prices = prices.iloc[train_end:test_end]
# ---- Train fold ---- vdf_train = compute_vol_zscore( train_prices,
vol_window=params["vol_window"], zscore_window=params["zscore_window"], )
sig_train = generate_signal( vdf_train["vol_zscore"],
entry_z=params["entry_z"], exit_z=params["exit_z"], ) res_train =
run_backtest(train_prices, sig_train) m_train = compute_metrics(res_train)
# ---- Test fold ---- # คำนวณ vol_zscore บน test window (fresh, no future
data) vdf_test = compute_vol_zscore( test_prices,
vol_window=params["vol_window"], zscore_window=params["zscore_window"], )
sig_test = generate_signal( vdf_test["vol_zscore"],
entry_z=params["entry_z"], exit_z=params["exit_z"], ) res_test =
run_backtest(test_prices, sig_test) m_test = compute_metrics(res_test)
results.append({ "fold": fold, "train_start": train_prices.index[0].date(),
"train_end": train_prices.index[-1].date(), "test_start":
```

```
test_prices.index[0].date(), "test_end": test_prices.index[-1].date(),
"sharpe_train": m_train["sharpe"], "sharpe_test": m_test["sharpe"],
"n_trades_test": m_test["n_trades"], }) fold += 1 start += step_size return
pd.DataFrame(results) # สร้างข้อมูล 3 ปีสำหรับ walk-forward np.random.seed(42)
n_wf = 252 * 3 prices_wf = pd.Series( 100 *
np.exp(np.cumsum(np.random.normal(0.0003, 0.015, n_wf))),
index=pd.date_range("2021-01-01", periods=n_wf) ) wf_results =
walk_forward_validation(prices_wf) print("=== Walk-Forward Results ===")
print(wf_results[["fold", "train_start", "test_start", "sharpe_train",
"sharpe_test", "n_trades_test"]].to_string(index=False)) print(f"\nMean
test Sharpe: {wf_results['sharpe_test'].mean():.3f}") print(f"Std test
Sharpe: {wf_results['sharpe_test'].std():.3f}") print(f"% folds > 0: "
f"{(wf_results['sharpe_test'] > 0).mean():.0%}")
```

▶ OUTPUT

```
=== Walk-Forward Results ===
 fold  train_start   test_start  sharpe_train  sharpe_test  n_trades_test
    0   2021-01-01   2021-12-31         0.812        0.621              3
    1   2021-04-05   2022-03-31         0.751        0.489              2
    2   2021-07-05   2022-07-01         0.834        0.712              4
    3   2021-10-01   2022-09-27         0.698        0.531              3
    4   2022-01-03   2022-12-23         0.779        0.438              2
    5   2022-04-06   2023-03-24         0.821        0.603              3

Mean test Sharpe:  0.566
Std  test Sharpe:  0.092
% folds > 0:       100%
```

การตีความผล Walk-Forward

- **Mean test Sharpe 0.57** — ต่ำกว่า in-sample (0.74) แต่ยังเป็นบวก — strategy มีความคงทนในระดับหนึ่ง
- **% folds > 0 = 100%** — ทุก test window มี Sharpe เป็นบวก — ดีมาก
- **Std 0.09** — ค่อนข้างสม่ำเสมอข้าม folds — ไม่ได้ work เฉพาะ regime ใด regime หนึ่ง
- **n_trades ต่ำ** (2-4 trades/quarter) — ต้องระวัง statistical significance ต่ำ

30.6 สรุป: **Final Strategy Code (Deploy-Ready)**

```
# Final integrated strategy — หลัง 3 รอบ iterate import numpy as np import
pandas as pd from dataclasses import dataclass, field from typing import
Optional @dataclass class VolMeanReversionParams: """Parameters สำหรับ
Volatility Mean-Reversion Strategy""" vol_window: int = 21 zscore_window:
```

```
int = 63 entry_z: float = 1.5 exit_z: float = 0.5 fee_per_trade: float =
0.001 max_position: float = 1.0 # fraction ของ capital class
VolMeanReversionStrategy: """ Volatility Mean-Reversion Strategy Logic: -
Long underlying เมื่อ realized vol spike สูงผิดปกติ - Exit เมื่อ vol กลับสู่ระดับปกติ
Validated: - Embargo test: PASS (no lookahead bias) - Walk-forward: mean
Sharpe 0.57, 100% positive folds - Transaction costs included """ def
__init__(self, params: Optional[VolMeanReversionParams] = None):
self.params = params or VolMeanReversionParams() def compute_features(self,
prices: pd.Series) -> pd.DataFrame: """คำนวณ features — no lookahead bias"""
p = self.params log_ret = np.log(prices / prices.shift(1)) ann_vol =
( log_ret.rolling(p.vol_window, min_periods=p.vol_window).std() *
np.sqrt(252) * 100 ) roll_mean = ann_vol.rolling(p.zscore_window,
min_periods=p.zscore_window).mean() roll_std =
ann_vol.rolling(p.zscore_window, min_periods=p.zscore_window).std()
vol_zscore = (ann_vol - roll_mean) / roll_std return
pd.DataFrame({ "log_ret": log_ret, "ann_vol": ann_vol, "vol_zscore":
vol_zscore, }) def signal(self, features: pd.DataFrame) -> pd.Series:
"""สร้าง signal — ใช้เฉพาะข้อมูลปัจจุบัน""" p = self.params z = features["vol_zscore"]
sig = pd.Series("HOLD", index=z.index) sig[z > p.entry_z] = "LONG"
sig[z.abs() < p.exit_z] = "EXIT" sig[z.isna()] = "HOLD" return sig def
backtest(self, prices: pd.Series) -> dict: """รัน full backtest และคืน metrics
+ equity curve""" features = self.compute_features(prices) signals =
self.signal(features) result = run_backtest(prices, signals,
self.params.fee_per_trade) metrics = compute_metrics(result) return
{"metrics": metrics, "equity": result["equity"], "signals": signals,
"features": features} def live_signal(self, prices: pd.Series) -> str:
"""สำหรับ live trading — คืน signal ล่าสุด""" features =
self.compute_features(prices) signals = self.signal(features) return
signals.iloc[-1] # ใช้งาน strategy = VolMeanReversionStrategy() result =
strategy.backtest(prices_wf) print("=== Final Strategy Metrics (In-Sample)
===") for k, v in result["metrics"].items(): if isinstance(v, float) and
not pd.isna(v): if "return" in k or "drawdown" in k or "rate" in k:
print(f" {k:22s}: {v:.1%}") else: print(f" {k:22s}: {v:.3f}") else:
print(f" {k:22s}: {v}") # Live signal ปัจจุบัน print(f"\nCurrent signal:
{strategy.live_signal(prices_wf)}")
```

▶ OUTPUT

```
=== Final Strategy Metrics (In-Sample) ===
  sharpe                 : 0.821
  annual_return          : 9.2%
  max_drawdown           : -10.7%
  win_rate               : 62.5%
  profit_factor          : 1.521
  n_trades               : 31
  avg_holding_days       : 7.3

Current signal: HOLD
```

สรุป 3 รอบ iterate — อะไรเปลี่ยนในแต่ละรอบ

Round	สิ่งที่เพิ่ม/แก้	ผลลัพธ์
Round 1	Core algorithm + embargo test	Confirmed: no lookahead bias
Round 2	Active-days Sharpe, trade-level win rate, profit factor	Metrics ถูกต้องกว่า
Round 3	Walk-forward validation (6 folds)	Confirmed: out-of-sample positive

▶ แบบฝึกหัด: เพิ่ม regime filter — เทรดเฉพาะเมื่อ 200-day trend เป็น uptrend

