

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
-

Gulf to Bay Analytics — Modernization Case Study

- 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

Gulf to Bay Analytics — Modernization Case Study

****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

Gulf to Bay Analytics — Modernization Case Study

- 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

Gulf to Bay Analytics — Modernization Case Study

The image displays three screenshots of Databricks notebooks, each showing the workspace interface with multiple tabs open. The notebooks are titled "Bronze layer — truncate + exception logging", "Silver Layer — Standardization and Type Enforcement", and "Gold Layer — Dimensional Modeling (Star Schema)". Each screenshot shows the notebook content, including a list of pipeline steps and a code editor with Python code snippets.

Bronze layer — truncate + exception logging

This notebook uses a deterministic TRUNCATE-based kill-and-fill pattern:

- Tables persist permanently
- Each run clears them with TRUNCATE TABLE
- Clean rows append into Bronze tables
- Malformed rows append into exception tables
- Schemas are explicitly defined
- Exception tables include _corrupt_record for traceability
- Logic is Fabric/Delta-native and transferable to other modern stacks

Pipeline steps

- Define schemas for customers, products, and sales.
- Ensure Bronze and exception tables exist (schema-only, no CTAS).
- TRUNCATE all Bronze and exception tables.
- Load clean rows into Bronze tables.
- Split clean vs malformed rows via file.
- Append clean rows into Bronze tables.
- Append malformed rows into exception tables.
- Print row counts for quick validation.

```
1 # -----
2 # # Bronze ingestion - kill and fill using TRUNCATE + exception logging
3 # -----
4
5 from pyspark.sql.types import (
6     StructType, StructField, StringType, IntegerType, FloatType
7 )
8 from pyspark.sql.functions import col
9
10 # -----
```

Silver Layer — Standardization and Type Enforcement

The Silver layer transforms raw Bronze data into clean, typed, and standardized tables.

It:

- Trims all string fields
- Enforces numeric types on selected columns
- Routes failed casts into Silver exception tables
- Preserves row-level fidelity for debugging and junior-friendly querying

This notebook uses a deterministic TRUNCATE-based kill-and-fill pattern:

- Tables persist permanently
- Each run clears them with TRUNCATE_TABLE
- Clean rows append into Silver tables
- Bad type conversions append into exception tables

Pipeline steps

- Ensure Silver and Silver exception tables exist (schema from Bronze).
- TRUNCATE all Silver and exception tables.
- Load Bronze tables.
- Trim all string fields.
- Enforce numeric types using a safe, lineage-stable cast helper.
- Route failed casts into exception tables.
- Append clean rows into Silver tables.
- Append exception rows into exception tables.
- Print row counts for quick validation.

```
1 #
2 # # Silver Transformations - Kill and Fill Using TRUNCATE
3 # -----
```

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:

- Buidls confirmed dimensions (Customer, Product)
- Buidls a clean FactData 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

- Ensure Gold and Gold exception tables exist.
- TRUNCATE all Gold and exception tables.
- Load Silver tables.
- Build dimensions with surrogate keys.
- Build FactData table with surrogate keys.
- Route join failures into exception tables.
- Append clean rows into Gold tables.
- Append exception rows into exception tables.
- Print row counts for validation.

```
1 #
2 # # Gold Modelling - Star Schema Build (Kill and Fill)
3 # -----
4
5 from pyspark.sql.functions import col, monotonically_increasing_id
6
7 # -----
8 # 1. Ensure Gold + exception tables exist
```

