OP - Options Pipeline: A Comprehensive Market Data Analytics System

A resilient, multi-stage analytics pipeline for Indian index options (NIFTY 50, BANKNIFTY, SENSEX) that has evolved through multiple development phases to become a sophisticated system handling data collection, processing, health monitoring, and advanced analytics.

Table of Contents

System Evolution & Architecture

Evolution Timeline

```
timeline
    title System Evolution Timeline
    section Phase 1 : Basic Collection
        2024-Q1 : Logger Runner
                : ATM Option Collector
                : Basic InfluxDB writes
    section Phase 2 : Health & Monitoring
        2024-Q2 : Health monitoring system
                : Grafana dashboards
                : Error handling & recovery
    section Phase 3 : Advanced Analytics
        2024-Q3 : CSV daily splits
```

```
                : Minute merge logic
                : Weekday master profiles
    section Phase 4 : Testing & Reliability
        2024-Q4 : Comprehensive test suite
                : TEST_MODE implementation
                : Mock framework
```

Current Architecture Overview

```
flowchart TD
    subgraph "Data Sources"
        API[Kite/Broker API]
        MARKET[Market Feeds]
    end

    subgraph "Core Collection Layer"
        RUNNER[logger_runner.py<br/>Main Orchestrator]
        ATM[atm_option_collector.py<br/>Option Chain Data]
        OV[overview_collector.py<br/>Index Overview]
    end

    subgraph "Data Processing Pipeline"
        JSON[JSON Snapshots<br/>Raw Legs]
        CSV_SIDE[CSV Sidecars<br/>Tabular Format]
        MERGE[minute_merge.py<br/>CE+PE Joining]
        SPLITS[Daily Split CSVs<br/>Per Index/Expiry/Offset]
    end

    subgraph "Advanced Analytics"
        AGG[adv_aggregator.py<br/>Streaming + EOD]
        MASTERS[Weekday Masters<br/>Time Profile Analytics]
        PAIRS[Pair Offsets<br/>p1m1, p2m2]
    end

    subgraph "Storage & Monitoring"
        INFLUX[(InfluxDB<br/>Time Series)]
        GRAFANA[Grafana Dashboards<br/>Live Monitoring]
        HEALTH[Health System<br/>Status & Alerts]
    end

    API --> RUNNER
    MARKET --> RUNNER
    RUNNER --> ATM
    RUNNER --> OV
    ATM --> JSON
    ATM --> CSV_SIDE
    OV --> JSON
    JSON --> MERGE
    CSV_SIDE --> MERGE
    MERGE --> SPLITS
    SPLITS --> AGG
    AGG --> MASTERS
    AGG --> PAIRS
    ATM --> INFLUX
    OV --> INFLUX
```

```
    AGG --> INFLUX
    HEALTH --> INFLUX
    INFLUX --> GRAFANA
```

Repository Structure

```
OP/
├── app/
│   ├── advanced/                    # Advanced Analytics Layer
│   │   ├── adv_config.py           # Configuration & Environment
│   │   ├── adv_io.py               # Atomic CSV I/O Operations
│   │   ├── adv_aggregator.py       # Main aggregation engine
│   │   ├── adv_influx_writer.py    # Advanced metrics to InfluxDB
│   │   ├── adv_ledger.py           # EOD idempotence tracking
│   │   └── adv_instruments.py      # Instrument validation
│   │
│   ├── brokers/                     # Broker Integration
│   │   ├── kite_client.py          # Zerodha Kite API client
│   │   ├── kite_helpers.py         # Utility functions
│   │   ├── kite_instruments.py     # Instrument mapping & ATM logic
│   │   └── expiry_discovery.py     # Weekly/Monthly expiry detection
│   │
│   ├── collectors/                  # Data Collection Core
│   │   ├── atm_option_collector.py # ATM±2 option chain collector
│   │   ├── overview_collector.py   # Index overview metrics
│   │   ├── csv_sidecar.py         # JSON→CSV mirroring
│   │   ├── minute_merge.py         # CE+PE merging per offset
│   │   └── csv_daily_split_writer.py # Authoritative daily splits
│   │
│   ├── sinks/                       # Data Output Handlers
│   │   └── influx_sink.py         # Type-safe InfluxDB writes
│   │
│   ├── storage/                     # Storage Abstractions
│   │   └── influx_writer.py       # InfluxDB client wrapper
│   │
│   ├── utils/                       # Shared Utilities
│   │   ├── time_utils.py          # Time handling & IST conversion
│   │   └── constants.py           # System-wide constants
│   │
│   └── monitors/                    # Health & Performance
│       └── health_writer.py       # System health metrics
│
├── scripts/                         # Entry Points & Tools
│   ├── logger_runner.py           # Main data collection orchestrator
│   ├── logger_health.py           # Health check coordinator
│   ├── logger_utils.py            # Console output & timing
│   ├── influx_smoke.py            # InfluxDB connectivity test
│   ├── health_monitor.py          # System health monitoring
│   └── run_analytics_supervisor.py # Analytics orchestration
│
├── data/                            # Data Storage Hierarchy
│   ├── raw_snapshots/             # JSON snapshots (audit trail)
│   │   ├── options/               # Option leg snapshots
│   │   └── overview/              # Index overview snapshots
```

```
│   │
│   ├── csv_data/                  # Daily split CSVs (authoritative)
│   │   ├── NIFTY/                # Per-index organization
│   │   ├── BANKNIFTY/
│   │   └── SENSEX/
│   │
│   └── data_adv/                  # Advanced analytics outputs
│       └── weekday_masters/       # Weekday time profiles
│
├── tests/                          # Test Suite
│   ├── test_atm_option_collector.py
│   ├── test_overview_collector.py
│   ├── test_logger_runner.py
│   ├── test_influx_writer.py
│   ├── test_influx_roundtrip.py
│   └── conftest.py                # Test configuration
│
├── dashboards/                     # Grafana Dashboard Configs
│   ├── market-sentiment-updated.json
│   ├── option-liquidity-pricing.json
│   ├── greeks-positioning.json
│   └── technical-levels-dashboard.json
│
├── config/                         # Configuration Files
│   ├── .env.template              # Environment template
│   └── logging.conf               # Logging configuration
│
├── requirements.txt               # Python dependencies
├── docker-compose.yml             # Local development stack
├── README.md                      # This comprehensive guide
└── .gitignore                     # Git ignore rules
```