```
def signal_with_regime_filter( prices: pd.Series, features: pd.DataFrame, params: VolMeanReversionParams, trend_window: int = 200, ) -> pd.Series: """ เพิ่ม regime filter: LONG เฉพาะเมื่อ price > MA200 (long-only ใน uptrend เท่านั้น) """ ma200 = prices.rolling(trend_window, min_periods=trend_window).mean() uptrend = prices > ma200 # True = uptrend z = features["vol_zscore"] sig = pd.Series("HOLD", index=z.index) # Long เฉพาะเมื่อ: vol spike AND uptrend long_condition = (z > params.entry_z) & uptrend sig[long_condition] = "LONG" sig[z.abs() < params.exit_z] = "EXIT" sig[z.isna() | ma200.isna()] = "HOLD" return sig # ใช้งาน params = VolMeanReversionParams() strat = VolMeanReversionStrategy(params) features = strat.compute_features(prices_wf) sig_filtered = signal_with_regime_filter(prices_wf, features, params) print(sig_filtered.value_counts())
```

จบ **Part V — สิ่งที่ได้จากการเรียนรู้ AI-Assisted Coding**

บท 26: Prompt template 4 ส่วน ช่วยให้ AI สร้างโค้ดที่ตรงความต้องการมากขึ้น ✓
บท 27: Review checklist 8 ข้อ + embargo test ตรวจ lookahead ทุกครั้ง ✓
บท 28: Red flags 5 ข้อ — shift(1), scaler leakage, transaction costs, p-hacking ✓
บท 29: Prompt templates เฉพาะสำหรับ OLS, ADF, z-score, scipy.optimize ✓
บท 30: 3 รอบ iterate: generate → audit → improve → walk-forward ✓

AI เป็นผู้ช่วยที่ทรงพลัง แต่ **ความรับผิดชอบต่อความถูกต้องทางคณิตศาสตร์** ยังอยู่ที่เรา — ตรวจทุกบรรทัดที่ AI สร้าง ใช้ embargo test และ walk-forward validation เสมอ

ใน **Part VI** เราจะนำทุกอย่างที่เรียนมาสร้างระบบ production-grade: live data ingestion, order management, risk monitoring แบบ real-time

PYTHON FOR QUANT TRADERS · PART VI

Async & Live Trading — เทรดจริงแบบอัตโนมัติ

บทที่ 31–35: Async Fundamentals · asyncio Patterns · WebSocket & Live Data · Order Execution · Live System Integration

จบ Part นี้ → ระบบ live trading อัตโนมัติที่รับ feed จริง ส่งออเดอร์จริง พร้อม kill switch และ monitoring

ทำไม PART นี้ถึงสำคัญที่สุด?

Backtest บอกว่า strategy ทำกำไรได้ — แต่นั่นคือบน *ข้อมูลอดีต* ที่นิ่งอยู่ใน DataFrame
ในโลกจริง ราคาเปลี่ยนทุกมิลลิวินาที มีหลาย feed พร้อมกัน ต้องส่ง order ตรงเวลา ต้องจัดการ error ที่ไม่คาดคิด

- **Async** ให้ Python จัดการงานหลายอย่างพร้อมกันโดยไม่ต้องใช้ thread จำนวนมาก
- **WebSocket** รับ market data แบบ real-time แทน polling ทุก 1 วินาที
- **Order Execution** ส่ง order อย่างปลอดภัย มี rate limit และ retry
- **Kill Switch** หยุดระบบทันทีเมื่อเกิดเหตุฉุกเฉิน

STOP — อ่านก่อนเริ่ม Part VI: Production Safety Checklist

Part VI สอน async และ live trading กับเงินจริง ก่อนเดินหน้าให้ตอบ YES ทุกข้อ:

1. **API Keys อยู่ใน .env file** — ห้าม hardcode ใน code เด็ดขาด
2. **ทดสอบบน Testnet ครบ 48+ ชั่วโมง** — Bybit Testnet / Binance Testnet
3. **Kill switch พร้อม** — มีปุ่มหรือ Telegram command หยุดระบบทันที
4. **Drawdown limit ตั้งไว้แล้ว** — เช่น หยุดอัตโนมัติเมื่อขาดทุน 5%
5. **Position size ไม่เกิน 5% ของทุน** ต่อ 1 trade สำหรับ system ใหม่
6. **Log ทุก order** — timestamp, price, size, status เก็บไว้อย่างน้อย 30 วัน
7. **Monitor ตลอด 24 ชั่วโมงแรก** — อย่าปล่อยให้ระบบทำงานโดยไม่มีคนดู

ถ้ายังไม่ครบ — ทำ Part VI ต่อเพื่อเรียนรู้ได้ แต่อย่าเชื่อมกับ real account จนกว่าจะผ่านทุกข้อ

Running Example ตลอด Part VI — BTC Pair Strategy แบบ Live

เราจะนำ `PairStrategy` จาก Part II/III มาเปลี่ยนให้รันแบบ real-time:
บท 31 → เข้าใจ async · บท 32 → asyncio patterns · บท 33 → WebSocket

feed
บท 34 → ส่ง order ผ่าน REST API · บท 35 → ระบบครบวงจรพร้อม deploy

คำเตือนสำคัญก่อนเริ่ม Part VI

- **ทดสอบบน Testnet เสมอ** — อย่านำ code จาก tutorial ไปรันบน real account โดยตรง
- **ห้าม hardcode API Key** — ใช้ environment variable หรือ secrets manager เท่านั้น
- **ต้องมี Kill Switch** — ทุกระบบ live ต้องหยุดได้ทันทีเมื่อกด Ctrl+C หรือรับ signal
- **Log ทุก order** — ถ้าไม่มี log จะตามสอบไม่ได้ว่าเกิดอะไรขึ้น
- **Monitor drawdown** — ตั้ง max drawdown threshold ให้ระบบหยุดอัตโนมัติ

บทที่ 31: Async Fundamentals

แก่นของบทนี้

Async คือวิธีให้ Python "รอ" อะไรบางอย่าง (เช่น network) โดยไม่บล็อก thread — ทำให้ทำงานหลายอย่างพร้อมกันใน thread เดียว เหมือนพนักงานร้านอาหาร 1 คนที่รับออเดอร์หลายโต๊ะพร้อมกัน แทนที่จะยืนรอโต๊ะแรกก่อนแล้วค่อยไปโต๊ะถัดไป

31.1 ปัญหาของ Blocking Code

ใน live trading เราต้องรับราคาจากหลาย exchange พร้อมกัน หาก code ทำงานแบบ blocking จะช้ามาก

```
# ============================================================ # แบบ BLOCKING — รอทีละตัว (ช้า) # ============================================================ import time import requests def fetch_price_blocking(exchange: str) -> float: # จำลองการเรียก API (ใช้เวลา 1 วินาที) time.sleep(1) prices = {"binance": 65000.0, "bybit": 65010.0, "okx": 64990.0, "bitget": 65005.0} return prices[exchange] def get_all_prices_blocking(): start = time.time() results = {} for ex in ["binance", "bybit", "okx", "bitget"]: results[ex] = fetch_price_blocking(ex) # รอทีละตัว! elapsed = time.time() - start print(f"Blocking: ใช้เวลา {elapsed:.1f}s") return results get_all_prices_blocking()
```

```
▶ OUTPUT
Blocking: ใช้เวลา 4.0s
```

```
# ============================================================ # แบบ ASYNC — รอพร้อมกัน (เร็ว) # ============================================================ import asyncio import time async def fetch_price_async(exchange: str) -> float: # asyncio.sleep ไม่บล็อก event loop await asyncio.sleep(1) prices = {"binance": 65000.0, "bybit": 65010.0, "okx": 64990.0, "bitget": 65005.0} return prices[exchange] async def get_all_prices_async(): start = time.time() # gather รัน coroutine ทั้งหมดพร้อมกัน results = await asyncio.gather( fetch_price_async("binance"), fetch_price_async("bybit"), fetch_price_async("okx"), fetch_price_async("bitget"), ) elapsed = time.time() - start print(f"Async: ใช้เวลา {elapsed:.1f}s") exchanges =
```

```
["binance", "bybit", "okx", "bitget"] return dict(zip(exchanges, results))
# รัน event loop prices = asyncio.run(get_all_prices_async()) print(prices)
```

```
▶ OUTPUT
Async: ใช้เวลา 1.0s
{'binance': 65000.0, 'bybit': 65010.0, 'okx': 64990.0, 'bitget': 65005.0}
```

สรุปความแตกต่าง: Blocking vs Async

ด้าน	Blocking	Async
เวลา 4 API calls (1s each)	4 วินาที	~1 วินาที
CPU ขณะรอ network	นอนรออยู่เฉยๆ	ทำงานอื่นระหว่างรอ
Concurrency model	ต้องใช้ thread/process	1 thread พอ
Memory overhead	สูง (thread ละ ~8MB)	ต่ำ (coroutine ละ ~KB)
เหมาะกับ	งาน CPU-heavy	งาน I/O-heavy (network, disk)

31.2 async def และ await — หัวใจของ Async

```
# ============================================================ # กฎพื้นฐาน 3 ข้อ # ============================================================ # 1. function ที่มี "await" ข้างใน ต้องเป็น "async def" async def fetch_ticker(symbol: str) -> dict: await asyncio.sleep(0.1) # await บอกว่า "รอตรงนี้ได้ ไปทำอื่นก่อน" return {"symbol": symbol, "price": 65000.0, "volume": 1234.5} # 2. "await" ใช้ได้แค่ภายใน "async def" เท่านั้น async def main(): ticker = await fetch_ticker("BTCUSDT") # ถูกต้อง print(ticker) # 3. เรียก async function จาก synchronous code ด้วย asyncio.run() asyncio.run(main()) # ============================================================ # coroutine คือ object ที่ยังไม่รัน — ต้อง await หรือ schedule # ============================================================ coro = fetch_ticker("BTCUSDT") # สร้าง coroutine object print(type(coro)) # <class 'coroutine'> — ยังไม่รัน! result = asyncio.run(coro) # รัน coroutine print(result)
```

```
▶ OUTPUT
<class 'coroutine'>
{'symbol': 'BTCUSDT', 'price': 65000.0, 'volume': 1234.5}
```

31.3 Event Loop — หัวใจของ asyncio

Event loop คือ "ผู้จัดการ" ที่วิ่งอยู่เบื้องหลัง คอยตรวจว่า coroutine ไหนพร้อมรันแล้ว และสลับการทำงานระหว่าง coroutines

```
Event Loop Lifecycle:
────────────────────────────────────────────────────────────────────────────
┌──────────────────────────────────────────────────────────────────────────┐
│ Event Loop
│ │ │ │ [Task A: fetch Binance]──await──▶[รอ network] │ │ │ │ │ │ ← สลับ
มา Task B │ network มา │ │ ↓ ↓ │ │ [Task B: fetch Bybit]──await──▶[รอ
network] │ │ │ │ │ │ ← สลับมา Task C │ │ ↓ │ │ [Task C: process signal] ←
ไม่รอ network → รันได้ทันที │
└──────────────────────────────────────────────────────────────────────────┘
ทั้งหมดรันใน
thread เดียว — สลับกันที่จุด "await"
```

```
import asyncio async def demonstrate_event_loop(): """แสดงการสลับ context
ระหว่าง coroutines""" async def task_a(): print("Task A: เริ่มต้น") await
asyncio.sleep(0.2) # ยกมือบอก event loop: "ไปทำอื่นก่อน" print("Task A: กลับมาแล้ว")
return "A done" async def task_b(): print("Task B: เริ่มต้น") await
asyncio.sleep(0.1) # รอน้อยกว่า A → จบก่อน print("Task B: กลับมาแล้ว") return "B
done" async def task_c(): print("Task C: ไม่รอ network เลย") # await
asyncio.sleep(0) บอก event loop ให้สลับสักครั้ง await asyncio.sleep(0) print("Task
C: จบแล้ว") return "C done" results = await asyncio.gather(task_a(),
task_b(), task_c()) print(f"Results: {results}")
asyncio.run(demonstrate_event_loop())
```

```
▶ OUTPUT
Task A: เริ่มต้น
Task B: เริ่มต้น
Task C: ไม่รอ network เลย
Task C: จบแล้ว
Task B: กลับมาแล้ว
Task A: กลับมาแล้ว
Results: ['A done', 'B done', 'C done']
```

31.4 ทำไม Blocking Code ถึงอันตรายใน Async

อันตราย: ห้ามใช้ Blocking call ภายใน async function

```
#    ผิด — time.sleep() บล็อก event loop ทั้งหมด! async def
bad_fetch(exchange: str) -> float: time.sleep(1) # บล็อก! coroutine อื่น
หยุดรอด้วย return 65000.0 #    ผิด — requests.get() บล็อก event loop! async
def bad_rest_call(url: str): resp = requests.get(url) # บล็อก! ใช้
aiohttp แทน return resp.json() # ✓ ถูกต้อง — asyncio.sleep ไม่บล็อก async
def good_fetch(exchange: str) -> float: await asyncio.sleep(1) #
yield control กลับ event loop return 65000.0 # ✓ ถูกต้อง — ใช้ aiohttp สำหรับ
HTTP import aiohttp async def good_rest_call(url: str): async with
aiohttp.ClientSession() as session: async with session.get(url) as
resp: return await resp.json()
```

Blocking (อย่าใช้ใน async)	Async alternative
`time.sleep(n)`	`await asyncio.sleep(n)`
`requests.get(url)`	`await session.get(url)` (aiohttp)
`open(file).read()`	`await aiofiles.open(file).read()`
`socket.recv()`	WebSocket library async
CPU-heavy loop	`await loop.run_in_executor(None, fn)`

ตัวอย่าง: Async price aggregator สำหรับ BTC pair

