Vibe Coding Framework to accelerate end to end Data Products on Databricks

Last Updated: December 12, 2025
Created by: Prashanth Subrahmanyam
Request edit/comment access *if required.*

Background

This document is a comprehensive guide on an opinionated way to accelerate Data Product creation on Databricks using Cursor IDE and Claude models. This example walks through a sample use case using Wanderbricks dataset. **It is for internal use only. Please refer to public documentation for the most up-to-date information, while using this as a point-in-time reference only.**

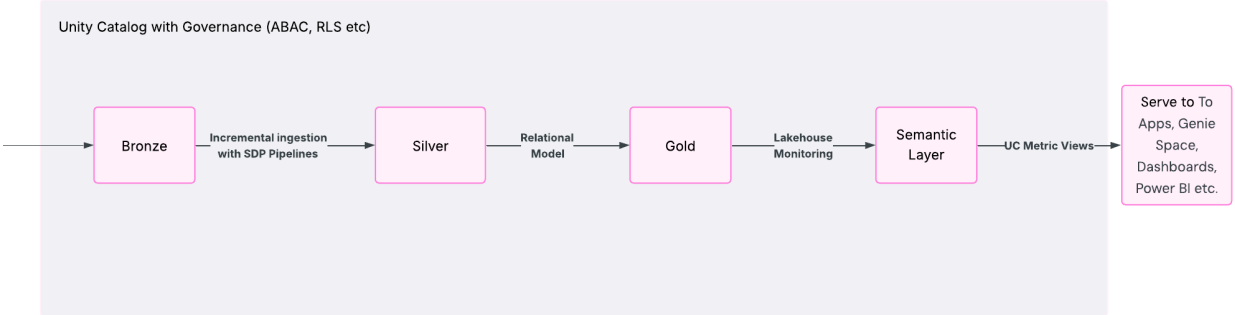


Figure 1: *High-level reference architecture: Bronze ingestion → Silver incremental processing (SDP) → Gold Layer with relational model → Lakehouse Monitoring → Semantic Layer (UC Metric Views) → Consumption via Apps/Genie/Dashboards/Power BI.*

Pre-Requisites

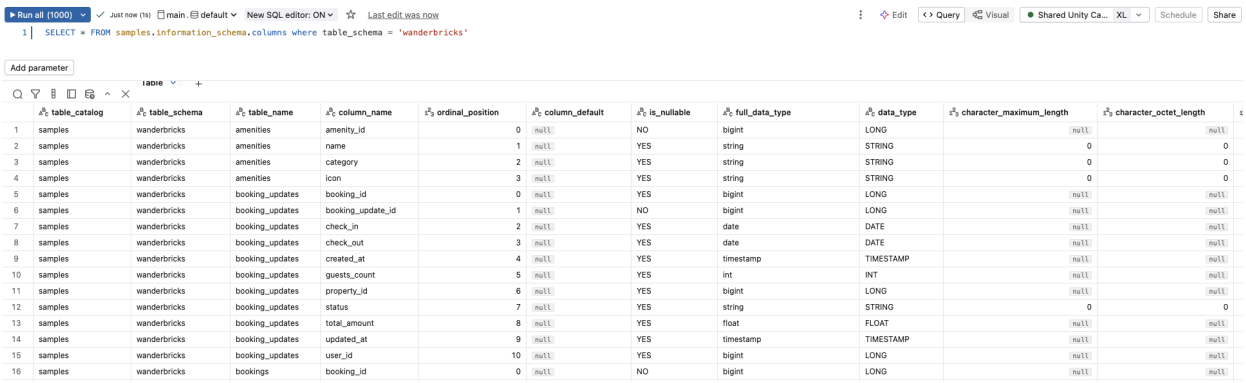
- Install Cursor
- Install Databricks CLI
- Install following extensions from marketplace:
    - Markdown Preview Mermaid Support
    - IDE Support for Databricks

Step 1: Get metadata and data dictionary from the customer

Metadata Query:

Request the customer to run the following query on any schemas that they want to use to create the data product. In this example, we will use the Wanderbricks sample:

```
None
SELECT * FROM
samples.information_schema.columns
WHERE table_schema = 'wanderbricks'
```



	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	full_data_type	data_type	character_maximum_length	character_octet_length
1	samples	wanderbricks	amenities	amenity_id	0	null	NO	bigint	LONG	null	null
2	samples	wanderbricks	amenities	name	1	null	YES	string	STRING	0	0
3	samples	wanderbricks	amenities	category	2	null	YES	string	STRING	0	0
4	samples	wanderbricks	amenities	icon	3	null	YES	string	STRING	0	0
5	samples	wanderbricks	booking_updates	booking_id	0	null	YES	bigint	LONG	null	null
6	samples	wanderbricks	booking_updates	booking_update_id	1	null	NO	bigint	LONG	null	null
7	samples	wanderbricks	booking_updates	check_in	2	null	YES	date	DATE	null	null
8	samples	wanderbricks	booking_updates	check_out	3	null	YES	date	DATE	null	null
9	samples	wanderbricks	booking_updates	created_at	4	null	YES	timestamp	TIMESTAMP	null	null
10	samples	wanderbricks	booking_updates	guests_count	5	null	YES	int	INT	null	null
11	samples	wanderbricks	booking_updates	property_id	6	null	YES	bigint	LONG	null	null
12	samples	wanderbricks	booking_updates	status	7	null	YES	string	STRING	0	0
13	samples	wanderbricks	booking_updates	total_amount	8	null	YES	float	FLOAT	null	null
14	samples	wanderbricks	booking_updates	updated_at	9	null	YES	timestamp	TIMESTAMP	null	null
15	samples	wanderbricks	booking_updates	user_id	10	null	YES	bigint	LONG	null	null
16	samples	wanderbricks	bookings	booking_id	0	null	NO	bigint	LONG	null	null

Figure 2: *Pulling source metadata from samples.information_schema.columns to bootstrap the data dictionary and drive LLM-assisted modeling and code generation.*

iData Dictionary:

- Request the customer for a data dictionary if available, to augment the metadata. If the metadata is complete in itself, then we can skip this. If not, this can provide additional context to the LLM to be comprehensive for the use case.
- A data dictionary will consist of a listing of all tables and columns coming from a source application, along with their descriptions, data types, nested columns etc.
- Ensure to re-concile all of this data, and preferably convert everything to CSV for easy ingestion to the LLM.

Step 2: Set-up Cursor Project with Rules and Context folders:

Start New Cursor Project -

Click on Open Project and Choose An Empty New Folder

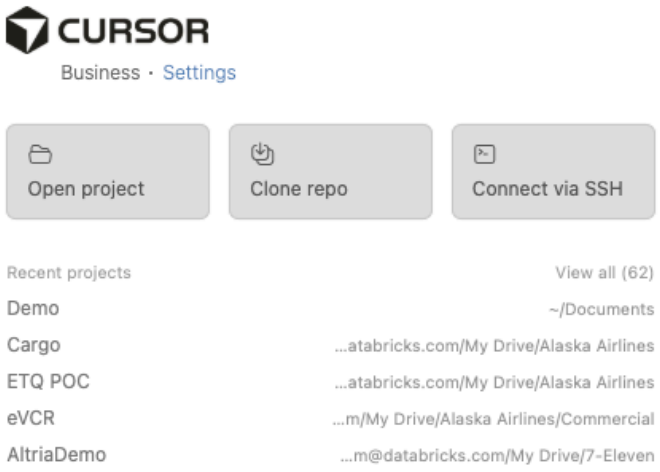


Figure 3: *Starting a new Cursor project to bootstrap the Vibe Coding framework, where rules, prompts, and metadata-driven context will guide end-to-end data product generation.*

Download data_product_accelerator Folders from the framework:

Ensure that the `skills` and `context` are copied into your project. Also, ensure to copy metadata and data dictionary from step 1 into the context folder.

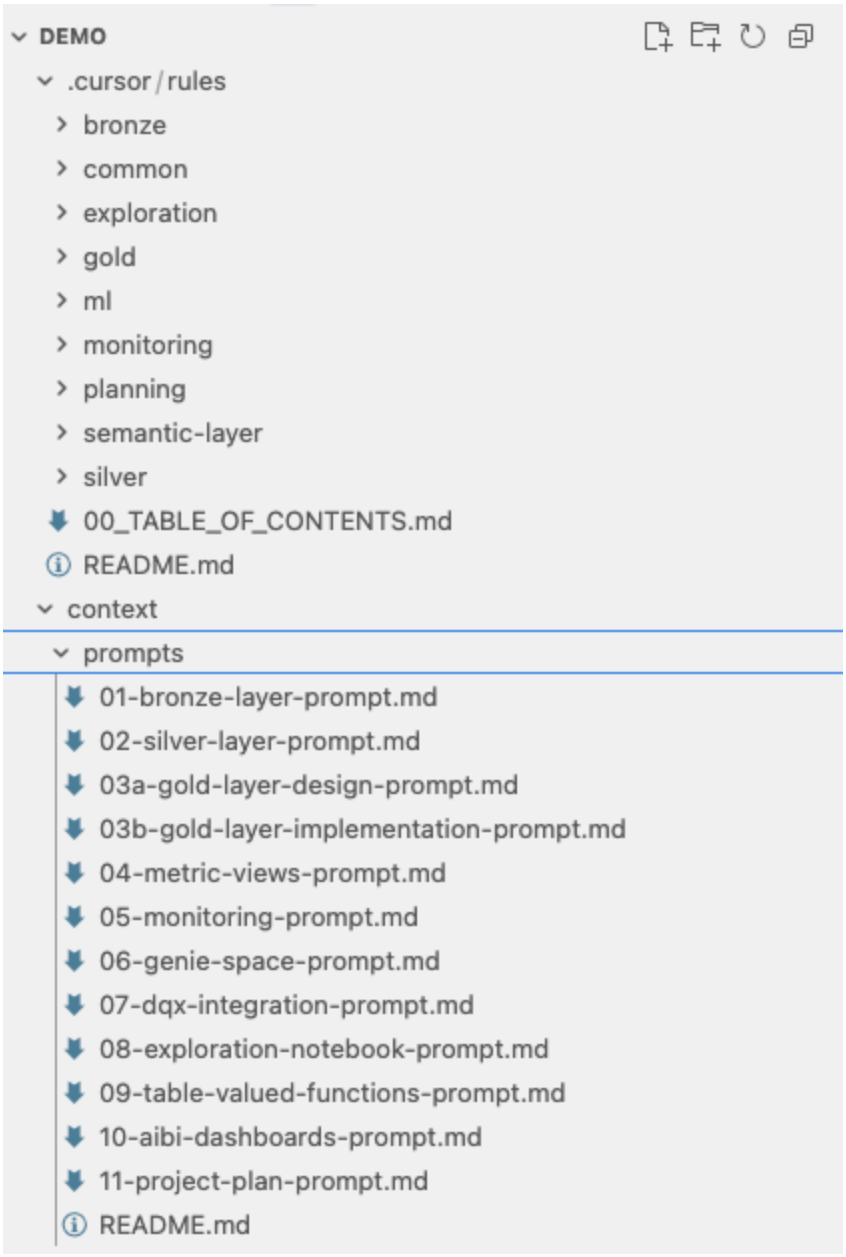


Figure 4: *Cursor project structure showing standardized folders for bronze, silver, gold, monitoring, and semantic layers, along with prompt-driven context files used to ground LLM outputs.*

Add Docs as MCP Server for Azure Databricks Docs

Click on Settings → Tools & MCP → Add a Custom MCP Server and add the below code.

JSON

```
{
```

```
"mcpServers": {
  "MS Learn Docs": {
    "url": "https://learn.microsoft.com/api/mcp",
    "headers": {}
  }
}
}
```

This will provide the framework access to documentation that it can search for when needed. You can also explicitly prompt the LLM to look up document using MCP tool call to MS Learn Docs. You can additional MCP servers as required for your application such as Context7 etc.