Data Flow Architecture

Primary Data Flow

```
flowchart LR
    subgraph "Ingestion Layer"
        A[Market API] --> B[ATM Collector]
        A --> C[Overview Collector]
    end

    subgraph "Processing Layer"
        B --> D[JSON Legs]
        C --> E[JSON Overview]
        D --> F[CSV Sidecar]
        D --> G[Minute Merge]
        F --> G
        G --> H[Daily Split CSV]
    end

    subgraph "Analytics Layer"
        H --> I[Advanced Aggregator]
```

```
        I --> J[Weekday Masters]
        I --> K[Pair Offsets]
    end

    subgraph "Storage & Viz"
        B --> L[(InfluxDB)]
        C --> L
        I --> L
        L --> M[Grafana Dashboards]
    end
```

Data Contract Specifications

Time Contract

- **IST Bucketing**: All minute-level aggregation uses Asia/Kolkata timezone
- **UTC Storage**: Metadata timestamps stored in UTC ISO8601
- **Last Tick Wins**: Multiple ticks per minute → last value persists

File Contract

- **Authoritative Source**: Daily split CSVs are canonical input for advanced analytics
- **Path Structure**: `MIRROR_SPLIT_ROOT/<INDEX>/<expiry_code>/<strike_offset>/<YYYY-MM-DD>.csv`
- **Required Columns**: Timestamp column + either `total_premium`/`tp_sum` OR `call_last_price`+`put_last_price`

Index Contract

- **Input Naming**: NIFTY, BANKNIFTY, SENSEX (folder structure)
- **Output Naming**: "NIFTY 50", "NIFTY BANK", "SENSEX" (canonical display)
- **Strike Steps**: NIFTY=50, BANKNIFTY=100, SENSEX=100

Core vs Supporting Modules

Core Modules (Non-Negotiable)

```
graph TD
    subgraph "Essential Core"
        A[logger_runner.py] --> B[atm_option_collector.py]
        A --> C[overview_collector.py]
        B --> D[minute_merge.py]
        C --> D
        D --> E[csv_daily_split_writer.py]
        E --> F[adv_aggregator.py]
    end
```

```
    subgraph "Supporting Infrastructure"
        G[kite_client.py] --> A
        H[influx_writer.py] --> A
        I[health_writer.py] --> A
    end

    style A fill:#e1f5fe
    style B fill:#e1f5fe
    style C fill:#e1f5fe
    style D fill:#e1f5fe
    style E fill:#e1f5fe
    style F fill:#e1f5fe
```

Core Functionality Requirements:

- ✅ **Data Collection**: ATM option collector + overview collector
- ✅ **Data Processing**: Minute merge + daily split generation
- ✅ **Advanced Analytics**: Weekday master aggregation
- ✅ **Health Monitoring**: System status tracking
- ✅ **Storage**: InfluxDB time series + CSV file persistence

Supporting Modules (Enhancing)

Module	Purpose	Impact if Removed
`csv_sidecar.py`	JSON→CSV mirroring	Loss of immediate tabular access
`adv_influx_writer.py`	Advanced metrics to DB	No advanced dashboard data
`adv_ledger.py`	EOD idempotence	Potential duplicate processing
`health_monitor.py`	Continuous health checks	Manual health verification needed
`influx_smoke.py`	DB connectivity testing	Manual connection validation

Optional Modules (Nice-to-Have)

- `adv_instruments.py` - Instrument validation (can use manual verification)
- `expiry_discovery.py` - Automated expiry detection (can use manual config)
- Advanced dashboard JSON configs (can use basic monitoring)

Script-by-Script Deep Dive

Primary Orchestrator: `logger_runner.py`

```
# Core Logic Flow
def main():
    # 1. Authentication & Client Setup
    kite = get_kite_client()
    ensure_token = lambda: _oauth_login(kite)._KiteConnect__access_token

    # 2. Infrastructure Initialization
    writer = InfluxWriter()
    health = HealthScheduler(writer=writer)

    # 3. Collector Initialization
    atm_col = ATMOptionCollector(kite, ensure_token, writer)
    ov_col = OverviewCollector(kite, ensure_token, atm_col, writer)

    # 4. Main Collection Loop
    while market_is_open():
        # Collect option data
        legs_res = atm_col.collect(offsets=OFFSETS)
        ov_data = ov_col.collect()

        # Process & merge
        merged = merge_call_put_to_rows(legs_res['legs'])

        # Write daily splits
        for (idx, exp, off), rows in merged['merged_map'].items():
            append_rows(CSV_ROOT, idx, exp, off, date, rows)

        # Health monitoring
        health.tick(broker_state, csv_stats)
        time.sleep(LOOP_INTERVAL)
```

