PRD: AI Analyst Builder

Author: Sankar Kumar Palaniappan (sankaar@icloud.com) **Date:** June 10, 2026
Status: Draft — Stage 4: Solution Review **GitHub:** github.com/sankarkumarpalaniappan/AI_Analyst_Builder

---

Hypothesis

We believe that automating the end-to-end analytics pipeline — from business question to validated slide deck — will reduce analysis cycle time from 5 days to under 2 hours for IC product analysts at data-light companies, measured by time-to-deck across the first 10 live analyses.

Problem

Who has this problem

Primary persona: IC product analyst at a Series B–D SaaS company (100–500 employees). No dedicated data engineering team. Expected to answer 3–5 business questions per week from PMs and leadership. Uses BigQuery or Postgres, writes SQL daily, has strong analytical judgment — but spends the majority of time on mechanical work: finding tables, writing boilerplate SQL, building charts, formatting decks.

Secondary persona: Product manager who can read SQL but not write it. Relies on a 1–2 person analytics team with a 2–3 day SLA on data requests. By the time the answer arrives, the conversation has moved on.

How bad is it

A standard analysis cycle looks like this:

Day	Work
Monday	Business question surfaces in a meeting
Tue–Wed	

Day	Work
	Data exploration: find the right tables, understand schema, write exploratory queries
Thursday	Build charts, validate numbers, spot-check aggregations
Friday	Write the narrative, build the deck
Following Monday	Present findings

By the time the insight lands, the business context has often shifted. The decision was already made on gut feel. The analyst spent five days on work that arrived too late to use.

Industry context: Studies on analyst workflows consistently show that 60–80% of analyst time is spent on data preparation and formatting — not on the reasoning and interpretation that requires human judgment. The bottleneck is mechanical, not intellectual. [NEED: cite specific survey before publish — Fishtown Analytics State of Analytics or similar]

Current workarounds

- Reusing last quarter's deck with updated numbers — misses new dimensions and breaks when schema changes
- Answering in Slack with raw, unvalidated numbers — errors reach decision-makers
- Delegating to a BI dashboard — shows *what* happened, never *why*
- Skipping validation entirely under time pressure — SQL bugs in stakeholder decks are common and costly

The workarounds prove the pain is real. When analysts skip validation under time pressure, wrong numbers reach executives.

If we don't solve it

Teams make decisions on stale or unvalidated data. Analysts burn out on mechanical work — writing the same JOIN for the fifth time, rebuilding the same funnel chart — leaving no time for the judgment and interpretation that actually requires a human. The insight-to-action loop stays broken.

Strategic Fit

Why agentic AI, not a rule-based pipeline

A scripted pipeline can't solve this. Every business question requires a different analytical approach — a funnel question needs different SQL, different chart types, and a different narrative than a churn question or a cohort analysis. A static pipeline doesn't know which agents to invoke, which dimensions to drill into when an anomaly surfaces, or when to halt because data quality is insufficient.

The critical capabilities that require a reasoning model: - **Adaptive framing:** decomposing an ambiguous natural language question into a specific analytical plan - **Iterative root cause investigation:** drilling through dimensions until reaching a specific, actionable cause — not just a pre-defined set of cuts - **Quality judgment:** deciding whether a finding is strong enough to present (A-F confidence grade) or whether to halt and surface the issue - **Self-correction:** reading user feedback in natural language, extracting the correction, and applying it to future SQL — impossible with regex or keyword matching

The unstructured inputs that require LLM reasoning: the business question itself (free-form natural language), user corrections ("that column is revenue, not GMV"), schema documentation written in prose, and business glossary entries that vary per organization.

Why now

Claude Code with Opus 4 is the first model capable of reliably orchestrating 18-agent analytical workflows with the reasoning quality needed to catch its own errors. Multi-agent orchestration at this quality level wasn't viable 18 months ago.

Why this, not ChatGPT / Claude Code alone / existing tools

Tool	What it does	What's missing
ChatGPT / Claude (chat)	Generates SQL and charts on request	No memory across sessions. No validation. No business context. Restarts from zero every time.
Claude Code alone	Writes the SQL you describe	A coding assistant, not an analyst. You drive every step manually. No framing, no validation pipeline, no deck output.