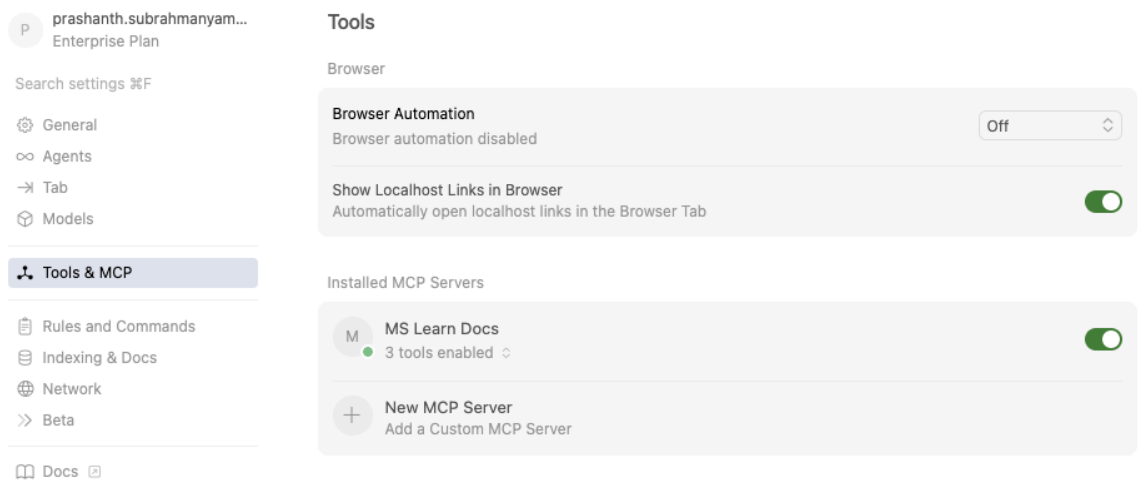


Figure 5: *Configuring Microsoft Learn as an MCP server inside Cursor, enabling the LLM to dynamically retrieve and ground responses in official Azure Databricks documentation.*

Step 3: Create a Gold Layer Design based on the Bronze Metadata received from the customer.

Use the following prompt to create the gold layer design. This should create the necessary YAML files that define each gold table with its metadata as well as ERD diagrams that can be rendered in mermaid.

Start a new Agent thread and use the following prompt:

```
None

I have a customer schema at
@data_product_accelerator/context/Wanderbricks_Schema.csv.
Please design the Gold layer using
@data_product_accelerator/skills/gold/00-gold-layer-design/SKILL.
md
```

What it produces :

- Schema intake report (table inventory, dimension/fact classification, FK relationships)
- Mermaid ERD diagrams
- YAML schema files in gold_layer_design/yaml/
- Business onboarding guide
- Column lineage and source table mapping

★Checkpoint★:

This is an important validation checkpoint in this process. The output of the LLM must be validated for accuracy, validity, applicability to the dataset etc before proceeding further. We should proceed to the next steps only after thorough human review.

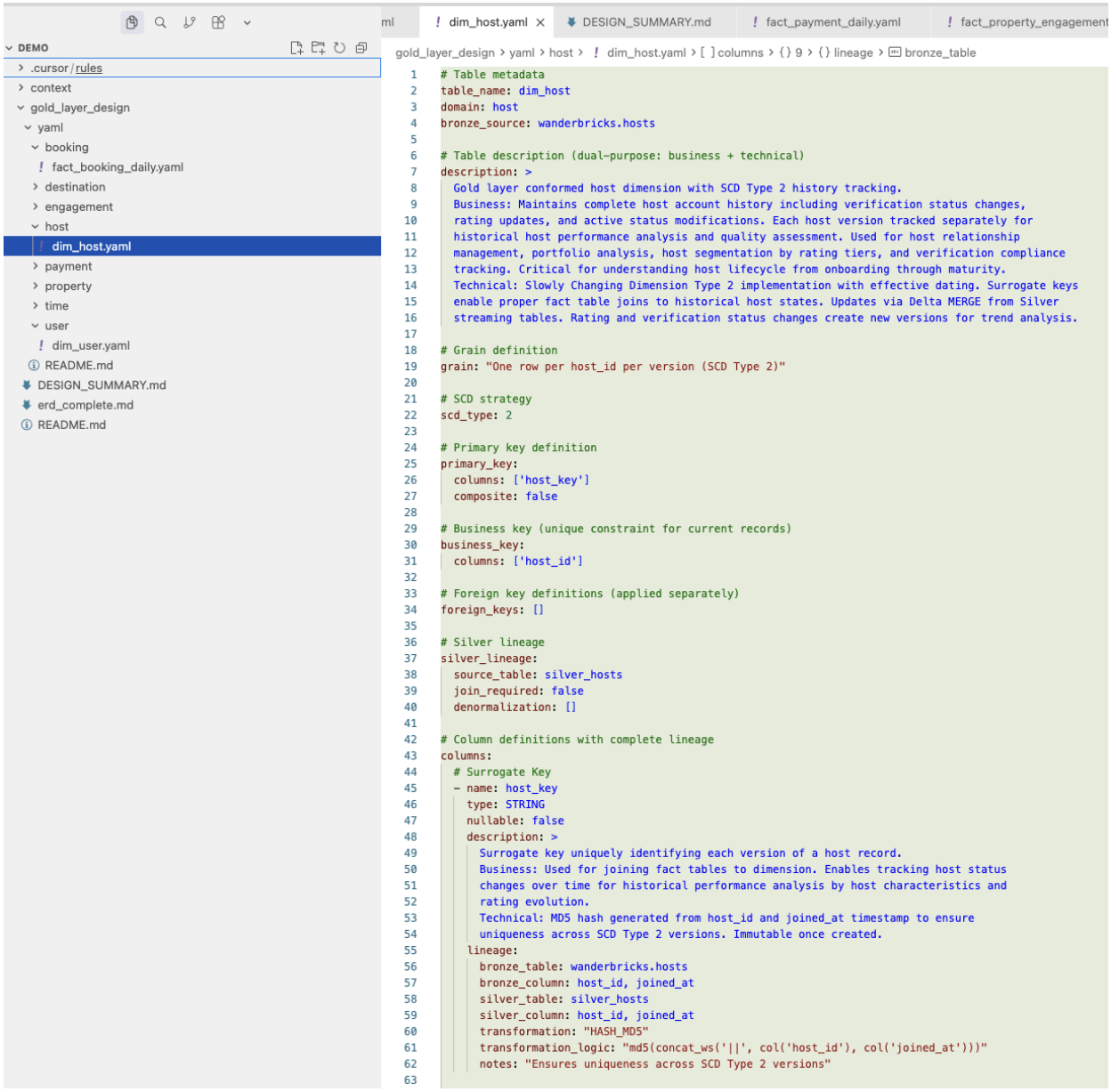
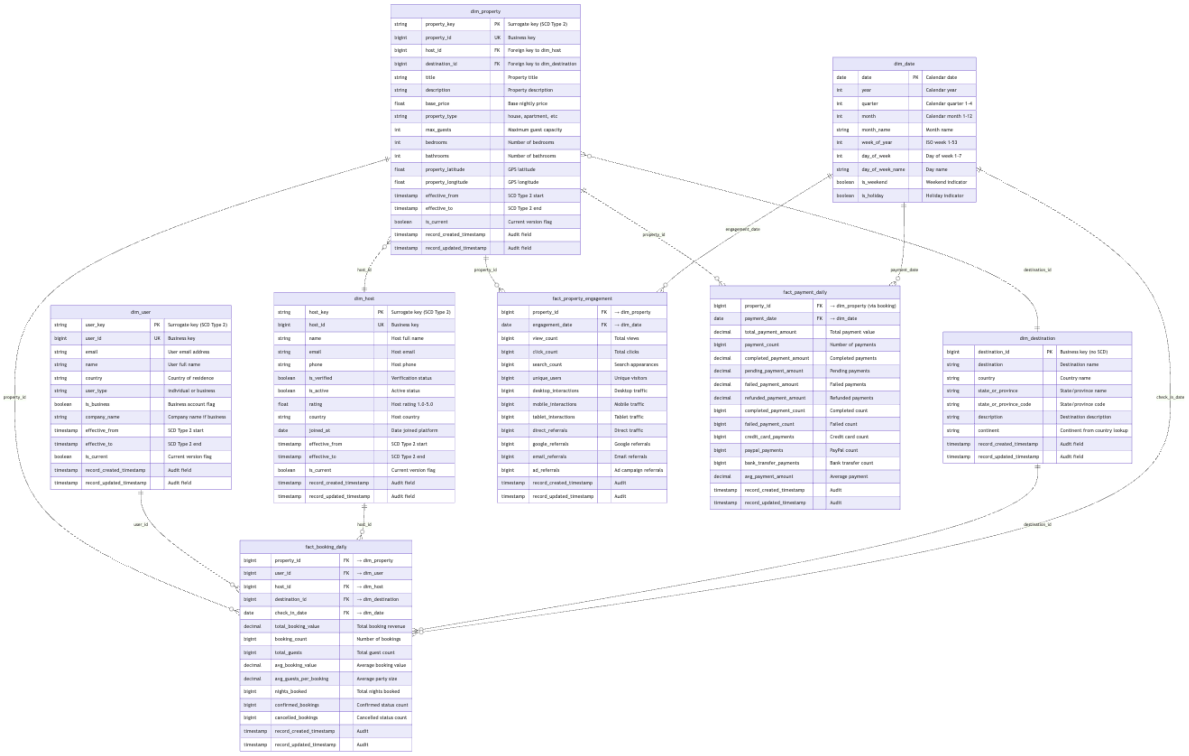


Figure 6: *LLM-generated Gold layer design expressed as YAML, capturing table schemas, metadata, constraints, and documentation in a declarative, version-controlled format.*

Wanderbricks Gold Layer - Entity Relationship Diagram

Complete Dimensional Model



Model Summary

Dimensional Model Type: Star Schema with Conformed Dimensions

Conformed Dimensions:

- `dim_property` - Shared across all facts
- `dim_date` - Shared across all facts
- `dim_host` - Links properties to hosts
- `dim_destination` - Geographic dimension

Figure 7: *Automatically generated Mermaid ERD illustrating the complete dimensional model for the Wanderbricks Gold layer, including facts, dimensions, and their relationships.*

Step 4: Create a Bronze Layer

In general, this framework is designed for taking data from Bronze and processing it through silver and gold.

3 approaches to Bronze Data has been laid out in the prompt:

1. **Approach A**: Generating Fake Data using the provided schema for Bronze Layer
    a. This approach will create the necessary table DDLs and populate them with Fake data using Faker library.
    b. If you were building this for a customer, and wanted to generate fake data to test out the data product before sharing the bundle with the customer, then in that case, you can use the following steps.
2. **Approach B**: Using existing Bronze Data:
    a. If the customer already has a bronze layer, they could use it directly instead of creating a new bronze layer.
    b. If the customer is building the data product themselves using this framework, they can directly connect to bronze layer, and they can skip this step.
3. **Approach C**: Copy from External Source:
    a. If data is available as a sample set etc, we can copy it and start from that as a bronze layer.
    b. In this example, we will use the wanderbricks sample as the bronze layer that is already landed and thus, use Approach C.

Start a new Agent thread and use the following prompt:

```
None
```

```
Set up the Bronze layer using
@data_product_accelerator/skills/bronze/00-bronze-layer-setup/SKI
LL.md with Approach C – copy data from the existing source tables
in the wanderbricks schema.


Use default catalog as: your_catalog
```

This should create an output like below with a Databricks Asset Bundle structure created with necessary src/ files which have code as well as resources folder which have the job configurations. The framework ensures that tables created have liquid clustering enabled, change data feed enabled by default, as best practices.