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

Gulf to Bay Analytics — Modernization Case Study

⌚ 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

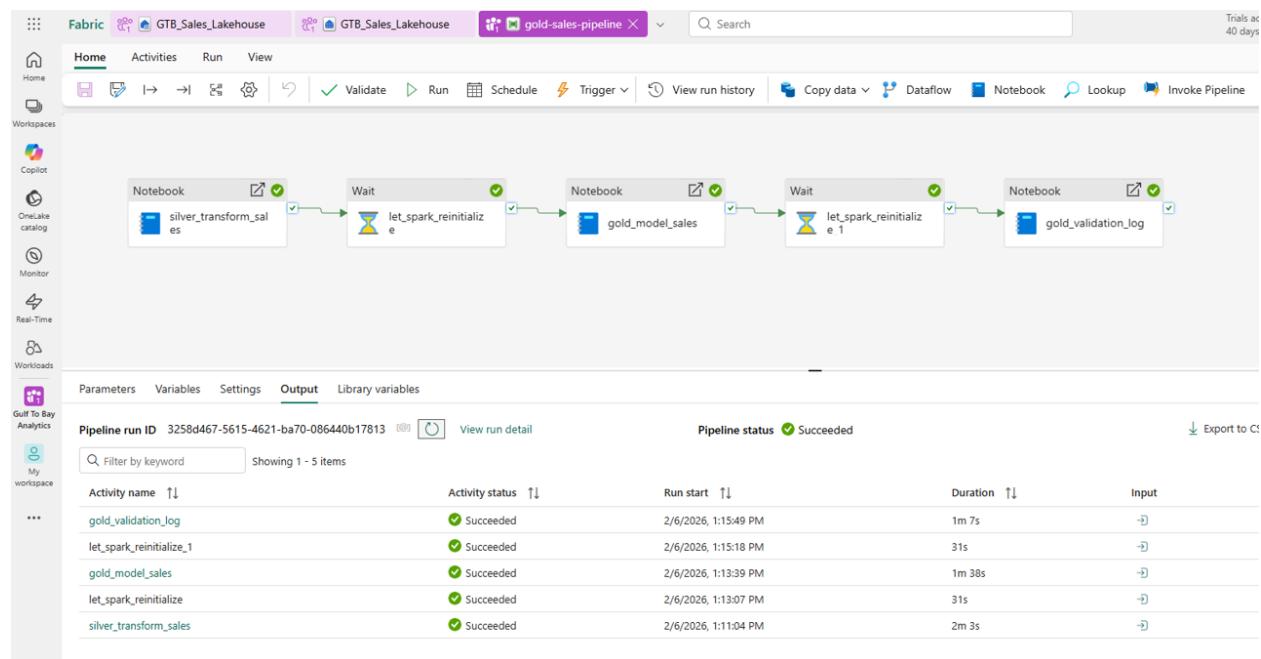
1. Ensure Gold and Gold exception tables exist.
2. TRUNCATE all Gold and exception tables.
3. Load Silver tables.
4. Build dimensions with surrogate keys.
5. Build FactSales with validated foreign keys.
6. Route join failures into exception tables.
7. Append clean rows into Gold tables.
8. Append exception 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()
164 )
```

Job 90 succeeded						
>	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				

Gulf to Bay Analytics — Modernization Case Study

⌚ ETL Components (Fabric Pipeline)



6A. Databricks Lakehouse & Medallion Pipeline

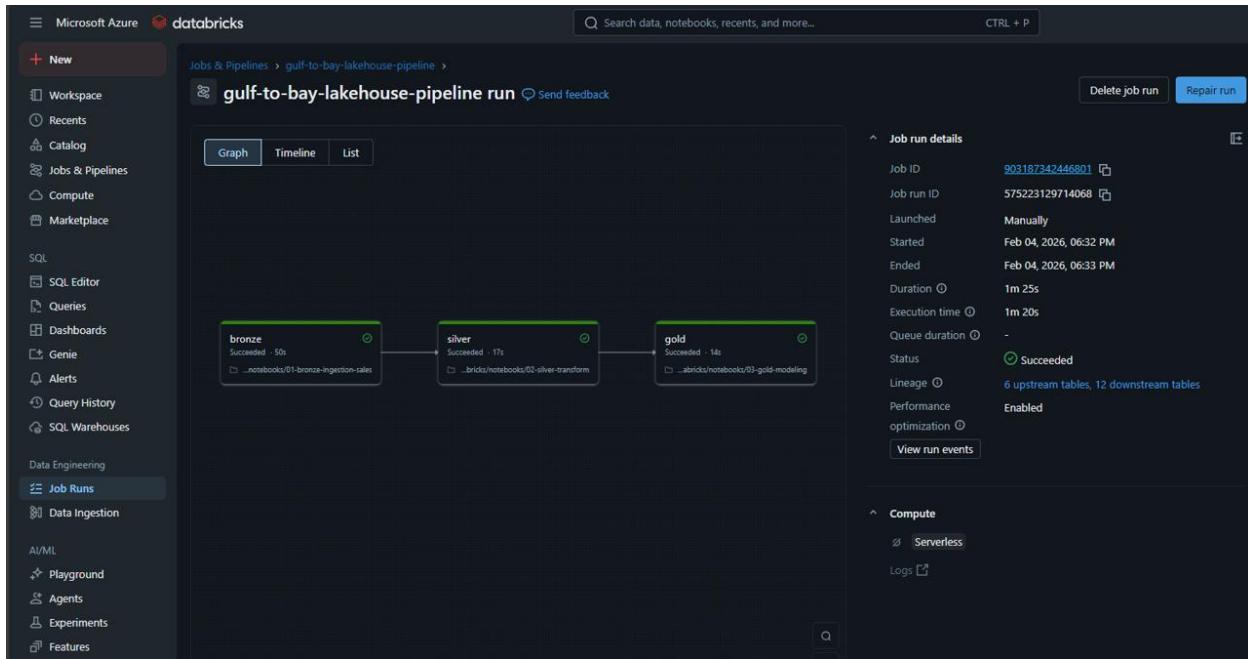
In addition to the Fabric Lakehouse implementation, a full Databricks medallion pipeline was developed using PySpark notebooks and Databricks Jobs. This parallel architecture demonstrates cloud-native engineering patterns, Spark-based transformations, and scalable orchestration aligned with enterprise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

Gulf to Bay Analytics — Modernization Case Study

🔥 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.

Gulf to Bay Analytics — Modernization Case Study

7. Data Modeling & Transformation Layer (Dataflows, SQL Views, Semantic Models)

The screenshot shows two instances of the Microsoft Power Query Advanced Editor interface. The top instance displays a table titled 'customers' with 26 rows of data. The columns include customer_id, first_name, last_name, address1, address2, city, state_province, country, and postal_code. The bottom instance shows the M code used to generate this table. The M code includes various Power Query functions such as #Imported CSV, Table.RenameColumns, and Table.TransformColumnTypes.

customer_id	first_name	last_name	address1	address2	city	state_province	country	postal_code