Tool	What it does	What's missing
BI tools (Looker, Tableau)	Persistent dashboards	Answer "what" not "why." No narrative, no root cause, no recommendation.
AI SQL tools (Julius AI, Noteable)	Natural language to SQL	No validation layer, no framing, no storyboard, no deck. Closed SaaS — no customization.
Jupyter + Copilot	AI-assisted notebook	Still requires the analyst to drive every step. No orchestration. No deck output.

AI Analyst Builder goes further than any of these: question → validated root cause → branded slide deck, automated and resumable, with a self-learning memory that gets smarter per correction.

MOAT: The corrections system, query archaeology, and business glossary create compounding value that SaaS tools can't replicate. Every correction logged makes future analyses more accurate. Every proven SQL pattern retrieved saves a query write. After 50 analyses, the system knows your schema, your metric definitions, your common errors, and your business context. A generic SaaS tool starts from scratch every session.

User Stories

- As an IC analyst, I want to run a full analysis from a business question to a validated deck without writing SQL manually, so I can answer stakeholder questions the same day they're asked instead of five days later.
- As a PM without dedicated analyst support, I want to ask a data question in plain English and receive a validated finding with supporting charts, so I can make decisions without waiting days for analyst capacity.
- As an analyst who made a SQL mistake last week, I want the system to remember the correction and apply it proactively next time, so the same error never reaches a stakeholder twice.
- As an analyst sharing findings with a VP, I want a confidence grade (A–F) on every analysis, so I know before presenting whether the numbers are solid enough to act on.

Solution

What We're Building

A Claude Code-powered toolkit that orchestrates 18 specialized agents across a 4-phase pipeline, turning a natural language business question into a validated analysis and branded slide deck. Everything runs from the terminal via slash commands. Every agent, skill, and rule is a readable, editable markdown file — nothing hidden, nothing locked.

Primary Flow: End-to-End Pipeline

```
User types: /run-pipeline question="Why is checkout conversion
dropping?"

Phase 1 — Frame (parallel)
  ├─ Question Framing: structures the question, defines hypotheses
  └─ Data Explorer: profiles the active dataset, flags quality issues

Phase 2 — Analyze (sequential)
  ├─ Source Tie-Out: dual-path verification (pandas vs DuckDB) —
HALTS on mismatch
  ├─ Descriptive Analytics: segmentation, funnel, drivers analysis
  ├─ Root Cause Investigator: drills dimensions iteratively to
specific cause
  ├─ Validation: 4-layer check (structural, logical, business rules,
Simpson's Paradox)
  └─ Opportunity Sizer: business impact with sensitivity analysis

Phase 3 — Story (sequential)
  ├─ Story Architect: Context-Tension-Resolution storyboard
  ├─ Coherence Reviewer: checks story flow before any chart is made
  ├─ Chart Maker: SWD-styled charts with collision detection
  └─ Design Critic: 16-point quality review per chart

Phase 4 — Deck (sequential)
  ├─ Storytelling: stakeholder-ready narrative
  ├─ Deck Creator: branded Marp slide deck with HTML components
  ├─ Slide Reviewer: lint gate (frontmatter, component variety,
pacing)
  └─ Comms Drafter: Slack summary + email brief

Output: analysis report, charts, PDF deck, HTML deck, speaker notes
```

Secondary Flows

User intent	Command	What runs
Quick data question	Ask in plain English	Direct query + chart (under 2 min)
Explore a new dataset	`/explore`	Interactive profiling, no analysis committed
Single chart	"Make a funnel chart of checkout"	Chart Maker + Design Critic only
Resume interrupted run	`/resume-pipeline`	Reads `working/pipeline_state.json`, resumes at last step
Redo deck only	`/run-pipeline plan=refresh_deck`	Phases 3–4, reuses Phase 2 analysis
Validate only	`/run-pipeline plan=validate_only`	Source Tie-Out + Validation only

AI Behavior: Input/Output Examples

Input	Expected output
"Why is conversion dropping?"	Frames as funnel analysis. Segments by device, traffic source, cohort. Drills to specific step and dimension.
"What's our conversion rate?"	L1 fast path. Direct query + number. No pipeline invoked.
"Analyze revenue"	Clarifies: "Revenue by what dimension and time range? What decision will this inform?"
"Just tell me the answer"	Runs analysis, skips user-facing checkpoints, still runs validation.
Column name that doesn't exist	Checks corrections index. If found, applies fix. If not, halts with: "Column X not found. Available columns: [list]. Did you mean Y?"
Question requiring data that doesn't exist	Flags immediately: "This question requires [column/table] which isn't in the active dataset. Options: [alternatives or data needed]."

