Gulf to Bay Analytics

End-to-End BI Modernization

A Portfolio Case Study by Michael Lloyd

Business Intelligence Developer

Gulf to Bay Analytics — Clearwater, FL

Modernizing the Enterprise Analytics Ecosystem

**From Legacy SQL Server, SSIS, SSAS, and SSRS →
A Unified Microsoft Fabric Lakehouse Architecture**

Technologies & Platforms

- Microsoft Fabric (Lakehouse, Pipelines, Dataflows, Semantic Models)
- Python ETL (Pandas, PySpark)
- Power BI (DAX, Modeling, KPI Framework)
- Power Apps (Operational Interfaces)
- Power Automate (Refresh & Notifications)
- Git-based SDLC (Feature Branching, Repo Hygiene)

Contact

GitHub: github.com/michaelraylloyd
LinkedIn: linkedin.com/in/michael-lloyd-7aa62250
Email: mrlloyd9@gmail.com

Gulf to Bay Analytics — End-to-End BI Modernization

Portfolio Overview — Michael Lloyd

Executive Summary

Gulf to Bay Analytics modernized its legacy Microsoft BI environment into a unified, cloud-native analytics platform built on Microsoft Fabric. The transformation replaced fragmented SSIS/SSAS/SSRS components with a governed Lakehouse architecture, Python-based ETL, standardized semantic modeling, automated refresh pipelines, and a clean SDLC supported by Git-based version control.

This case study outlines the full modernization journey—from legacy assessment to cloud architecture, ETL migration, modeling, reporting, automation, and engineering standards.

---

1. Introduction

This modernization initiative was designed to eliminate technical debt, improve data reliability, and establish a scalable analytics foundation. The new platform supports governed data ingestion, standardized transformations, automated workflows, and consistent KPI reporting across the organization.

---

2. Legacy Environment Overview

The original Gulf to Bay Analytics environment was built on a traditional Microsoft BI stack with limited automation, fragmented logic, and no centralized governance. While functional, the system had accumulated technical debt over time, making enhancements slow and operational reliability inconsistent.

Key Characteristics

- SQL Server as the primary data store
- SSIS packages orchestrating nightly ETL with tightly coupled logic
- SSAS Tabular models requiring manual refresh
- SSRS reports with independent datasets and duplicated transformations
- Manual refresh cycles and inconsistent data lineage

- No version control, SDLC, or deployment standards

This landscape created operational friction, inconsistent reporting, and limited scalability. These constraints directly informed the modernization strategy and established the need for a unified, cloud-aligned analytics platform.

---

3. Modernization Goals

The modernization effort focused on establishing a scalable, governed, cloud-native analytics foundation that could replace fragmented legacy components with a unified Microsoft Fabric ecosystem. These objectives shaped every architectural and engineering decision throughout the project.

Core Objectives

- Decouple business logic from SSIS and migrate ETL to Python and Fabric
- Implement a Lakehouse-based medallion architecture (Bronze → Silver → Gold)
- Rebuild semantic models for consistency, performance, and governance
- Automate refreshes and operational workflows with Power Automate
- Standardize documentation, lineage, and metadata across the platform
- Establish Git-based version control and a clean SDLC with feature branching
- Improve discoverability, maintainability, and long-term scalability

These goals provided the blueprint for the modernized architecture described in the next section.

---

4. Architecture Overview

The modernized Gulf to Bay Analytics platform is built on a unified Microsoft Fabric ecosystem that replaces fragmented on-prem components with a clean, cloud-native architecture. The solution integrates ingestion, transformation, modeling, reporting, and operational workflows into a single, governed analytics environment.

🏗️ Platform Components Summary

Data Ingestion

- Python notebooks for structured ingestion into Bronze
- Fabric Dataflows for lightweight ingestion and CSV processing
- Metadata-driven patterns for repeatable ingestion

Lakehouse & Medallion Architecture
- Bronze → raw ingestion
- Silver → standardization, cleansing, and DQ enforcement
- Gold → star-schema modeling for analytics and reporting
- SQL views for semantic alignment

ETL / ELT Processing
- Python notebooks for transformation and synchronization
- Fabric Pipelines for orchestration and scheduling
- Integrated Data Quality subsystem

Semantic Modeling
- Fabric Lakehouse semantic layer
- Power BI semantic model with standardized DAX
- Conformed dimensions and fact tables

Reporting & Analytics
- Power BI dashboards and KPI models
- Drill-through, detail pages, and consistent branding
- Automated refresh and monitoring

Operational Applications
- Power Apps KPI Explorer for business workflows
- Embedded Power BI visuals
- Role-based access patterns

Automation & Orchestration
- Power Automate for dataset refreshes and notifications
- Pipeline-driven execution of notebooks and DQ rules

SDLC & Governance
- Git-based version control with feature branching
- Dev/Main/Prod separation
- Branch protection rules and structured commit history

- Folder-level documentation and repo hygiene standards

---

5. Lakehouse Architecture (Bronze → Silver → Gold)

The Lakehouse serves as the core of the modernized analytics platform, replacing fragmented on-prem data stores with a unified, cloud-native medallion architecture. This structure enables clean lineage, modular transformations, and scalable analytics across the entire ecosystem.

🏛️ Lakehouse Architecture Overview

Bronze — Raw Ingestion

- Direct ingestion from Python notebooks and Dataflows
- Minimal transformation to preserve source fidelity
- Standardized metadata and ingestion patterns