11000	Jon	Yang	3761 N. 14th St	NULL	Rochampton	Queensland	Australia	4700
11001	Eugene	Huang	2243 W. St.	NULL	Seaford	Victoria	Australia	3198
11002	Ruben	Tones	5844 Linden Land	NULL	Hobart	Tasmania	Australia	7001
11003	Christy	Zhu	1825 Village Pk.	NULL	North Ryde	New South Wales	Australia	2113
11004	John	Peterson	1234 Main Street	NULL	Melbourne	Victoria	Australia	2500
11005	Jane	Ross	7305 Humphrey Drive	NULL	East Brisbane	Queensland	Australia	4199
11006	Janet	Alvarez	2812 Berry Dr	NULL	Mataura	New South Wales	Australia	2398
11007	Marco	Marta	942 Brook Street	NULL	Warrnambool	Victoria	Australia	3280
11008	Rob	Vernoff	624 Ready Road	NULL	Bendigo	Victoria	Australia	3550
11009	Shannon	Carson	3839 Normgate Road	NULL	Henvey Bay	Queensland	Australia	4655
11010	Jacquelyn	Suarez	7300 Corrine Court	NULL	east Brisbane	Queensland	Australia	4199
11011	Curtis	Lu	1224 Oceanic	NULL	east Brisbane	Queensland	Australia	4199
11012	Lauren	Walker	4785 Scott Street	NULL	Brentwood	Washington	United States	98312
11013	Ian	Jenkins	7902 Hudson Ave.	NULL	Lebanon	Oregon	United States	97355
11014	Sydney	Bennett	8011 Tank Drive	NULL	Redmond	Washington	United States	98052
11015	Chris	Young	244 Willow Pass Road	NULL	Burbank	California	United States	91502
11016	Wyatt	Hill	9666 Normridge Ct.	NULL	Imperial Beach	California	United States	91932
11017	Shannon	Wang	7330 Saddlehill Lane	NULL	Sunbury	Victoria	Australia	3429
11018	Claudence	Rai	244 Riverview	NULL	Bendigo	Victoria	Australia	3550
11019	Luke	Lai	7832 Landing Dr.	NULL	Langley	British Columbia	Canada	V3A 4R2
11020	Jordan	King	3156 Rose Dr.	NULL	Methow	British Columbia	Canada	V9
11021	Wilson	White	6120 15th St.	NULL	Des Moines	Washington	United States	98105
11022	Ethan	Zhang	1769 Nicholas Drive	NULL	Bellingham	Washington	United States	98223
11023	Seth	Edwards	4499 Valley Crest	NULL	Bellflower	California	United States	90706
11024	Russell	Xie	8734 Oxford Place	NULL	Concord	California	United States	94519
11025	Alejandro	Beck	2396 Franklin Canyon Road	NULL	Hawthorne	Queensland	Australia	4771

Advanced editor M code:

```
[#File = "Files", ItemKind = "Folder"]
[Datas]
#Navigation 3 =
#Navigation 2 =
{[Name = "customers.csv"]
|Content;
}
#Imported CSV =
Csv.Document(
#Navigation 3,
#"Imported CSV",
[Delimiter = ",",
Column1 = "customer_id",
Encoding = 65001,
QuoteStyle = QuoteStyle.None
]
)
#Renamed columns =
Table.RenameColumns(
#"Imported CSV",
{("Column1", "customer_id")}
)
#Changed column type =
Table.TransformColumnTypes(
#"Renamed columns",
{
("customer_id", Int64.Type),
("Column2", Type.Text),
("Column3", Type.Text),
("Column4", Type.Text),
("Column5", Type.Text),
("Column6", Type.Text),
("Column7", Type.Text),
("Column8", Type.Text),
("Column9", Type.Text)
}
)
```

Gulf to Bay Analytics — Modernization Case Study

Lakehouse SQL Views & Metadata Alignment

The screenshot shows the Fabric Data Explorer interface. On the left, the sidebar includes sections for Home, Help, Workspace, Copilot, Create catalog, Monitor, Real-Time, Workloads, Gulf To Bay Analytics, and My workspace. The main area displays the schema and table structure for the 'GTB_Sales_Lakehouse' database, specifically the 'dbo' schema. A query editor window titled 'qry_sales_model' is open, showing a complex SELECT statement that joins multiple views and tables. Below the editor is a 'Results' tab displaying a table of sales data with columns like sales_key, order_id, order_date, customer_key, first_name, last_name, city, state_province, product_key, category, subcategory, quantity, and unit_price. The results are limited to 10,000 rows.

Semantic Modeling (Fabric Lakehouse → Power BI)