Non-Negotiable Behaviors:

- Must maintain market hours awareness
- Must handle authentication token refresh
- Must emit health metrics every loop
- Must write both InfluxDB and CSV outputs

Data Collection Engine: `atm_option_collector.py`

```
class ATMOptionCollector:
    def collect(self, offsets=[-2,-1,0,1,2]):
        legs = []
        for index in SUPPORTED_INDICES:
            spot = self._spot_price(index)  # Get current spot price
            atm = round(spot/STEP_SIZES[index]) * STEP_SIZES[index]

            # Discover expiry buckets
            weeklies, monthlies = self._discover_expiries(index, atm)
```

```
            for bucket, expiry in [('this_week', weeklies[0]), ...]:
                # For each offset around ATM
                for offset in offsets:
                    strike = atm + offset * STEP_SIZES[index]

                    # Find CE and PE instruments
                    ce_token, pe_token = self._find_instruments(strike, expiry)

                    # Fetch quotes
                    quotes = safe_call(self.kite, 'quote', [ce_token, pe_token])

                    # Process each leg
                    for side, token in [('CALL', ce_token), ('PUT', pe_token)]:
                        leg_data = self._build_leg_record(quotes[token], ...)
                        legs.append(leg_data)

                        # Write outputs
                        self._write_json_snapshot(leg_data)
                        write_atm_leg(leg_data, self.influx_writer)

        return {'legs': legs, 'overview_aggs': overview_data}
```

Key Features:

- ✅ **Dynamic Expiry Discovery**: Automatically finds weekly/monthly expiries
- ✅ **ATM Strike Calculation**: Precise strike rounding per index
- ✅ **Implied Volatility Fallback**: Black-Scholes calculation when broker IV unavailable
- ✅ **Multi-Format Output**: JSON snapshots + InfluxDB points
- ✅ **Error Recovery**: Safe API calls with retry logic

Advanced Analytics Engine: `adv_aggregator.py`

```
class AdvancedAggregator:
    def process_streaming(self):
        """Per-minute streaming updates"""
        for index in self.indices:
            for expiry in self.expiries:
                for offset in self.offsets:
                    # Read today's split CSV
                    df = self._read_split_csv(index, expiry, offset, today)

                    # Bucket by IST minute (HH:MM)
                    bucketed = self._bucket_by_minute(df)

                    # Update weekday master
                    weekday = today.weekday()  # 0=Monday
                    master_path = self._get_master_path(index, expiry, offset, weekday)

                    for minute, total in bucketed.items():
                        self._update_master_counters(master_path, minute, total)

    def process_eod(self, date):
```

```
        """End-of-day canonical reconciliation"""
        # Similar to streaming but with pair derivation
        for index in self.indices:
            self._derive_missing_pairs(index, date)
            self._reconcile_all_masters(index, date)
```

Processing Logic:

1. **CSV Reading**: Robust parsing with Windows file lock handling
2. **Time Bucketing**: IST HH:MM aggregation with last-tick-wins
3. **Total Calculation**: `total_premium` OR `call_price + put_price`
4. **Master Updates**: Incremental counters (n, sum, avg, min, max)
5. **Pair Derivation**: p1m1 = avg(+1 offset, -1 offset) when both present

Dashboard Ecosystem

Current Grafana Dashboards

1. Market Sentiment Dashboard (`market-sentiment-updated.json`)

```
graph LR
    subgraph "Market Sentiment Panels"
        A[Index Overview<br/>NIFTY/BANKNIFTY/SENSEX]
        B[Option Chain Heatmap<br/>Strike vs Time]
        C[Put-Call Ratio<br/>Trending Analysis]
        D[Implied Volatility<br/>Term Structure]
    end

    subgraph "Data Sources"
        E[(InfluxDB)]
        F[index_overview measurement]
        G[atm_option_quote measurement]
    end

    F --> A
    G --> B
    G --> C
    G --> D
    E --> F
    E --> G
```

Key Metrics:

- Real-time index levels with day change %
- ATM implied volatility tracking
- Put-call ratio for sentiment analysis
- Strike-wise option activity heatmaps

2. Option Liquidity & Pricing (`option-liquidity-pricing.json`)

Panels:

- Volume & Open Interest by strike
- Bid-ask spread analysis
- Time decay visualization
- Liquidity depth metrics

3. Greeks & Positioning (`greeks-positioning.json`)

Advanced Analytics:

- Delta positioning (dealer gamma stance)
- Charm levels (time decay acceleration)
- Gamma exposure by strike
- Dealer positioning inference

4. Technical Levels (`technical-levels-dashboard.json`)

Support/Resistance:

- Key option strike clustering
- Max pain calculations
- Support/resistance from option OI
- Breakout probability analysis

Dashboard Data Dependencies

```
flowchart TD
    subgraph "InfluxDB Measurements"
        A[atm_option_quote]
        B[index_overview]
        C[weekday_master_updates]
        D[monitor_status]
    end

    subgraph "Dashboard Categories"
        E[Market Sentiment]
        F[Liquidity Analysis]
        G[Greeks & Positioning]
        H[Technical Levels]
        I[System Health]
    end

    A --> E
    A --> F
    A --> G
```

```
    A --> H
    B --> E
    B --> H
    C --> G
    D --> I
```

