MyIndicator (Static + Dynamic Asset Allocation System)

본 프로젝트는 **정적(Static) 자산배분 전략**과
동적(Dynamic) 모멘텀 기반 리밸런싱 전략을 결합한
백테스트 및 라이브 리밸런싱 프레임워크입니다.

로컬에 저장된 시계열 가격 데이터(parquet)를 기반으로

- 과거 성과를 검증하는 **백테스트**
- 주기적으로 실행되는 **라이브 리밸런싱(실거래 연계)**
  를 동일한 전략 구조로 수행할 수 있도록 설계되었습니다.

1. 프로젝트 개요

- Static / Dynamic 자산배분 전략 결합
- parquet 기반 고속 백테스트
- YAML 설정 중심 전략 제어
- 백테스트 ↔ 라이브 전략 구조 통일
- KIS API 기반 실거래 연계 가능 구조

2. 프로젝트 전체 구조

```
MyIndicator-main/
├── config/                 # YAML 설정 파일
│   ├── data.yaml           # 데이터 다운로드 설정
│   └── backtester.yaml     # 백테스트 / 전략 설정
│
├── infra/                  # 인프라 계층
│   ├── config.py
│   ├── data_manager/
│   │   └── downloader.py   # 주가 데이터 다운로드
│   └── api/
│       └── broker/
│           └── kis.py      # KIS API 래퍼
│
├── research/               # 백테스트 계층
│   └── backtester/
│       ├── cli.py
│       ├── runner.py
│       ├── master.py
│       ├── universe.py
│       ├── data_loader.py
│       └── strategies/
│           ├── static_engine.py
│           └── dynamic_engine.py
│
├── core/                   # 전략 코어
│   ├── engine.py
│   ├── factors.py
│   ├── scoring.py
│   ├── signals.py
│   ├── allocators.py
│   └── rebalancing/
│       ├── triggers.py
│       └── methods.py
│
├── live/                   # 라이브 리밸런싱
│   ├── cli.py
│   ├── engine.py
│   ├── state.py
│   ├── execution/
│   │   └── order_manager.py
```

```
│   └── strategy/
│       ├── static.py
│       └── dynamic.py
│
└── data/
    ├── raw/
    └── processed/
```

3. 데이터 다운로드 및 저장

```
python -m infra.data_manager.downloader --config data
```

- `config/data.yaml` 기준 주가 데이터 수집
- parquet 형식으로 로컬 저장
- 증분 다운로드 및 스킵 기능 지원
- raw / processed 데이터 분리 관리

4. 백테스트 실행

```
python -m research.backtester.cli --config backtester
```

백테스트 흐름

1. 설정 로드
2. 로컬 parquet 데이터 로딩
3. Static / Dynamic 전략 초기화
4. 날짜별 리밸런싱 여부 판단
5. 포트폴리오 가치(NAV) 계산
6. 성과 지표 및 결과 저장

5. 리밸런싱 전략 개요

본 프로젝트는 **정적 리밸런싱**과 **동적 리밸런싱**을 함께 사용합니다.

6. 정적(Static) 리밸런싱

핵심 개념

> 정적 리밸런싱은 **indicator를 사용하지 않는 전략**

사용 요소

- 종가(Close price): 자산 가치 평가
- 시간(Time): 리밸런싱 트리거

특징

- 사전에 정의된 자산 및 비중 유지
- Monthly / Quarterly 등 주기 기반
- 안정성 중심의 패시브 전략

7. 동적(Dynamic) 리밸런싱 요약

> **N일 수익률 기반 모멘텀으로 종목을 랭킹 →**
> **상관계수로 분산 필터링 →**
> **설정된 주기에 따라 목표 비중으로 리밸런싱**

사용 Indicator 요약

- 모멘텀 (N일 수익률)
- 변동성 (표준편차, 보조)
- 종목 간 상관계수
- 랭킹 기반 Top-N 선정

RSI, MACD, 이동평균 등 전통적 기술적 지표는 사용하지 않습니다.

8. 동적 리밸런싱 상세 (Indicator 기준)

8.1 입력 데이터

- 종가(close price)
- 설정 기반 룩백 윈도우

8.2 모멘텀 Indicator (핵심)

```
returns = prices.pct_change(window)
momentum = returns.iloc[-1]
```

- 단순 수익률
- 최근 시점 값만 사용

8.3 변동성 Indicator (보조)

```
vol = prices.pct_change().rolling(window).std()
```

- 고변동성 종목 억제 목적

8.4 상관계수 Indicator

```
corr = returns.corr()
```

- 종목 간 중복 제거
- 분산 투자 강제

8.5 랭킹 및 종목 선정

- 모멘텀 점수 내림차순 정렬

- Top-N 종목 선택
- 최소/최대 보유 종목 수 제어

9. 정적 vs 동적 Indicator 비교

구분	정적	동적
Indicator 사용	❌	✅
주요 지표	없음	모멘텀, 상관계수
판단 기준	시간	가격 신호
전략 성격	패시브	세미 액티브
목적	안정성	초과수익

10. 라이브 리밸런싱

```
python -m live.cli
```

라이브 1회 사이클

1. 유니버스 구성
2. 시장 데이터 로딩
3. 계좌 상태 조회
4. 동적 종목 선정
5. 정적/동적 비중 합성
6. 손절 로직 적용
7. 리밸런싱 날짜 판정
8. 주문 실행
9. 상태 저장

⚠️ `MarketDataProvider` 구현은 별도 보완 필요

11. 실행 요약

```
python -m infra.data_manager.downloader --config data
python -m research.backtester.cli --config backtester
python -m live.cli
```

12. 한 문장 요약

> 정적 리밸런싱은 지표 없이 시간 기준으로 비중을 유지하는 전략이며,
> 동적 리밸런싱은 모멘텀·변동성·상관계수 등 가격 기반 indicator를 활용해
> 종목을 선택하고 주기적으로 포트폴리오를 재구성하는 전략이다.