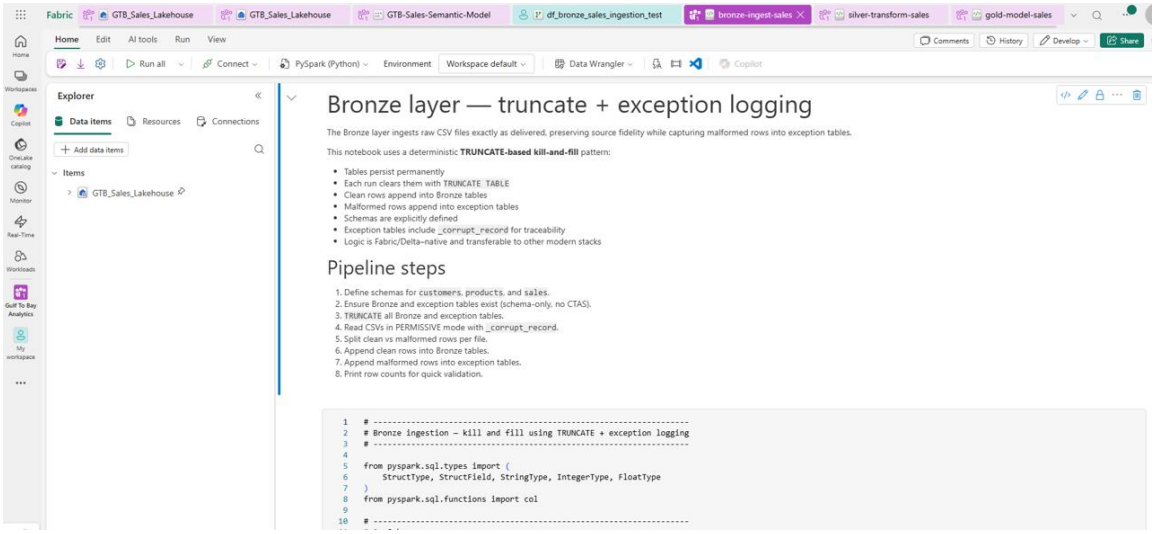
Silver — Standardization & Data Quality

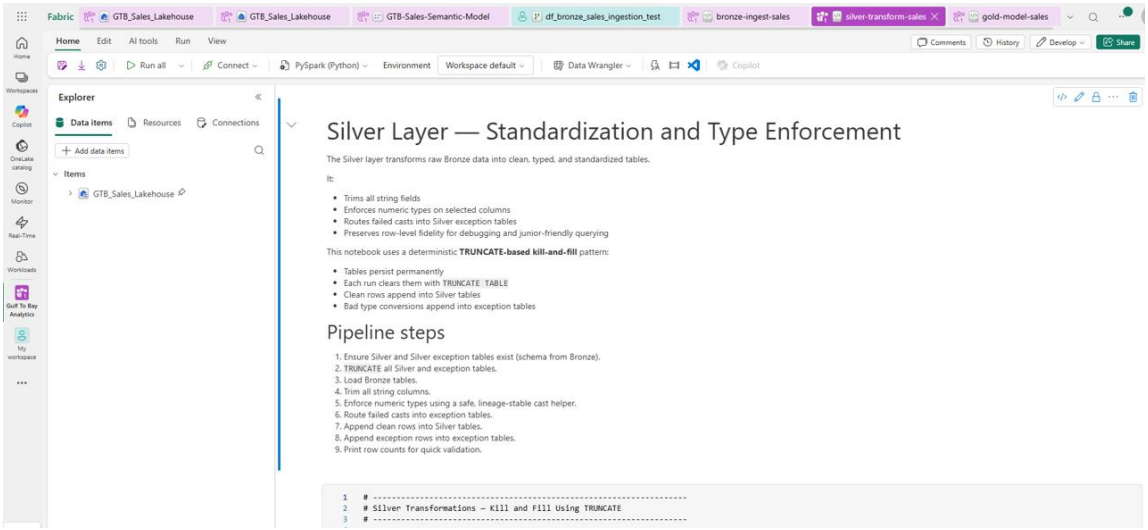
- Cleansing, normalization, and type enforcement
- Execution of Data Quality (DQ) rules
- Conformed structures ready for modeling
- Integration with metadata-driven transformation logic

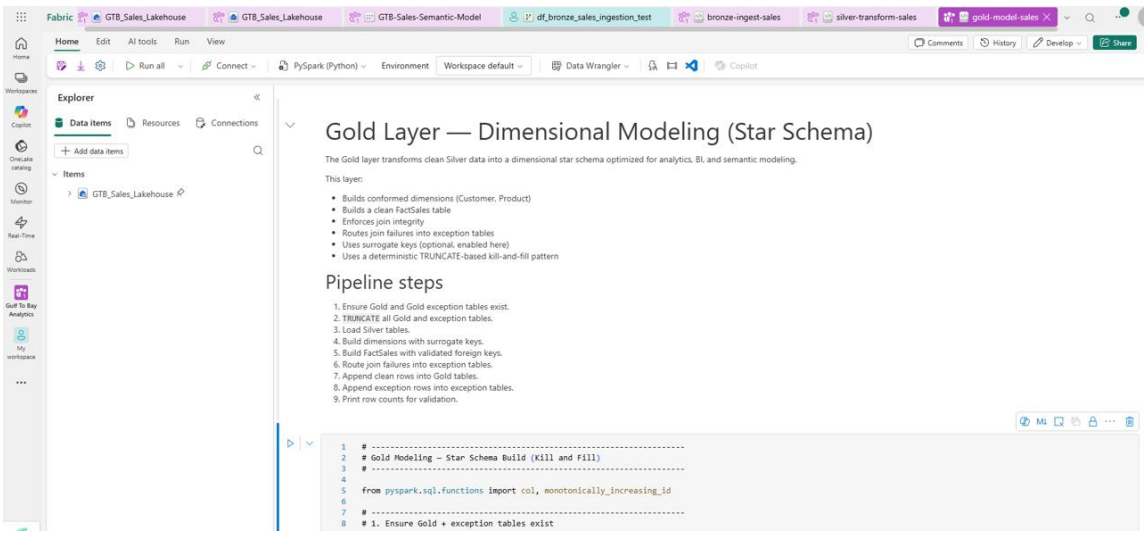
Gold — Star Schema Modeling

- Fact and dimension tables aligned to business processes
- Surrogate key logic and conformed dimensions
- Optimized for semantic modeling and Power BI reporting
- SQL views for consistent downstream consumption

Lakehouse Notebooks







6. Modern ETL/ELT Architecture (Python, Dataflows, Pipelines)

The ETL layer was fully modernized from tightly coupled SSIS packages into a modular, cloud-native architecture built on Python notebooks, Fabric Dataflows, and Fabric Pipelines.

🔄 Evolution of the ETL Layer

Legacy SSIS

- Embedded business logic
- Limited parameterization
- Manual refresh cycles

Azure Data Factory (Transition Phase)

- Parameterized pipelines
- Early cloud orchestration

Fabric Lakehouse (Final Architecture)

- Python notebooks for ingestion and transformation
- Dataflows for lightweight ingestion
- Fabric Pipelines for orchestration
- Integrated Data Quality subsystem

ETL Components (Fabric Python Notebook)

Gold Layer — Dimensional Modeling (Star Schema)

The Gold layer transforms clean Silver data into a dimensional star schema optimized for analytics, BI, and semantic modeling.