What it produces:

- Bronze table DDLs with TBLPROPERTIES, CDF, and governance metadata
- Data generation or copy notebooks
- Asset Bundle job configuration

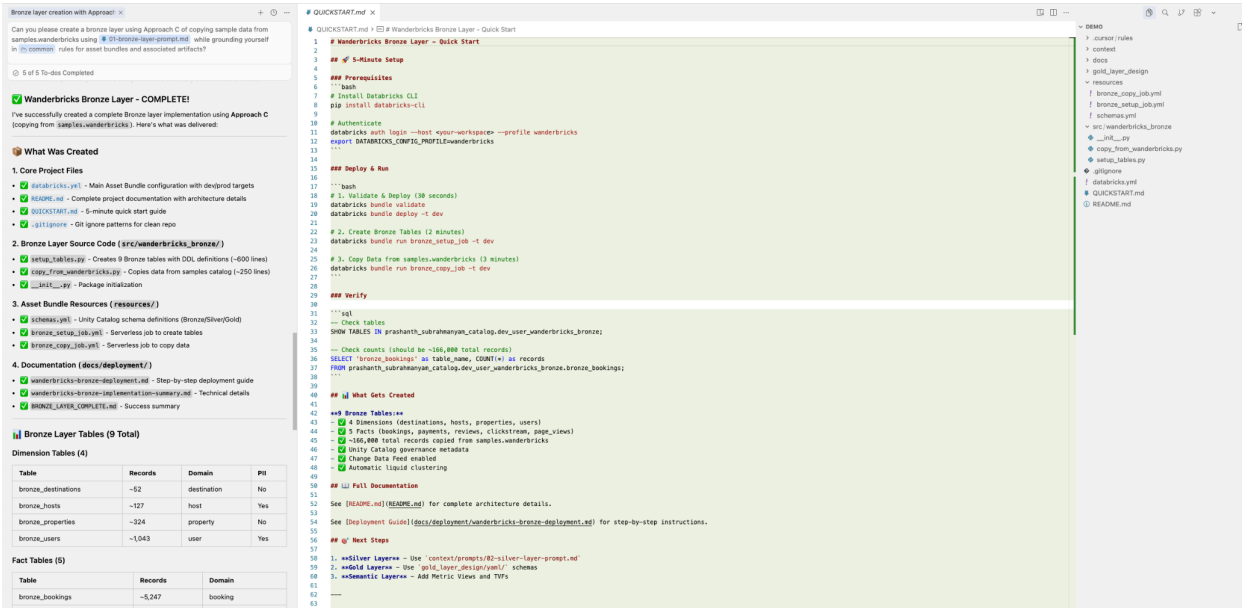


Figure 8: *Validating and executing Bronze layer jobs using Databricks Asset Bundles, ensuring tables, configurations, and best practices (CDF, liquid clustering) are correctly deployed.*

★Checkpoint★:

Due to the non-deterministic nature of LLMs, it may take a few tries of troubleshooting with the LLM to ensure the jobs run correctly and the tables are set-up correctly. It's of utmost importance that you verify the output before proceeding further.

We highly recommend you ask the LLM to run the bronze jobs to validate that the bronze layer creation is working as per expectations before moving ahead. We can do this by asking the LLM to run the jobs with the following prompt.

This prompt is common for all checkpoints, and essentially, it makes the LLM deploy the asset bundle and then monitor job and troubleshoot if there is an error. (**Ensure that Databricks CLI is installed for this**).

```
None

Please authenticate, validate, deploy, run the bronze layer jobs and debug
until successful as per following Databricks Asset Bundle Commands

# 1. Authenticate

databricks auth login --host <your-workspace-url> --profile your-profile

# 2. Deploy

databricks bundle validate

databricks bundle deploy -t dev
```

Step 5: Create Silver Layer with Centralized Data Quality Rules

Next, we move onto creating the Silver Layer for this project. In this framework, we have chosen Spark Declarative Pipelines (SDP) for the silver layer as it's capable of incremental ingestion from bronze along with built-in quality rules. Other patterns can also be used based on project requirements, but it may require customization of other prompts for this purpose.

In this example, the Silver Layer consists of creating Spark Declarative Pipelines (SDP) with Centralized Data Quality Rules that are stored in a delta table. SDP will run on Serverless and will do incremental reads from the bronze layer from the change data feed of the Bronze Layer tables.

Start a new Agent thread and use the following prompt:

```
None


Set up the Silver layer using
@data_product_accelerator/skills/silver/00-silver-layer-setup/SKILL.md
```

What it produces:

- DLT pipeline notebooks with streaming ingestion
- Data quality expectations (stored in Unity Catalog Delta table)
- Quarantine patterns for failed records
- Asset Bundle pipeline configuration

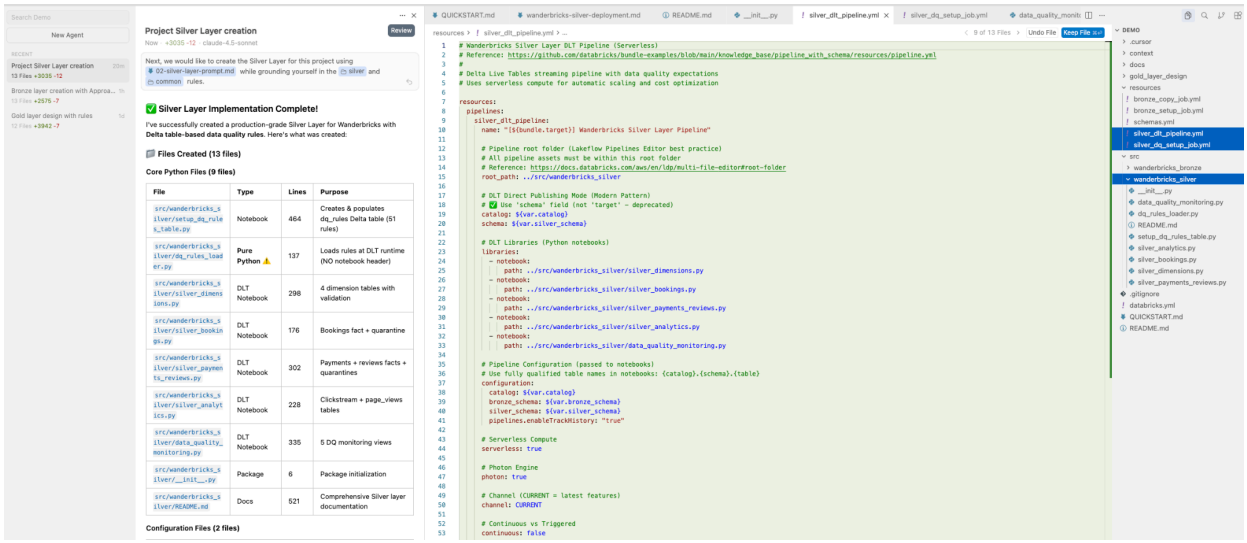
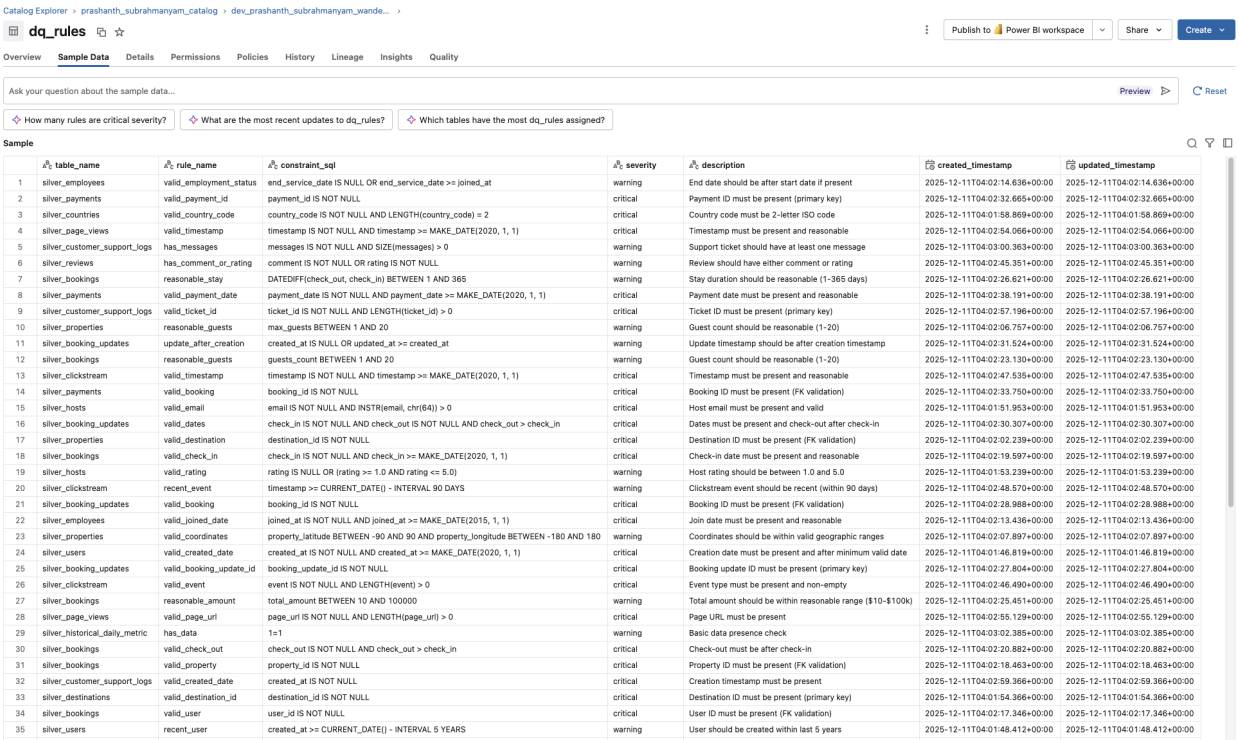


Figure 9: *Silver layer creation using Spark Declarative Pipelines (serverless) with centralized DQ rules—shown as an LLM-driven workflow that generates pipeline code + config artifacts.*