Input	Expected output
User correction mid-session	Logs to `.knowledge/corrections/`. Applied proactively in all future sessions on this dataset.
Ambiguous metric name	Checks business glossary. If found, uses definition. If not, asks for clarification before querying.

Key System Behaviors

Self-correction loop: User corrections are captured by the Feedback Capture skill, stored in `.knowledge/corrections/`, and retrieved before every SQL write. The same mistake is never made twice.

Confidence grades: Every analysis receives an A–F grade synthesized from all 4 validation layers. Grade C and below = findings stay internal. Grade B and above = safe to share with stakeholders.

Checkpoints: Two user-facing (question framing review, storyboard review) and two automated (analysis quality gate, deck lint). Say "just do it" to skip user-facing checkpoints.

Fallback chain: Primary connection fails → local DuckDB → CSV via pandas. Always tells the user which source is active.

Edge Cases

Situation	Behavior
Source tie-out fails	HALT. Surfaces mismatch. Asks: investigate or proceed with caution? Never presents unvalidated findings.
SQL errors	Simplifies query, shows what failed, offers alternatives.
Chart collision unresolved after 3 fix strategies	HALT at chart stage. Returns to storyboard.
3 agents fail in same tier	Circuit breaker fires. Pipeline halts with diagnostic report.
Agent timeout (5 min)	One retry. Critical agents halt; non-critical agents (Design Critic) degrade gracefully.
Context too long (15+ queries)	Saves all working files. Prompts `/resume-pipeline` in a fresh session.

Situation	Behavior
Schema differs from expected	Agents resolve tables/columns from active manifest — never from hardcoded names.

Non-Goals

- **No predictive modeling.** Descriptive and product analytics only. `/forecast` is a lightweight add-on, not a data science platform.
- **No dashboards.** Output is analyses and decks. For persistent dashboards: Looker, Metabase, Grafana.
- **No GUI.** Terminal only, via Claude Code. A web UI is explicitly out of scope — the target users are comfortable in a terminal.
- **No multi-user collaboration in V1.** Knowledge system, corrections, and profile are single-user. Team sharing is V2.
- **No built-in datasets.** Ships clean. Users connect their own data. Practice dataset exists for testing only.
- **Not a replacement for the analyst.** Handles the mechanical 80%. The analyst frames the question, validates output, and decides what findings mean for the business. This is an amplifier.

Success Metrics

Metric	Type	Baseline	Target	Timeframe
Time-to-deck per analysis	North Star	~5 days (manual)	< 2 hours	First 10 analyses
GitHub stars	Community	0	200	90 days post-publish
Pipeline success rate (no halts)	Quality	[NEED: log first 10 runs]	> 80%	60 days
Chart design pass rate (first submission)	Quality	[NEED: log first 10 runs]	> 70%	60 days
		0		60 days

Metric	Type	Baseline	Target	Timeframe
Corrections applied proactively	Self-learning		> 5 corrections retrieved per 10 runs	

Guardrail metrics: - Confidence grade: no finding below B reaches a stakeholder - Source tie-out: never bypassed — every run passes or explicitly halts - Test suite: 606 tests pass after every change; no regressions

Baseline collection plan: Log time, halt events, chart rejections, and resume events for the first 10 pipeline runs before publishing. This closes the [NEED] baselines above.

Rollout Plan

1. **Personal use (now):** Run on real analyses. Catch bugs. Build corrections index. Measure time-to-deck.
2. **GitHub publish (v1.0):** Public repo. Clean README. Tag the release.
3. **Community seeding:** Share in Lenny's Newsletter Slack, Mind the Product, r/datascience, r/ProductManagement.
4. **No A/B test:** Single-user tool. Success is personal productivity gain + community adoption.

Rollback plan: SQL logic errors that pass validation → log correction, applied proactively from next run. Systemic issues → `git revert` to last tagged release.

Open Questions

Question	Owner	Deadline
Time 3 manual analyses to set the time-to-deck baseline	Sankar	Before v1.0 publish
Source the analyst time-allocation stat (Gartner / Fishtown / DAMA)	Sankar	Before v1.0 publish
Ship practice dataset as git submodule or download script?	Sankar	Decision before v1.0 tag

Question	Owner	Deadline
Recommend MotherDuck specifically or keep README warehouse-agnostic?	Sankar	Before docs finalization
Which community channels have highest PM/analyst open-source signal?	Sankar	Week of launch