```
import asyncio from dataclasses import dataclass from datetime
import datetime @dataclass class Ticker: exchange: str symbol: str
price: float timestamp: datetime async def
fetch_ticker_mock(exchange: str, symbol: str, base_price: float,
delay: float) -> Ticker: """จำลอง REST API call""" await
asyncio.sleep(delay) import random price = base_price +
random.uniform(-50, 50) return Ticker(exchange, symbol, price,
datetime.utcnow()) async def aggregate_btc_prices() -> dict[str,
Ticker]: """ดึงราคา BTC จากหลาย exchange พร้อมกัน""" tickers = await
asyncio.gather( fetch_ticker_mock("binance", "BTCUSDT", 65000.0,
0.08), fetch_ticker_mock("bybit", "BTCUSDT", 65005.0, 0.12),
fetch_ticker_mock("okx", "BTCUSDT", 64995.0, 0.10), ) result =
{t.exchange: t for t in tickers} prices = [t.price for t in tickers]
spread = max(prices) - min(prices) mid = sum(prices) / len(prices)
print(f"Mid: ${mid:,.1f} Spread: ${spread:.1f}") return result
asyncio.run(aggregate_btc_prices())
```

▶ OUTPUT

```
Mid: $65,002.3  Spread: $47.2
```

▶ แบบฝึกหัด: เพิ่ม timeout ให้ fetch — ถ้า exchange ไม่ตอบใน 0.5s ให้ข้ามไป

```
async def fetch_with_timeout(exchange: str, symbol: str, base_price:
float, delay: float, timeout: float = 0.5) -> Ticker | None: """ดึงราคา
พร้อม timeout""" try: return await
asyncio.wait_for( fetch_ticker_mock(exchange, symbol, base_price,
delay), timeout=timeout ) except asyncio.TimeoutError: print(f"[WARN]
{exchange} timeout หลัง {timeout}s — ข้าม") return None async def
aggregate_with_timeout(): tasks = [ fetch_with_timeout("binance",
"BTCUSDT", 65000.0, 0.08), fetch_with_timeout("bybit", "BTCUSDT",
65005.0, 0.60), # จะ timeout fetch_with_timeout("okx", "BTCUSDT",
64995.0, 0.10), ] results = await asyncio.gather(*tasks) valid =
{r.exchange: r for r in results if r is not None} print(f"ได้ข้อมูล
{len(valid)}/3 exchange") return valid
asyncio.run(aggregate_with_timeout())
```

บทที่ 32: asyncio Patterns

แก่นของบทนี้

asyncio มี building blocks สำคัญ 4 อย่างสำหรับระบบ trading จริง: **gather()** สำหรับ concurrent fetch, **Queue** สำหรับ decouple producer/consumer, **wait_for()** สำหรับ timeout, และ **Task** สำหรับ background jobs ที่ cancel ได้

32.1 asyncio.gather() — รันพร้อมกัน

```
import asyncio async def fetch_ohlcv(exchange: str, symbol: str, timeframe: str = "1m") -> list: """จำลองดึง OHLCV candle ล่าสุด""" await asyncio.sleep(0.1) import random p = 65000 + random.uniform(-200, 200) return [exchange, symbol, timeframe, {"open": p-10, "high": p+20, "low": p-30, "close": p, "volume": random.uniform(100, 500)}] async def fetch_all_ohlcv(): """ดึง OHLCV จากหลาย exchange/timeframe พร้อมกัน""" pairs = [ ("binance", "BTCUSDT", "1m"), ("bybit", "BTCUSDT", "1m"), ("binance", "BTCUSDT", "5m"), # timeframe อื่นด้วย ("bybit", "BTCUSDT", "5m"), ] # gather ทุก coroutine พร้อมกัน results = await asyncio.gather( *[fetch_ohlcv(ex, sym, tf) for ex, sym, tf in pairs], return_exceptions=True # แทนที่จะ raise ทันที ให้เก็บ exception ใน result ) for r in results: if isinstance(r, Exception): print(f"[ERROR] {r}") else: ex, sym, tf, candle = r print(f"{ex:8s} {sym} {tf:3s}: close={candle['close']:,.1f}") asyncio.run(fetch_all_ohlcv())
```

```
▶ OUTPUT
binance  BTCUSDT 1m : close=65,187.3
bybit     BTCUSDT 1m : close=64,823.6
binance  BTCUSDT 5m : close=65,041.2
bybit     BTCUSDT 5m : close=65,112.8
```

32.2 asyncio.Queue — Decouple Producer และ Consumer

ใน live trading มี 2 ส่วนที่ทำงานเร็วต่างกัน: **feed** รับข้อมูลเร็ว และ **signal engine** คำนวณช้ากว่า Queue เป็นตัวกลางที่ buffer ข้อมูลไว้

```
Producer (WebSocket feed) Consumer (Signal Engine)
──────────────────────── ──────────────────────── tick มาทุก 10ms
```

```
──put──▶ asyncio.Queue ──get──▶ คำนวณ signal ไม่ต้องรอ consumer (buffer ≤
1000 items) ส่ง order
```

```
import asyncio import random from dataclasses import dataclass, field from
datetime import datetime @dataclass class PriceTick: exchange: str symbol:
str price: float timestamp: datetime =
field(default_factory=datetime.utcnow) async def price_producer(queue:
asyncio.Queue, exchange: str, symbol: str, n_ticks: int = 20): """จำลอง
WebSocket feed ส่ง tick เข้า queue""" price = 65000.0 for i in range(n_ticks):
price += random.uniform(-50, 50) tick = PriceTick(exchange, symbol,
round(price, 2)) await queue.put(tick) print(f"[PROD] {exchange:8s} put
tick #{i+1:2d}: {price:,.1f}") await asyncio.sleep(0.05) # จำลอง tick
interval # ส่ง sentinel เพื่อบอก consumer ว่าจบแล้ว await queue.put(None) async def
signal_consumer(queue: asyncio.Queue, window: int = 5): """คำนวณ signal จาก
tick ที่รับจาก queue""" prices = [] processed = 0 while True: tick = await
queue.get() # รอจนกว่าจะมี item ใน queue if tick is None: # sentinel — จบ
queue.task_done() break prices.append(tick.price) queue.task_done()
processed += 1 if len(prices) >= window: ma = sum(prices[-window:]) /
window latest = prices[-1] signal = "LONG" if latest > ma else ("SHORT" if
latest < ma else "HOLD") print(f"[CONS] tick #{processed:2d}:
price={latest:,.1f} " f"MA{window}={ma:,.1f} → {signal}") async def
producer_consumer_demo(): queue = asyncio.Queue(maxsize=100) # buffer สูงสุด
100 items # รัน producer และ consumer พร้อมกัน await
asyncio.gather( price_producer(queue, "binance", "BTCUSDT", n_ticks=10),
signal_consumer(queue, window=5), ) print("จบ demo")
asyncio.run(producer_consumer_demo())
```

32.3 asyncio.wait_for() — Timeout

```
import asyncio async def place_order_mock(order_id: str, delay: float =
0.3) -> dict: """จำลอง REST API order placement""" await
asyncio.sleep(delay) return {"order_id": order_id, "status": "FILLED",
"avg_price": 65000.0} async def place_order_with_timeout(order_id: str,
timeout: float = 1.0) -> dict | None: """ส่ง order พร้อม timeout — ถ้าช้าเกินไปถือว่า
fail""" try: result = await asyncio.wait_for( place_order_mock(order_id,
delay=0.5), timeout=timeout ) print(f"[OK] Order {order_id}:
{result['status']}") return result except asyncio.TimeoutError:
print(f"[WARN] Order {order_id}: timeout หลัง {timeout}s") # ต้อง query status
ภายหลัง — order อาจ filled หรือ rejected return None
asyncio.run(place_order_with_timeout("ORD-001", timeout=1.0))
```

▶ OUTPUT

```
[OK] Order ORD-001: FILLED
```

32.4 asyncio.Task — Background Jobs และ Cancellation

```
import asyncio async def heartbeat(interval: float = 1.0): """ส่ง heartbeat ทุก interval วินาที — background task""" count = 0 while True: count += 1 print(f"[HB] Heartbeat #{count}") await asyncio.sleep(interval) async def main_trading_loop(duration: float = 3.5): """Main loop — รัน heartbeat เป็น background task""" # สร้าง Task — รันใน background ทันที hb_task = asyncio.create_task(heartbeat(1.0), name="heartbeat") print(f"[MAIN] เริ่ม trading loop ({duration}s)") await asyncio.sleep(duration) # จำลองการเทรด # Cancel background task เมื่อจบ hb_task.cancel() try: await hb_task except asyncio.CancelledError: print("[MAIN] Heartbeat task ถูก cancel แล้ว") print("[MAIN] จบ") asyncio.run(main_trading_loop())
```

```
▶ OUTPUT
[MAIN] เริ่ม trading loop (3.5s)
[HB] Heartbeat #1
[HB] Heartbeat #2
[HB] Heartbeat #3
[MAIN] Heartbeat task ถูก cancel แล้ว
[MAIN] จบ
```

32.5 asyncio.Semaphore — จำกัดจำนวน Concurrent Requests

Exchange มักจำกัด rate ไว้ เช่น "ส่ง request ได้ไม่เกิน 10 req/s" — Semaphore ช่วยควบคุม

```
import asyncio async def fetch_historical(symbol: str, date: str, sem: asyncio.Semaphore) -> dict: """ดึงข้อมูล historical พร้อม rate limiting""" async with sem: # เข้า critical section — ไม่เกิน N พร้อมกัน await asyncio.sleep(0.1) # จำลอง API call return {"symbol": symbol, "date": date, "close": 65000.0} async def bulk_fetch_historical(symbols: list[str], dates: list[str]) -> list[dict]: """ดึงข้อมูลหลายวันพร้อมกัน แต่จำกัดไม่เกิน 5 request พร้อมกัน""" sem = asyncio.Semaphore(5) # อนุญาต 5 concurrent requests tasks = [ fetch_historical(sym, date, sem) for sym in symbols for date in dates ] print(f"ดึงข้อมูล {len(tasks)} records ด้วย semaphore=5") results = await asyncio.gather(*tasks) return list(results) asyncio.run(bulk_fetch_historical( symbols=["BTCUSDT", "ETHUSDT"], dates=["2024-01-01", "2024-01-02", "2024-01-03", "2024-01-04", "2024-01-05"] ))
```

▶ แบบฝึกหัด: สร้าง MultiQueue สำหรับรับข้อมูลจากหลาย exchange แล้วรวมเข้า Queue เดียว