This layer:

- Builds conformed dimensions (Customer, Product)
- Builds a clean FactSales table
- Enforces join integrity
- Routes join failures into exception tables
- Uses surrogate keys (optional, enabled here)
- Uses a deterministic TRUNCATE-based kill-and-fill pattern

Pipeline steps

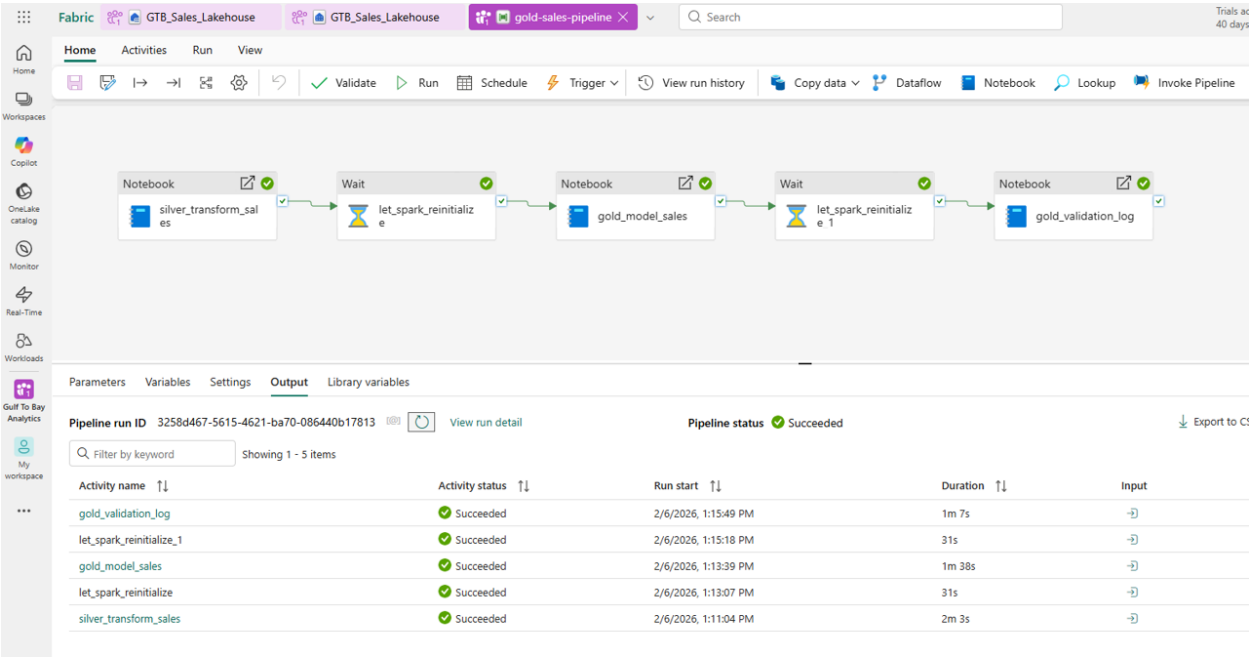
1. Ensure Gold and Gold exception tables exist.
2. `TRUNCATE` all Gold and exception tables.
3. Load Silver tables.
4. Build dimensions with surrogate keys.
5. Build FactSales with validated foreign keys.
6. Route join failures into exception tables.
7. Append clean rows into Gold tables.
8. Append exception rows into exception tables.
9. Print row counts for validation.

```
130  # ------------------------------------------------------------------
131  # 5. Build FactSales with validated foreign keys
132  # ------------------------------------------------------------------
133
134  # Join to dimensions
135  fact_joined = (
136      silver_sales.alias("s")
137          .join(dim_customer_df.alias("c"), col("s.customer_id") == col("c.customer_id"), "left")
138          .join(dim_product_df.alias("p"), col("s.product_id") == col("p.product_id"), "left")
139  )
140
141  # Identify join failures
142  exceptions_fact_sales_df = fact_joined.filter(
143      col("c.customer_sk").isNull() | col("p.product_sk").isNull()
144  ).select(
145      "s.order_id",
146      "s.order_date",
147      "s.customer_id",
148      "s.product_id",
149      "s.quantity",
150      "s.unit_price",
151      "s.discount",
152      "s.line_total",
153      (
154          col("c.customer_sk").isNull().cast("string")
155          .alias("reason")
156      )
157  )
158
159  # Keep only successful rows
160  fact_sales_df = fact_joined.filter(
161      col("c.customer_sk").isNotNull() & col("p.product_sk").isNotNull()
162  ).withColumn(
163      "sales_sk", monotonically_increasing_id()
```

Job 92	count at NativeMethodAccessorImpl.java:0	Succeeded	1/1	4/4 succeeded	87 ms	18	5.03 KB	3.77 KB
Job 91	$anonfun$recordDeltaOperationInternal$1 at SynapseLoggingShim.scala:111	Succeeded	1/1	4/4 succeeded	47 ms	12	6.73 KB	0 B
Job 90	$anonfun$submit$1 at FutureTask.java:264	Succeeded	1/1	1/1 succeeded	78 ms	8	886 B	0 B
Job 89	$anonfun$submit$1 at FutureTask.java:264	Succeeded	1/1	1/1 succeeded	32 ms	18	19.19 KB	0 B
Job 88	$anonfun$submit$1 at FutureTask.java:264	Succeeded	1/1	8/8 succeeded	175 ms	8	0 B	886 B
Job 87	$anonfun$submit$1 at FutureTask.java:264	Succeeded	1/1	2/2 succeeded	151 ms	18	0 B	19.19 KB
Job 86	insertInto at NativeMethodAccessorImpl.java:0	Succeeded	1/1	2/2 succeeded	376 ms	182242	326.05 KB	319.81 KB
Job 85	$anonfun$withThreadLocalCaptured$1 at FutureTask.java:264	Succeeded	1/1	1/1 succeeded	77 ms	18507	0 B	0 B
Job 84	$anonfun$withThreadLocalCaptured$1 at FutureTask.java:264	Succeeded	1/1	1/1 succeeded	74 ms	503	0 B	0 B

```
dim_customer 18507
dim_product 503
fact_sales 60451
exceptions_fact_sales 60926
```

ETL Components (Fabric Pipeline)



