Executive Summary

This project focuses on building a modern, dynamic data pipeline using the Twitter dataset with Azure Data Stack, demonstrating practical skills and yielding valuable insights.

Introduction

In this modern data landscape, everything is driven by data. Scalable, automated and analytical End-to-End pipelines are essential for enabling organisations to effectively process raw data into actionable insights that drive business value.

Data Source

- The Twitter dataset was sourced and generated by web scraping.
- There are two separate files: one containing over 4000 rows for the initial load and another file containing approximately 400 rows for the incremental process.

Ingestion Layer - Azure Data Factory

- The initial step is to ingest the raw data into the SQL database.
- Then, ADF was utilised to both ingest and orchestrate the pipeline execution.

Storage Layer - Azure Data Lake Storage Gen 2

- The ingested data is then staged within Azure Data Lake Storage Gen 2.

Transformation Layer - Databricks

- Data then flows from the Storage layer to Azure Databricks, where it undergoes transformation processing using PySpark.

Serving Layer

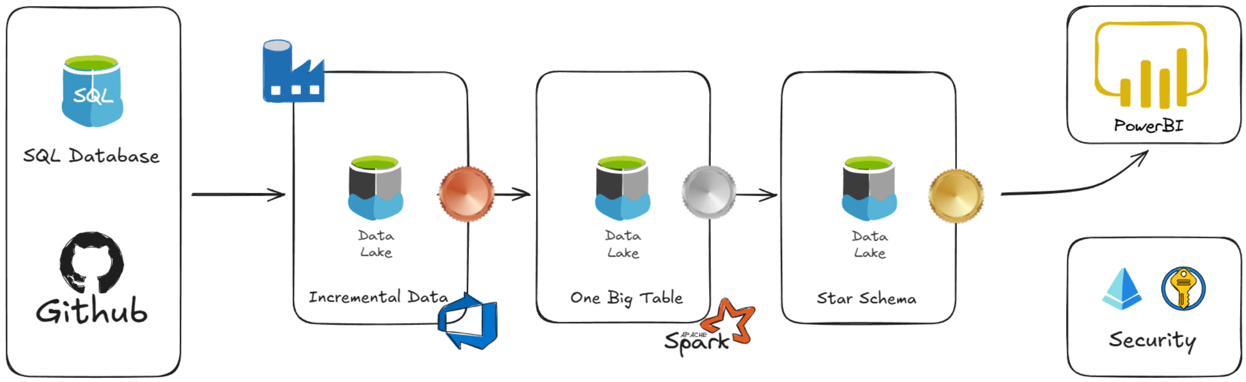
- Following the transformation, the resulting model is served directly.
- Based on the downstream usage, various reporting tools can then be utilised to generate comprehensive reports and dashboards.

Monitoring

- Implemented using Azure Factory Git CI/CD pipeline.

Overall Flow Summary

GitHub | SQL Database → ADF → Azure Data Lake Storage Gen 2 → Databricks → Power BI.



Architecture Diagram

The Medallion Architecture, a pattern adopted within Data Engineering, was implemented for this project. It comprises three distinct layers:

- Bronze layer: This layer holds the raw, source-system data.
- Silver layer: This layer stores validated, cleaned and harmonised data
- Gold layer: This layer contains refined, aggregated and feature-engineered data.