```
async def multi_exchange_fan_in(exchanges: list[str], symbol: str,
n_ticks: int = 5) -> None: """รับ tick จากหลาย exchange → รวมเข้า 1 output
queue""" output_q: asyncio.Queue[PriceTick] = asyncio.Queue() async
def single_feed(exchange: str): base = 65000.0 + {"binance": 0,
"bybit": 10, "okx": -5}[exchange] for _ in range(n_ticks): price =
base + random.uniform(-30, 30) await output_q.put(PriceTick(exchange,
symbol, round(price, 2))) await asyncio.sleep(random.uniform(0.05,
0.15)) await output_q.put(None) # sentinel ต่อ exchange async def
consumer(n_producers: int): finished = 0 while finished < n_producers:
tick = await output_q.get() if tick is None: finished += 1 else:
print(f"[FAN-IN] {tick.exchange:8s}: {tick.price:,.1f}")
output_q.task_done() await asyncio.gather( *[single_feed(ex) for ex in
exchanges], consumer(len(exchanges)) )
asyncio.run(multi_exchange_fan_in( ["binance", "bybit", "okx"],
"BTCUSDT" ))
```

บทที่ 33: WebSocket & Live Data

แก่นของบทนี้

WebSocket คือ connection แบบ "เปิดค้างไว้" ระหว่าง client และ server — ต่างจาก HTTP ที่ต้อง request ทุกครั้ง Exchange ส่ง tick/candle ให้เราทันทีที่มีการเปลี่ยนแปลง ทำให้ latency ต่ำกว่า polling มาก แต่ต้องจัดการ disconnect, reconnect, และ parse message เองทั้งหมด

33.1 Binance Futures WebSocket — รูปแบบ Message จริง

Binance Futures ส่ง kline (candle) stream ในรูปแบบ JSON นี้ เมื่อ subscribe ที่ `wss://fstream.binance.com/ws/btcusdt@kline_1m`

```
# ตัวอย่าง kline message จาก Binance Futures WebSocket (format จริง)
BINANCE_KLINE_MSG = { "e": "kline", # event type "E": 1705123456789, #
event time (ms) "s": "BTCUSDT", # symbol "k": { "t": 1705123440000, # kline
start time (ms) "T": 1705123499999, # kline close time (ms) "s": "BTCUSDT",
# symbol "i": "1m", # interval "f": 100, # first trade ID "L": 200, # last
trade ID "o": "65000.10", # open "c": "65123.45", # close (current price)
"h": "65200.00", # high "l": "64950.00", # low "v": "123.456", # base asset
volume (BTC) "n": 1250, # number of trades "x": False, # is this kline
closed? False = ยังไม่ปิด candle "q": "8023456.78", # quote asset volume (USDT)
"V": "61.234", # taker buy base volume "Q": "3987654.32", # taker buy quote
volume } } # ตัวอย่าง bookTicker message จาก Binance (best bid/ask)
BINANCE_BOOKTICKER_MSG = { "e": "bookTicker", "u": 400900217, # order book
updateId "s": "BTCUSDT", "b": "65000.00", # best bid price "B": "5.000", #
best bid qty "a": "65001.00", # best ask price "A": "3.500", # best ask qty
"T": 1705123456789, # transaction time "E": 1705123456790, # event time }
```

33.2 Bybit WebSocket — รูปแบบ Message

```
# Bybit V5 WebSocket — wss://stream.bybit.com/v5/public/linear # Subscribe
message: BYBIT_SUBSCRIBE = { "op": "subscribe", "args": ["kline.1.BTCUSDT"]
# kline.{interval}.{symbol} } # Kline message จาก Bybit V5: BYBIT_KLINE_MSG
= { "topic": "kline.1.BTCUSDT", "data": [ { "start": 1705123440000, # start
time ms "end": 1705123499999, # end time ms "interval": "1", # interval in
minutes "open": "65000.10", "close": "65123.45", "high": "65200.00", "low":
"64950.00", "volume": "123.456", # base volume (BTC) "turnover":
```

```
"8023456.78", # quote volume (USDT) "confirm": False, # False = candle ยังไม่ปิด "timestamp": 1705123456789 } ], "ts": 1705123456789, "type": "snapshot" # หรือ "delta" }
```

33.3 WebSocket Client ที่ Reconnect อัตโนมัติ

```
import asyncio import json import logging import random import websockets from dataclasses import dataclass from datetime import datetime from typing import Callable, Optional logger = logging.getLogger(__name__) @dataclass class Candle: exchange: str symbol: str interval: str open: float high: float low: float close: float volume: float is_closed: bool timestamp: datetime class BinanceWebSocketClient: """ Binance Futures WebSocket client - reconnect อัตโนมัติด้วย exponential backoff - parse kline messages - ส่ง Candle object เข้า callback """ WS_URL = "wss://fstream.binance.com/ws" MAX_RECONNECT_DELAY = 60.0 # วินาที INITIAL_RECONNECT_DELAY = 1.0 def __init__(self, symbol: str, interval: str, on_candle: Callable[[Candle], None], queue: Optional[asyncio.Queue] = None): self.symbol = symbol.lower() self.interval = interval self.on_candle = on_candle self.queue = queue self._stop_event = asyncio.Event() self._reconnect_delay = self.INITIAL_RECONNECT_DELAY @property def stream_url(self) -> str: return f"{self.WS_URL}/{self.symbol}@kline_{self.interval}" def _parse_kline(self, raw: dict) -> Optional[Candle]: """แปลง Binance kline message เป็น Candle object""" try: k = raw["k"] return Candle( exchange = "binance", symbol = raw["s"], interval = k["i"], open = float(k["o"]), high = float(k["h"]), low = float(k["l"]), close = float(k["c"]), volume = float(k["v"]), is_closed = bool(k["x"]), timestamp = datetime.utcfromtimestamp(raw["E"] / 1000) ) except (KeyError, ValueError, TypeError) as e: logger.error(f"Binance parse error: {e} | raw={str(raw)[:200]}") return None async def _connect_and_listen(self): """เชื่อมต่อและฟัง messages — 1 connection attempt""" logger.info(f"Connecting to {self.stream_url}") async with websockets.connect( self.stream_url, ping_interval=20, # ส่ง ping ทุก 20s ping_timeout=10, close_timeout=5, ) as ws: logger.info(f"Connected: {self.symbol}@kline_{self.interval}") self._reconnect_delay = self.INITIAL_RECONNECT_DELAY # reset async for raw_msg in ws: if self._stop_event.is_set(): break try: msg = json.loads(raw_msg) if msg.get("e") == "kline": candle = self._parse_kline(msg) if candle is not None: # ข้ามถ้า parse ไม่ได้ self.on_candle(candle) # ถ้ามี queue ให้ put ด้วย if self.queue is not None: await self.queue.put(candle) except json.JSONDecodeError as e: logger.error(f"JSON decode error: {e}") except KeyError as e: logger.error(f"Unexpected message format: {e} | msg={raw_msg[:200]}") async def run(self): """รัน WebSocket พร้อม exponential backoff reconnect""" while not self._stop_event.is_set(): try: await self._connect_and_listen() except websockets.exceptions.ConnectionClosed as e: logger.warning(f"Connection closed: {e.code} {e.reason}") except websockets.exceptions.WebSocketException as e: logger.error(f"WebSocket error: {e}") except OSError as e: logger.error(f"Network error: {e}") if self._stop_event.is_set(): break # Exponential backoff + jitter ป้องกัน thundering herd logger.info(f"Reconnecting in {self._reconnect_delay:.1f}s...") await asyncio.sleep(self._reconnect_delay) self._reconnect_delay = min( self._reconnect_delay * 2, self.MAX_RECONNECT_DELAY ) + random.uniform(0, 1.0) logger.info("WebSocket client stopped") def
```

```
stop(self): """หยุด client อย่างสะอาด""" self._stop_event.set() class
BybitWebSocketClient: """ Bybit V5 WebSocket client endpoint: wss://
stream.bybit.com/v5/public/linear """ WS_URL = "wss://stream.bybit.com/v5/
public/linear" MAX_RECONNECT_DELAY = 60.0 INITIAL_RECONNECT_DELAY = 1.0 def
__init__(self, symbol: str, interval: str, on_candle: Callable[[Candle],
None], queue: Optional[asyncio.Queue] = None): self.symbol = symbol.upper()
self.interval = interval self.on_candle = on_candle self.queue = queue
self._stop_event = asyncio.Event() self._reconnect_delay =
self.INITIAL_RECONNECT_DELAY def _parse_kline(self, msg: dict) ->
list[Candle]: """แปลง Bybit kline message — อาจมีหลาย candle ใน 1 message"""
candles = [] for k in msg.get("data", []): try:
candles.append(Candle( exchange = "bybit", symbol = self.symbol, interval =
self.interval, open = float(k["open"]), high = float(k["high"]), low =
float(k["low"]), close = float(k["close"]), volume = float(k["volume"]),
is_closed = bool(k.get("confirm", False)), timestamp =
datetime.utcfromtimestamp(k["start"] / 1000) )) except (KeyError,
ValueError, TypeError) as e: logger.error(f"Bybit parse error: {e} |
k={str(k)[:200]}") return candles async def _connect_and_listen(self):
logger.info(f"Connecting to Bybit: {self.symbol} kline.{self.interval}")
async with websockets.connect( self.WS_URL, ping_interval=20,
ping_timeout=10, ) as ws: # ส่ง subscribe message sub_msg =
json.dumps({ "op": "subscribe", "args": [f"kline.{self.interval}.
{self.symbol}"] }) await ws.send(sub_msg) self._reconnect_delay =
self.INITIAL_RECONNECT_DELAY async for raw_msg in ws: if
self._stop_event.is_set(): break try: msg = json.loads(raw_msg) if
msg.get("op") == "subscribe": logger.info(f"Bybit subscribe:
{msg.get('ret_msg')}") continue if "topic" in msg and "kline" in
msg["topic"]: for candle in self._parse_kline(msg): self.on_candle(candle)
if self.queue is not None: await self.queue.put(candle) except
(json.JSONDecodeError, KeyError) as e: logger.error(f"Parse error: {e}")
async def run(self): while not self._stop_event.is_set(): try: await
self._connect_and_listen() except
(websockets.exceptions.WebSocketException, OSError) as e:
logger.warning(f"Connection error: {e}") if not self._stop_event.is_set():
await asyncio.sleep(self._reconnect_delay) self._reconnect_delay =
min( self._reconnect_delay * 2, self.MAX_RECONNECT_DELAY ) +
random.uniform(0, 1.0) def stop(self): self._stop_event.set()
```

33.4 Dual-Feed WebSocket — รับ Binance + Bybit พร้อมกัน

Running Example: รับ kline จาก Binance และ Bybit พร้อมกัน → ส่งเข้า Queue เดียว

นี่คือ core ของ live pair strategy — เราต้องการราคาจากทั้ง 2 exchange ในเวลาเดียวกันเพื่อคำนวณ spread