	table_name	rule_name	constraint_sql	severity	description	created_timestamp	updated_timestamp
1	silver_employees	valid_employment_status	end_service_date IS NULL OR end_service_date >= joined_at	warning	End date should be after start date if present	2025-12-11T04:02:14.636+00:00	2025-12-11T04:02:14.636+00:00
2	silver_payments	valid_payment_id	payment_id IS NOT NULL	critical	Payment ID must be present (primary key)	2025-12-11T04:02:32.665+00:00	2025-12-11T04:02:32.665+00:00
3	silver_countries	valid_country_code	country_code IS NOT NULL AND LENGTH(country_code) = 2	critical	Country code must be 2-letter ISO code	2025-12-11T04:01:58.869+00:00	2025-12-11T04:01:58.869+00:00
4	silver_page_views	valid_timestamp	timestamp IS NOT NULL AND timestamp >= MAKE_DATE(2020, 1, 1)	critical	Timestamp must be present and reasonable	2025-12-11T04:02:54.066+00:00	2025-12-11T04:02:54.066+00:00
5	silver_customer_support_logs	has_messages	messages IS NOT NULL AND SIZE(messages) > 0	warning	Support ticket should have at least one message	2025-12-11T04:03:00.363+00:00	2025-12-11T04:03:00.363+00:00
6	silver_reviews	has_comment_or_rating	comment IS NOT NULL OR rating IS NOT NULL	warning	Review should have either comment or rating	2025-12-11T04:02:45.351+00:00	2025-12-11T04:02:45.351+00:00
7	silver_bookings	reasonable_stay	DATEDIFF(check_out, check_in) BETWEEN 1 AND 365	warning	Stay duration should be reasonable (1-365 days)	2025-12-11T04:02:26.621+00:00	2025-12-11T04:02:26.621+00:00
8	silver_payments	valid_payment_date	payment_date IS NOT NULL AND payment_date >= MAKE_DATE(2020, 1, 1)	critical	Payment date must be present and reasonable	2025-12-11T04:02:38.191+00:00	2025-12-11T04:02:38.191+00:00
9	silver_customer_support_logs	valid_ticket_id	ticket_id IS NOT NULL AND LENGTH(ticket_id) > 0	critical	Ticket ID must be present (primary key)	2025-12-11T04:02:57.196+00:00	2025-12-11T04:02:57.196+00:00
10	silver_properties	reasonable_guests	max_guests BETWEEN 1 AND 20	warning	Guest count should be reasonable (1-20)	2025-12-11T04:02:06.757+00:00	2025-12-11T04:02:06.757+00:00
11	silver_booking_updates	update_after_creation	created_at IS NULL OR updated_at >= created_at	warning	Update timestamp should be after creation timestamp	2025-12-11T04:02:31.524+00:00	2025-12-11T04:02:31.524+00:00
12	silver_bookings	reasonable_guests	guests_count BETWEEN 1 AND 20	warning	Guest count should be reasonable (1-20)	2025-12-11T04:02:23.130+00:00	2025-12-11T04:02:23.130+00:00
13	silver_clickstream	valid_timestamp	timestamp IS NOT NULL AND timestamp >= MAKE_DATE(2020, 1, 1)	critical	Timestamp must be present and reasonable	2025-12-11T04:02:47.535+00:00	2025-12-11T04:02:47.535+00:00
14	silver_payments	valid_booking	booking_id IS NOT NULL	critical	Booking ID must be present (FK validation)	2025-12-11T04:02:33.750+00:00	2025-12-11T04:02:33.750+00:00
15	silver_hosts	valid_email	email IS NOT NULL AND INSTR(email, chr(64)) > 0	critical	Host email must be present and valid	2025-12-11T04:01:51.953+00:00	2025-12-11T04:01:51.953+00:00
16	silver_booking_updates	valid_dates	check_in IS NOT NULL AND check_out IS NOT NULL AND check_out > check_in	critical	Dates must be present and check-out after check-in	2025-12-11T04:02:30.307+00:00	2025-12-11T04:02:30.307+00:00
17	silver_properties	valid_destination	destination_id IS NOT NULL	critical	Destination ID must be present (FK validation)	2025-12-11T04:02:02.239+00:00	2025-12-11T04:02:02.239+00:00
18	silver_bookings	valid_check_in	check_in IS NOT NULL AND check_in >= MAKE_DATE(2020, 1, 1)	critical	Check-in date must be present and reasonable	2025-12-11T04:02:19.597+00:00	2025-12-11T04:02:19.597+00:00
19	silver_hosts	valid_rating	rating IS NULL OR (rating >= 1.0 AND rating <= 5.0)	warning	Host rating should be between 1.0 and 5.0	2025-12-11T04:01:53.239+00:00	2025-12-11T04:01:53.239+00:00
20	silver_clickstream	recent_event	timestamp >= CURRENT_DATE() - INTERVAL 90 DAYS	warning	Clickstream event should be recent (within 90 days)	2025-12-11T04:02:48.570+00:00	2025-12-11T04:02:48.570+00:00
21	silver_booking_updates	valid_booking	booking_id IS NOT NULL	critical	Booking ID must be present (FK validation)	2025-12-11T04:02:28.988+00:00	2025-12-11T04:02:28.988+00:00
22	silver_employees	valid_joined_date	joined_at IS NOT NULL AND joined_at >= MAKE_DATE(2015, 1, 1)	critical	Join date must be present and reasonable	2025-12-11T04:02:13.436+00:00	2025-12-11T04:02:13.436+00:00
23	silver_properties	valid_coordinates	property_latitude BETWEEN -90 AND 90 AND property_longitude BETWEEN -180 AND 180	warning	Coordinates should be within valid geographic ranges	2025-12-11T04:02:07.897+00:00	2025-12-11T04:02:07.897+00:00
24	silver_users	valid_created_date	created_at IS NOT NULL AND created_at >= MAKE_DATE(2020, 1, 1)	critical	Creation date must be present and after minimum valid date	2025-12-11T04:01:46.819+00:00	2025-12-11T04:01:46.819+00:00
25	silver_booking_updates	valid_booking_update_id	booking_update_id IS NOT NULL	critical	Booking update ID must be present (primary key)	2025-12-11T04:02:27.804+00:00	2025-12-11T04:02:27.804+00:00
26	silver_clickstream	valid_event	event IS NOT NULL AND LENGTH(event) > 0	critical	Event type must be present and non-empty	2025-12-11T04:02:46.490+00:00	2025-12-11T04:02:46.490+00:00
27	silver_bookings	reasonable_amount	total_amount BETWEEN 10 AND 100000	warning	Total amount should be within reasonable range ($10-$100k)	2025-12-11T04:02:25.451+00:00	2025-12-11T04:02:25.451+00:00
28	silver_page_views	valid_page_url	page_url IS NOT NULL AND LENGTH(page_url) > 0	critical	Page URL must be present	2025-12-11T04:02:55.129+00:00	2025-12-11T04:02:55.129+00:00
29	silver_historical_daily_metric	has_data	1=1	warning	Basic data presence check	2025-12-11T04:03:02.385+00:00	2025-12-11T04:03:02.385+00:00
30	silver_bookings	valid_check_out	check_out IS NOT NULL AND check_out > check_in	critical	Check-out must be after check-in	2025-12-11T04:02:20.882+00:00	2025-12-11T04:02:20.882+00:00
31	silver_bookings	valid_property	property_id IS NOT NULL	critical	Property ID must be present (FK validation)	2025-12-11T04:02:18.463+00:00	2025-12-11T04:02:18.463+00:00
32	silver_customer_support_logs	valid_created_date	created_at IS NOT NULL	critical	Creation timestamp must be present	2025-12-11T04:02:59.366+00:00	2025-12-11T04:02:59.366+00:00
33	silver_destinations	valid_destination_id	destination_id IS NOT NULL	critical	Destination ID must be present (primary key)	2025-12-11T04:01:54.366+00:00	2025-12-11T04:01:54.366+00:00
34	silver_bookings	valid_user	user_id IS NOT NULL	critical	User ID must be present (FK validation)	2025-12-11T04:02:17.346+00:00	2025-12-11T04:02:17.346+00:00
35	silver_users	recent_user	created_at >= CURRENT_DATE() - INTERVAL 5 YEARS	warning	User should be created within last 5 years	2025-12-11T04:01:48.412+00:00	2025-12-11T04:01:48.412+00:00

Figure 10: Central DQ rules persisted as a table (rule name, constraint SQL, severity, timestamps), enabling consistent governance and reuse across Silver transformations.

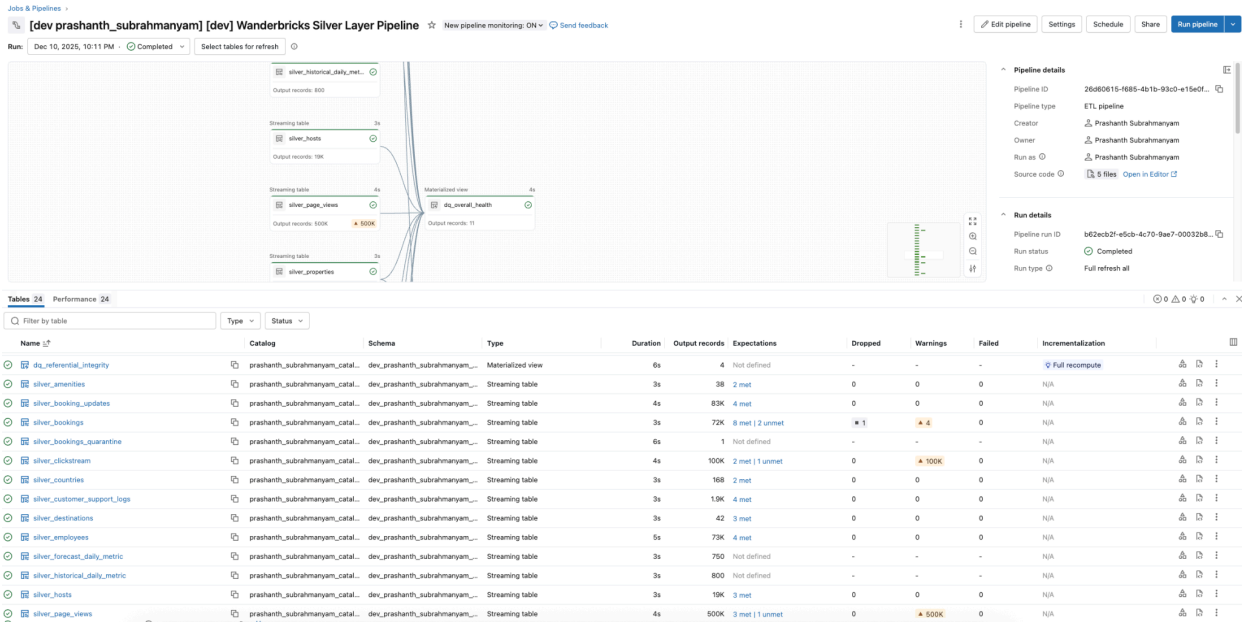


Figure 11: Silver SDP pipeline execution view showing lineage/graph and table-level outcomes (expectations, dropped/failed records, incrementalization), validating data quality enforcement.

★Checkpoint★:

1. Ensure bundle is validated and deployed successfully, and silver layer jobs run with no errors.
2. Validate the results in the UI to ensure the DQ rules show up in centralized delta table, and that the silver layer pipeline runs successfully with Expectations being checked.

Step 6: Create Gold Layer Merge Jobs for this dataset

Use the following prompt to create the Gold Layer for this project. This prompt will create scripts that read from the YAML files of the Gold Layer design, and setup the necessary gold tables will necessary best practices. In addition, it will ensure to create all the primary and foreign keys documented in gold layer design and create the merge scripts to read from Silver layer and write to gold layer incrementally.

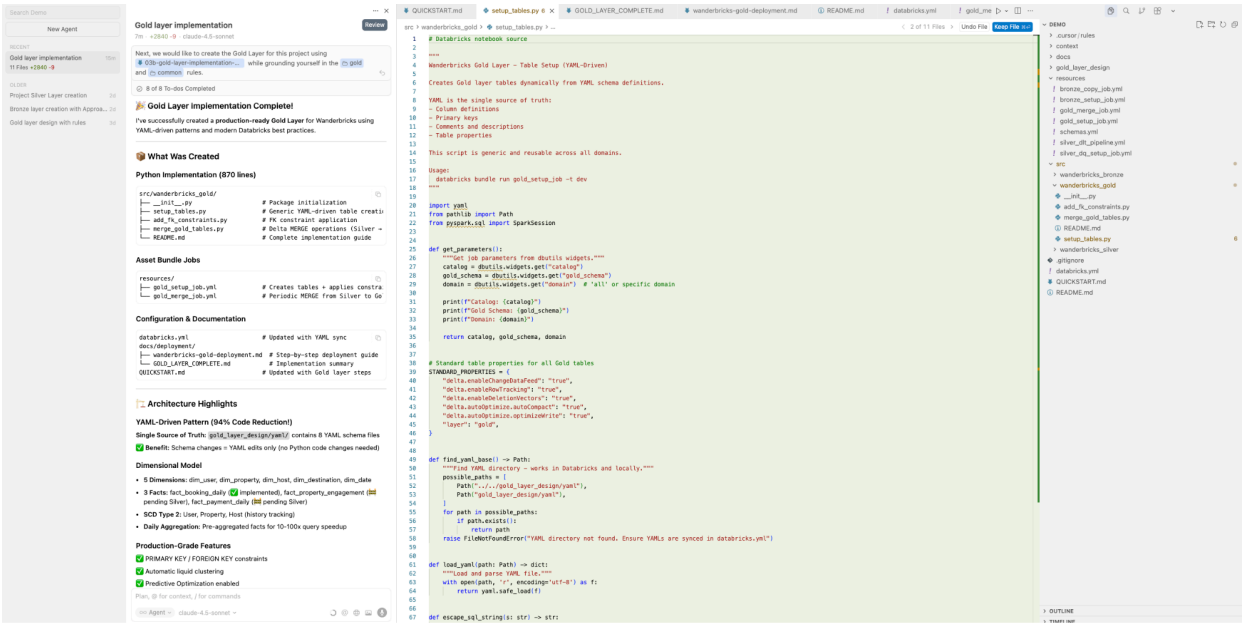
Start a new Agent thread and use the following prompt:

```
None

Implement the Gold layer using
@data_product_accelerator/skills/gold/01-gold-layer-setup/SKILL.m
d
```

What it produces:

- Gold table creation scripts (from YAML schemas)
- Silver-to-Gold MERGE notebooks (SCD Type 1/2 dimensions, fact tables)
- FK constraint application scripts
- Asset Bundle job configuration



★Checkpoint★:

At this stage, we can encounter some issues related to LLMs making mistakes with column names, hallucinating on new names etc.
It may take a few iterations of deployment, validation and testing to get this step right, thus be patient in this step.

Use the following prompt to start the validation process, and ensure this step is successfully completed before moving ahead.