Architecture Overview

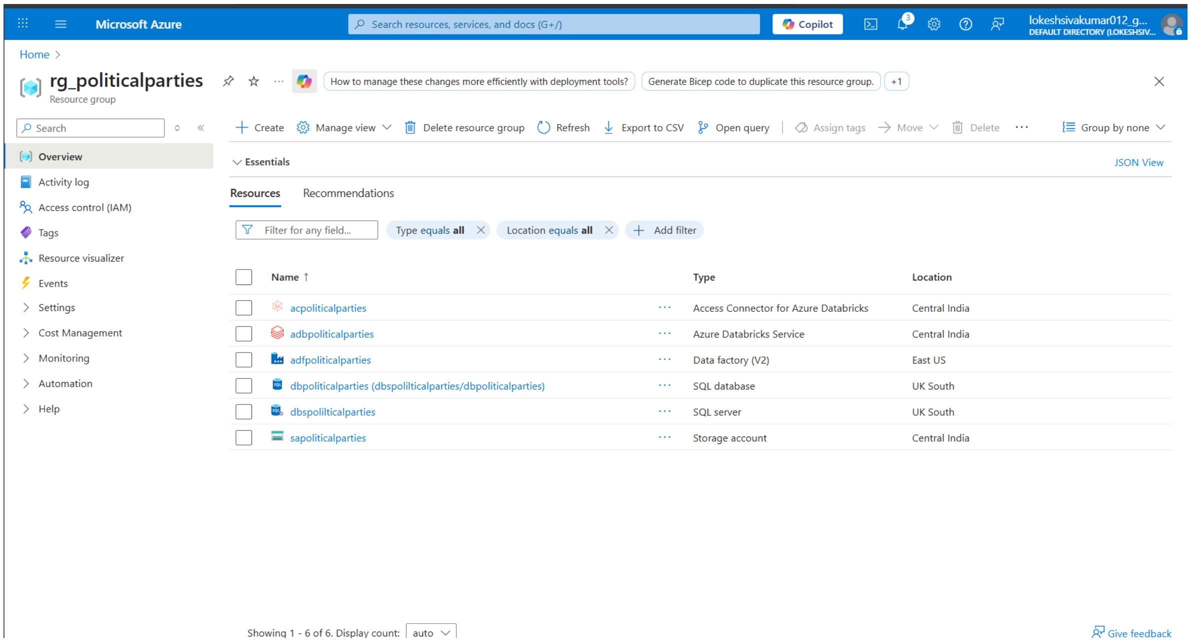
The following components facilitate the pipeline:

- GitHub: Used to store and manage the raw source data.
- SQL Databases: The raw data is initially copied into the local SQL database for staging.
- Azure Data Factory: ADF is used to ingest data into the bronze layer.
- Azure Data Lake Storage Gen 2: This is where all our data gets stored.
- Azure Databricks: Data transformation and refinement occur here, utilising PySpark.
- Azure Key Vault: This service enhances security and secures access credentials.

Implementation Details

Environment

Microsoft offers a free 30-day trial period to access its services. I have created the following resource group named rg_politicalparties. Within it, I have used the required services, namely Azure Data Factory, Azure Storage Account, Azure Data Lake Storage Gen 2, Azure Databricks, and Azure Databricks Access Connector, as shown in the image below.



Azure Services

Data Source

We needed a place to hold our raw data before starting the pipeline. I chose a SQL database for this step. From here, our data is ingested into our bronze layer. So I created a username and password for secure access to our SQL database.

Data Ingestion Using Azure Data Factory

Connecting SQL Server and Azure Data Factory

ADF needed a way to talk to our SQL Server database. They didn't have a connection yet. To fix this, I created a Linked Service to act as a secure bridge between ADF and our SQL database, along with the Dataset so that ADF know exactly which table to read.

Data Ingestion

I have used the following activities in the ADF to build a flexible pipeline with a parameterisation approach.

- LookUp: To find out the last time we ran the pipeline.
- Set variable: To get the current date using the utcnow function.
- Copy data: From SQL into the Bronze layer.
- If Condition: To check if new data was read, if then
- Script: To get the max value of the last updated cdc column (date), and
- Copy data: To update the last updated date from ingestion.
- Delete: To remove the temporary files.

