

QIM Indian Indices Engine

A Complete Technical Translation from Excel Logic to a Deterministic Python System

QIM Framework – Structural-Behavioural Interpretation for Research Use

© 2026 QIM Framework. All rights reserved.

QIM Indian Indices Engine – Python Automation Architecture

A Complete Technical Translation from Excel Logic to a Deterministic Python System

Author: Ashwin — Founder, QIM Framework

Version: 3.0 (Python Automated Edition)

Release Date: September 2026

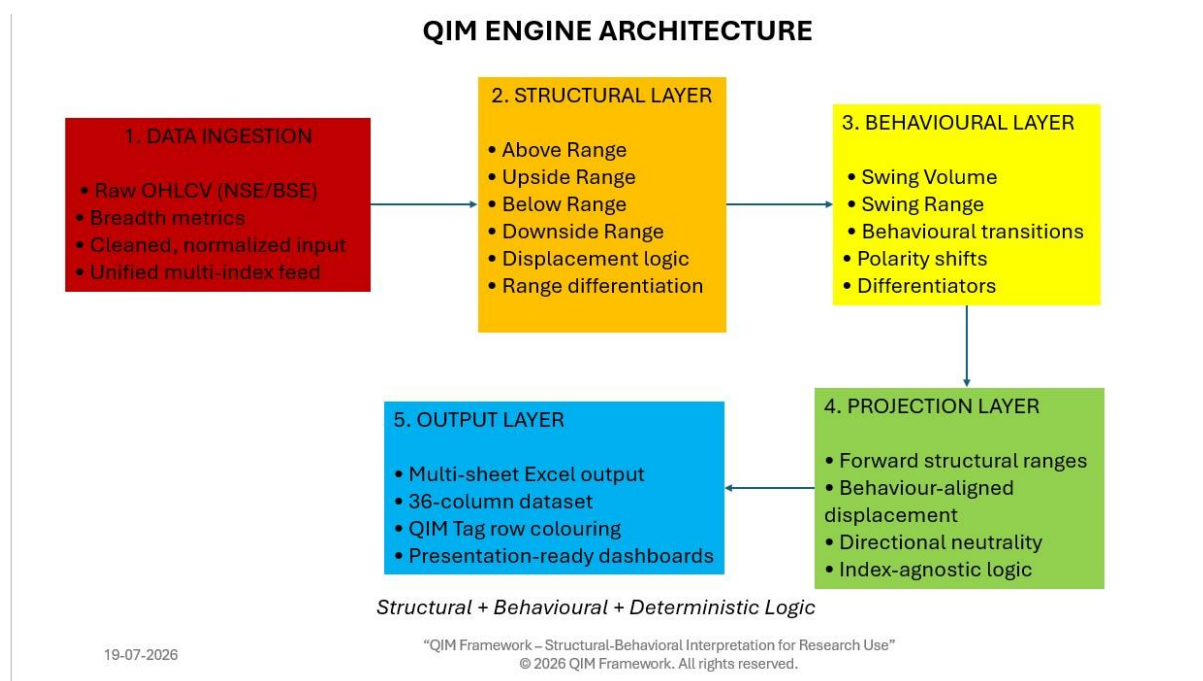
Website: qimframework.com

Executive Summary

The QIM Indian Indices Engine is a deterministic structural-behavioural analytical system designed to interpret Nifty 50, Nifty Bank, and Sensex 30 using mathematically derived ranges, behavioural metrics, displacement logic, and forward projections. Originally implemented through multi-sheet Excel calculations requiring approximately **30 minutes per index**, the engine has now been fully translated into Python, reducing computation time to **under 5 minutes for all three indices combined**.

This document presents the complete 8-Block Python architecture, the libraries used, engineering hurdles encountered, debugging strategies applied, and the final accomplishment of converting a complex Excel-based framework into a fully automated Python engine. The translation preserves the deterministic nature of QIM while enabling scalability, reproducibility, and performance improvements.

Architecture Diagram (Placeholder)



Python Libraries Utilized

The engine relies on a minimal, stable, and deterministic set of Python libraries:

- **pandas** — DataFrame manipulation, slicing, rolling windows
- **numpy** — mathematical operations, vectorized computation
- **openpyxl** — multi-sheet Excel export
- **datetime** — date handling and alignment

- **os** — file path management and output directory control

These libraries collectively replace the entire Excel calculation logic, enabling a clean automation pipeline.

Eight-Block Python Architecture

Block 1 — Foundation & Structural Base Preparation

Purpose: Prepare raw OHLCV data for deterministic structural computation.

Operations:

- Date normalization
- High/Low/Close alignment
- Volume standardization
- Removal of noise rows
- Base structural columns
- Range pre-computation

Outcome: A stable foundation DataFrame for all subsequent blocks.

Block 2 — QIM Tag Assignment & Polarity Logic

Purpose: Assign structural behaviour tags and directional polarity.