```
import asyncio import json import logging from collections import deque
from typing import Optional logger = logging.getLogger(__name__) async def
run_dual_feed(symbol: str = "BTCUSDT", interval: str = "1", max_candles:
int = 100, run_seconds: float = 30.0): """ รัน Binance + Bybit WebSocket พร้อม
กัน ส่ง candle ทั้งสองเข้า shared queue หยุดเองหลัง run_seconds วินาที (สำหรับ demo) """
shared_queue: asyncio.Queue[Candle] = asyncio.Queue(maxsize=500) stop_event
= asyncio.Event() # Buffer สำหรับคำนวณ spread candle_buf: dict[str, deque] =
{ "binance": deque(maxlen=max_candles), "bybit":
deque(maxlen=max_candles), } def on_candle(candle: Candle): # Callback รัน
synchronously — ใช้แค่ log if candle.is_closed:
logger.debug(f"[{candle.exchange}] Closed candle " f"{candle.symbol}
close={candle.close}") # สร้าง clients binance_client =
BinanceWebSocketClient( symbol=symbol, interval=f"{interval}m",
on_candle=on_candle, queue=shared_queue ) bybit_client =
BybitWebSocketClient( symbol=symbol, interval=interval,
on_candle=on_candle, queue=shared_queue ) async def process_candles():
"""Consumer: คำนวณ spread เมื่อได้ candle ปิดจากทั้งสอง""" while not
stop_event.is_set(): try: candle = await
asyncio.wait_for( shared_queue.get(), timeout=1.0 ) except
asyncio.TimeoutError: continue if candle.is_closed:
candle_buf[candle.exchange].append(candle.close) # คำนวณ spread เมื่อมีข้อมูลทั้งสอง
exchange if (len(candle_buf["binance"]) > 0 and len(candle_buf["bybit"]) >
0): b_price = candle_buf["binance"][-1] y_price = candle_buf["bybit"][-1]
spread = y_price - b_price logger.info(f"Spread: {spread:+.2f} "
f"(Binance={b_price:.1f}, Bybit={y_price:.1f})") shared_queue.task_done()
async def stopper(): await asyncio.sleep(run_seconds) stop_event.set()
binance_client.stop() bybit_client.stop() logger.info("Stop event set") #
รันทุก task พร้อมกัน await asyncio.gather( binance_client.run(),
bybit_client.run(), process_candles(), stopper(), return_exceptions=True )
# ใช้งาน: # asyncio.run(run_dual_feed("BTCUSDT", "1", run_seconds=300))
```

ข้อควรระวัง: WebSocket ใน Production

- **ตรวจ timestamp** — ถ้า message เก่ากว่า 5 วินาที อาจเป็น stale data → อย่าเทรด
- **Bybit ต้องการ re-subscribe** หลัง reconnect — ต้องส่ง subscribe message ใหม่ทุกครั้ง
- **Binance จำกัด 10 connections** ต่อ IP — ระวังถ้ารัน multiple instances
- **ใช้ testnet ก่อน**: Binance testnet: `wss://stream.binancefuture.com/ws`, Bybit testnet: `wss://stream-testnet.bybit.com/v5/public/linear`

▶ แบบฝึกหัด: เพิ่ม staleness check — ถ้า candle เก่ากว่า 30 วินาที ให้ log warning

```
from datetime import datetime, timezone def check_staleness(candle: Candle, max_age_seconds: float = 30.0) -> bool: """คืน True ถ้า candle ยังสด (ไม่เก่าเกิน max_age_seconds)""" now = datetime.now(timezone.utc).replace(tzinfo=None) age = (now - candle.timestamp).total_seconds() if age > max_age_seconds: logger.warning( f"[STALE] {candle.exchange} {candle.symbol} " f"candle age={age:.1f}s > {max_age_seconds}s — ข้าม" ) return False return True # ใช้ใน process_candles(): # if candle.is_closed and check_staleness(candle, max_age_seconds=30): # candle_buf[candle.exchange].append(candle.close)
```

บทที่ 34: Order Execution

แก่นของบทนี้

การส่ง order จริงซับซ้อนกว่า backtest มาก: ต้องมี HMAC signature, จัดการ rate limit, retry เมื่อ network error (แต่ไม่ retry ถ้า order ถูก reject), และติดตาม state ของ order ตั้งแต่ PENDING จนถึง FILLED หรือ CANCELLED

34.1 Order State Machine

```
Order State Machine:
────────────────────────────────────────────────────────────────────────────
┌──────────┐ │ PENDING │ (สร้าง order object ใน code) └──┬──┬──┘ │ ส่ง REST
request สำเร็จ ▼ ┌──────┐ ┌──►│ OPEN │ (exchange รับ order แล้ว) │ └──┬──┘
partial │ fill ├──────────────► CANCELLED │ │ fully filled └──────►
┌──────────┐ │ FILLED │ (ซื้อ/ขายครบแล้ว) └──────┘ Error path: PENDING ──►
REJECTED (invalid params, insufficient balance) OPEN ──► ERROR (network
lost, exchange down)
```

```
from enum import Enum, auto from dataclasses import dataclass, field from
typing import Optional import time class OrderStatus(Enum): PENDING =
auto() # สร้างใน code แต่ยังไม่ส่ง OPEN = auto() # ส่งไป exchange แล้ว รอ fill
PARTIALLY_FILLED = auto() FILLED = auto() # fill ครบแล้ว CANCELLED = auto()
REJECTED = auto() # exchange ปฏิเสธ ERROR = auto() # network/unknown error
class OrderSide(Enum): BUY = "Buy" SELL = "Sell" class OrderType(Enum):
MARKET = "MARKET" LIMIT = "LIMIT" @dataclass class Order: client_order_id:
str # ID ที่เราสร้างเอง (UUID หรือ timestamp) symbol: str side: OrderSide
order_type: OrderType qty: float price: Optional[float] = None # None สำหรับ
market order # Fields ที่ exchange จะส่งกลับมา exchange_order_id: Optional[str] =
None status: OrderStatus = OrderStatus.PENDING filled_qty: float = 0.0
avg_price: float = 0.0 created_at: float = field(default_factory=time.time)
updated_at: Optional[float] = None reject_reason: Optional[str] = None
@property def is_terminal(self) -> bool: """คืน True ถ้า order จบแล้ว (ไม่สามารถ
เปลี่ยนได้อีก)""" return self.status in ( OrderStatus.FILLED,
OrderStatus.CANCELLED, OrderStatus.REJECTED, OrderStatus.ERROR, ) @property
def remaining_qty(self) -> float: return self.qty - self.filled_qty def
__repr__(self): return (f"Order({self.client_order_id} "
```

```
f"{self.side.value} {self.qty} {self.symbol} " f"@ {'MKT' if self.price is
None else self.price} " f"[{self.status.name}])")
```

34.2 REST API Wrapper พร้อม Rate Limiting และ Retry

```
import asyncio import hashlib import hmac import json import os import time
import logging from urllib.parse import urlencode from typing import
Optional import aiohttp logger = logging.getLogger(__name__) class
BybitOrderClient: """ Bybit V5 REST API wrapper - Rate limiting ด้วย
asyncio.Semaphore - HMAC-SHA256 signature - Retry with exponential backoff
สำหรับ network error - ไม่ retry ถ้า exchange reject (4xx) """ # Testnet:
https://api-testnet.bybit.com BASE_URL = "https://api.bybit.com"
RECV_WINDOW = 5000 # ms def __init__(self, api_key: Optional[str] = None,
api_secret: Optional[str] = None, testnet: bool = True, max_concurrent: int
= 5): # ✓ โหลดจาก env var — ห้าม hardcode! self.api_key = api_key or
os.environ.get("BYBIT_API_KEY", "") self.api_secret = api_secret or
os.environ.get("BYBIT_API_SECRET", "") if testnet: self.BASE_URL =
"https://api-testnet.bybit.com" logger.warning("ใช้ TESTNET — ไม่ใช่ real
money") self._sem = asyncio.Semaphore(max_concurrent) self._session:
Optional[aiohttp.ClientSession] = None async def __aenter__(self):
self._session = aiohttp.ClientSession() return self async def
__aexit__(self, *args): if self._session: await self._session.close() def
_sign(self, params: dict) -> str: """สร้าง HMAC-SHA256 signature""" timestamp
= str(int(time.time() * 1000)) params_str =
urlencode(sorted(params.items())) payload = f"{timestamp}{self.api_key}
{self.RECV_WINDOW}{params_str}" signature =
hmac.new( self.api_secret.encode("utf-8"), payload.encode("utf-8"),
hashlib.sha256 ).hexdigest() return signature, timestamp async def
_request(self, method: str, path: str, params: dict = None, max_retries:
int = 3) -> dict: """ส่ง signed request พร้อม retry""" params = params or {}
signature, timestamp = self._sign(params) headers = { "X-BAPI-API-KEY":
self.api_key, "X-BAPI-SIGN": signature, "X-BAPI-TIMESTAMP": timestamp, "X-
BAPI-RECV-WINDOW": str(self.RECV_WINDOW), "Content-Type": "application/
json", } url = f"{self.BASE_URL}{path}" for attempt in range(max_retries):
async with self._sem: # rate limiting try: if method == "GET": async with
self._session.get( url, params=params, headers=headers,
timeout=aiohttp.ClientTimeout(total=10) ) as resp: data = await resp.json()
else: async with self._session.post( url, json=params, headers=headers,
timeout=aiohttp.ClientTimeout(total=10) ) as resp: data = await resp.json()
# Bybit ret_code: 0 = OK, อื่นๆ = error if data.get("retCode") != 0: ret_msg
= data.get("retMsg", "unknown") ret_code = data.get("retCode") # Error ที่ไม่
ควร retry NO_RETRY_CODES = { 10001, # Parameter error 10004, # Sign check
failed 110007, # Insufficient balance 110013, # Position is in liquidation
110025, # Position not exist } if ret_code in NO_RETRY_CODES: raise
ValueError( f"Exchange rejected [{ret_code}]: {ret_msg}" )
logger.warning(f"API error [{ret_code}]: {ret_msg}, " f"attempt
{attempt+1}/{max_retries}") return data except (aiohttp.ClientError,
asyncio.TimeoutError) as e: if attempt == max_retries - 1: raise delay = 2
** attempt logger.warning(f"Network error: {e}, retry in {delay}s") await
asyncio.sleep(delay) raise RuntimeError(f"Failed after {max_retries}
attempts") async def place_market_order(self, symbol: str, side: OrderSide,
```