Tracking New Data

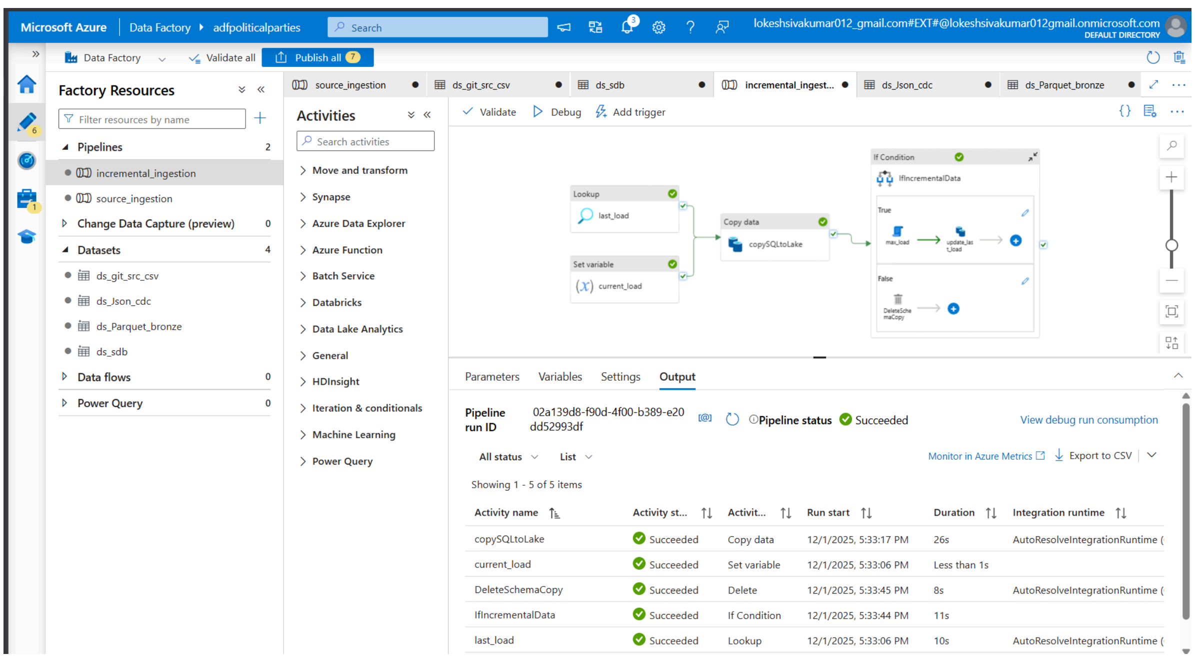
We need a way to remember the last time we loaded data. I also added a back date function. This allows us to re-run data from a past day if needed. I used two files for this:

1. cdc.json: Stores the date of the last successful data load.
2. empty.json: Helps in interchange of the last ingested date.
3. back_date: allows us to rerun old data if needed.

The above parameters help our loop to work effectively.

Scheduling

After setting up the pipeline, I used the Debug option to run a test. Next, I tested it with the incremental date using an Add Trigger option. Every time it runs, it fetches data from the SQL database. This keeps our Bronze data fresh.