Operations:

- Behavioural displacement
- Polarity determination
- Structural transitions
- Tag consistency enforcement

Outcome: A deterministic structural map of index behaviour.

Block 3 — Swing Metrics & Emotional Strength Gating

Purpose: Compute behavioural metrics for each structural segment.

Operations:

- Swing Range
- Swing Volume
- Polarized Volume
- Emotional Strength (forward rolling sum)
- Pivot (forward rolling mean)

Engineering Hurdle: Volume contamination caused repeated `TypeError: int + str`. Resolved through a permanent Volume-cleaning pipeline:

- comma removal
- whitespace stripping
- numeric coercion
- NaN replacement
- segment-safe cleaning inside Block 3

Outcome: A complete behavioural profile of each structural segment.

Block 4 — Displacement & Time Metrics

Purpose: Compute structural displacement and time-based behaviour.

Operations:

- Upward displacement
- Downward displacement
- Net displacement
- Trading days
- Trading hours
- Required displacement
- Estimated velocity

Outcome: A deterministic displacement model without indicators or heuristics.

Block 5 — Structural Price Zones

Purpose: Compute the four QIM structural boundaries.

Operations:

- Above Range
- Upside Range
- Below Range
- Downside Range
- Boundary differentiation
- Zone mapping

Outcome: A structural map showing index position relative to mathematically derived boundaries.

Block 6 — Differentiators & Behavioural Filters

Purpose: Enhance interpretation through behavioural modifiers.

Operations:

- Strength filters
- Behavioural differentiators
- Emotional Strength gating integration
- Structural-behavioural alignment

Outcome: A refined behavioural layer that explains how the index is moving.

Block 7 — Formatting & Presentation Logic

Purpose: Prepare the dataset for multi-sheet Excel output.

Operations:

- Column ordering
- QIM Tag row colouring logic
- Sheet-ready formatting
- Presentation alignment

Outcome: A clean, presentation-ready dataset suitable for dashboards and reporting.

Block 8 — Multi-Index Execution & Output Generation

Purpose: Execute the engine for all benchmark indices and generate output.

Operations:

- Nifty 50 computation
- Nifty Bank computation
- Sensex 30 computation
- Multi-sheet Excel export
- Unified structural-behavioural output

Outcome: A complete analytical engine that fires in under 5 minutes.

Engineering Hurdles & Debugging Journey

1. Volume Contamination Errors

Repeated failures occurred due to mixed data types in the Volume column (int + str).

Resolution required:

- strict string conversion
- comma removal
- whitespace stripping
- numeric coercion
- NaN replacement
- cleaning inside Block 3 before segment slicing

This fix stabilized the behavioural layer permanently.

2. Structural Tag Misalignment

Early versions mis-tagged transitions between RALLY and DROP. Resolved by rewriting the tag-transition logic to enforce deterministic behaviour.

3. Segment Boundary Errors

Segment slicing (`df.iloc[i:end]`) exposed unclean rows. Resolved by integrating cleaning logic directly inside Block 3.

4. Excel Writer Conflicts

Openpyxl required strict sheet naming and controlled writer context. Resolved through a unified index-sheet mapping.

5. Emotional Strength Performance Bottlenecks

Large forward windows caused performance degradation. Resolved by optimizing rolling window computation.

Resilience & Engineering Discipline

The engine required multiple rewrites:

- Block 3 rebuilt three times
- Block 4 refined twice
- Block 2 polarity logic corrected once
- Ingestion pipeline rebuilt once

Each misfire strengthened the architecture. Each error clarified the logic. Each debug step made QIM more deterministic.

This is engineering resilience in practice.

Performance Benchmark — From 30 Minutes to 5 Minutes

Excel Workflow (Before)

- manual sorting
- manual filtering
- manual displacement calculation
- manual behavioural tagging
- manual projection logic
- manual formatting
- manual multi-sheet output

Total time: 30 minutes per index.

Python Workflow (After)

- ingestion
- computation
- tagging
- slicing
- projection
- formatting
- export

Total time: 5 minutes for all three indices combined.

This is not just automation. This is architectural transformation.

Future Extensions

- QIM Equity Engine
- QIM Sector Engine
- Global Indices Engine
- Real-time structural updates
- API-based ingestion
- QIM Dashboard (Web + Mobile)
- Behavioural anomaly detection
- Structural-behavioural forecasting modules

These extensions will expand QIM into a complete analytical ecosystem.

Founder's Note

“The QIM Engine represents years of structural-behavioural research translated into a deterministic Python system. This work is dedicated to analysts, researchers, and learners who seek clarity in market structure. The successful translation from Excel to Python demonstrates the robustness of the QIM Framework and establishes a scalable foundation for future analytical engines.”

Conclusion

The QIM Indian Indices Engine is now a fully automated, deterministic Python system capable of structural-behavioural interpretation across India's major benchmark indices. The successful translation from Excel logic to Python demonstrates the robustness of the QIM Framework and establishes a scalable foundation for future analytical engines.