

AI Analytics Trust Sprint: Executive Readiness Report (SAMPLE)

SAMPLE, FICTIONAL DATA. This shows the format of the sprint's report, filled from the public demo project [dbt-revenue-variance-demo](#) (commit 243d6f0 , CI green). No client, no client data. It is not a Cortex Analyst audit and not a Snowflake run: the demo has neither. Section 2 says what a real engagement adds.

Client: none (fictional demo project)
Domain shown: monthly revenue and budget variance (dbt models and semantic layer, DuckDB)
Prepared: 2026-10-07 (sample)
Prepared by: Simon Sangla, independent analytics and AI engineer
Verdict: CONDITIONAL GO (format example only; no real system was assessed). Each condition traces to a finding in section 4.

1. Executive summary

	RESULT
Checks run	6 re-runnable proofs, each with its claim, test and result in the repo; CI re-runs them on every push
Reference figure	revenue 53,546.30 EUR (clean build), recomputed from the raw orders
Faults injected into the data	3. The original build caught 1 of 3; with two tests added for them it catches 3 of 3
Changes that left a plain dbt build green	4 distinct: two of the data faults, one metric aggregation change, one dashboard-query filter change
Data residency	Not a Snowflake run. This sample ran on DuckDB with fictional seeds.

2. What a real engagement adds (not shown in this sample)

This sample covers the metric-trust and validation part of the work. The sprint itself is a fixed-scope, 5-day evaluation of **your** Snowflake Cortex or text-to-SQL setup, and the report adds:

- **Per-question results.** A set of your business questions is run through your Cortex or text-to-SQL setup and each answer is compared with a ground-truth query: pass, fail or warning, with the reason.
- **A governed semantic view** for the audited domain, built in a sandbox or clone.
- **A validation harness** you can re-run, so the check outlives the sprint.
- **A go/no-go recommendation** with the blockers to fix first, and a debrief with your team.
- **Data residency, stated step by step.** Read-only on your production data; where each step ran is written in the report, and any step that is not in-account is said so.

3. Metric trust map

METRIC	GOLDEN DEFINITION	GRAIN	MANDATORY FILTER	OWNER (ILLUSTRATIVE ROLE)	TRUST STATUS
Revenue	sum of quantity x unit price over completed orders	month x product	status = completed (refunds excluded)	Finance lead	VERIFIED at total grain: the semantic-layer metric equals the raw-orders recomputation, 53,546.30 EUR. A change to the metric definition would still pass the build until condition 2 is in place (finding 2)
Variance bridge (volume, price)	budget + volume effect + price effect = actual revenue	month x product	same as Revenue	Finance lead	VERIFIED: tied to revenue recomputed from the orders, so a changed filter fails the build (counting refunds turns it red, finding 4)
Budget coverage	every budget month has completed orders	month	none	Data engineering	VERIFIED after a test was added (finding 1)
Dashboard revenue	sum of actual revenue from the monthly model	total	must match the Revenue metric	BI lead	WARNING: a second definition of the same number, guarded only by a parity check (finding 3)

Grain: the level one row stands for. Known-answer test: recomputes a figure independently from the raw data and compares it with the model's.

4. Findings

Findings below come from **deliberate fault injection on a public demo**, not from discovery on a real system.

Finding 1 (high): a fault class a green pipeline can miss. Three faults were injected into the orders seed of a copy of the project. Clean revenue is 53,546.30 EUR.

INJECTED FAULT	REVENUE	ORIGINAL BUILD	AFTER TWO TESTS ADDED FOR THEM
Five completed order lines loaded twice	53,776.60 (+230.30)	caught (unique_stg_orders_order_id)	caught
Refunded lines relabelled completed	61,606.60 (+8,060.30, +15%)	not caught , PASS=27, ERROR=0	caught (assert_refunds_present_each_month)
March 2026 orders missing	35,950.20 (-17,596.10, -33%)	not caught , PASS=27, ERROR=0	caught (assert_orders_cover_budget_months)

The known-answer tests did not help: they recompute revenue from the same seed, so when the seed is wrong both sides agree. The missing month showed up only as a 100% budget miss, which reads like a business result, not a data fault. Tests added: two. Clean build PASS=29, ERROR=0; the three mutated builds fail with ERROR=1, 1 and 2.

Finding 2 (high): a change inside the semantic layer is invisible to the build. One word changed in the metric definition (agg: sum to agg: max): the metric returns 9,550.90 EUR against 53,546.30 EUR from the raw orders. dbt build stays green because no table test reads the semantic layer. Only a known-answer check, queried through the semantic layer itself, catches it.

Finding 3 (medium): two definitions of one number. A dashboard-style query and the revenue metric agree at 53,546.30 EUR. If the query drops the status filter it returns 61,606.60 EUR, and dbt build stays green (PASS=27, ERROR=0). A check that runs the query and the metric side by side catches it.

Finding 4 (resolved): a definition change must be visible. Counting refunded orders as revenue moves revenue to 61,606.60 EUR (+8,060.30, +15%) and turns the build red on two known-answer tests (ERROR=2). The change cannot reach the variance bridge silently: someone has to approve it.

5. Production go / no-go gate

Conditions before general rollout

1. Keep the two added data tests (budget-month coverage, refunds present each month) in the build. Finding 1.
2. Run the metric known-answer check through the semantic layer in CI. Finding 2.
3. Run the dashboard-versus-metric parity check in CI, or retire the second definition. Finding 3.

Sources: `proofs/2026-10-07-*/CLAIM.md` in the demo repository, commit `243d6f0`. Fictional data, seed 42.