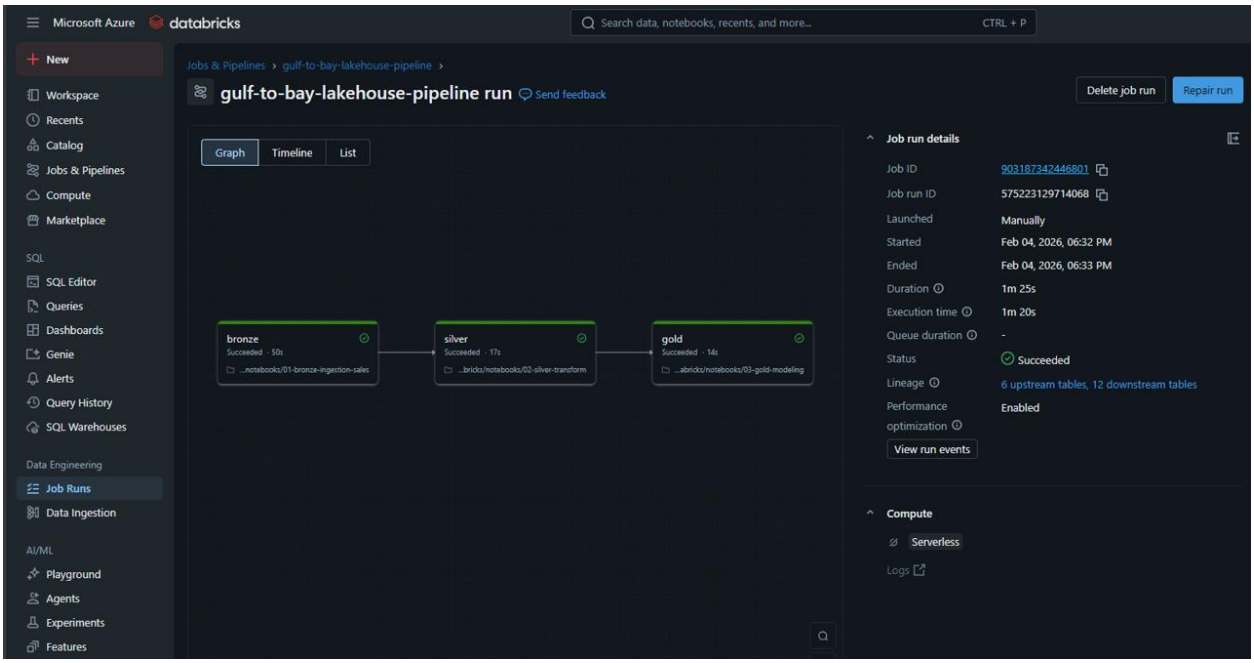
6A. Databricks Lakehouse & Medallion Pipeline

In addition to the Fabric Lakehouse implementation, a full Databricks medallion pipeline was developed using PySpark notebooks and Databricks Jobs. This parallel architecture demonstrates cloud-native engineering patterns, Spark-based transformations, and scalable orchestration aligned with enterprise data engineering standards.

Databricks Highlights

- PySpark notebooks for Bronze, Silver, and Gold layers
- Delta Lake storage with ACID transactions
- Databricks Jobs orchestrating medallion flow
- Serverless compute with autoscaling
- Lineage tracking across upstream and downstream tables
- Parameterized notebook execution
- Version-controlled notebooks in Git

🔥 Databricks Job Orchestration



🔥 Notebook Structure

- **01-bronze-ingestion-sales**
- **02-silver-standardization-sales**
- **03-gold-modeling-sales**

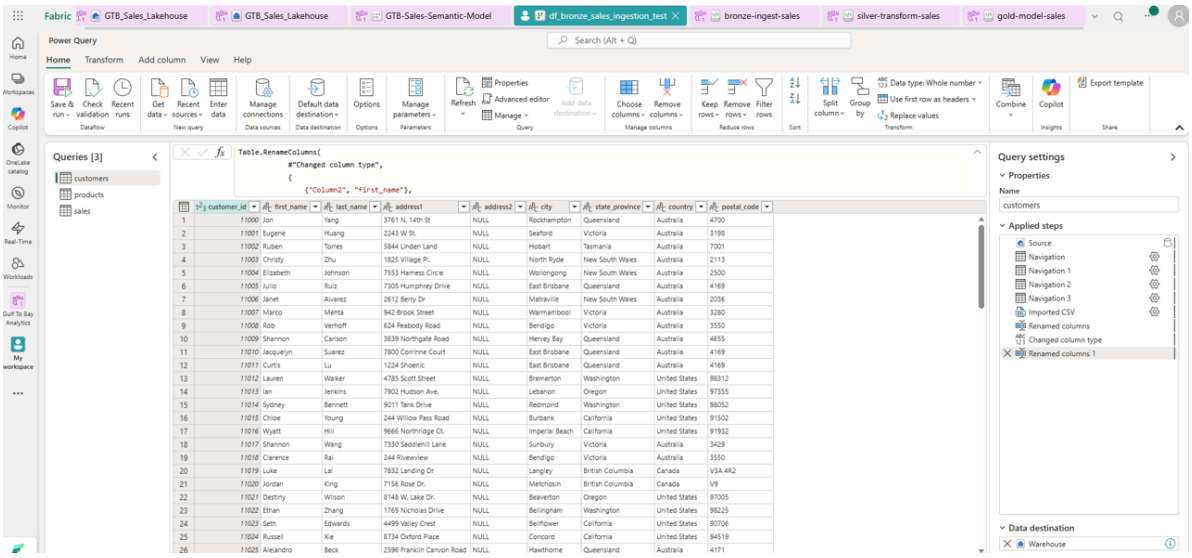
Each notebook applies the same medallion logic used in Fabric, demonstrating portability of engineering patterns across platforms.

