A) Step-by-step pipeline (using the intake table)

1. **Group**

- Group rows by `Segment → Factor → Layer`.

2. **Normalize each row**

- Convert raw input to `Value ∈ [0,1]` using the benchmark method in the row.

3. **Compute row confidence**

- `confidence = source_reliability × recency_weight × coverage_weight`.
    - `recency_weight = exp(-Δdays / half_life_days_for_layer)`
    - `coverage_weight` example for volume: `log(1+n) / log(1+n_ref)` clipped to [0,1].

4. **Per-layer aggregation**

- For all rows of the same layer that are **not events**:
    - `layer_value = Σ(Value_i × confidence_i) / Σ(confidence_i)`
    - `layer_confidence = clip01( Σ(confidence_i) / (Σ(confidence_i) + prior_weight) )`
- Apply **Qual Insight** rows:
    - Adjust `layer_confidence` by the specified factor (e.g., −0.3).
    - Store any `Diff_Tag` (e.g., `variance_high`) as metadata on the layer.

5. **Apply Event Shock overlays**

- For each event mapped to this layer or its factor:
    - `impact = base_impact × credibility × salience × scope`
    - Decay: `impact_t = impact × exp(-Δdays / event_half_life)`
    - Adjust value: `layer_value = clip01(layer_value + sign × impact_t)`
    - Event rows also carry their own confidence; if you prefer, multiply the impact by event confidence.

6. **Aggregate layers → factor**

- POC: equal-weight average of layer values within the factor.
- Optionally carry `factor_confidence = mean(layer_confidence)`.

7. **Aggregate factors → segment raw**

- Weighted sum using your segment factor weights.

8. **Apply segment adjustments**

- Multiply by scenario/stability/learning constants and add bias.
- Normalize as specified (clip or logistic) → `Segment score ∈ [0,1]`.

9. **Total score**

- Weighted sum of segment scores → apply global adjustments → logistic → `Validatus_Score`.

10. **Patterns and scenarios**

- Evaluate **Segment patterns** on layers or factor mini-scores.
- Evaluate **Cross-segment patterns** on segment scores and attached tags.
- Seed scenarios from:
    - Strong pattern hits,

    - Large positive or negative Event impacts,
    - Presence of differentiation tags.

11. **Persist provenance**

- Store per row: source, timestamp, normalization method, confidence, and any event IDs used.

That is the full loop.

B) Which libraries are used where (name them)

1. **Benchmark Catalog**
   File: `benchmarks.yaml`
   Purpose: min–max ranges, z-score params, half-lives per layer for normalization.

2. **Source Catalog**
   File: `sources.yaml`
   Purpose: base reliability per source type (gov = 0.9, industry report = 0.8, client export = 0.7, social = 0.4).

3. **Impact Library (Event Ontology + Impacts)**
   Files:

- `events_ontology.yaml` (event_type IDs, descriptions, default half-life)
- `impact_library.yaml` (event_type → affected Segment/Factor/Layer, base_impact, sign)
  Purpose: convert real-world events into numeric shocks.
  Yes, this is a separate library. Keep it small for POC.

4. **Pattern Library**
   File: `patterns.yaml` (single file with namespaces)

- `scope: segment | cross_segment`

- `segment: Consumer | Market | Product | Brand | Experience | null`
  Purpose: rule logic that consumes layers or segment scores and emits matched patterns with recommended actions.

These four are enough for the POC and scale later.

C) Where correlations with patterns live

- **Segment patterns** operate on **layers or factor-level mini-scores**. They catch local motifs.
  Example: `Consumer.Engagement > 0.7 AND Consumer.Trust < 0.4 → FragileUsage`.

- **Cross-segment patterns** operate on **segment scores** and metadata tags. They create the strategic narrative.
  Example: `Consumer.Demand > 0.7 AND Product.Readiness < 0.5 → ExecutionGap`.

- **Events affect patterns indirectly**. Events change layer or factor values via the Impact Library, which can make a pattern start or stop matching. You can also have event-triggered patterns if needed, but not required for POC.

D) Minimal schemas (copy to repo)

benchmarks.yaml

```
- layer_id: Consumer.Perception.SocialSentiment
  norm: {type: "sentiment", range: [-1, 1]}
  volume_ref: 2000
  half_life_days: 30

- layer_id: Market.Competition.PriceBand
  norm: {type: "minmax", min: 5000, max: 12000}
  half_life_days: 90
```

sources.yaml

```yaml
- source: "gov_stat"
  reliability: 0.90
- source: "industry_report"
  reliability: 0.80
- source: "client_export"
  reliability: 0.70
- source: "social_media"
  reliability: 0.40
```

events_ontology.yaml

```yaml
- event_type: "tariff_announcement"
  description: "Government announces import tariff intent"
  default_half_life_days: 60
```

impact_library.yaml

```yaml
- event_type: "tariff_announcement"
  impacts:
    - target: "Market.Stability.Regulatory"
      base_impact: -0.10
      sign: -1
    - target: "Product.Resilience.SupplyChainStability"
      base_impact: -0.06
      sign: -1
```

patterns.yaml

```yaml
- id: fragile_usage_v1
  scope: segment
  segment: Consumer
  logic:
    all:
      - gt: { Consumer.Adoption.Engagement: 0.70 }
      - lt: { Consumer.Adoption.Trust: 0.40 }
```

```
  action: { name: "Onboarding trust push" }

- id: execution_gap_v1
  scope: cross_segment
  segments: [Consumer, Product]
  logic:
    all:
      - gt: { Consumer.Demand: 0.70 }
      - lt: { Product.MarketReadiness: 0.50 }
  action: { name: "Gate marketing, accelerate readiness" }
```

Inputs from Client

Ground rule

We **never overwrite client data**. We **blend** or **branch**, and we always **show our work**.

