Assigment 5: Testing & CI in Financial Engineering

100 Points Possible

10/19/2025

Attempt 1

In Progress

NEXT UP: Submit Assignment

Add Comment

Unlimited Attempts Allowed

Details

Assignment 5: Testing & CI in Financial Engineering

You can clone a starter repo from here: **https://github.com/hyoung3/assignment5-testing-CI (https://github.com/hyoung3/assignment5-testing-CI)**

- **Duration:** ~5–6 hours
- **Focus:** Unit tests, coverage, and CI for a minimal daily-bar backtester (PnL is *not* the goal—engineering discipline is).

🎯 Learning Objectives

- Design testable components (data loader, strategy, broker, backtester).
- Write focused unit tests with `pytest`, fixtures, and mocks.
- Measure and enforce coverage (target ≥90%) and keep tests fast.
- Wire up GitHub Actions to run tests + coverage on every push/PR.

📘 What You'll Build

A tiny daily backtester with:

- **PriceLoader:** returns a `pandas.Series` of prices for a single symbol (use synthetic data for tests).
- **Strategy:** outputs daily signals (`-1, 0, +1` or booleans) from price history.
- **Broker:** accepts market orders, updates cash/position with no slippage/fees (keep deterministic for tests).
- **Backtester:** runs end-of-day loop: compute signal (t–1), trade at close (t), track cash/position/equity.

> You'll implement **one simple strategy** (e.g., *VolatilityBreakoutStrategy*). This strategy calculates a rolling x-day standard deviation of returns and buys when the current return is > this x-day figure.

> The assignment is graded on **tests + CI**, not alpha.

⚙️ Constraints

- Tests must **not** hit the network or external APIs — *mock or generate data.*
- Test suite must complete in **< 60 seconds** on GitHub Actions.
- Coverage **fails CI** if `< 90%` (branches optional, lines required).

🗂️ Repository Layout (Suggested)

```
trading-ci-lab/
  backtester/
    __init__.py
    price_loader.py
    strategy.py
    broker.py
    engine.py
  tests/
    test_strategy.py
    test_broker.py
    test_engine.py
    conftest.py
  requirements.txt
  pyproject.toml        # or setup.cfg for pytest/coverage options
  .github/workflows/ci.yml
  README.md
```

🧩 Part 1 — CI Wiring (30–45 min)

Create a GitHub repo and add workflow:

```
# .github/workflows/ci.yml
name: CI Pipeline
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: "3.11"
      - run: pip install -r requirements.txt
      - run: coverage run -m pytest -q
      - run: coverage report --fail-under=90
```

requirements.txt (minimum)

```
pytest
coverage
```

```
pandas
numpy
```

> 💡 *Tip:* Add a coverage badge locally with `coverage-badge` (optional).

🧩 Part 2 — Minimal Components (45–60 min)

Implement only what you need to support tests:

```
# backtester/strategy.py
import numpy as np
import pandas as pd

class VolatilityBreakoutStrategy:

    def __init__(self):
        pass

    def signals(self, prices: pd.Series) -> pd.Series:
        pass
```

```
# backtester/broker.py
class Broker:
    def __init__(self, cash: float = 1_000_000):
        self.cash = cash
        self.position = 0

    def market_order(self, side: str, qty: int, price: float):
        pass
```

```
# backtester/engine.py
import pandas as pd

class Backtester:
    def __init__(self, strategy, broker):
        self.strategy = strategy
        self.broker = broker

    def run(self, prices: pd.Series):
        pass
```

🧩 Part 3 — Tests, Fixtures, and Mocks (2–3 hours)

Write **focused** tests. Keep them deterministic and fast.

Required Tests

- **Strategy logic:** signal generation of x-day volatility breakout.
- **Broker behavior:** buy/sell adjusts cash/position correctly, rejects bad inputs, raises on insufficient cash/shares.
- **Engine loop:** executes trades; final equity matches cash + pos×price.

- **Edge cases:** empty series, constant price series, NaNs at head, very short series.
- **Failure handling:** demonstrate one mocked failure path (e.g., broker raising) and assert it propagates/logs as expected.

Example Fixtures & Mocks

```
# tests/conftest.py
import numpy as np, pandas as pd, pytest
from backtester.strategy import VolatilityBreakoutStrategy
from backtester.broker import Broker

@pytest.fixture
def prices():
    # deterministic rising series
    return pd.Series(np.linspace(100, 120, 200))

@pytest.fixture
def strategy():
    return VolatilityBreakoutStrategy()

@pytest.fixture
def broker():
    return Broker(cash=1_000)
```

```
# tests/test_strategy.py
def test_signals_length(strategy, prices):
    sig = strategy.signals(prices)
    assert len(sig) == len(prices)
```

```
# tests/test_broker.py
import pytest

def test_buy_and_sell_updates_cash_and_pos(broker):
    broker.market_order("BUY", 2, 10.0)
    assert (broker.position, broker.cash) == (2, 1000 - 20.0)

def test_rejects_bad_orders(broker):
    with pytest.raises(ValueError):
        broker.market_order("BUY", 0, 10)
```

```
# tests/test_engine.py
from unittest.mock import MagicMock
from backtester.engine import Backtester

# example
def test_engine_uses_tminus1_signal(prices, broker, strategy, monkeypatch):
    # Force exactly one buy at t=10 by controlling signals
    fake_strategy = MagicMock()
    fake_strategy.signals.return_value = prices*0
    fake_strategy.signals.return_value.iloc[9] = 1  # triggers buy at t=10
    bt = Backtester(fake_strategy, broker)
    eq = bt.run(prices)
    assert broker.position == 1
    assert broker.cash == 1000 - float(prices.iloc[10])
```

> Use `unittest.mock` / `MagicMock` and monkey-patching sparingly to isolate external dependencies; test your core logic directly.

🧩 Part 4 — Coverage & Reporting (30–45 min)

Run locally:

```
coverage run -m pytest -q
coverage report -m
coverage html
```

CI must **fail** if coverage < 90%:

```
coverage report --fail-under=90
```

Commit the HTML report (optional) or attach screenshots in the README.

✅ Deliverables (Checklist)

- [ ] Code for `PriceLoader`, `Strategy`, `Broker`, `Backtester` (minimal but clean).
- [ ] `tests/` with comprehensive unit tests and fixtures.
- [ ] Passing GitHub Actions run (link/screenshot).
- [ ] Coverage report showing **≥ 90%.**
- [ ] `README.md` with design notes, how to run tests, CI status, coverage summary.

🧮 Grading Rubric (100 pts)

Category	Points	Description
Unit tests quality & breadth	40	Clear, isolated, meaningful assertions; good edge-case coverage.
Coverage	20	≥90% lines (18 pts for 90–94, 20 pts for ≥95).
CI integration	20	Workflow runs on push/PR, fails on low coverage, fast and reliable.
Design & clarity	10	Simple, readable code; minimal but sensible abstractions.
Determinism & speed	10	No network, seeded/synthetic data, suite < 60s.

Keep in mind, this submission will count for everyone in your Fall 2025 group.

Choose a submission type

Text

Web URL

Upload

More

Submit Assignment