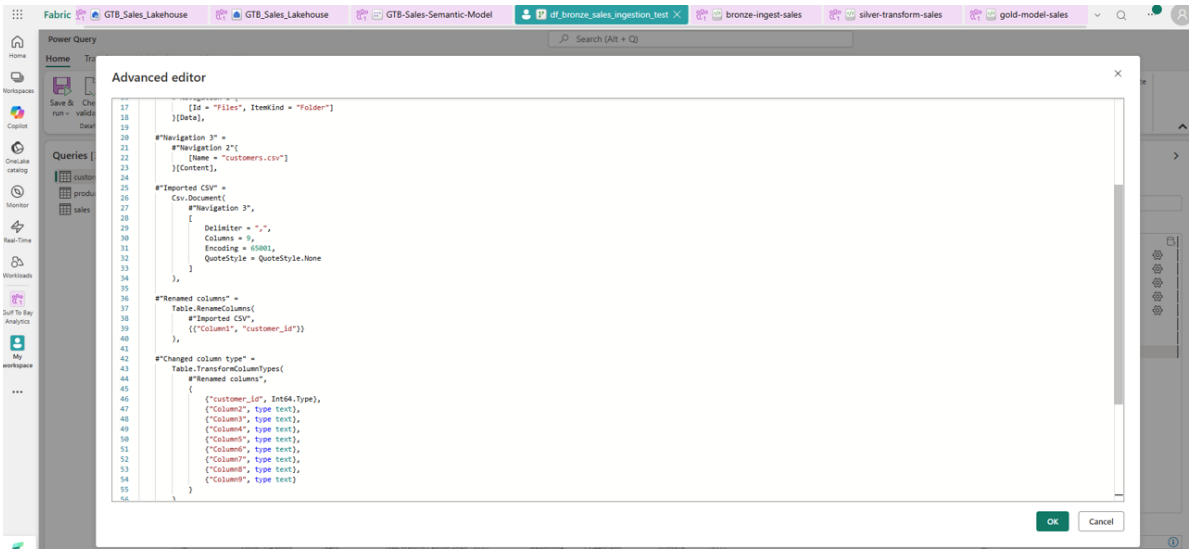
---

7. Data Modeling & Transformation Layer

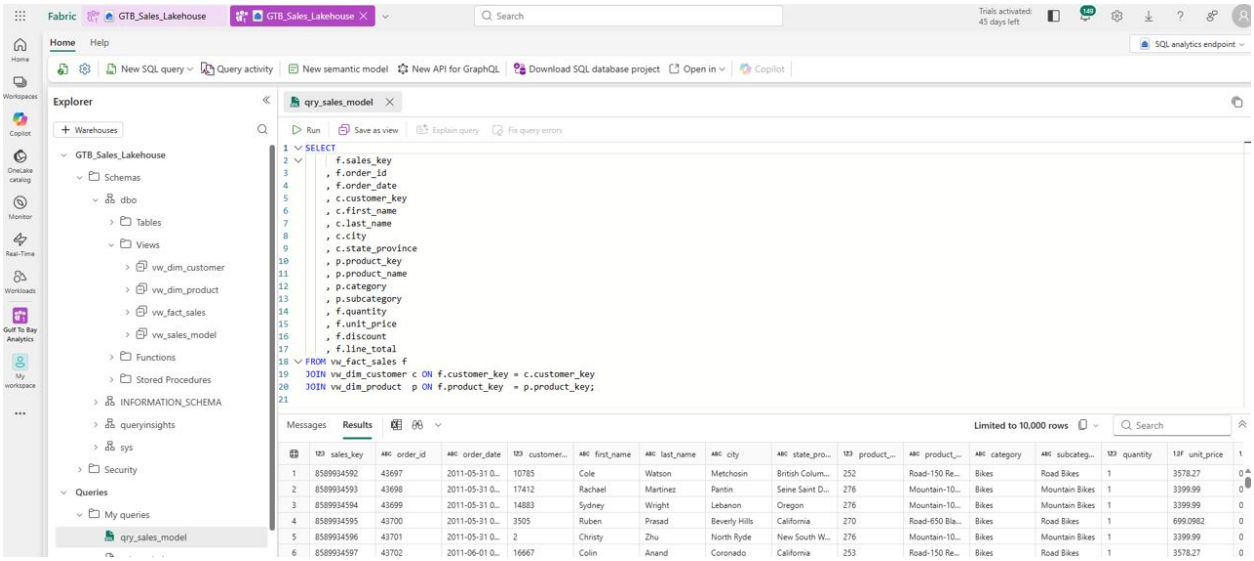
(Dataflows, SQL Views, Semantic Models)

Dataflows & M Transformations

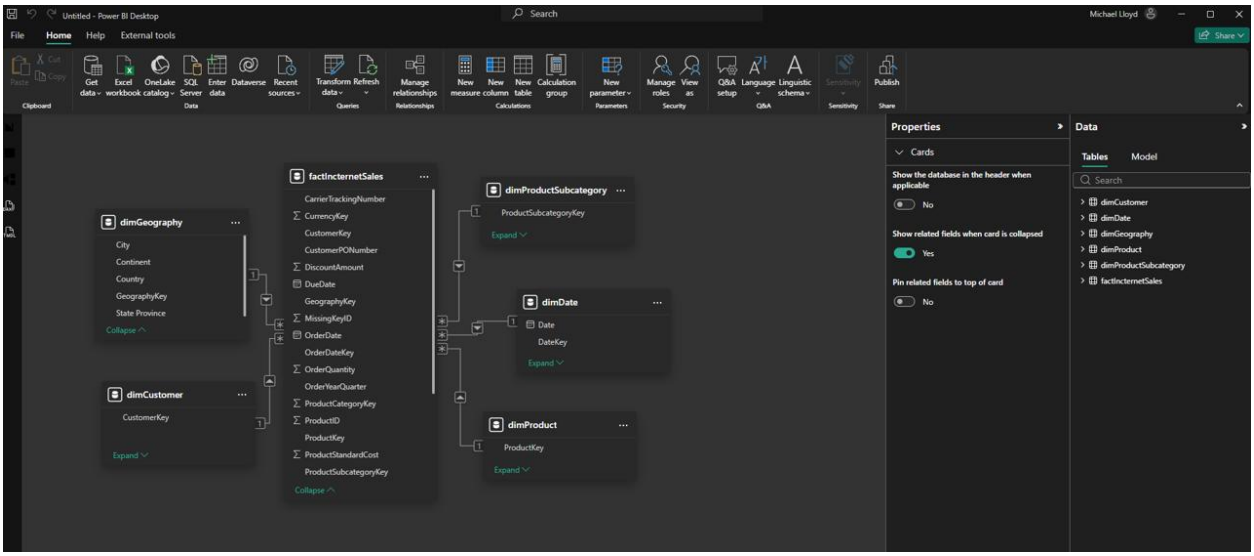




🪵 Lakehouse SQL Views & Metadata Alignment



📊 Semantic Modeling (Fabric Lakehouse → Power BI)



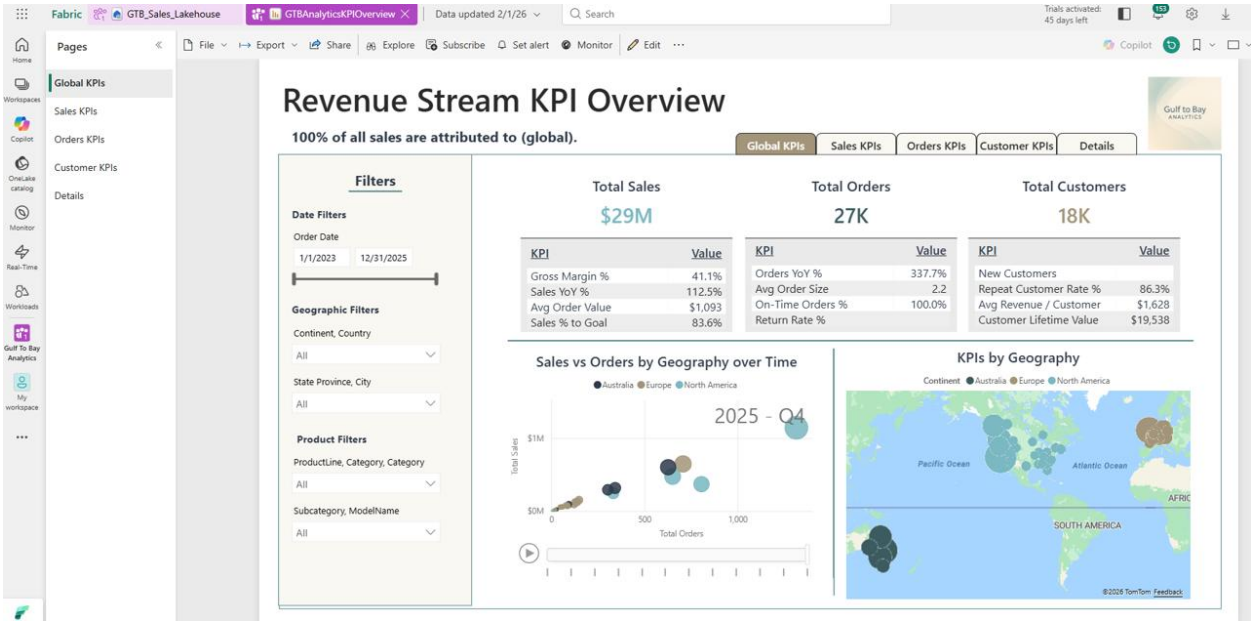
8. Reporting & Dashboards

📈 Reporting Highlights

- Rebuilt KPI model
- Standardized DAX
- Drill-through pages
- Consistent branding
- Automated refresh