A) Ingestion policy (per input)

Each datum gets these flags:

- `source_type`: client_export | client_estimate | industry_benchmark | web_scrape | analyst_note
- `as_reported`: raw value + units
- `norm_method`: how we map to 0–1
- `confidence_base`: default by source (client_export high; estimate lower)
- `recency_weight`, `coverage_weight`
- `assertion_level`: hard_assert | soft_assert | none

hard_assert = “treat this as truth unless impossible.”
soft_assert = “prefer this unless strong contradictory evidence.”

B) Priority & blending modes

At the **Segment layer** (where normalization happens), choose one of three modes per factor (configurable):

1. **Client-First (Assert)**
    - If `assertion_level = hard_assert`, set posterior = normalized client value.
    - Still run basic **sanity checks** (out of bounds, unit mismatch). If it fails, flag and fall back to Blended.
2. **Blended (Default)**

Bayesian-style precision weighting:

```
posterior = (w_prior*μ_prior + Σ w_obs*x_obs) / (w_prior + Σ w_obs)
```

-
    - Put client input as an observation with **higher weight** than web/benchmark.
    - If client and benchmark diverge by > Δ* (e.g., 0.25), **branch** (see D).

3. **Benchmark-Only (Sparse)**
    - Use when client has no data; posterior = prior.

Weights (typical defaults; tweak in `sources.yaml`):

- client_export: 0.85
- client_estimate: 0.60
- industry_benchmark: 0.75
- web_scrape: 0.45
- analyst_note: 0.50

Multiply by recency and coverage as usual.

C) We do not “override”—we annotate adjustments

When the posterior differs from the client value by more than a tolerance (e.g., 0.1 on 0–1 scale):

- Record:
    - `client_norm`: normalized client value

- `posterior`: blended value
- `delta`: posterior − client_norm
- `why`: list of contributing evidence with weights

UI copy (plain):

> "We blended your input (weight 0.85) with benchmark X (0.75) and source Y (0.45). Because the sources disagreed (Δ=0.18), we show both 'As reported' and 'Blended' in the panel. You can pin 'As reported' for scenarios."

D) Branch instead of overwrite (scenario-safe)

If disagreement is large (Δ ≥ Δ*), create **two parallel values** for downstream use:

- `value_as_reported` (pinned client)
- `value_blended` (posterior)

The Strategy layer runs scenarios on **both branches** and shows the spread. Patterns fire on the active branch but can show "would-fire" on the other.

E) Sanity checks (don’t surprise the client)

Before any blending:

- **Bounds & units**: reject or convert (log in provenance).
- **Plausibility band**: if client value is outside industry hard-bounds, mark `needs_review: true` and notify.
- **Time alignment**: if client data is old and contradicts fresh benchmark, downweight via recency.

F) Transparency objects (persist and display)

For every factor used:

```
{
  "factor_id": "Market.PriceElasticity",
  "mode": "blended",
  "value_as_reported": 0.72,
  "value_blended": 0.61,
  "active_value": "blended",                // or "as_reported"
```

```
  "delta": -0.11,
  "inputs": [
{"src":"client_export","norm":0.72,"w":0.85},
{"src":"statista_2025","norm":0.58,"w":0.75},
    {"src":"reddit_sent","norm":0.49,"w":0.40}
  ],
  "explanation":"Client preferred, but recent market data pulled elasticity down.",
  "assertion_level":"soft_assert",
  "tolerance":0.10
}
```

G) Pattern & score behavior with branches

- **Segment scores**: compute for **both** if branched; set one as **active** (toggle in UI).
- **Patterns**: evaluate on the active branch; show “alt-branch” hits in a collapsed row.
- **Action layer**: same as patterns.
- **Dashboard**: badge any card influenced by branched inputs: “Assumption-sensitive.”

H) “Trump said tariffs” example

- Map to Impact Library event.
- **Do not overwrite** client BOM or stability outright.
- Apply **shock overlay** (decaying) that adjusts the **blended** value, and show the delta:
    - “Regulatory Stability: −0.06 shock applied (cred 0.9 × salience 0.8 × scope 0.8), half-life 60 days.”
- If client says “ignore this” → set event `assertion_level = none` in this scenario; branch remains available.

I) Developer checklist (concise)

1. Ingest row → normalize → compute base confidence.
2. Merge by factor using mode: **assert / blended / benchmark-only**.
3. If `|client_norm - posterior| ≥ Δ*` → **branch** and mark.
4. Apply **event shocks** to each branch (with decay).
5. Compute segment and action scores for active branch; compute alt-branch in background.
6. Patterns on active branch; list alt-branch differences.
7. Persist full provenance and branch metadata.

J) When to prioritize client over benchmark

- `assertion_level = hard_assert` **or** regulatory/contractual obligation to use client numbers in reporting.
- Early POC with limited external data and the client is the primary truth source.

- But still run **sanity/bounds** and show a gentle warning if it’s far off.