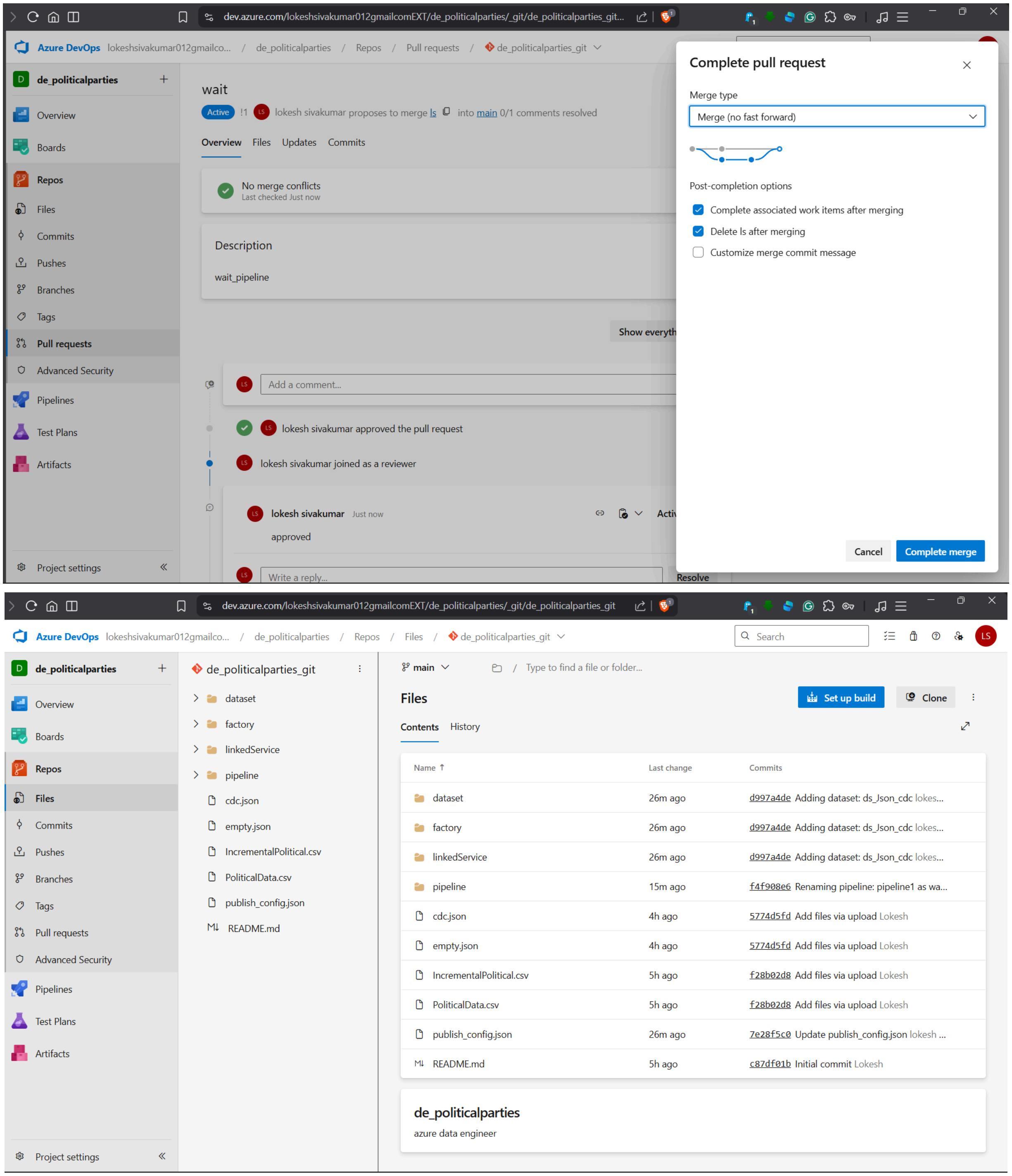


Incremental Ingestion Loop Pipeline

The raw data gets successfully moved from the SQL database into our Bronze layer. The raw data is stored in one big table: politicalparties.

Deployment and Monitoring

The ADF environment was integrated with a Git repository to track all changes. Any change merged into the main branch automatically triggers the build process, which automates the promotion of the validated pipeline from the development to testing and production.