The screenshot shows the Power BI Desktop interface with the semantic model canvas. The canvas displays a data flow between various dimensions and fact tables. Dimensions shown include 'dimGeography', 'dimCustomer', 'dimProductSubcategory', 'dimDate', and 'dimProduct'. Fact tables include 'factInternetSales'. Relationships are visualized as arrows connecting the keys of one dimension to the foreign key in another fact table. The 'Properties' pane on the right shows settings for cards, such as 'Show the database in the header when applicable' set to 'No'. The 'Data' pane on the right lists the tables used in the model.

8. Reporting & Dashboards

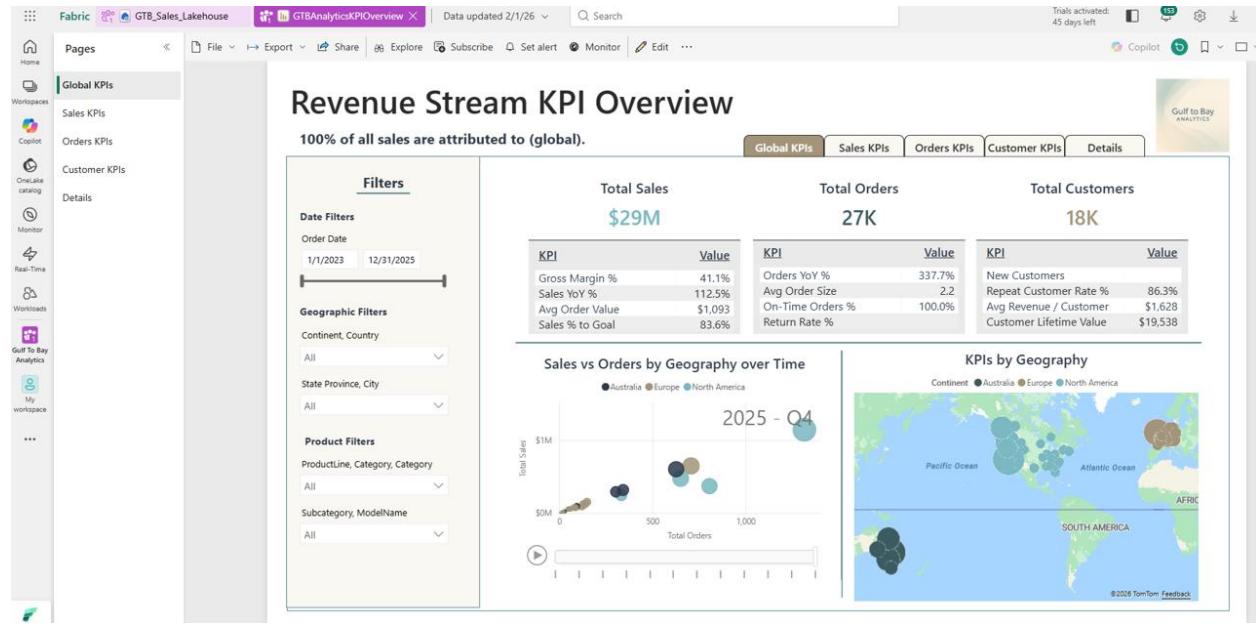
Reporting Highlights

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

Gulf to Bay Analytics — Modernization Case Study

KPI Overview (Public)

<https://app.powerbi.com/view?r=eyJrIjoiMzcyYTlzN2EtYzBjNi00MmY5LWJhY2UtZDk5MDkyZTYwNDExIwidCl6ImE0MzI2YTU4LWY3ZDktNDQ0ZC1iM2FhLWIwOTAyN2U1ZTg2NilslmMiOjF9>



Gulf to Bay Analytics — Modernization Case Study

9. Power Apps — KPI Explorer

Highlights

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

The screenshot shows the Microsoft Power Apps canvas editor. The top navigation bar includes Back, Insert, New screen, Theme, Open Sans, 13, Normal, and a gear icon. The left sidebar shows a tree view of the app structure under 'scHome' and various components like 'btnExploreKPIs', 'ibnViewDashboard', 'ibnAppTitle', etc. The main workspace displays a 'Revenue Stream KPI Overview' card. The card features three KPIs: Total Sales (\$29M), Total Orders (27K), and Total Customers (18K). It also includes a chart titled 'Sales vs Orders by Geography over Time' for the period 2025 - Q4, and a map titled 'KPIs by Geography'. A code editor window on the right shows a snippet of M language:

```
SortByColumns(
    AddColumns(
        Distinct("ML.vwCategoriesForDropDown", Category),
        SortOrder,
        If(Value = "All", 0, 1)
    ),
    "SortOrder",
    "Ascending",
    "Value",
    "Ascending"
)
```

Gulf to Bay Analytics — Modernization Case Study

10. Power Automate — Refresh & Notifications

🔔 Highlights

- Scheduled dataset refresh
- Failure notifications
- Semantic model triggers

The screenshot shows the Microsoft Power Automate interface. At the top, there's a blue header bar with the title "Power Automate" and a search bar. Below the header, a breadcrumb navigation shows "Back" and "Scheduled Daily at 2:00 AM Eastern -> Refresh AdventureWorksRevenue...". The main area has a left sidebar with tabs for "Parameters" (which is selected), "Settings", "Code view", "Testing", and "About". The "Parameters" tab displays fields for "Workspace" (set to "My Workspace") and "Dataset" (set to "GTBAnalyticsKPIOverview"). Below these fields, a note says "Connected to mlloyd9@pods9.onmicrosoft.com." with a "Change connection" link. To the right of the sidebar is a canvas where a flowchart is being built. The flowchart consists of three main steps connected by arrows: 1. A "Scheduled Daily at 2:00 AM Eastern" trigger step, represented by a blue square with a white alarm icon. 2. A "Refresh Master Semantic Model" action step, represented by a yellow square with a white document icon. 3. A "Refresh GTBAnalyticsKPIOverview Semantic Model" action step, also represented by a yellow square with a white document icon. Each step has a small edit icon (a circular arrow) to its right.

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

Branch rules

Restrict creations

Only allow users with bypass permission to create matching refs.

Restrict updates

Only allow users with bypass permission to update matching refs.

Restrict deletions

Only allow users with bypass permissions to delete matching refs.

Require linear history

Prevent merge commits from being pushed to matching refs.

Require deployments to succeed