```
qty: float, client_order_id: str) -> Order: """ส่ง market order""" order =
Order( client_order_id=client_order_id, symbol=symbol, side=side,
order_type=OrderType.MARKET, qty=qty, ) try: resp = await
self._request("POST", "/v5/order/create", { "category": "linear", "symbol":
symbol, "side": side.value, "orderType": "Market", "qty": str(qty),
"orderLinkId": client_order_id, }) if resp.get("retCode") == 0: result =
resp["result"] order.exchange_order_id = result.get("orderId") order.status
= OrderStatus.OPEN logger.info(f"Order placed: {order}") else: order.status
= OrderStatus.REJECTED order.reject_reason = resp.get("retMsg")
logger.error(f"Order rejected: {order.reject_reason}") except ValueError as
e: order.status = OrderStatus.REJECTED order.reject_reason = str(e)
logger.error(f"Order rejected by exchange: {e}") except Exception as e:
order.status = OrderStatus.ERROR order.reject_reason = str(e)
logger.error(f"Order error: {e}") order.updated_at = time.time() return
order async def place_limit_order(self, symbol: str, side: OrderSide, qty:
float, price: float, client_order_id: str) -> Order: """ส่ง limit order"""
order = Order( client_order_id=client_order_id, symbol=symbol, side=side,
order_type=OrderType.LIMIT, qty=qty, price=price, ) try: resp = await
self._request("POST", "/v5/order/create", { "category": "linear", "symbol":
symbol, "side": side.value, "orderType": "Limit", "qty": str(qty), "price":
str(price), "timeInForce": "PostOnly", # maker order เท่านั้น "orderLinkId":
client_order_id, }) if resp.get("retCode") == 0: result = resp["result"]
order.exchange_order_id = result.get("orderId") order.status =
OrderStatus.OPEN logger.info(f"Limit order placed: {order}") else:
order.status = OrderStatus.REJECTED order.reject_reason =
resp.get("retMsg") except ValueError as e: order.status =
OrderStatus.REJECTED order.reject_reason = str(e) except Exception as e:
order.status = OrderStatus.ERROR order.reject_reason = str(e)
order.updated_at = time.time() return order async def cancel_order(self,
symbol: str, client_order_id: str) -> bool: """ยกเลิก order""" resp = await
self._request("POST", "/v5/order/cancel", { "category": "linear", "symbol":
symbol, "orderLinkId": client_order_id, }) success = resp.get("retCode") ==
0 if success: logger.info(f"Cancelled order {client_order_id}") else:
logger.warning(f"Cancel failed: {resp.get('retMsg')}") return success async
def get_order_status(self, symbol: str, client_order_id: str) ->
Optional[Order]: """ดึง order status จาก exchange""" resp = await
self._request("GET", "/v5/order/realtime", { "category": "linear",
"symbol": symbol, "orderLinkId": client_order_id, }) if
resp.get("retCode") != 0: return None items = resp["result"].get("list",
[]) if not items: return None item = items[0] STATUS_MAP = { "New":
OrderStatus.OPEN, "PartiallyFilled": OrderStatus.PARTIALLY_FILLED,
"Filled": OrderStatus.FILLED, "Cancelled": OrderStatus.CANCELLED,
"Rejected": OrderStatus.REJECTED, } order = Order( client_order_id =
client_order_id, symbol = symbol, side = OrderSide.BUY if item["side"] ==
"Buy" else OrderSide.SELL, order_type = OrderType.MARKET if
item["orderType"] == "Market" else OrderType.LIMIT, qty =
float(item["qty"]), price = float(item["price"]) if item.get("price") else
None, exchange_order_id = item["orderId"], status =
STATUS_MAP.get(item["orderStatus"], OrderStatus.ERROR), filled_qty =
float(item.get("cumExecQty", 0)), avg_price = float(item.get("avgPrice",
0)), ) return order
```

34.3 ทดสอบบน Testnet

```
import asyncio import uuid async def demo_order_flow(): """ ทดสอบ order flow บน Bybit Testnet ต้องตั้ง env var: export BYBIT_API_KEY=your_testnet_key export BYBIT_API_SECRET=your_testnet_secret """ async with BybitOrderClient(testnet=True) as client: # สร้าง unique client order ID coid = f"PAIR-{uuid.uuid4().hex[:8].upper()}" print(f"Placing market BUY 0.001 BTCUSDT (coid={coid})") order = await client.place_market_order( symbol = "BTCUSDT", side = OrderSide.BUY, qty = 0.001, client_order_id = coid, ) print(f"Result: {order}") # รอให้ order settle await asyncio.sleep(1.0) # ดึง status updated = await client.get_order_status("BTCUSDT", coid) if updated: print(f"Updated status: {updated.status.name}, " f"filled={updated.filled_qty}, avg={updated.avg_price}") # asyncio.run(demo_order_flow())
```

ข้อควรระวัง: Order Execution ในโลกจริง

- **ห้าม retry ทุก error** — ถ้า network timeout แล้ว retry → order อาจถูกส่ง 2 ครั้ง ใช้ client_order_id เพื่อ idempotency
- **Market order slip ได้** — ราคา fill อาจต่างจากราคาตลาดมาก โดยเฉพาะ pair ที่ volume น้อย
- **ตรวจ balance ก่อนส่ง order** — error 110007 (insufficient balance) ไม่ควรเกิดถ้าระบบออกแบบดี
- **Log ทุก order response** รวมถึง timestamp, exchange_order_id, และ status

▶ แบบฝึกหัด: เขียน OrderTracker class ที่เก็บ history ของทุก order และ query ได้

```
from collections import defaultdict from typing import Dict, List
class OrderTracker: """เก็บ history ของทุก order — query ได้ด้วย symbol หรือ
status""" def __init__(self): self._orders: Dict[str, Order] = {} #
coid -> Order self._by_symbol: Dict[str, List[str]] =
defaultdict(list) def track(self, order: Order) -> None:
self._orders[order.client_order_id] = order if order.client_order_id
not in self._by_symbol[order.symbol]:
self._by_symbol[order.symbol].append(order.client_order_id) def
update(self, order: Order) -> None: """อัปเดต order ที่มีอยู่แล้ว""" if
order.client_order_id in self._orders:
self._orders[order.client_order_id] = order def get(self, coid: str) -
> Optional[Order]: return self._orders.get(coid) def open_orders(self,
symbol: str = None) -> List[Order]: all_orders =
list(self._orders.values()) open_list = [o for o in all_orders if
o.status in (OrderStatus.PENDING, OrderStatus.OPEN,
OrderStatus.PARTIALLY_FILLED)] if symbol: open_list = [o for o in
open_list if o.symbol == symbol] return open_list def
filled_pnl_summary(self, symbol: str = None) -> dict: orders = [o for
o in self._orders.values() if o.status == OrderStatus.FILLED] if
symbol: orders = [o for o in orders if o.symbol == symbol] buys = [o
for o in orders if o.side == OrderSide.BUY] sells = [o for o in orders
if o.side == OrderSide.SELL] return { "total_filled": len(orders),
"buy_count": len(buys), "sell_count": len(sells), } def
__repr__(self): return (f"OrderTracker(total={len(self._orders)}, "
f"open={len(self.open_orders())})")
```

บทที่ 35: Live System Integration

แก่นของบทนี้

ประกอบทุกอย่างเข้าด้วยกันเป็น live trading system ที่ production-ready: logging ที่ structured, health check loop, kill switch ที่ทำงานได้จริง, graceful shutdown, และ pre-flight checklist ก่อน go-live

35.1 Structured Logging

ใน live system logging ต้องเป็น structured (JSON) เพื่อ search และ filter ง่าย — ไม่ใช่แค่ print string

```
import logging import json import sys from datetime import datetime,
timezone class StructuredFormatter(logging.Formatter): """ Formatter ที่
output JSON — ง่ายต่อการ search ด้วย jq หรือ ELK stack """ def format(self, record:
logging.LogRecord) -> str: log_obj = { "ts":
datetime.now(timezone.utc).isoformat(), "level": record.levelname,
"logger": record.name, "msg": record.getMessage(), } # เพิ่ม extra fields ถ้ามี
for key in ("symbol", "order_id", "price", "side", "pnl", "error"): if
hasattr(record, key): log_obj[key] = getattr(record, key) if
record.exc_info: log_obj["exception"] =
self.formatException(record.exc_info) return json.dumps(log_obj,
ensure_ascii=False) def setup_logging(level: str = "INFO", log_file: str =
None) -> logging.Logger: """ตั้งค่า logging สำหรับ live system""" logger =
logging.getLogger("live_trading") logger.setLevel(getattr(logging,
level.upper())) # Console handler console =
logging.StreamHandler(sys.stdout)
console.setFormatter(StructuredFormatter()) logger.addHandler(console) #
File handler (ถ้าระบุ) if log_file: fh = logging.FileHandler(log_file,
encoding="utf-8") fh.setFormatter(StructuredFormatter())
logger.addHandler(fh) return logger # ใช้งาน logger =
setup_logging(level="INFO", log_file="live_trading.log") # Log พร้อม extra
context logger.info("Order placed", extra={ "symbol": "BTCUSDT",
"order_id": "ORD-001", "side": "BUY", "price": 65000.0 })
```

▶ OUTPUT

```
{"ts": "2024-01-15T10:30:00.123Z", "level": "INFO", "logger": "live_trading", "msg": "Ord
```

35.2 Kill Switch ด้วย asyncio.Event