CI/CD Pipeline

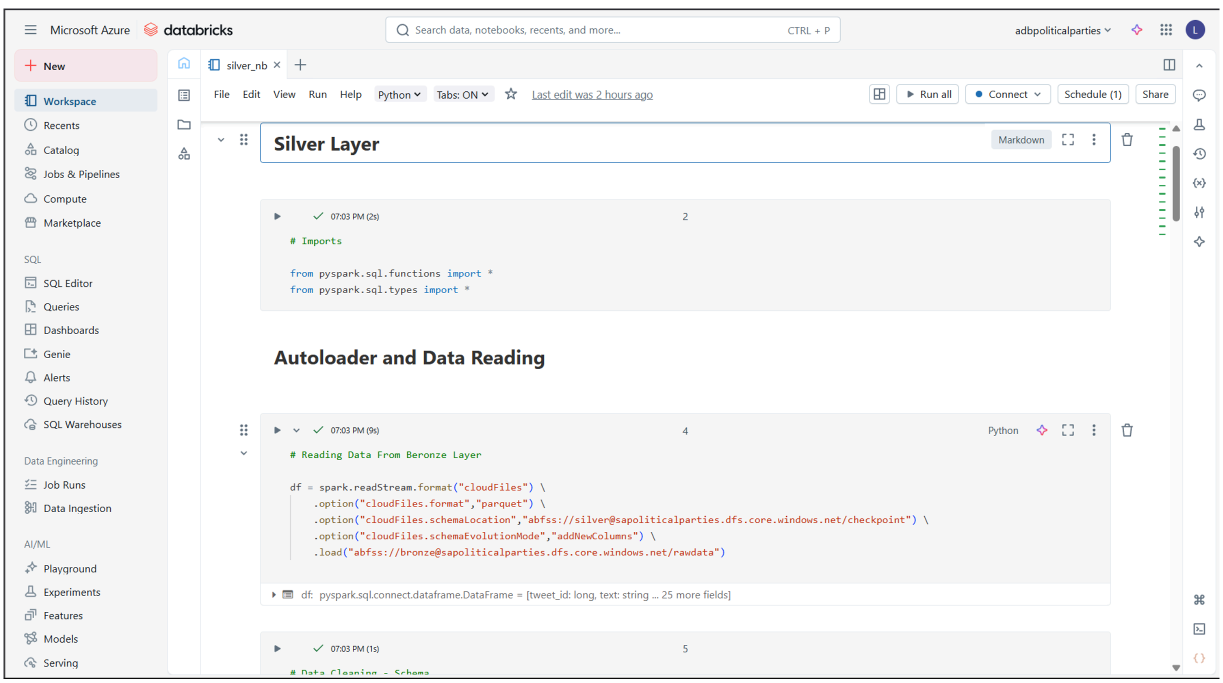
Data Transformation using Azure Databricks

I have leveraged several tools here:

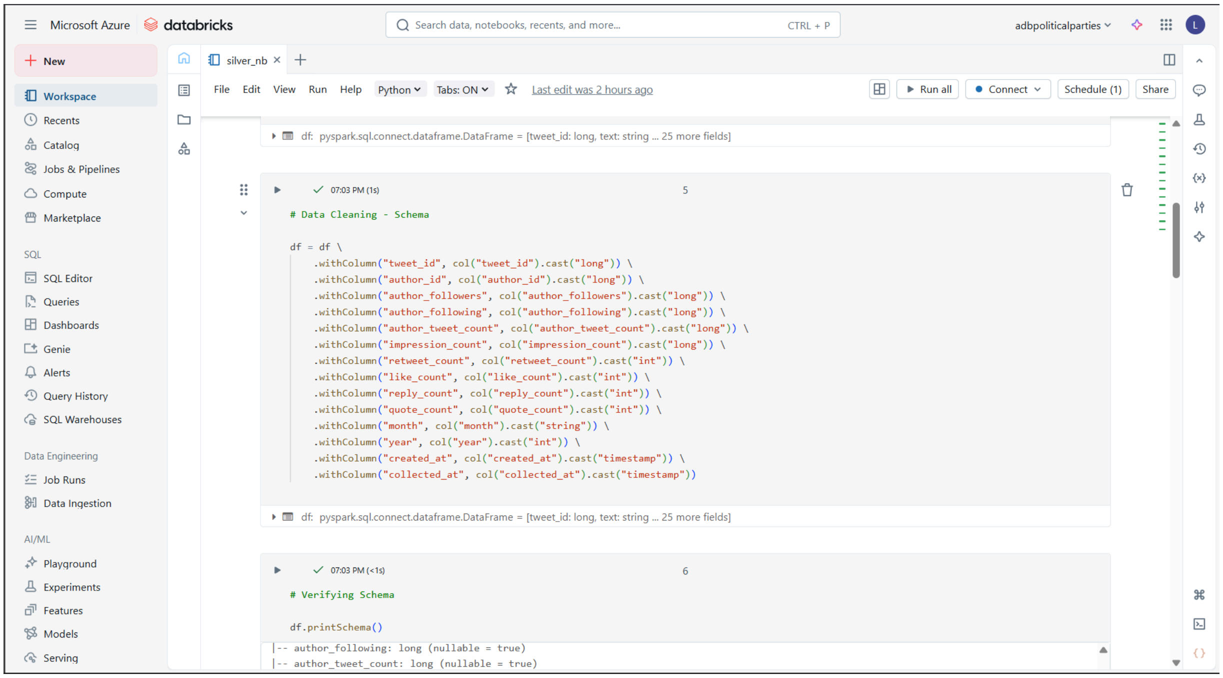
- Databricks Unity Catalog: This helps us manage and secure all our data and tables. It gives us one place to control everything.
- Databricks Access Connector: It uses a Managed Identity, a secure, password-less way for services to connect. It makes our Databricks securely read and write data to ADLS Gen2 without a secret key.
- Spark Streaming with Autoloader: This is the automatic way to handle new data as soon as it arrives. Autoloader watches the files and starts the process automatically.
- Pyspark Transformation: To write all the code for cleaning, validating, and shaping the data.