📈 KPI Overview (Public)

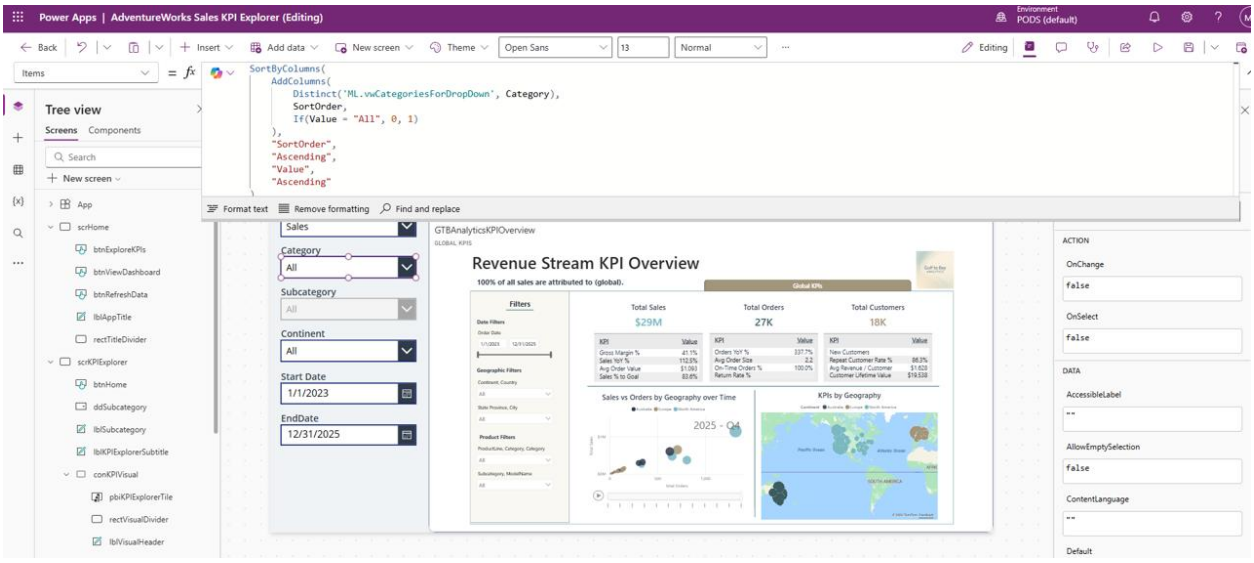
https://app.powerbi.com/view?r=eyJrIjoiMzcyYTIzN2EtYzBjNi00MmY5LWJhY2UtZDk5MDkyZTYwNDExIiwidCI6ImE0MzI2YTU4LWY3ZDktNDQ0ZC1iM2FhLWIwOTAyN2U1ZTg2NiIsImMiOjF9



9. Power Apps — KPI Explorer

Highlights

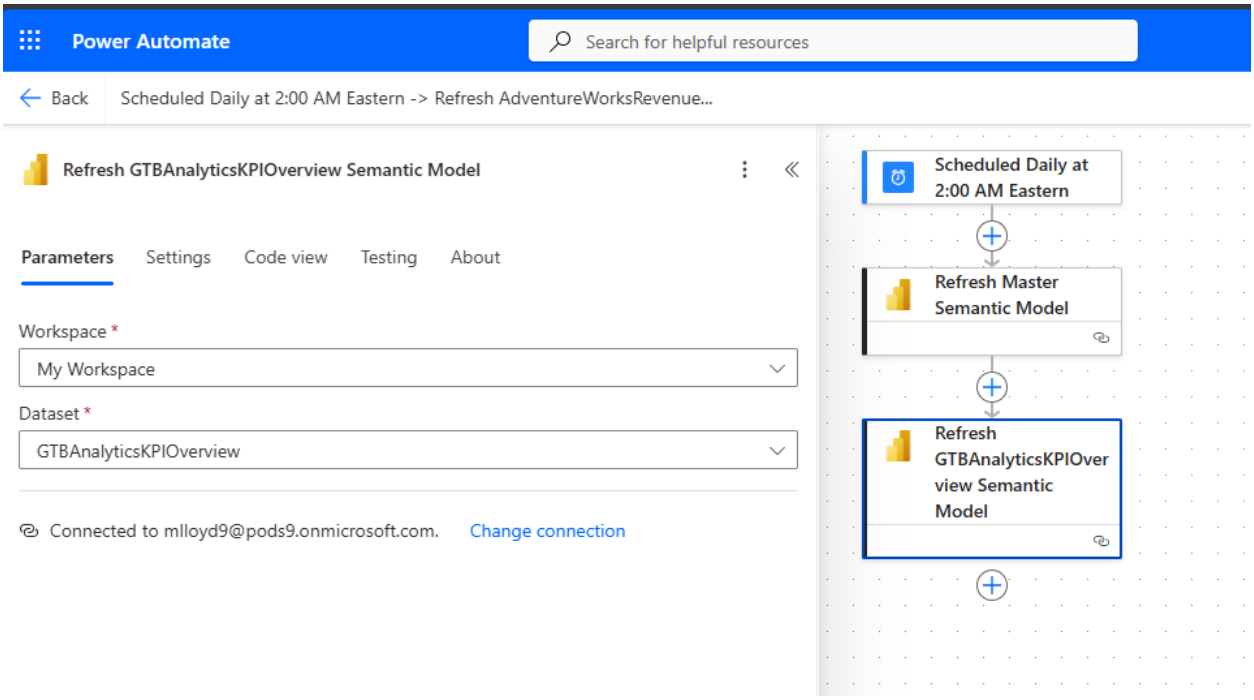
- Drill-down navigation
- Embedded Power BI
- Role-based access