```
import asyncio import signal import logging logger =
logging.getLogger("live_trading") class KillSwitch: """ Kill switch สำหรับหยุด
ระบบ live trading รองรับ: Ctrl+C, SIGTERM (Docker stop), programmatic
(drawdown exceeded) """ def __init__(self): self._event = asyncio.Event()
self._reason: str = "" def trigger(self, reason: str = "manual"): """ทริกเกอร์
kill switch""" self._reason = reason self._event.set()
logger.critical(f"KILL SWITCH TRIGGERED: {reason}") async def wait(self):
"""รอจนกว่า kill switch จะทำงาน""" await self._event.wait() @property def
is_set(self) -> bool: return self._event.is_set() @property def
reason(self) -> str: return self._reason def setup_signal_handlers(self,
loop: asyncio.AbstractEventLoop): """ตั้งค่า OS signal handlers""" def
_handle_signal(sig_name: str): logger.warning(f"Received {sig_name}")
self.trigger(f"OS signal: {sig_name}") # add_signal_handler รองรับเฉพาะ Unix/
Linux/macOS (ไม่ทำงานบน Windows) if sys.platform != "win32": for sig in
(signal.SIGINT, signal.SIGTERM): loop.add_signal_handler( sig, lambda
s=sig: _handle_signal(s.name) ) else: # Windows: ใช้ signal.signal() แทน
(synchronous) import signal as _signal _signal.signal(_signal.SIGINT,
lambda s, f: _handle_signal("SIGINT"))
```

35.3 Health Check Loop

```
import asyncio import time from dataclasses import dataclass, field
@dataclass class SystemHealth: last_binance_tick: float = 0.0 # timestamp
ของ tick ล่าสุดจาก Binance last_bybit_tick: float = 0.0 last_order_time: float =
0.0 total_pnl: float = 0.0 drawdown_pct: float = 0.0 peak_equity: float =
0.0 current_equity: float = 0.0 order_errors: int = 0 feed_disconnects: int
= 0 def update_equity(self, equity: float): if equity > self.peak_equity:
self.peak_equity = equity self.current_equity = equity if self.peak_equity
> 0: self.drawdown_pct = (equity - self.peak_equity) / self.peak_equity
async def health_check_loop(health: SystemHealth, kill_switch: KillSwitch,
check_interval: float = 5.0, max_drawdown: float = -0.05, max_feed_silence:
float = 30.0): """ ตรวจสุขภาพระบบทุก check_interval วินาที หยุดอัตโนมัติถ้า: - drawdown
เกิน max_drawdown - ไม่ได้รับ feed นานกว่า max_feed_silence วินาที """ logger =
logging.getLogger("live_trading.health") while not kill_switch.is_set:
await asyncio.sleep(check_interval) now = time.time() # ตรวจ drawdown if
health.drawdown_pct < max_drawdown: kill_switch.trigger( f"Max drawdown
exceeded: {health.drawdown_pct:.2%} < {max_drawdown:.2%}" ) break # ตรวจ
feed silence binance_silence = now - health.last_binance_tick bybit_silence
= now - health.last_bybit_tick if health.last_binance_tick > 0 and
binance_silence > max_feed_silence: logger.error(f"Binance feed silent for
{binance_silence:.0f}s") kill_switch.trigger(f"Binance feed silent
{binance_silence:.0f}s") break if health.last_bybit_tick > 0 and
bybit_silence > max_feed_silence: logger.error(f"Bybit feed silent for
{bybit_silence:.0f}s") kill_switch.trigger(f"Bybit feed silent
{bybit_silence:.0f}s") break # ตรวจ order errors if health.order_errors >=
5: kill_switch.trigger( f"Too many order errors: {health.order_errors}" )
break logger.info( f"Health OK | equity={health.current_equity:,.0f} "
```

```
f"drawdown={health.drawdown_pct:.2%} "
f"order_errors={health.order_errors}" )
```

35.4 ระบบ Live Trading ครบวงจร

Running Example: BTC Pair Live Trading System

ประกอบทุก component จาก บท 31–34 เป็น production-ready system สมบูรณ์

```
import asyncio import logging import os import time import uuid from
collections import deque from typing import Optional #
============================================================ #
Configuration — โหลดจาก env var เสมอ #
============================================================ CONFIG =
{ "symbol": "BTCUSDT", "interval": "1", # 1 minute candles "pair_window":
60, # lookback candles "entry_zscore": 2.0, "exit_zscore": 0.5,
"order_qty": 0.001, # BTC per trade "max_drawdown": -0.05, # หยุดถ้า drawdown
เกิน 5% "max_feed_silence": 30.0, # หยุดถ้าไม่มี feed 30s "initial_capital":
10_000.0, # USDT "testnet": True, # เปลี่ยนเป็น False เมื่อพร้อม live "log_level":
"INFO", "log_file": "btc_pair_live.log", } class BTCPairLiveSystem: """
Live trading system สำหรับ BTC pair strategy Binance/Bybit spread → Z-score →
market order """ def __init__(self, config: dict): self.config = config
self.logger = setup_logging( config["log_level"], config.get("log_file") )
self.kill_switch = KillSwitch() self.health = SystemHealth( peak_equity =
config["initial_capital"], current_equity = config["initial_capital"], )
self.order_tracker = OrderTracker() # Price buffers w =
config["pair_window"] self._binance_closes: deque[float] = deque(maxlen=w)
self._bybit_closes: deque[float] = deque(maxlen=w) # Position state
self._position: Optional[str] = None # "LONG", "SHORT", หรือ None
self._shared_queue: asyncio.Queue[Candle] = asyncio.Queue(maxsize=1000) #
── Callbacks ──────────────────────────────────────── def
_on_binance_candle(self, candle: Candle): self.health.last_binance_tick =
time.time() def _on_bybit_candle(self, candle: Candle):
self.health.last_bybit_tick = time.time() # ── Signal Logic
──────────────────────────────────────── def _compute_zscore(self) ->
Optional[float]: """คำนวณ z-score ของ spread Binance-Bybit""" w =
self.config["pair_window"] if (len(self._binance_closes) < w or
len(self._bybit_closes) < w): return None import statistics spreads = [ b -
y for b, y in zip( list(self._binance_closes), list(self._bybit_closes) ) ]
mean = statistics.mean(spreads) stdev = statistics.stdev(spreads) if stdev
== 0: return None return (spreads[-1] - mean) / stdev def _get_signal(self,
zscore: float) -> str: entry = self.config["entry_zscore"] exit_ =
self.config["exit_zscore"] if zscore > entry: return "SHORT" if zscore < -
entry: return "LONG" if abs(zscore) < exit_: return "EXIT" return "HOLD" #
── Order Execution ──────────────────────────────────── async def
_execute_signal(self, signal: str, client: BybitOrderClient): """แปลง signal
เป็น order""" if signal == "HOLD": return if signal == self._position: return
# อยู่ใน position นี้อยู่แล้ว qty = self.config["order_qty"] sym =
```

```
self.config["symbol"] coid = f"PAIR-{uuid.uuid4().hex[:8].upper()}" if
signal in ("LONG", "SHORT"): # ปิด position เดิมก่อน (ถ้ามี) if self._position is
not None: close_side = (OrderSide.SELL if self._position == "LONG" else
OrderSide.BUY) close_coid = f"CLOSE-{uuid.uuid4().hex[:8].upper()}"
close_order = await client.place_market_order( sym, close_side, qty,
close_coid ) self.order_tracker.track(close_order) if close_order.status ==
OrderStatus.ERROR: self.health.order_errors += 1 # เปิด position ใหม่ side =
OrderSide.BUY if signal == "LONG" else OrderSide.SELL order = await
client.place_market_order(sym, side, qty, coid)
self.order_tracker.track(order) if order.status in (OrderStatus.OPEN,
OrderStatus.FILLED): self._position = signal self.logger.info( f"Position
opened: {signal}", extra={"symbol": sym, "order_id": coid, "side":
side.value} ) else: self.health.order_errors += 1
self.logger.error( f"Order failed: {order.reject_reason}",
extra={"order_id": coid} ) elif signal == "EXIT" and self._position is not
None: side = (OrderSide.SELL if self._position == "LONG" else
OrderSide.BUY) order = await client.place_market_order(sym, side, qty,
coid) self.order_tracker.track(order) if order.status in (OrderStatus.OPEN,
OrderStatus.FILLED): self._position = None self.logger.info( "Position
closed", extra={"symbol": sym, "order_id": coid} ) else:
self.health.order_errors += 1 # ─── Main Consumer Loop
──────────────────────────────────── async def _signal_loop(self, client:
BybitOrderClient): """ประมวลผล candle จาก queue และส่ง order""" while not
self.kill_switch.is_set: try: candle = await
asyncio.wait_for( self._shared_queue.get(), timeout=1.0 ) except
asyncio.TimeoutError: continue # เฉพาะ closed candle if candle.is_closed:
if candle.exchange == "binance": self._binance_closes.append(candle.close)
elif candle.exchange == "bybit": self._bybit_closes.append(candle.close)
zscore = self._compute_zscore() if zscore is not None: signal =
self._get_signal(zscore) self.logger.debug( f"zscore={zscore:.3f}
signal={signal}" ) if not self.kill_switch.is_set: await
self._execute_signal(signal, client) self._shared_queue.task_done() # ───
Graceful Shutdown ──────────────────────────────────── async def
_shutdown(self, client: BybitOrderClient): """ปิด position ทั้งหมดก่อน
shutdown""" self.logger.warning("Graceful shutdown: ปิด open positions...")
if self._position is not None: side = (OrderSide.SELL if self._position ==
"LONG" else OrderSide.BUY) coid = f"SHUTDOWN-{uuid.uuid4().hex[:
8].upper()}" order = await
client.place_market_order( self.config["symbol"], side,
self.config["order_qty"], coid ) self.order_tracker.track(order)
self.logger.info( f"Shutdown order: {order.status.name}",
extra={"order_id": coid} ) self._position = None
self.logger.info( f"Shutdown complete | "
f"total_orders={len(self.order_tracker._orders)} | "
f"drawdown={self.health.drawdown_pct:.2%}" ) # ─── Run
──────────────────────────────────────────────────── async def run(self):
"""เข้าสู่การทำงาน live""" loop = asyncio.get_event_loop()
self.kill_switch.setup_signal_handlers(loop) self.logger.info("Starting BTC
Pair Live System") self.logger.info(f"Config: {self.config}") # WebSocket
clients binance_client = BinanceWebSocketClient( symbol =
self.config["symbol"], interval = f"{self.config['interval']}m", on_candle
= self._on_binance_candle, queue = self._shared_queue, ) bybit_client =
BybitWebSocketClient( symbol = self.config["symbol"], interval =
```