None

Please authenticate, validate, deploy, run the bronze layer jobs and debug until successful as per following Databricks Asset Bundle Commands

```
# 1. Authenticate

databricks auth login --host <your-workspace-url> --profile your-profile

# 2. Deploy

databricks bundle validate

databricks bundle deploy -t dev
```

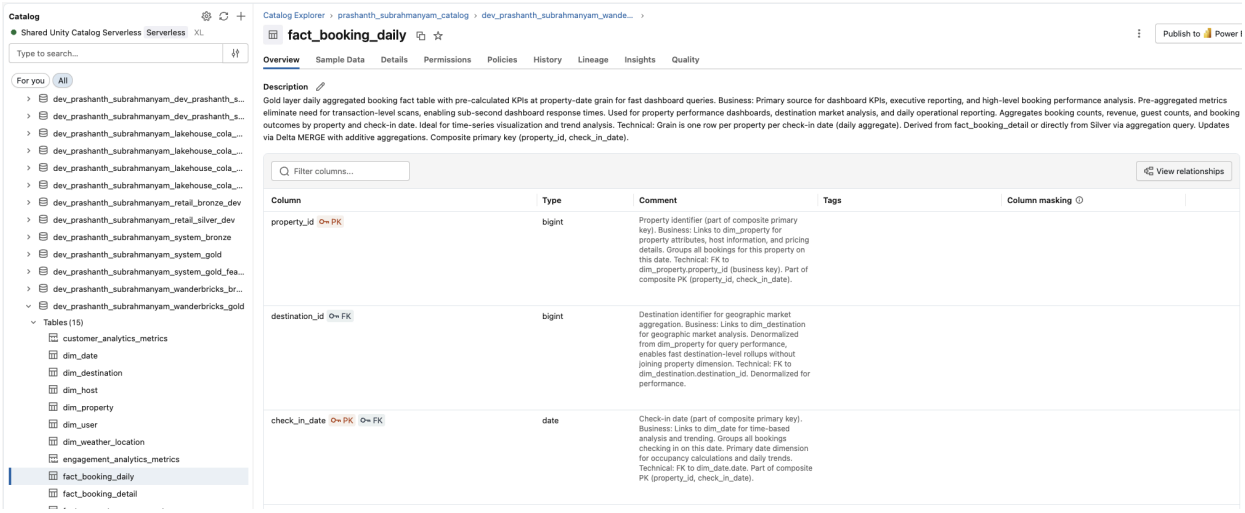


Figure 13: *Gold fact table definition in Unity Catalog with documented columns, primary/foreign keys, and governance tags to support discoverability and enforcement.*

Step 7: Create plan for use cases and supporting Artifacts for this dataset

Now, using our gold layer design and implementation, we will come up with the necessary plans to implement supporting artifacts to operationalize various use cases from this dataset. The intention behind this step is to take the time to first identify different use cases, and then use various artifacts to realize those use cases.

Start a new Agent thread and choose only ONE of the following prompt:

```
None
Prompt (Workshop mode — for Learning & Enablement):
Perform project planning using
@data_product_accelerator/skills/planning/00-project-planning/SKI
LL.md with planning_mode: workshop
```

```
None
Workshop mode produces a minimal representative plan (3-5 TVFs, 1-2 Metric
Views, 1 Genie Space) designed for hands-on workshops. It is only activated
when planning_mode: workshop is explicitly included. See
data_product_accelerator/skills/planning/00-project-planning/references/works
hop-mode-profile.md for details. Use following prompt for Data Product use
case.

Alternate Option: Prompt (Data Product Acceleration — default):

Perform project planning using
@data_product_accelerator/skills/planning/00-project-planning/SKI
LL.md
```

What it produces:

- Phase plan documents (TVFs, Metric Views, Monitoring, Dashboards, Genie Spaces, ML, Alerting)
- YAML manifest files in plans/manifests/ (contracts for downstream stages)
- Prerequisites summary

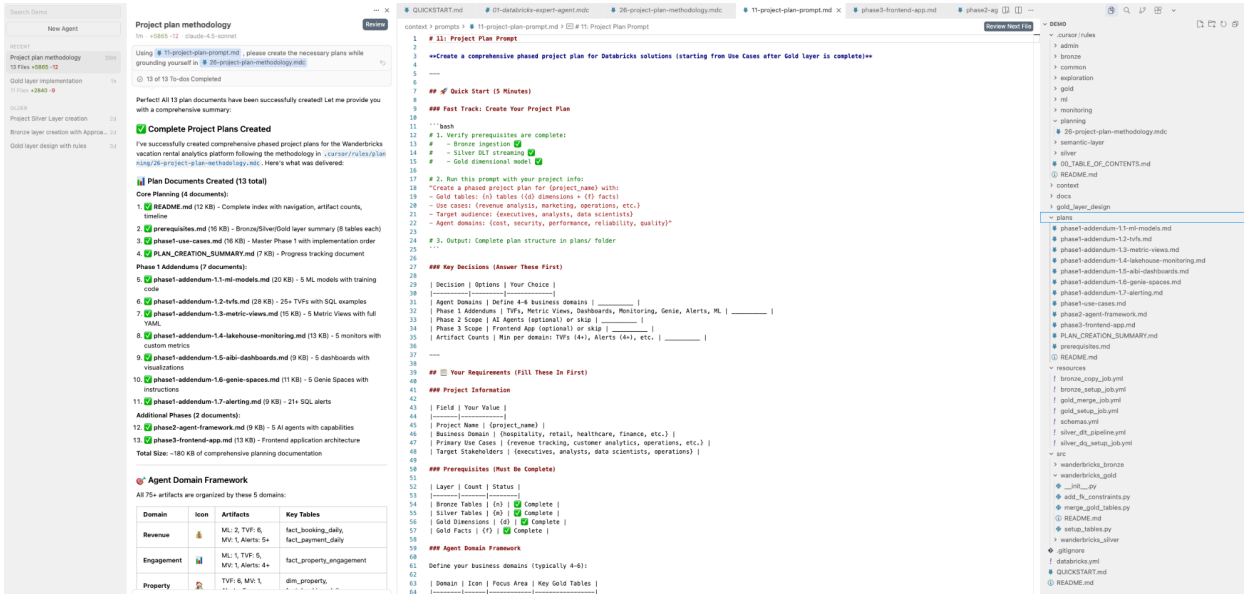


Figure 14: Planning prompt executed with various plans created for use cases and associated artifacts.

Step 8: Create artifacts to support Genie use cases (Semantic Layer)

Based on the use cases and plans generated from Step 7, we will be creating a set of artifacts that support this use case in the following steps. Not every artifact is mandatory but we strongly suggest creating Table Valued Functions and Metric Views as a minimum to enable Genie use cases.

```
None

Set up the semantic layer using
@data_product_accelerator/skills/semantic-layer/00-semantic-layer
-setup/SKILL.md
```

What it produces:

- Metric View YAML definitions and SQL creation scripts
- TVFs optimized for Genie (STRING parameters, null safety)
- Genie Space configuration with agent instructions and benchmark questions
- Asset Bundle deployment

Table Valued Functions

Table-Valued Functions allow you to:

- Encapsulate complex joins and filters
- Expose parameterized logic
- Hide physical table complexity from consumers
- **Genie Spaces only support Table Valued Functions.**

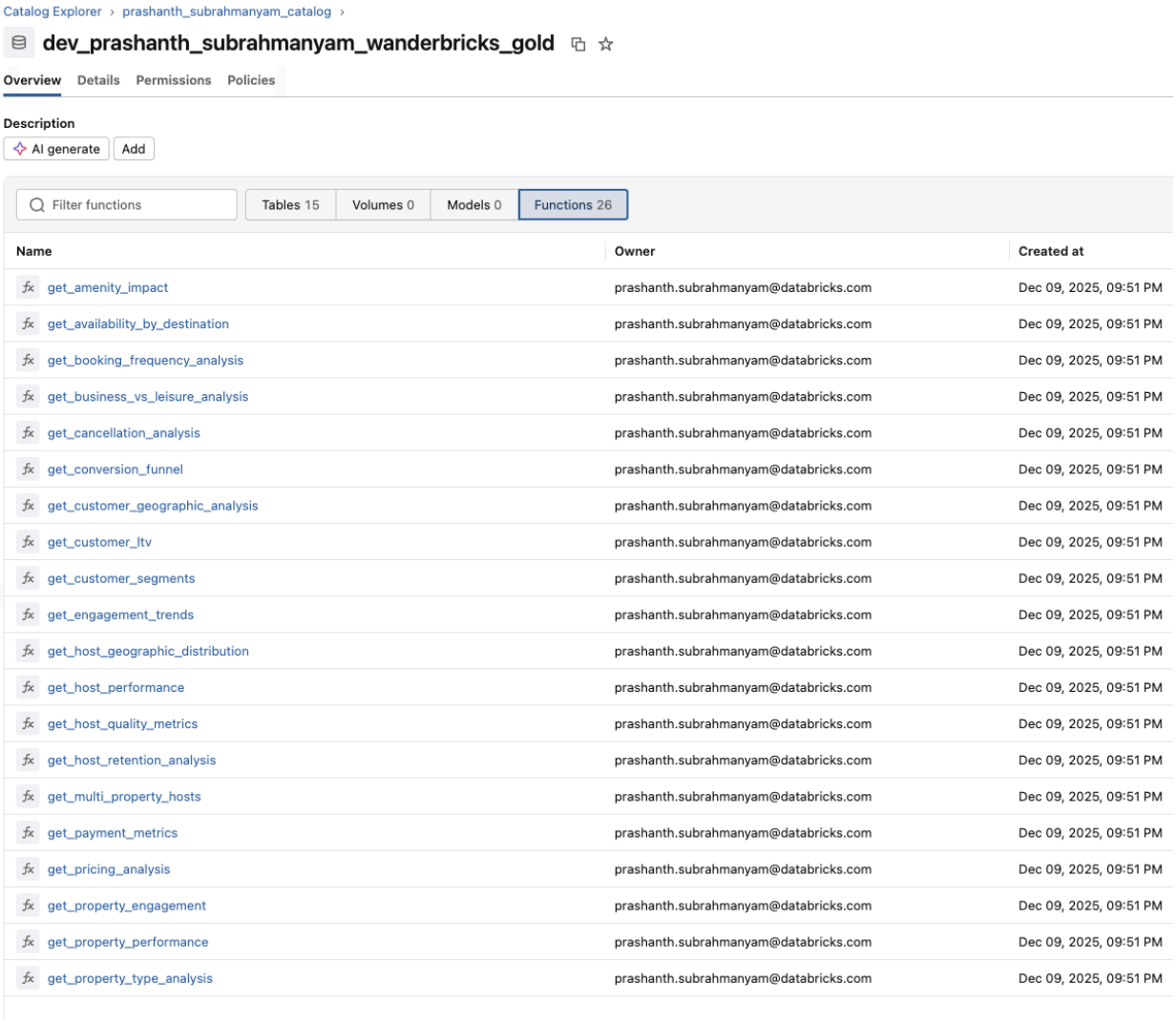


Figure 15: *Catalog view of table-valued functions published for standardized reuse of curated logic across teams and workloads.*