10. Power Automate — Refresh & Notifications

🔔 Highlights

- Scheduled dataset refresh
- Failure notifications
- Semantic model triggers



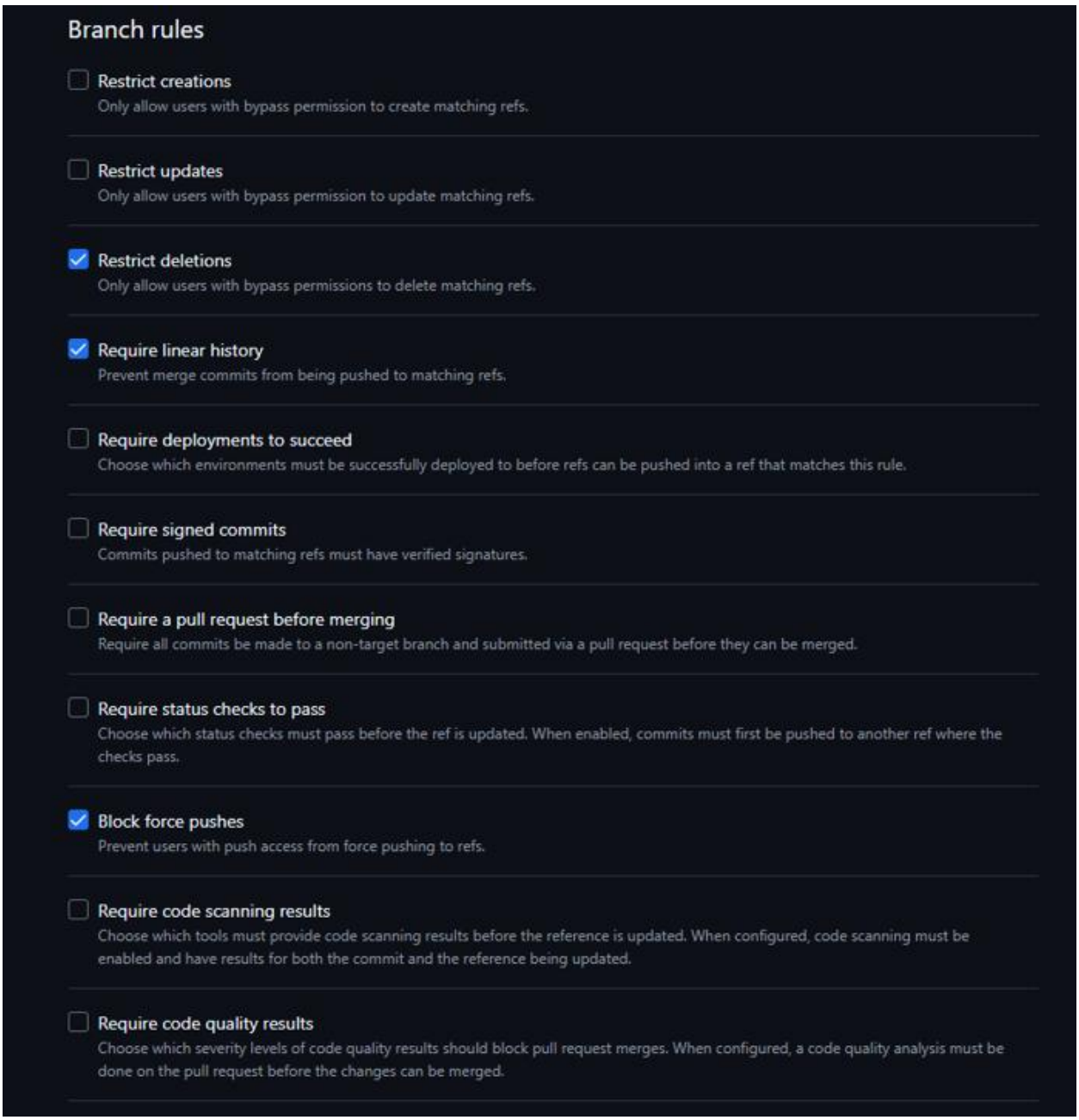
11. SDLC, Version Control & Engineering Standards

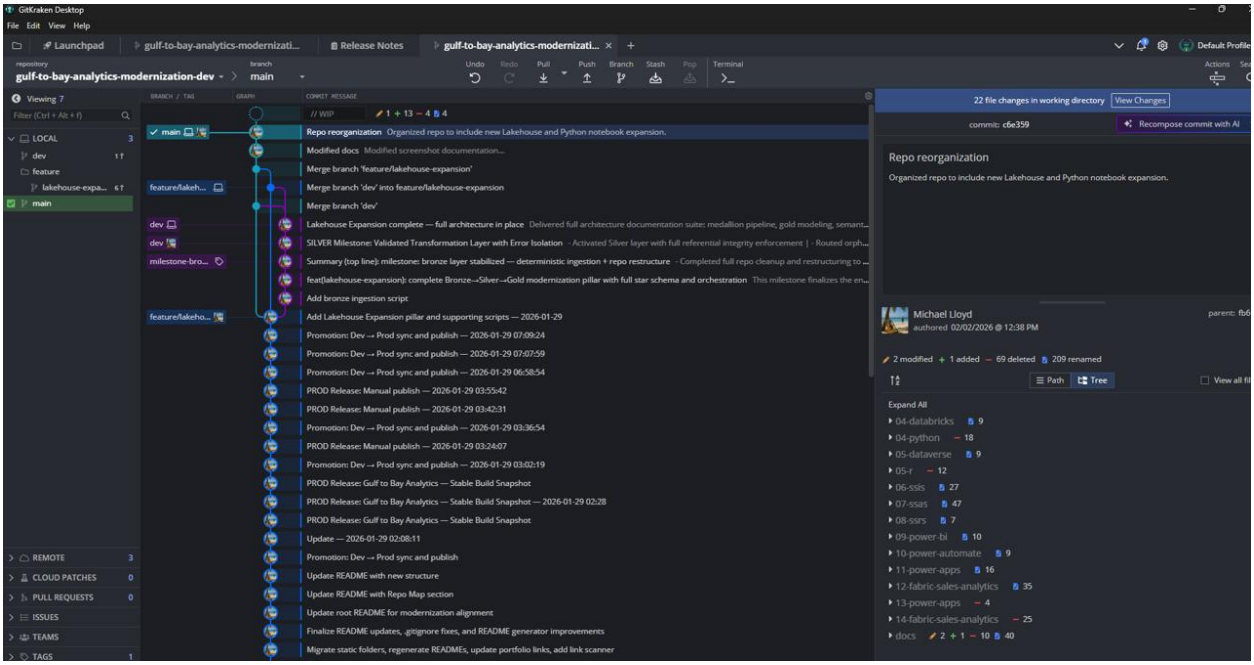
🔧 SDLC Evolution

- No version control → PowerShell checks → GitKraken branching
- Feature Branch → Dev → Main → Manual PROD publish