Testing Framework

Current Test Structure

```
graph TD
    subgraph "Unit Tests"
        A[test_atm_option_collector.py]
        B[test_overview_collector.py]
        C[test_influx_writer.py]
        D[test_logger_runner.py]
    end

    subgraph "Integration Tests"
        E[test_influx_roundtrip.py]
        F[test_end_to_end_flow.py]
    end

    subgraph "Mock Framework"
        G[TEST_MODE Environment]
        H[Kite Client Mocks]
        I[InfluxDB Mocks]
    end

    A --> G
    B --> G
    C --> I
    D --> H
    E --> I
    F --> G
```

Test Categories & Coverage

Unit Tests (Isolated Component Testing)

```
# Example: test_atm_option_collector.py
class TestATMOptionCollector(unittest.TestCase):
    @patch('app.brokers.kite_client.get_kite_client')
    def test_collect_five_strikes(self, mock_kite):
        # Mock setup
        mock_kite.return_value.quote.return_value = MOCK_QUOTES

        # Test execution
        collector = ATMOptionCollector(mock_kite(), lambda: 'token')
        result = collector.collect(offsets=[-2,-1,0,1,2])
```

```
        # Assertions
        self.assertEqual(len(result['legs']), 10)  # 5 strikes × 2 sides
        self.assertIn('strike', result['legs'][0])
        self.assertIn('iv', result['legs'][0])
```

Integration Tests (End-to-End Flow)

```
# Example: test_end_to_end_flow.py
class TestEndToEndFlow(unittest.TestCase):
    def test_complete_pipeline(self):
        """Test: Collection → Processing → Storage → Analytics"""
        with tempfile.TemporaryDirectory() as tmpdir:
            # Setup test environment
            os.environ['TEST_MODE'] = '1'
            os.environ['MIRROR_SPLIT_ROOT'] = tmpdir

            # Execute pipeline
            self._run_collection_loop()
            self._verify_csv_splits_created()
            self._run_advanced_aggregator()
            self._verify_weekday_masters_updated()
```

Development Testing Strategy

Online/Live Branch Testing

```
flowchart LR
    subgraph "Live Market Testing"
        A[Health Monitor<br/>Real-time Checks] --> B[Alert System<br/>Slack/Email]
        C[Smoke Tests<br/>Every 5 minutes] --> D[Auto Recovery<br/>Process Restart]
        E[Data Quality Checks<br/>Missing legs, IV outliers] --> F[Quality Reports<br/>Er
    end
```

Live Testing Components:

- `health_monitor.py` - Continuous system health verification
- `influx_smoke.py` - Database connectivity validation
- Quality checks for missing option legs or IV anomalies
- Automated alerts for system failures

Offline Branch Testing (Market Closed)

```
flowchart TD
    subgraph "Development Testing"
        A[Mock Data Playback<br/>Historical JSON snapshots]
        B[Unit Test Suite<br/>All components isolated]
        C[Integration Tests<br/>Full pipeline simulation]
```

```
        D[Performance Tests<br/>Load & memory profiling]
    end

    subgraph "Test Data Sources"
        E[Historical CSV Splits]
        F[Mock API Responses]
        G[Synthetic Option Chains]
    end

    E --> A
    F --> B
    G --> C
```

Offline Testing Features:

- Historical data replay for development
- Comprehensive mock framework for broker APIs
- Performance benchmarking with synthetic loads
- Database schema migration testing

Expanding the Testing Module

Proposed Test Enhancements

1. **Property-Based Testing**

```
from hypothesis import given, strategies as st

@given(st.floats(min_value=15000, max_value=25000))
def test_atm_calculation_properties(spot_price):
    """Test ATM calculation maintains expected properties"""
    atm = calculate_atm_strike(spot_price, index='NIFTY')
    assert atm % 50 == 0  # Must be multiple of step size
    assert abs(atm - spot_price) < 50  # Within one step
```

2. **Chaos Engineering Tests**

```
def test_network_failure_recovery():
    """Test system behavior during network interruptions"""
    with NetworkFailureSimulator(failure_rate=0.1):
        collector = ATMOptionCollector(...)
        results = collector.collect_with_retry(max_retries=3)
        assert len(results['legs']) > 0  # Should recover
```

3. **Data Quality Validation**

```
def test_iv_outlier_detection():
    """Test detection of suspicious IV values"""
    data = load_test_option_data()
```

```
    outliers = detect_iv_outliers(data, threshold=3.0)
    assert all(outlier['iv'] > 2.0 for outlier in outliers)
```

Health Monitoring System

Health Architecture

```
flowchart TD
    subgraph "Health Components"
        A[HealthScheduler<br/>Coordinator] --> B[InfluxDB Health<br/>Connection & Latency
        A --> C[Broker Health<br/>API Rate Limits]
        A --> D[CSV Health<br/>File Write Status]
        A --> E[System Health<br/>CPU, Memory, Disk]
    end

    subgraph "Monitoring Outputs"
        F[(InfluxDB<br/>Health Metrics)]
        G[Console Logs<br/>Structured Output]
        H[Alert System<br/>Critical Issues]
    end

    B --> F
    C --> F
    D --> F
    E --> F
    A --> G
    F --> H
```