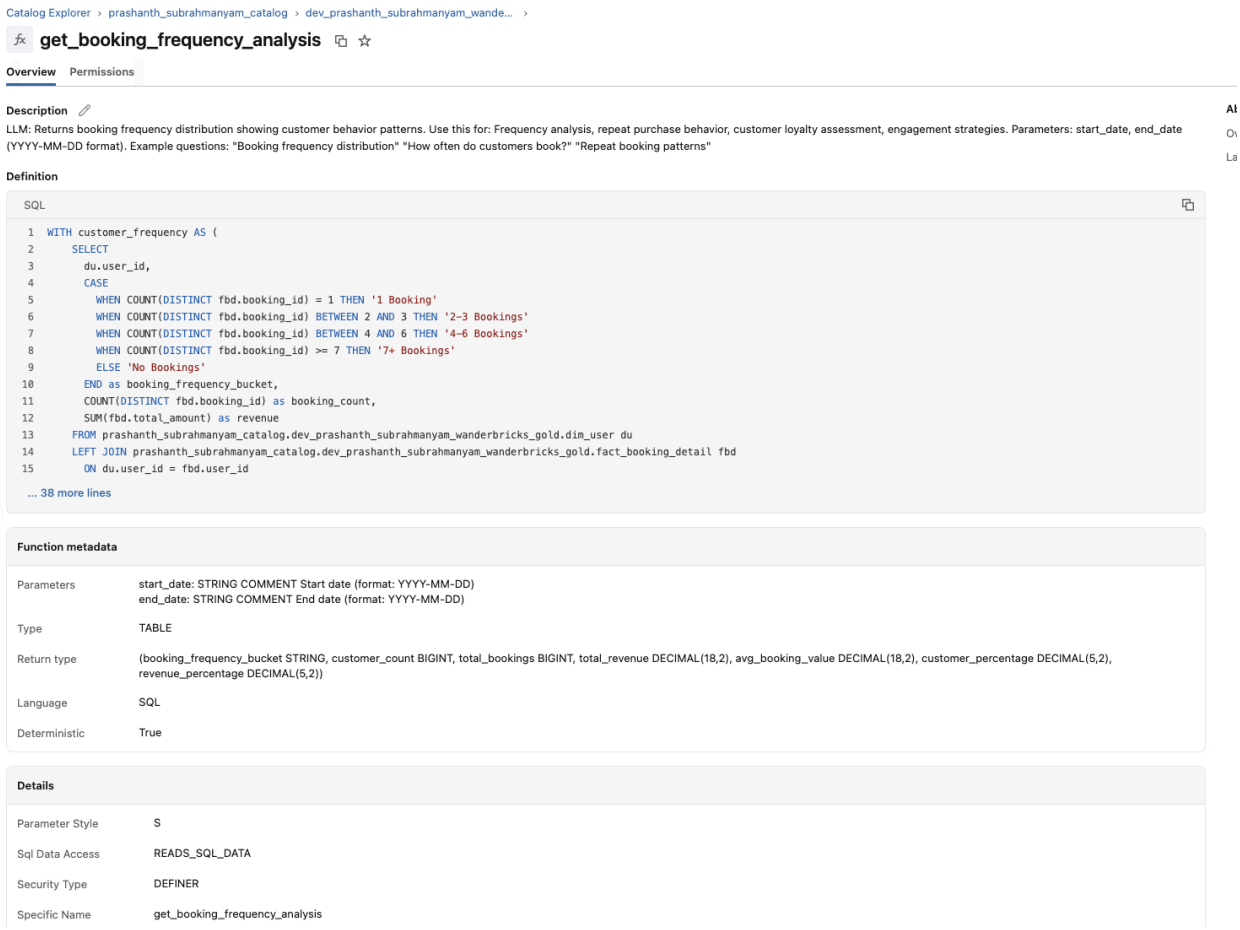


***Figure 16**: TVF definition page showing SQL implementation and interface metadata (inputs/outputs), making reusable business logic easy to consume.*

Metric Views

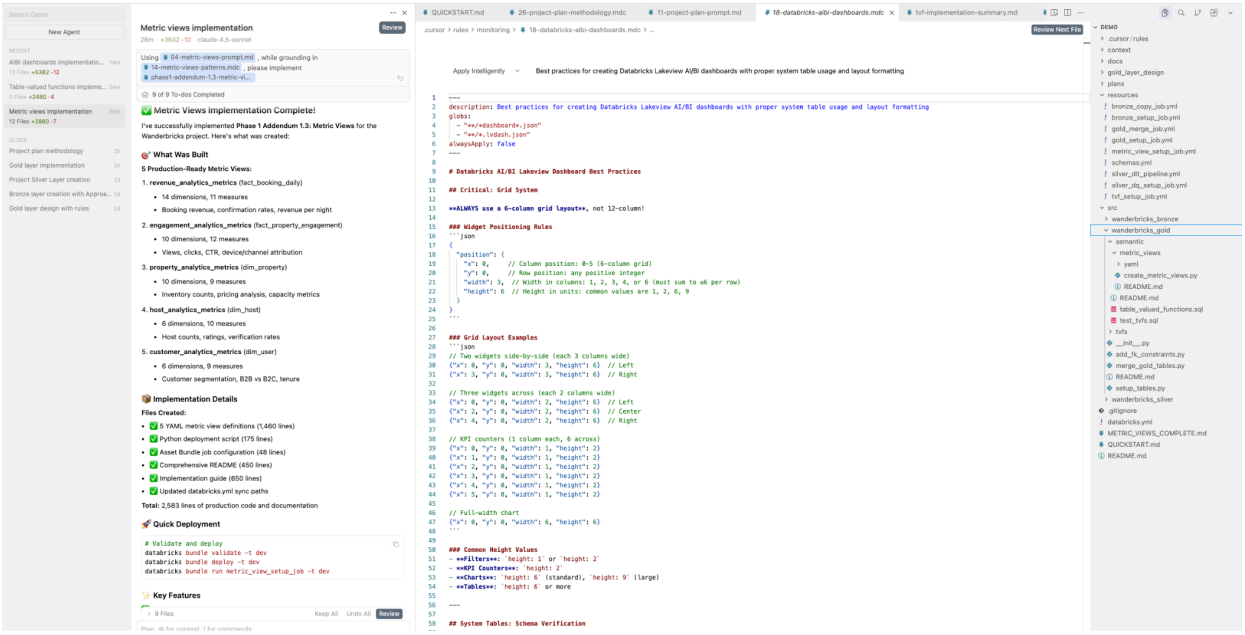


Figure 17: Authoring Metric Views using a prompt-driven workflow, where LLM-generated semantic definitions (measures, dimensions, metadata) are validated and refined before registration in Unity Catalog.

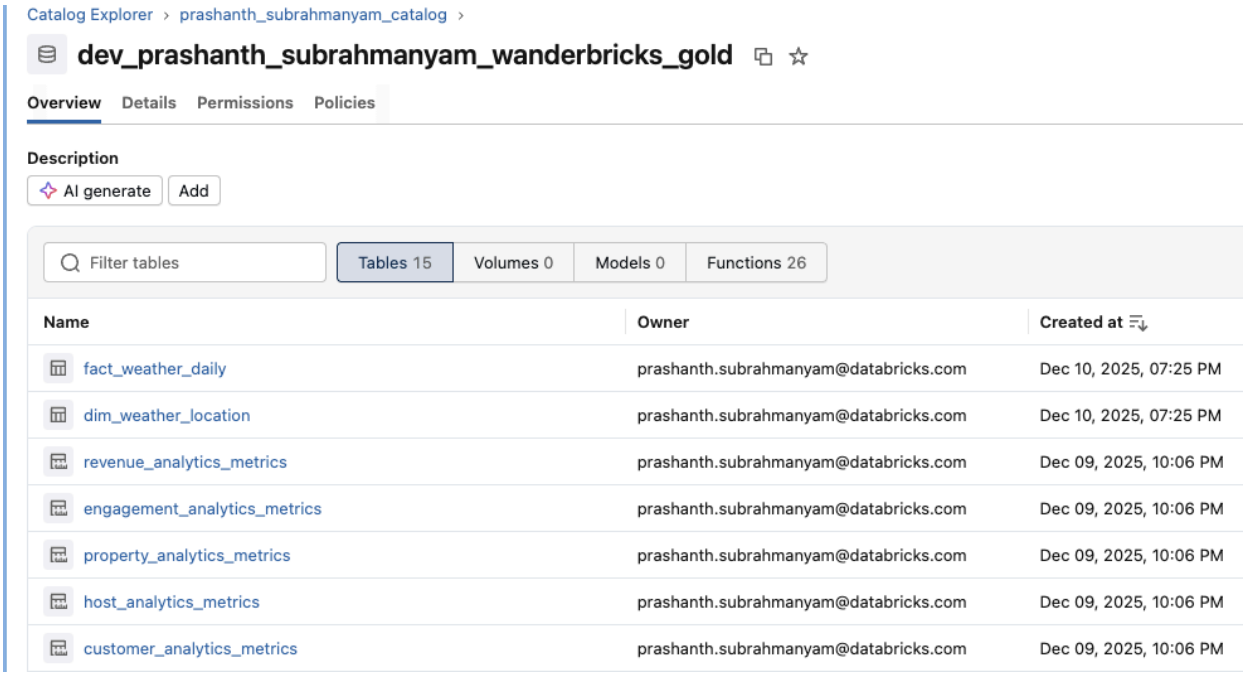


Figure 18: *Metric Views registered on top of Gold to standardize KPI definitions and ensure consistent analytics consumption.*

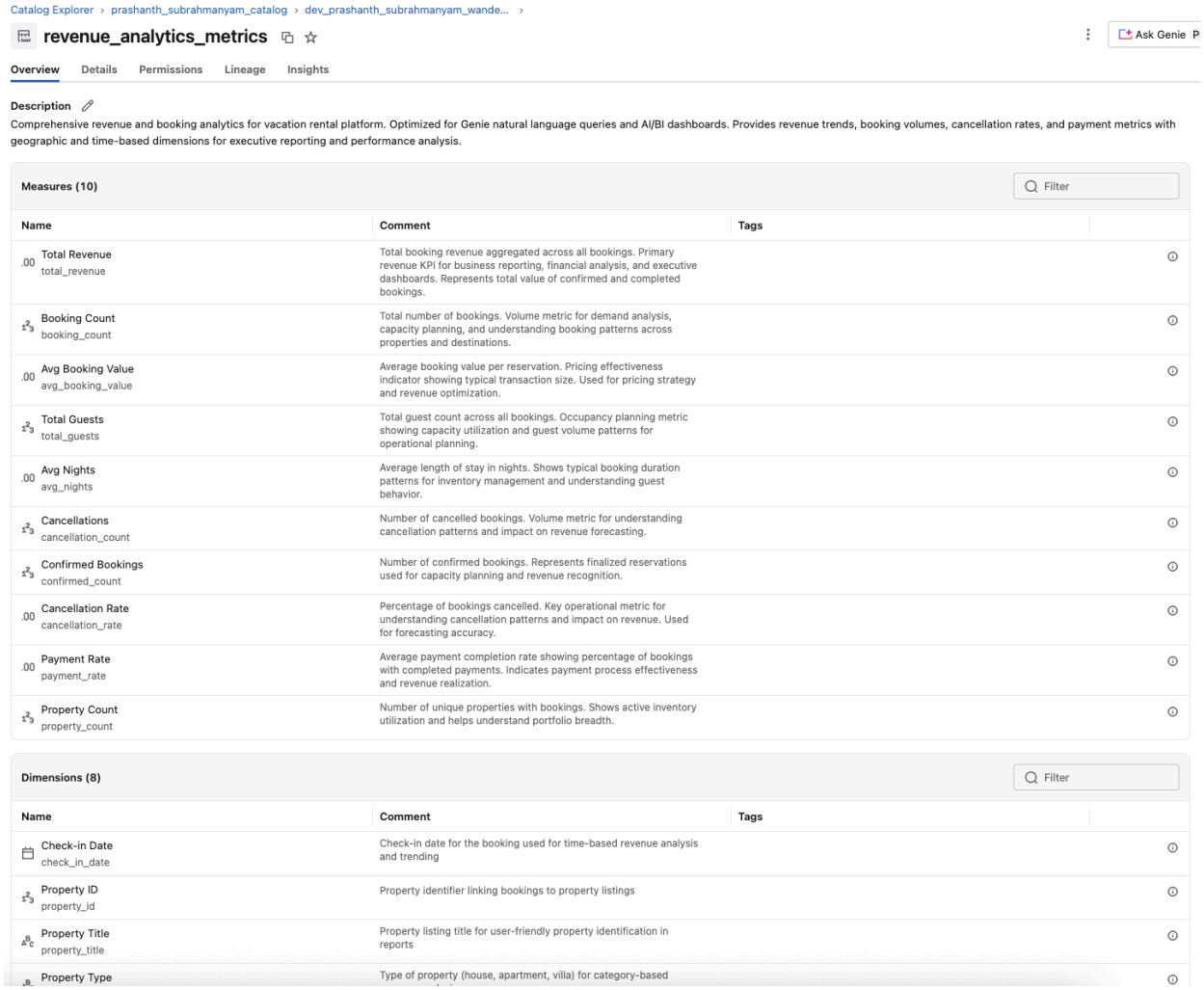


Figure 19: *Metric View semantic definition (measures, dimensions, descriptions/metadata) used to drive governed, repeatable BI semantics.*