🔧 Repo Hygiene

- Modular READMEs
- PowerShell utilities
- Branch protection rules





12. Results & Impact

🚀 Platform Impact

- Unified Fabric ecosystem
- Clean lineage
- Automated refresh cycles

⚙️ Engineering Impact

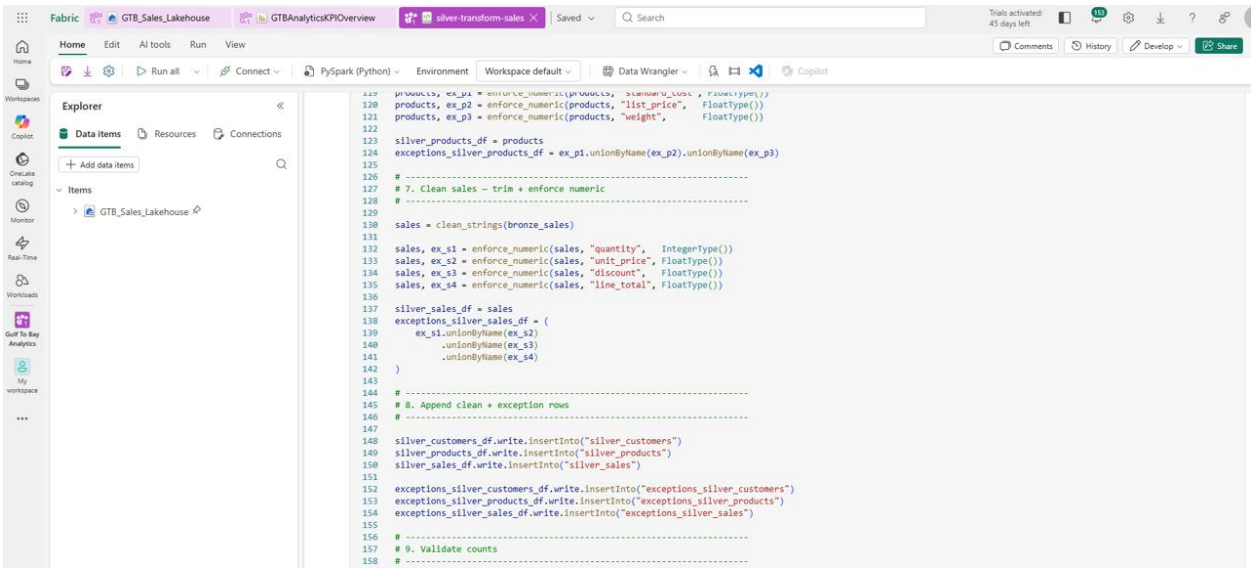
- Git-based SDLC
- Deterministic promotion
- Standardized documentation

📊 Business Impact

- Faster access to KPIs
- Consistent global metrics
- Improved trust in data

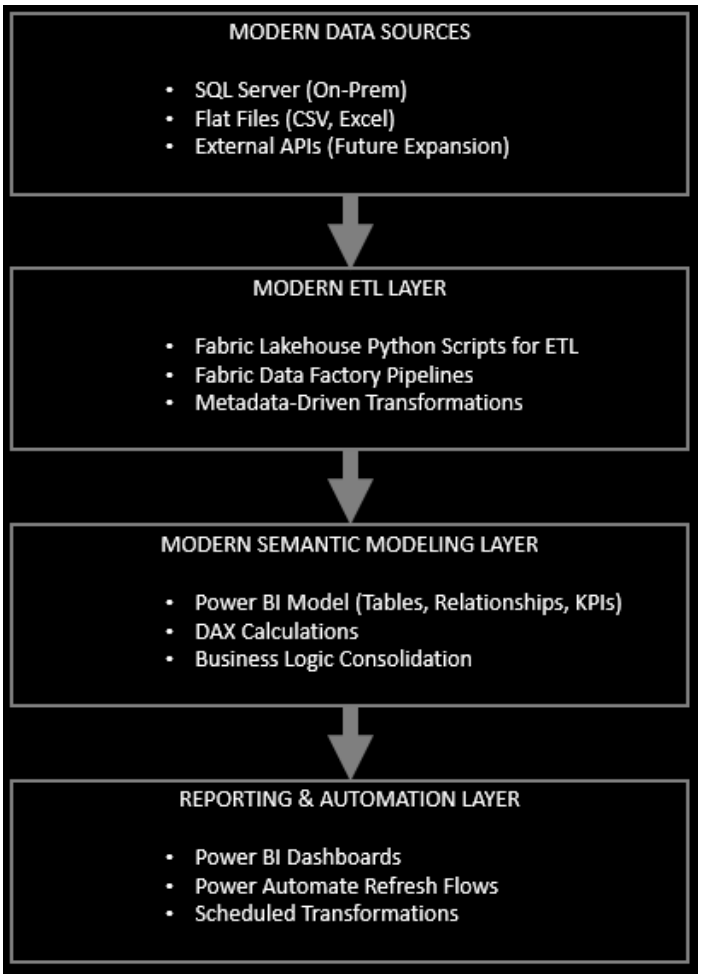
Appendix

Appendix A — Data Quality Subsystem



```
products, ex_p2 = enforce_numeric(products, "list_price",   FloatType())
products, ex_p3 = enforce_numeric(products, "weight",       FloatType())

silver_products_df = products
exceptions_silver_products_df = ex_p1.unionByName(ex_p2).unionByName(ex_p3)

# ------------------------------------------------------------
# 7. Clean sales — trim + enforce numeric
# ------------------------------------------------------------

sales = clean_strings(bronze_sales)

sales, ex_s1 = enforce_numeric(sales, "quantity",   IntegerType())
sales, ex_s2 = enforce_numeric(sales, "unit_price", FloatType())
sales, ex_s3 = enforce_numeric(sales, "discount",   FloatType())
sales, ex_s4 = enforce_numeric(sales, "line_total", FloatType())

silver_sales_df = sales
exceptions_silver_sales_df = (
    ex_s1.unionByName(ex_s2)
         .unionByName(ex_s3)
         .unionByName(ex_s4)
)

# ------------------------------------------------------------
# 8. Append clean + exception rows
# ------------------------------------------------------------

silver_customers_df.write.insertInto("silver_customers")
silver_products_df.write.insertInto("silver_products")
silver_sales_df.write.insertInto("silver_sales")

exceptions_silver_customers_df.write.insertInto("exceptions_silver_customers")
exceptions_silver_products_df.write.insertInto("exceptions_silver_products")
exceptions_silver_sales_df.write.insertInto("exceptions_silver_sales")

# ------------------------------------------------------------
# 9. Validate counts
# ------------------------------------------------------------
```

Appendix B — Additional Architecture Notes



Appendix C — About the Developer

****Michael Lloyd****

Business Intelligence Developer

Gulf to Bay Analytics

Clearwater, FL

Appendix D — Contact Information

- **GitHub:** https://github.com/michaelraylloyd
- **LinkedIn:** https://www.linkedin.com/in/michael-lloyd-7aa62250/
- **Email:** mrlloyd9@gmail.com