Health Metrics Specification

System Status Measurements

Measurement	Tags	Fields	Purpose
`monitor_status`	env, host, app, status	loop_duration_ms, legs_count, error_count	Overall system health
`pipeline_tick`	env, host, app	atm_collect_ms, overview_collect_ms, records_written	Performance tracking
`broker_health`	env, host, api	quote_latency_ms, http_429_count, error_rate_percent	API monitoring
`influx_write_stats`	env, host, measurement	points_written, points_rejected, flush_latency_ms	DB performance

Health Check Logic

```
class HealthScheduler:
    def startup_probe(self):
        """Initial system readiness check"""
        if self.influx_enabled:
            check_influx(self.influx_url, self.writer)

    def tick(self, broker_state, csv_stats):
        """Periodic health assessment"""
        now = time.time()
        if now - self._last_tick < self.interval:
            return

        # Check all subsystems
        if self.broker_enabled:
            check_broker(broker_state, self.writer)
        if self.csv_enabled:
            check_csv(csv_stats, self.writer)
        if self.disk_enabled:
            check_disk(self.disk_path, self.writer)

        self._last_tick = now
```

Configuration Management

Environment Configuration

```
# Core System Settings
ENV=production
TEST_MODE=0
LOGGER_LOOP_INTERVAL_SEC=30
LOGGER_OFFSETS="-2,-1,0,1,2"

# Market Session Parameters
MARKET_TIMEZONE=Asia/Kolkata
SESSION_START_HHMM=09:15
SESSION_END_HHMM=15:30

# Data Storage Paths
CSV_SPLIT_ROOT=data/csv_data
MIRROR_SPLIT_ROOT=data/csv_data
RAW_SNAPSHOTS_ROOT=data/raw_snapshots

# InfluxDB Configuration
INFLUXDB_URL=http://localhost:8086
INFLUXDB_ORG=your-org
INFLUXDB_BUCKET=your-bucket
INFLUXDB_TOKEN=***

# Broker API Configuration
KITE_API_KEY=***
```

```
KITE_API_SECRET=***

# Advanced Analytics Settings
ADV_ENABLE_STREAMING=true
ADV_LOG_SUMMARY_ONLY=true
ADV_LOG_LAST_TOTALS=false
ADV_ENABLE_INFLUX_WRITES=true

# Health Monitoring
HEALTH_ENABLED=true
HEALTH_INTERVAL_SEC=30
HEALTH_INFLUX_ENABLED=true
HEALTH_BROKER_ENABLED=true
HEALTH_CSV_ENABLED=true
HEALTH_DISK_ENABLED=true
HEALTH_SYSTEM_ENABLED=true
```

Configuration Hierarchy

```
graph TD
    A[.env File<br/>Environment Variables] --> B[adv_config.py<br/>Advanced Settings]
    A --> C[logger_runner.py<br/>Core Settings]
    A --> D[health_scheduler<br/>Monitoring Config]

    B --> E[Advanced Analytics<br/>Processing Logic]
    C --> F[Data Collection<br/>Main Loop]
    D --> G[Health System<br/>Monitoring Logic]
```

Redundancy Analysis

Functional Redundancies

Data Format Redundancies ⚠

```
flowchart LR
    A[Raw Market Data] --> B[JSON Snapshots<br/>Audit Trail]
    A --> C[CSV Sidecars<br/>Tabular Access]
    A --> D[Daily Split CSVs<br/>Authoritative]

    B -.-> E[Development<br/>Debugging]
    C -.-> F[Ad-hoc Analysis<br/>Quick Access]
    D --> G[Advanced Analytics<br/>Production Use]

    style D fill:#c8e6c9
    style B fill:#fff3e0
    style C fill:#fff3e0
```

Analysis:

- ✅ **Intentional Redundancy**: JSON for audit, CSV sidecars for immediate access, splits for analytics
- ⚠️ **Storage Overhead**: ~3x storage requirement for same data
- 🔧 **Recommendation**: Maintain all three during development, consider archiving JSON/sidecars in production

Processing Redundancies ⚠️

Component	Primary Function	Redundant With	Action
CSV Sidecar Writer	JSON→CSV conversion	Daily Split Writer	Keep for immediate access
Streaming Aggregator	Real-time updates	EOD Reconciliation	Keep both (different purposes)
Pair Offset Calculation	In split writer	In advanced aggregator	Prefer split writer, fallback in aggregator

Data Flow Redundancies

```
flowchart TD
    subgraph "Potential Inefficiencies"
        A[Multiple CSV Scans<br/>Per minute for all splits]
        B[Duplicate Pair Calculations<br/>Split writer + Aggregator]
        C[Repeated File I/O<br/>Each streaming tick]
    end

    subgraph "Optimization Opportunities"
        D[Incremental Reading<br/>Track last processed minute]
        E[Single Pair Source<br/>Prefer split, fallback aggregate]
        F[Batched Updates<br/>Buffer multiple minutes]
    end

    A --> D
    B --> E
    C --> F
```

Variable Redundancies

Timestamp Column Variations

- `ts`, `ts_ist`, `timestamp` - **Fix**: Standardize on `ts` across all writers
- `last_updated` in multiple formats - **Fix**: Always UTC ISO8601