Genie Spaces

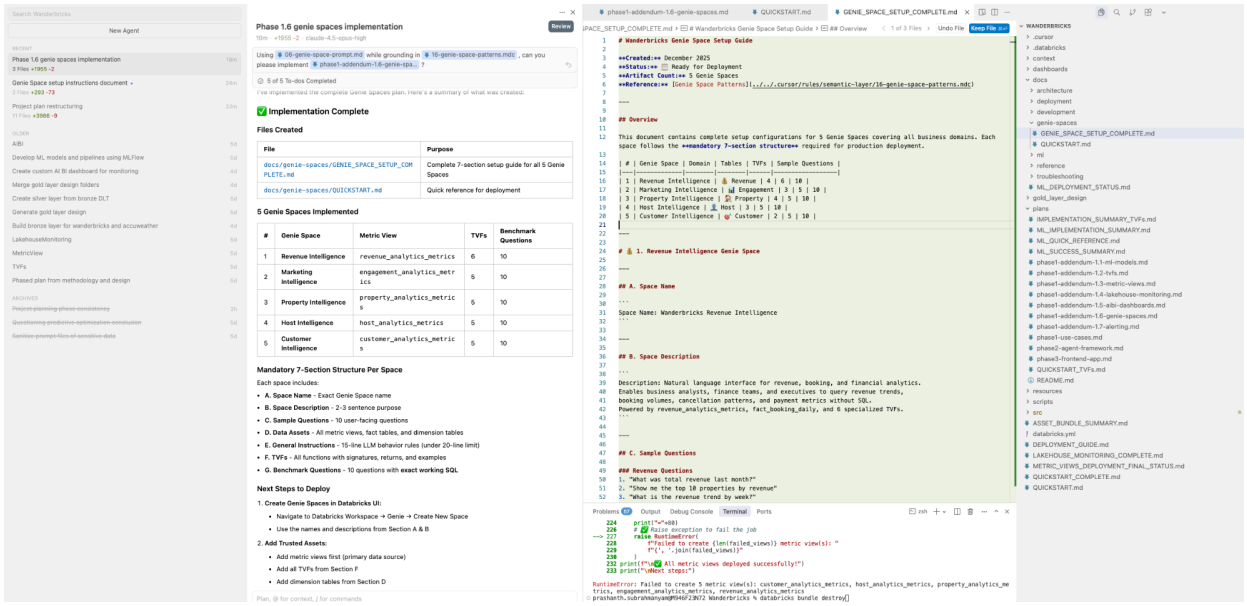


Figure 20: *Genie Spaces implementation prompt, showing how the framework generates a document that can guide you through everything needed to set-up a Genie Space in few minutes, constrained to approved semantic artifacts (TVFs + Metric Views).*

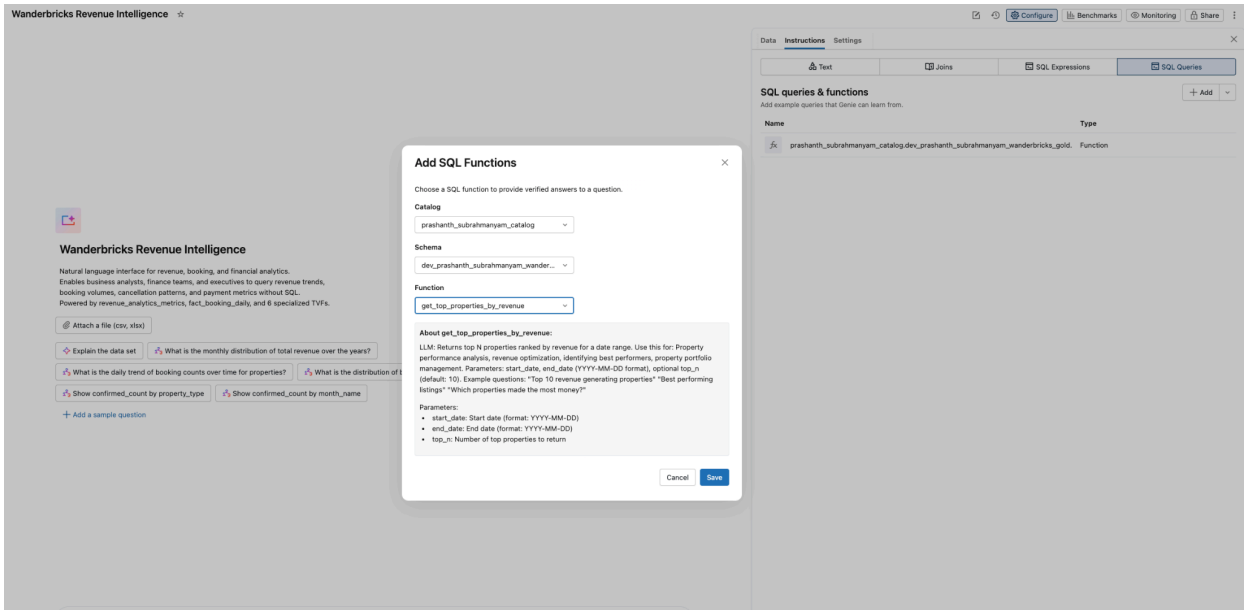


Figure 21: Genie Spaces implementation workflow, showing how approved semantic artifacts like Table-Valued Functions can be added for an improved quality genie space.

Step 9: (Optional) Create artifacts to support new use cases

ML Models

Machine learning artifacts in this framework are built directly on top of the curated Gold and Semantic layers, ensuring that features are business-aligned, well-documented, and consistent with analytical use cases.

The framework leverages MLflow for experiment tracking, model versioning, and lifecycle management, and Unity Catalog for governance and access control. Models are treated as first-class assets alongside tables, views, and functions, enabling discoverability, auditability, and controlled deployment.

The following prompt will create the necessary feature stores, training pipelines as well as batch inference jobs for ML. Further customization for real-time model serving can be done, but this has not been implemented in this example.

None

```
Set up the ML pipeline using
@data_product_accelerator/skills/ml/00-ml-pipeline-setup/SKILL.md
```

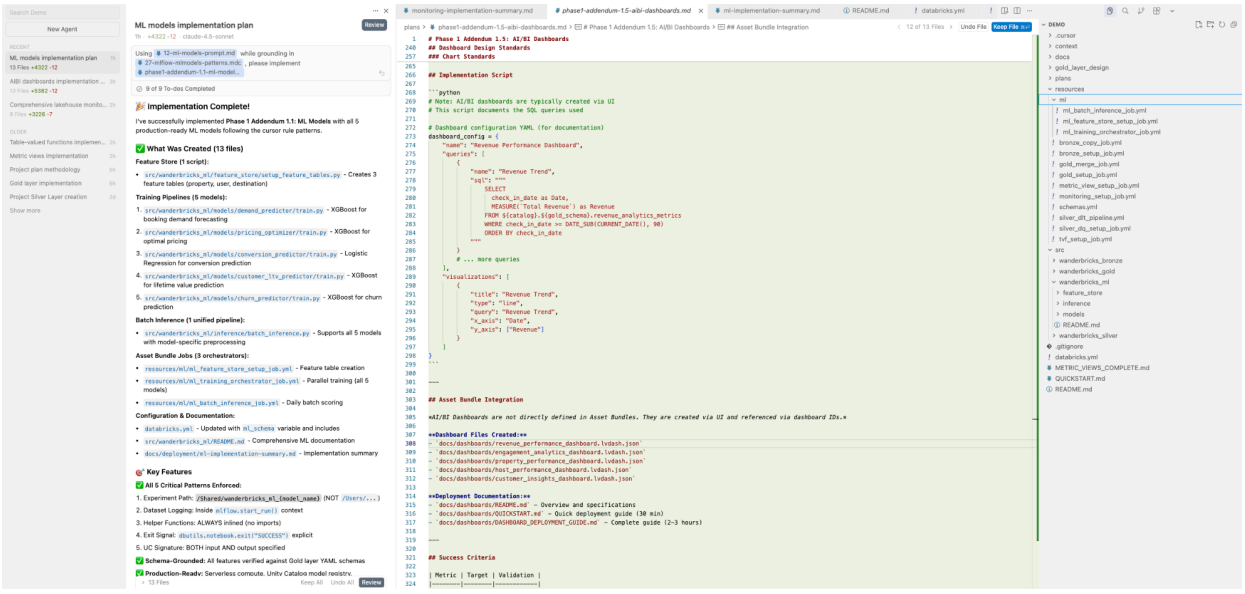


Figure 22: *End-to-end ML model development workflow, combining LLM-assisted code generation, MLflow experiment tracking, and model registration as governed assets in Unity Catalog.*

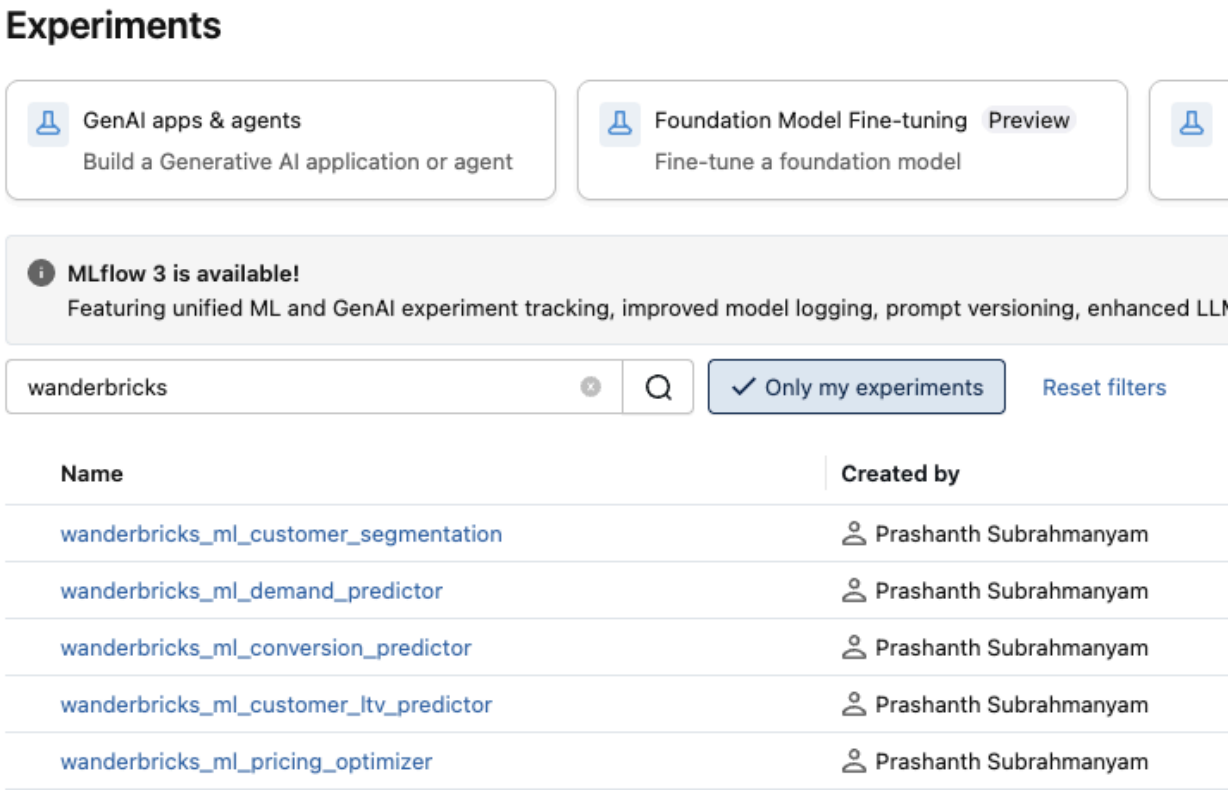


Figure 23: *MLflow experiments for the Wanderbricks use cases, tracking training runs and enabling reproducibility across ML workflows.*

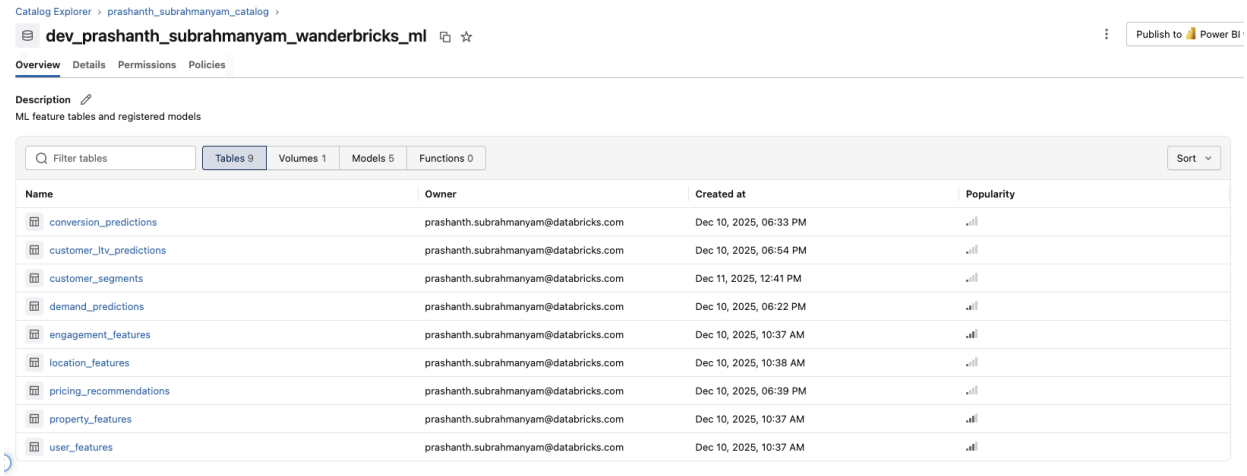


Figure 24: *Unity Catalog schema that organizes ML-related tables/assets (feature-like tables and supporting datasets) alongside governed ownership metadata.*