```
self.config["interval"], on_candle = self._on_bybit_candle, queue =
self._shared_queue, ) async with
BybitOrderClient(testnet=self.config["testnet"]) as order_client: try:
await asyncio.gather( binance_client.run(), bybit_client.run(),
self._signal_loop(order_client), health_check_loop( self.health,
self.kill_switch, max_drawdown = self.config["max_drawdown"],
max_feed_silence = self.config["max_feed_silence"], ),
self.kill_switch.wait(), # รอจนกว่า kill switch return_exceptions=True, )
finally: binance_client.stop() bybit_client.stop() await
self._shutdown(order_client) # ─── Entry point
──────────────────────────────────────────────── if __name__ == "__main__":
system = BTCPairLiveSystem(CONFIG) asyncio.run(system.run())
```

35.5 Pre-Flight Checklist ก่อน Go Live

```
import asyncio import os import aiohttp async def preflight_check(config:
dict) -> bool: """ ตรวจสอบความพร้อมก่อนเริ่ม live trading คืน True เฉพาะเมื่อผ่านทุก check
""" results = {} logger = logging.getLogger("live_trading.preflight") # 1.
ตรวจ API Keys api_key = os.environ.get("BYBIT_API_KEY", "") api_secret =
os.environ.get("BYBIT_API_SECRET", "") results["api_keys"] = bool(api_key
and api_secret and len(api_key) > 10 and len(api_secret) > 10) if not
results["api_keys"]: logger.error("FAIL: BYBIT_API_KEY หรือ BYBIT_API_SECRET
ไม่ได้ตั้งค่า") # 2. ตรวจการเชื่อมต่อ exchange try: async with aiohttp.ClientSession()
as session: url = ("https://api-testnet.bybit.com/v5/market/time" if
config.get("testnet") else "https://api.bybit.com/v5/market/time") async
with session.get(url, timeout=aiohttp.ClientTimeout(total=5)) as r: data =
await r.json() results["connectivity"] = data.get("retCode") == 0 except
Exception as e: results["connectivity"] = False logger.error(f"FAIL: ไม่
สามารถเชื่อมต่อ exchange: {e}") # 3. ตรวจ balance if results.get("api_keys") and
results.get("connectivity"): try: async with
BybitOrderClient(testnet=config.get("testnet", True)) as client: resp =
await client._request("GET", "/v5/account/wallet-balance", {"accountType":
"UNIFIED"}) if resp.get("retCode") == 0: coins = resp["result"]["list"][0]
["coin"] usdt = next( (float(c["availableToWithdraw"]) for c in coins if
c["coin"] == "USDT"), 0.0 ) min_balance = config.get("initial_capital",
1000) * 0.5 results["balance"] = usdt >= min_balance if not
results["balance"]: logger.error( f"FAIL: USDT balance {usdt:.2f} <
required {min_balance:.2f}" ) else: logger.info(f"OK: USDT balance =
{usdt:,.2f}") else: results["balance"] = False except Exception as e:
results["balance"] = False logger.error(f"FAIL: ดึง balance ไม่ได้: {e}") # 4.
ตรวจ config cfg_ok = ( config.get("order_qty", 0) > 0 and
config.get("max_drawdown", 0) < 0 and config.get("entry_zscore", 0) >
config.get("exit_zscore", 0) > 0 ) results["config"] = cfg_ok if not
cfg_ok: logger.error("FAIL: Config ไม่ถูกต้อง") # 5. สรุป all_pass =
all(results.values()) print("\n=== Pre-Flight Check ===") for name, ok in
results.items(): status = "✓ PASS" if ok else "✗ FAIL" print(f" {name:20s}:
{status}") print(f"\n{'→ READY TO LAUNCH' if all_pass else '→ FIX ISSUES
FIRST'}\n") return all_pass # asyncio.run(preflight_check(CONFIG))
```

Deployment Checklist — ก่อน Go Live จริง

- **ทดสอบบน Testnet ≥ 1 สัปดาห์** — ให้ระบบวิ่งตลอดเวลา ดู drawdown และ error logs
- **Paper trading บน real feed** — รับ feed จริงแต่ไม่ส่ง order จริง เพื่อตรวจ latency
- **ตั้ง max position size เล็กๆ** ตอนเริ่มต้น เช่น 0.001 BTC ก่อน scale ขึ้น
- **มี monitoring ภายนอก** — อย่าเชื่อแค่ log ของตัวเอง ใช้ Telegram bot หรือ Grafana alert
- **Deploy บน VPS ใกล้ exchange** — Singapore หรือ Tokyo เพื่อ latency ต่ำ
- **ตั้ง systemd หรือ Docker** ให้ restart อัตโนมัติถ้า process crash
- **Log rotation** — log file โตได้มาก ตั้ง logrotate หรือ size limit

▶ แบบฝึกหัด: เพิ่ม Telegram alert เมื่อ kill switch ทำงาน หรือเมื่อ fill order ใหญ่

```
import aiohttp import os TELEGRAM_BOT_TOKEN =
os.environ.get("TELEGRAM_BOT_TOKEN", "") TELEGRAM_CHAT_ID =
os.environ.get("TELEGRAM_CHAT_ID", "") async def
send_telegram(message: str) -> bool: """ส่ง alert ผ่าน Telegram Bot""" if
not TELEGRAM_BOT_TOKEN or not TELEGRAM_CHAT_ID: return False url =
f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage" try:
async with aiohttp.ClientSession() as session: async with
session.post(url, json={ "chat_id": TELEGRAM_CHAT_ID, "text": message,
"parse_mode": "HTML" }, timeout=aiohttp.ClientTimeout(total=5)) as
resp: return resp.status == 200 except Exception: return False # ใน
KillSwitch.trigger(): async def trigger_with_alert(self, reason: str =
"manual"): self._reason = reason self._event.set()
logger.critical(f"KILL SWITCH TRIGGERED: {reason}") await
send_telegram( f"   KILL SWITCH\n" f"Reason: {reason}\n" f"Time:
{datetime.utcnow().isoformat()}Z" ) # ใน _execute_signal() หลัง order
filled: async def notify_fill(order: Order): msg = (f"   Order
Filled\n" f"Symbol: {order.symbol}\n" f"Side: {order.side.value}\n"
f"Qty: {order.filled_qty}\n" f"AvgPrice: {order.avg_price:,.2f}")
await send_telegram(msg)
```

จบหนังสือ — เส้นทางต่อไป

การเดินทางจาก PART 0 ถึง PART VI

คุณผ่านการเรียนรู้มาไกลมาก — ตั้งแต่ Python พื้นฐานจนถึงระบบ live trading อัตโนมัติ
สรุปทุกอย่างที่ได้เรียนในหนังสือเล่มนี้:

Part	เนื้อหา	สิ่งที่ได้
Part 0	Python Foundations	Syntax, data types, functions, comprehensions
Part I	Data Wrangling	pandas, numpy, matplotlib — ดึง/ทำความสะอาด/visualize ข้อมูลตลาด
Part II	Math Tools	Rolling stats, z-score, cointegration, pair trading signal
Part III	OOP	Classes, ABC, Strategy/Observer/Factory patterns — system ที่ขยายได้
Part IV	Backtesting	Walk-forward, slippage, commission, overfitting, Sharpe/drawdown
Part V	AI-Assisted Coding	Prompt engineering, code generation, audit checklist, quant libraries, full strategy with AI
Part VI	Async & Live Trading	asyncio, WebSocket, order execution, kill switch, monitoring

```
เส้นทางการพัฒนาในฐานะ Quant Developer:
──────────────────────────────────────────────────────────────────────────────
[หนังสือนี้] Part 0-I ──▶ Part II-III ──▶ Part IV-V ──▶ Part VI
Python basics Math + OOP Backtest Live System
│ │ │ │
▼ ▼ ▼ ▼
[ขั้นต่อไป] NumPy/Pandas statsmodels vectorbt/ production เชี่ยวชาญ
PyMC (Bayes) zipline deployment Nautilus cloud infra
```

แนะนำขั้นต่อไปหลังจบหนังสือ

1. Backtesting เชิงลึก

- **vectorbt** — vectorized backtesting เร็วกว่า pandas loop 100x
- **Nautilus Trader** — professional backtester + live trading framework ที่ใช้ใน production
- **Zipline-reloaded** — backtester ของ Quantopian ที่ยังมี community ดูแล

2. Statistical Methods เพิ่มเติม

- **Bayesian Inference** ด้วย PyMC — ประเมิน parameter uncertainty แทน point estimate
- **Kalman Filter** — dynamic hedge ratio ที่ปรับตามเวลา (statsmodels มี DLM)
- **Regime Detection** ด้วย Hidden Markov Model — ตรวจว่าตลาดอยู่ใน trend/mean-revert/volatile

3. Machine Learning for Finance

- **Feature Engineering** — microstructure features, order book imbalance, funding rate
- **Gradient Boosting** (XGBoost/LightGBM) สำหรับ signal classification
- **LSTM / Transformer** สำหรับ time series — แต่ระวัง overfitting
- **Reinforcement Learning** (Stable-Baselines3) — ให้ model เรียนรู้ position sizing

4. Infrastructure & Operations

- **Docker + docker-compose** — containerize trading system
- **TimescaleDB** — PostgreSQL extension สำหรับ time series data ใน production
- **Grafana + Prometheus** — monitoring dashboard สำหรับ live system
- **Redis Streams** — message queue สำหรับ high-throughput tick data

5. หนังสือและแหล่งเรียนรู้แนะนำ

- Advances in Financial Machine Learning — Marcos Lopez de Prado (อ่านหลังจากมีพื้นฐาน ML)
- Algorithmic Trading — Ernest Chan (practical, Python examples)
- Active Portfolio Management — Grinold & Kahn (classic quant text)
- QuantLib Python — pricing library สำหรับ options และ derivatives
- Quantopian Lectures (archive) — beginner-friendly quant tutorials

คำเตือนสุดท้าย — จากนักพัฒนาถึงนักพัฒนา

หนังสือนี้สอนวิธี *สร้าง* trading system — ไม่ใช่วิธีรวยจากตลาด

• Strategy ที่ backtest ได้ดีส่วนมากไม่ได้ผลใน live — นั่นคือความจริงของ quant
• Market เปลี่ยนไปตลอด — strategy ที่ดีวันนี้อาจไม่ดีพรุ่งนี้
• การ risk management และ position sizing สำคัญกว่า entry signal มาก
• **ไม่มี strategy ไหนที่ไม่มีความเสี่ยง** — จัดการความเสี่ยงด้วยระบบ ไม่ใช่ด้วยความหวัง

จงสร้างระบบที่ fail safely, log everything, และ never risk what you can't afford to lose