Index Naming Inconsistencies

- Input: "NIFTY", Output: "NIFTY 50" - **Status**: Acceptable (input/output boundary)
- File paths vs display names - **Status**: Handled by mapping functions

Efficiency Assessment

Current Efficiency Score: 7.2/10

```
radar
    title System Efficiency Radar
    x-axis 0 --> 10
    "Data Collection" : [9]
    "Processing Speed" : [6]
    "Storage Efficiency" : [5]
    "Memory Usage" : [7]
    "Error Handling" : [8]
    "Maintainability" : [9]
    "Scalability" : [6]
```

Performance Analysis

Strengths ✅

- **Robust Error Handling**: Comprehensive retry logic and graceful degradation
- **Clean Architecture**: Well-separated concerns with clear data contracts
- **Comprehensive Monitoring**: Extensive health metrics and alerting
- **Test Coverage**: Good unit and integration test foundation

Performance Bottlenecks ⚠️

1. **File I/O Overhead**
   - **Issue**: Full CSV scan every minute for streaming updates
   - **Impact**: O(n) read cost per minute where n = total rows
   - **Solution**: Implement incremental reading with minute cursors
2. **Windows File Lock Contention**
   - **Issue**: Multiple processes accessing same CSV files
   - **Impact**: PermissionError and retry overhead
   - **Solution**: Reader/writer coordination or message queues
3. **Triple Write Pattern**
   - **Issue**: JSON + CSV sidecar + daily split for same data
   - **Impact**: 3x write operations and storage overhead

- **Solution**: Async writing or consolidate to 2 formats

Scalability Concerns

```
graph TD
    subgraph "Current Limitations"
        A[Single-threaded Collection<br/>Sequential index processing]
        B[File-based Coordination<br/>No distributed capability]
        C[Full-scan Analytics<br/>No incremental processing]
    end

    subgraph "Scalability Solutions"
        D[Parallel Collection<br/>Per-index workers]
        E[Message Queue Coordination<br/>Redis/RabbitMQ]
        F[Incremental Processing<br/>Change data capture]
    end

    A --> D
    B --> E
    C --> F
```

Failure Points & Mitigations

Predicted Failure Scenarios

1. Authentication Token Expiry HIGH RISK

```
sequenceDiagram
    participant App as Application
    participant Kite as Kite API
    participant User as User/Browser

    App->>Kite: API Request with Token
    Kite->>App: 403 Token Expired
    App->>User: Launch Browser Auth
    Note over User: Manual intervention required
    User->>Kite: Complete OAuth Flow
    Kite->>App: New Token
    App->>Kite: Resume API Calls
```

Mitigation Strategy:

```
def enhanced_token_management():
    """Improved token handling with automatic refresh"""
    if token_expires_in_minutes() < 30:
        try:
            refresh_token_silently()
        except TokenRefreshError:
```

```
            schedule_maintenance_window()
            alert_operators("Manual token refresh required")
```

2. InfluxDB Connection Loss HIGH RISK

Failure Modes:

- Network connectivity issues
- InfluxDB server restart
- Authentication token expiry
- Bucket quota exceeded

Current Mitigation:

```
def resilient_influx_write(data, max_retries=3):
    """Write with exponential backoff and local buffering"""
    for attempt in range(max_retries):
        try:
            influx_writer.write(data)
            break
        except (ConnectionError, TimeoutError) as e:
            if attempt == max_retries - 1:
                buffer_locally(data)  # Fallback to local storage
            time.sleep(2 ** attempt)  # Exponential backoff
```

3. Market Data Feed Issues ⚠ MEDIUM RISK

Symptoms:

- Missing option legs (incomplete chains)
- Stale prices (delayed quotes)
- IV calculation failures

Detection & Recovery:

```
def validate_option_chain_completeness(legs):
    """Ensure all expected strikes and sides are present"""
    expected_legs = len(OFFSETS) * 2 * len(EXPIRY_BUCKETS)
    if len(legs) < expected_legs * 0.8:  # 80% threshold
        alert_operators("Incomplete option chain detected")
        return False
    return True
```

4. Disk Space Exhaustion ⚠ MEDIUM RISK

Monitoring:

```
def disk_space_monitoring():
    """Proactive disk space management"""
    free_gb = psutil.disk_usage('/').free / (1024**3)
    if free_gb < 2.0:  # Critical threshold
        cleanup_old_snapshots(days=7)
        compress_historical_data()
        alert_operators("Low disk space - cleanup initiated")
```

Recovery Procedures

Automated Recovery

- **Token Refresh**: Automatic browser-based OAuth when possible
- **Connection Retry**: Exponential backoff with local buffering
- **Data Validation**: Skip incomplete data points, continue collection
- **Health Monitoring**: Self-healing restart on critical failures

Manual Recovery

- **Database Reset**: Measurement deletion and schema recreation
- **Authentication Reset**: Manual token generation via broker console
- **Data Backfill**: Historical data replay from JSON snapshots or CSV archives

Development Roadmap

Initial Goals ✅ ACHIEVED

- [x] **Reliable Data Collection**: Minute-level option chain capture
- [x] **Multi-Format Storage**: JSON audit trail + CSV analysis + InfluxDB time series
- [x] **Health Monitoring**: Comprehensive system status tracking
- [x] **Advanced Analytics**: Weekday master profiles and paired offsets
- [x] **Dashboard Integration**: Grafana visualization with live updates
- [x] **Testing Framework**: Unit and integration test coverage