Choose which environments must be successfully deployed to before refs can be pushed into a ref that matches this rule.

Require signed commits

Commits pushed to matching refs must have verified signatures.

Require a pull request before merging

Require all commits be made to a non-target branch and submitted via a pull request before they can be merged.

Require status checks to pass

Choose which status checks must pass before the ref is updated. When enabled, commits must first be pushed to another ref where the checks pass.

Block force pushes

Prevent users with push access from force pushing to refs.

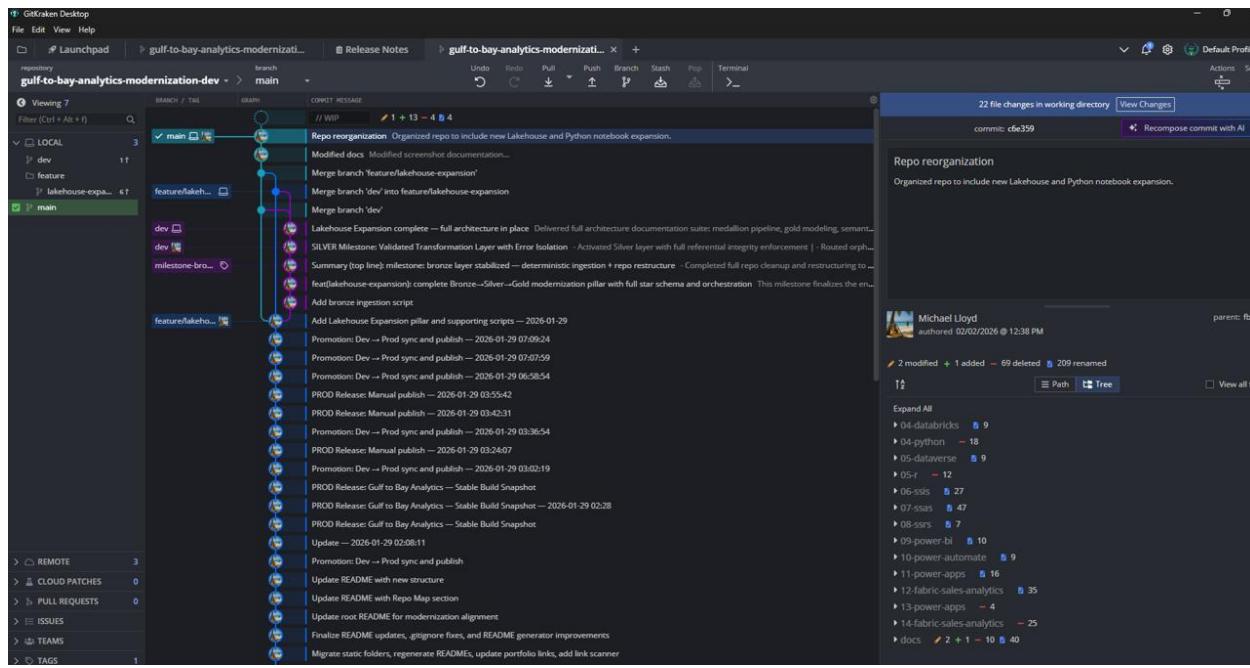
Require code scanning results

Choose which tools must provide code scanning results before the reference is updated. When configured, code scanning must be enabled and have results for both the commit and the reference being updated.

Require code quality results

Choose which severity levels of code quality results should block pull request merges. When configured, a code quality analysis must be done on the pull request before the changes can be merged.

Gulf to Bay Analytics — Modernization Case Study



12. Results & Impact

🚀 Platform Impact

- Unified Fabric ecosystem
- Clean lineage
- Automated refresh cycles

⚙️ Engineering Impact

- Git-based SDLC
- Deterministic promotion
- Standardized documentation

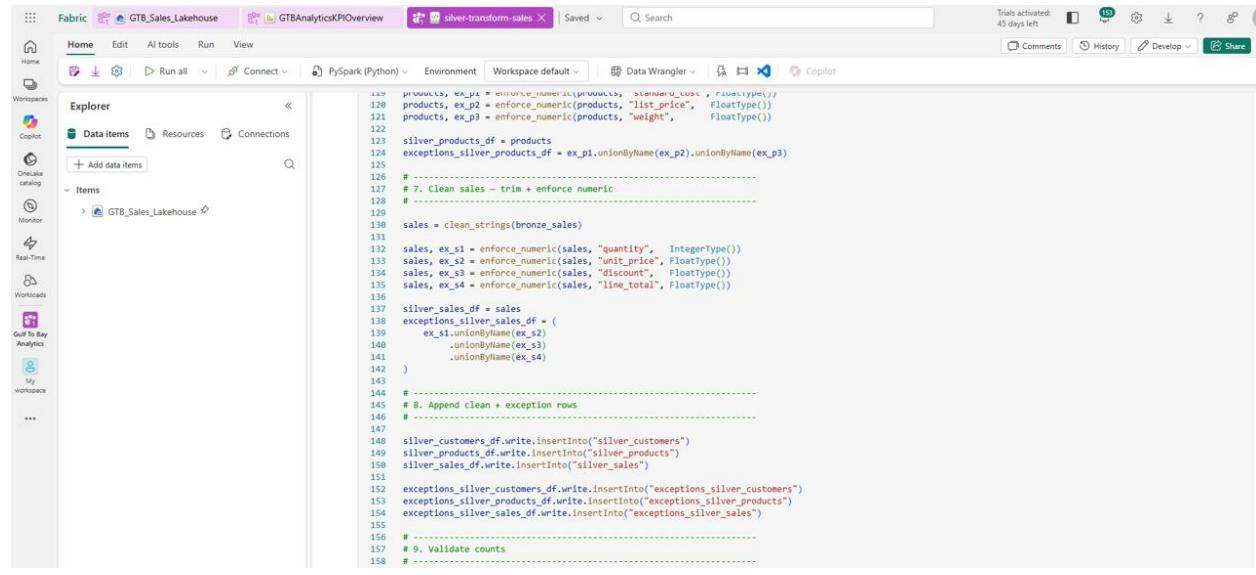
📊 Business Impact

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

Gulf to Bay Analytics — Modernization Case Study

Appendix

Appendix A — Data Quality Subsystem



The screenshot shows the Fabric Data Wrangler interface. On the left, the sidebar includes sections for Home, Edit, AI tools, Run, View, Workspace, Monitor, Real-Time, Workloads, and Gulf to Bay Analytics. The main area has tabs for Home, Edit, AI tools, Run, View, Connect, PySpark (Python), Environment, Workspace default, Data Wrangler, and Capilot. The Data Wrangler tab is active. The Explorer panel on the left shows 'Data items' and 'Items' under 'GTB_Sales_Lakehouse'. The central workspace contains a Python code editor with the following script:

```
products, ex_p1 = enforce_numeric(products, "standard_cost", FloatType())
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())

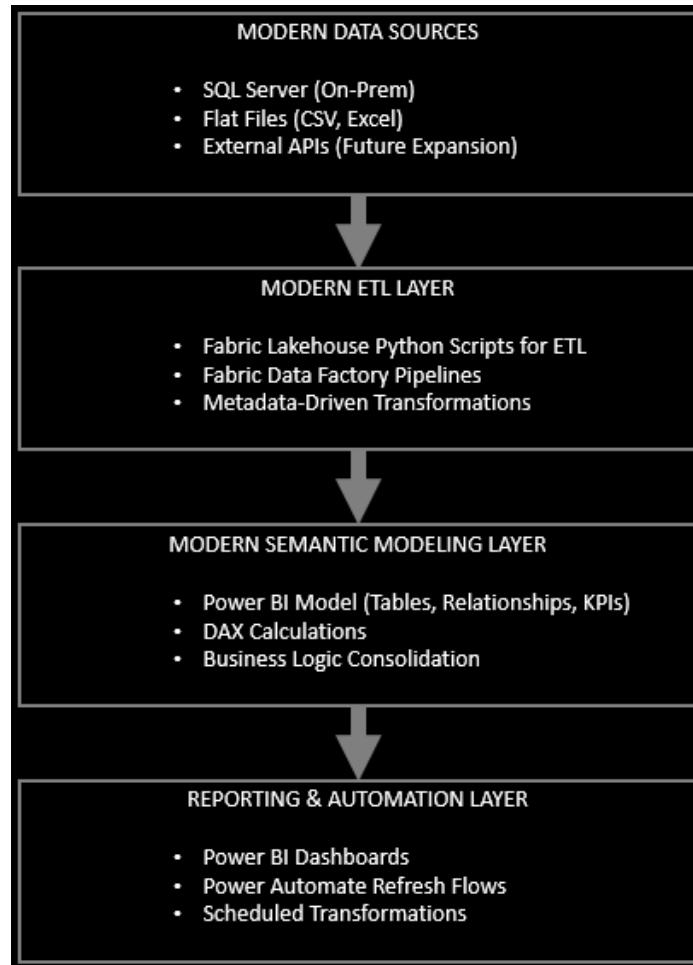
silver_sales_df = sales
exceptions_silver_sales_df = (
    ex_s1.unionByName(ex_s2)
    .unionByName(ex_s3)
    .unionByName(ex_s4)
)

# -----
# 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")
silver_sales_df.write.insertInto("exceptions_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



Gulf to Bay Analytics — Modernization Case Study

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