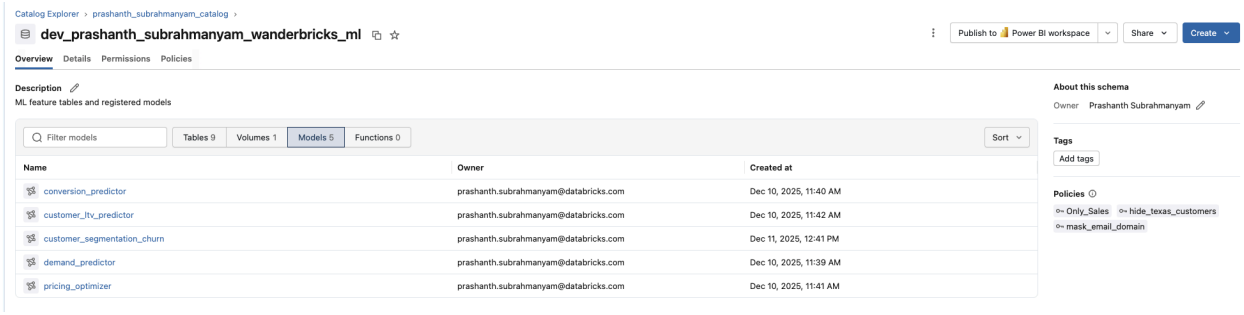


Figure 25: *Published/registered models visible in Unity Catalog, enabling governed discovery and consistent deployment practices.*

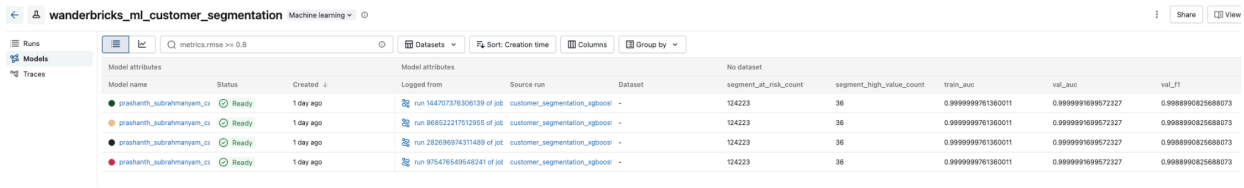


Figure 26: Operational monitoring view capturing run history and evaluation-style metrics, supporting observability for ML/data-product behavior over time.

Step 10: (Optional) Create monitoring and observational artifacts

Set up Lakehouse Monitors, AI/BI Dashboards, and SQL Alerts.

- Lakehouse Monitors with custom business metrics
- AI/BI Dashboard JSON definitions
- Asset Bundle deployment

```
None
Set up observability using
@data_product_accelerator/skills/monitoring/00-observability-setup/SKILL
.md
```

Lakehouse Monitoring

Lakehouse Monitoring closes the loop between data ingestion, transformation, and consumption by providing continuous visibility into time-series profiles for normal tables as well as inference tables. Because downstream analytics, BI, and ML workloads depend on stable and trustworthy data, observability is treated as a core requirement rather than an afterthought.

In this framework, monitoring artifacts are generated declaratively and integrated directly with Gold layers, allowing teams to detect issues early and maintain confidence in the data product over time. The lakehouse monitoring includes Custom Business Metrics that are generated by the LLM and can be used to monitor data slices.

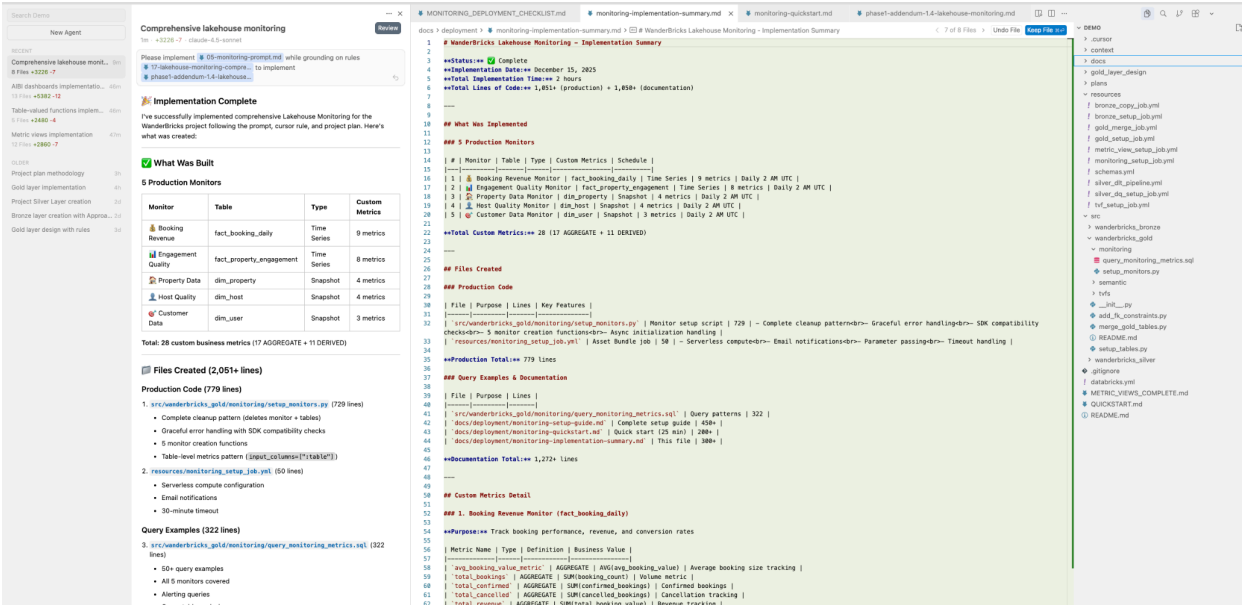


Figure 27: *Lakehouse Monitoring implementation prompt and workflow, generating monitoring assets to track data quality, freshness, volume, and distribution signals for ongoing observability of the data product.*

AI / BI Dashboards

AI / BI Dashboards represent the primary human-facing consumption layer of the data product. These dashboards are intentionally built on top of Metric Views and semantic

artifacts rather than raw tables, ensuring that visualizations remain thin, maintainable, and consistent.

By minimizing embedded business logic in dashboards, teams can evolve metrics and rules centrally without requiring widespread rework across reports.

Databricks AI/BI dashboards are serialized JSON files, and hence can be generated by an LLM. The generated files may have some minor issues (such as referencing right catalog, schema in queries, and choosing right parameters on visuals etc), which can be fixed in the UI, but in general, this can be a tremendous accelerator to dashboard development.

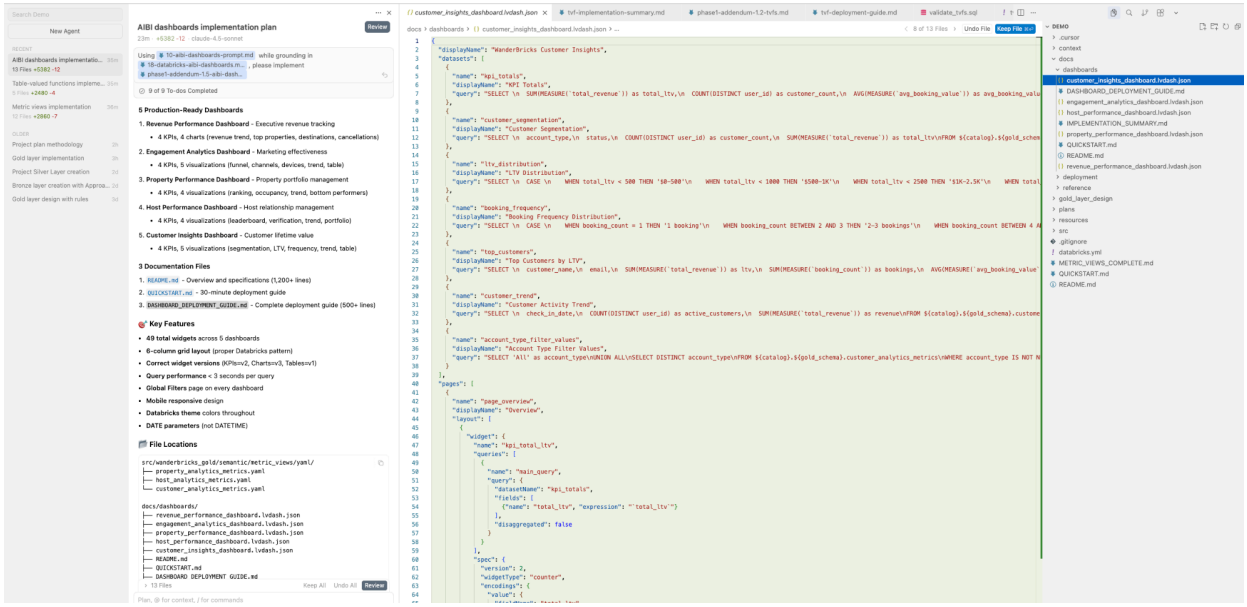


Figure 28: *AI/BI Dashboard generation prompt, showing how dashboards are created on top of governed Metric Views/semantic definitions to keep BI logic centralized and consistent.*

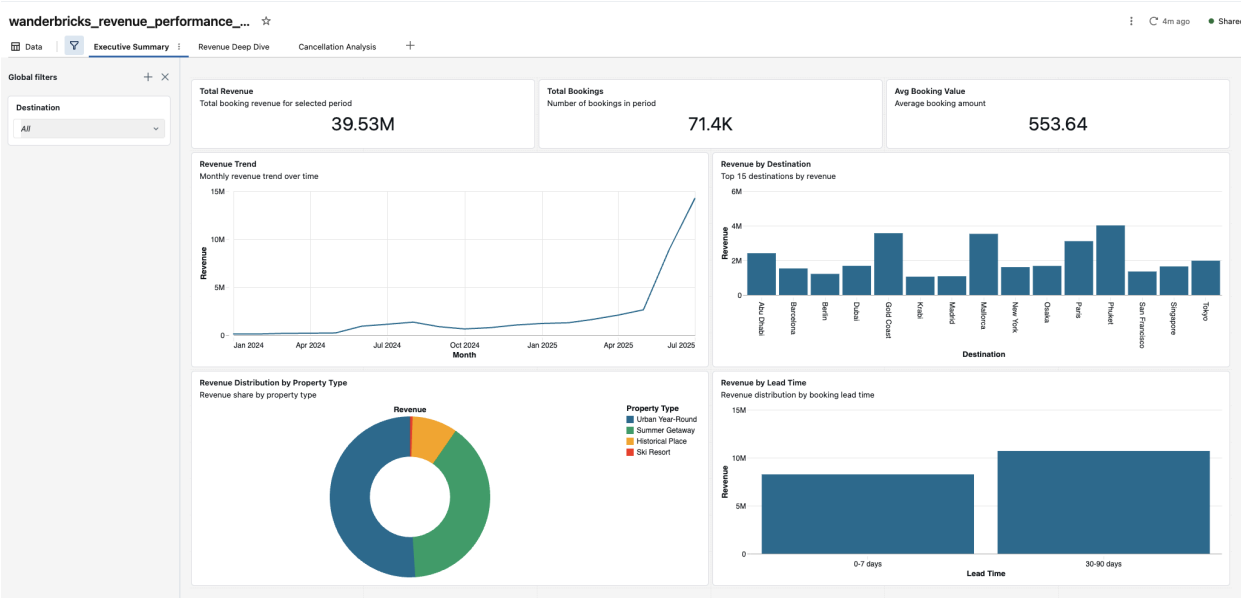


Figure 29: *Executive BI dashboard built on the semantic layer, showcasing business KPIs and trends powered by the Gold model and governed metrics.*