Current Achievements ✅

```
gantt
    title Development Progress Timeline
    dateFormat  X
    axisFormat %d

    section Data Collection
    Basic Collection    :done, 1, 5
    Health Monitoring   :done, 3, 8
    Error Recovery      :done, 6, 10

    section Advanced Analytics
    Daily Splits        :done, 8, 12
    Weekday Masters     :done, 10, 14
    Pair Offsets        :done, 12, 16

    section Testing & Reliability
    Unit Tests          :done, 14, 18
    Integration Tests   :done, 16, 20
    Mock Framework      :done, 18, 22
```

Short-term Roadmap (Next 3 Months)

Phase 1: Performance Optimization

- [ ] **Incremental Data Processing**: Cursor-based CSV reading
- [ ] **Parallel Collection**: Multi-threaded index processing
- [ ] **Memory Optimization**: Streaming data processing with fixed memory footprint
- [ ] **I/O Optimization**: Batched writes and read-ahead caching

Phase 2: Advanced Analytics Expansion

- [ ] **IV Analytics**: Term structure, skew, rank calculations
- [ ] **Greeks Computation**: Delta, gamma, theta, vega from market data
- [ ] **Dealer Positioning**: Gamma exposure and charm analysis
- [ ] **Technical Levels**: Support/resistance from option OI clustering

Phase 3: Infrastructure Improvements

- [ ] **Containerization**: Docker-based deployment with orchestration
- [ ] **Message Queue Integration**: Decoupled data flow with Redis/RabbitMQ
- [ ] **Distributed Processing**: Horizontal scaling capability
- [ ] **Enhanced Monitoring**: ML-based anomaly detection

Long-term Vision (6-12 Months)

Advanced Features

```
roadmap
    title Long-term Feature Roadmap
    x-axis 2025-Q1 --> 2025-Q4

    section Analytics Engine
        Advanced Greeks      :2025-Q1, 2025-Q2
        ML Predictions       :2025-Q2, 2025-Q3
        Risk Management      :2025-Q3, 2025-Q4

    section Infrastructure
        Cloud Deployment     :2025-Q1, 2025-Q2
        Auto-scaling         :2025-Q2, 2025-Q3
        Multi-region         :2025-Q3, 2025-Q4

    section Integration
        Multiple Brokers     :2025-Q1, 2025-Q3
        External Data        :2025-Q2, 2025-Q4
        API Services         :2025-Q3, 2025-Q4
```

- **Machine Learning Integration**: Predictive models for IV and price movements
- **Multi-Broker Support**: Integration with additional data providers
- **Real-time Alerting**: Advanced notification system for trading opportunities
- **API Services**: RESTful API for external data access
- **Cloud Deployment**: AWS/GCP deployment with auto-scaling

Efficiency Improvement Recommendations

Immediate Optimizations (High Impact, Low Effort)

1. Incremental CSV Processing

```
def incremental_csv_reader(file_path, last_position=0):
    """Read only new rows since last position"""
    with open(file_path, 'r') as f:
        f.seek(last_position)
        new_rows = f.readlines()
        new_position = f.tell()
    return new_rows, new_position
```

2. Timestamp Standardization

- **Action**: Standardize on `ts` column across all CSV writers
- **Impact**: Eliminates column detection logic overhead
- **Effort**: Low (config change + migration script)

3. Batched InfluxDB Writes

```
class BatchedInfluxWriter:
    def __init__(self, batch_size=100, flush_interval=10):
        self.batch_size = batch_size
        self.flush_interval = flush_interval
        self.buffer = []
        self.last_flush = time.time()

    def write_point(self, point):
        self.buffer.append(point)
        if len(self.buffer) >= self.batch_size or \
           time.time() - self.last_flush > self.flush_interval:
            self.flush()
```

Medium-term Optimizations (High Impact, Medium Effort)

1. Parallel Index Processing

```
from concurrent.futures import ThreadPoolExecutor

def parallel_collection():
    """Collect data for all indices in parallel"""
    with ThreadPoolExecutor(max_workers=3) as executor:
        futures = []
        for index in ['NIFTY', 'BANKNIFTY', 'SENSEX']:
            future = executor.submit(collect_index_data, index)
            futures.append(future)

        results = [future.result() for future in futures]
    return combine_results(results)
```

2. Memory-Mapped File Access

```
import mmap

def memory_mapped_csv_reader(file_path):
    """Use memory mapping for large CSV files"""
    with open(file_path, 'r') as f:
        with mmap.mmap(f.fileno(), 0, access=mmap.ACCESS_READ) as mmapped_file:
            return process_mmapped_data(mmapped_file)
```

3. Caching Layer Implementation

```
from functools import lru_cache
import redis

class DataCache:
    def __init__(self):
        self.redis_client = redis.Redis()

    @lru_cache(maxsize=1000)
    def get_instrument_token(self, symbol, expiry, strike, option_type):
        """Cache frequently accessed instrument mappings"""
        cache_key = f"instrument:{symbol}:{expiry}:{strike}:{option_type}"
        cached = self.redis_client.get(cache_key)
        if cached:
            return json.loads(cached)

        # Fetch from broker API
        token = self.broker_api.get_instrument_token(...)
        self.redis_client.setex(cache_key, 3600, json.dumps(token))
        return token
```

Long-term Architectural Improvements

1. Event-Driven Architecture

```
flowchart LR
    subgraph "Event-Driven Flow"
        A[Data Collector] --> B[Message Queue<br/>Redis/RabbitMQ]
        B --> C[Processing Workers<br/>Parallel Consumers]
        C --> D[Event Store<br/>InfluxDB + Files]
        D --> E[Analytics Engine<br/>Stream Processing]
    end
```

2. Microservices Decomposition

- **Collection Service**: Pure data ingestion
- **Processing Service**: Data transformation and merging
- **Analytics Service**: Advanced computations
- **API Gateway**: External access and authentication
- **Monitoring Service**: Health and alerting

3. Cloud-Native Deployment

```
# Kubernetes deployment example
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: option-collector
spec:
  replicas: 3
  selector:
    matchLabels:
      app: option-collector
  template:
    spec:
      containers:
      - name: collector
        image: option-pipeline:latest
        resources:
          requests:
            memory: "512Mi"
            cpu: "500m"
          limits:
            memory: "1Gi"
            cpu: "1000m"
```

Repository Restructuring Recommendations

Current Structure Issues

- **Mixed Concerns**: Scripts and application logic intermingled
- **Flat Module Structure**: Limited hierarchical organization
- **Configuration Spread**: Settings scattered across multiple files

Proposed Improved Structure

```
OP/
├── services/                          # Microservice-ready components
│   ├── collection/                    # Data collection service
│   │   ├── collectors/
│   │   ├── brokers/
│   │   └── health/
│   │
│   ├── processing/                    # Data processing service
│   │   ├── mergers/
│   │   ├── writers/
│   │   └── validators/
│   │
│   ├── analytics/                     # Analytics service
│   │   ├── aggregators/
│   │   ├── computers/
│   │   └── models/
│   │
│   └── api/                           # API service
│       ├── endpoints/
│       ├── middleware/
│       └── schemas/
│
```

```
├── shared/                          # Shared utilities
│   ├── config/
│   ├── utils/
│   ├── constants/
│   └── types/
│
├── infrastructure/                  # Infrastructure as code
│   ├── docker/
│   ├── kubernetes/
│   ├── terraform/
│   └── monitoring/
│
├── data/                           # Data storage (unchanged)
├── tests/                          # Comprehensive test suite
├── docs/                           # Documentation
└── scripts/                        # Deployment and utility scripts
```

Migration Strategy

Phase 1: Service Extraction

1. Extract collection logic into `services/collection/`
2. Move processing logic to `services/processing/`
3. Isolate analytics in `services/analytics/`

Phase 2: Shared Library Creation

1. Consolidate utilities in `shared/`
2. Centralize configuration management
3. Standardize error handling and logging

Phase 3: Infrastructure Modernization

1. Containerize each service
2. Add Kubernetes deployment manifests
3. Implement service mesh for communication

Quick Start Guide

Prerequisites

```
# System Requirements
Python 3.10+
Docker Desktop
Git
```

```
# Install Dependencies
pip install -r requirements.txt
```

Local Development Setup

1. Clone and Configure

```
git clone https://github.com/ayushrajani07/OP.git
cd OP

# Create environment configuration
cp config/.env.template .env
# Edit .env with your settings
```

2. Start Infrastructure

```
# Start InfluxDB and Grafana
docker-compose up -d

# Verify services
docker-compose ps
```

3. Initialize InfluxDB

```
# Test connectivity
python scripts/influx_smoke.py

# Setup initial measurements (if needed)
python scripts/setup_influxdb.py
```

4. Run Collection Pipeline

```
# Test mode (no real API calls)
export TEST_MODE=1
python scripts/logger_runner.py

# Production mode (requires valid Kite credentials)
unset TEST_MODE
python scripts/logger_runner.py
```

5. Access Dashboards

- **InfluxDB UI**: http://localhost:8086
- **Grafana**: http://localhost:3000 (admin/admin)

Testing

```
# Run all tests
python -m unittest discover tests -v

# Run with coverage
pip install coverage
coverage run -m unittest discover tests
coverage html
```

Health Monitoring

```
# Check system health
python scripts/health_monitor.py

# View live logs
tail -f logs/application.log
```

Conclusion

The OP (Options Pipeline) system represents a mature, production-ready analytics platform that has evolved through multiple development phases to address real-world challenges in options data processing. With its robust architecture, comprehensive monitoring, and extensive testing framework, the system provides a solid foundation for advanced market analytics.

Key Strengths

- **Battle-Tested Reliability**: Proven error handling and recovery mechanisms
- **Comprehensive Coverage**: End-to-end data flow from collection to visualization
- **Monitoring Excellence**: Extensive health metrics and alerting capabilities
- **Testing Maturity**: Solid unit and integration test foundation
- **Documentation Quality**: Detailed documentation enabling system reconstruction

Growth Opportunities

- **Performance Optimization**: Incremental processing and parallel execution
- **Architectural Evolution**: Microservices and cloud-native deployment
- **Advanced Analytics**: ML integration and predictive capabilities
- **Scalability Enhancement**: Distributed processing and horizontal scaling

The system's modular design and clear separation of concerns positions it well for future enhancements while maintaining operational stability. The comprehensive testing framework and monitoring capabilities ensure reliable production operation, making it suitable for critical financial data processing workflows.

Last Updated: August 2025
System Version: 2.4
Documentation Version: 1.0