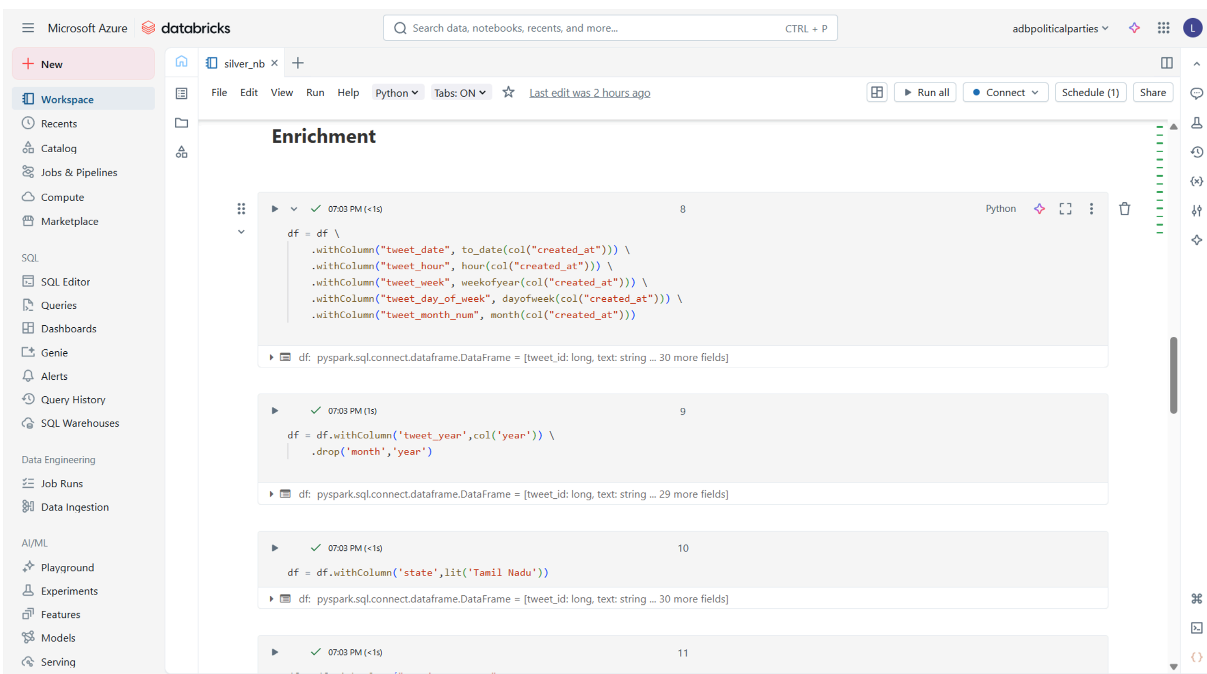
The Silver layer

The raw data from the Bronze layer, which we read using the cloudFiles method. This allows us to use streaming for efficiency.

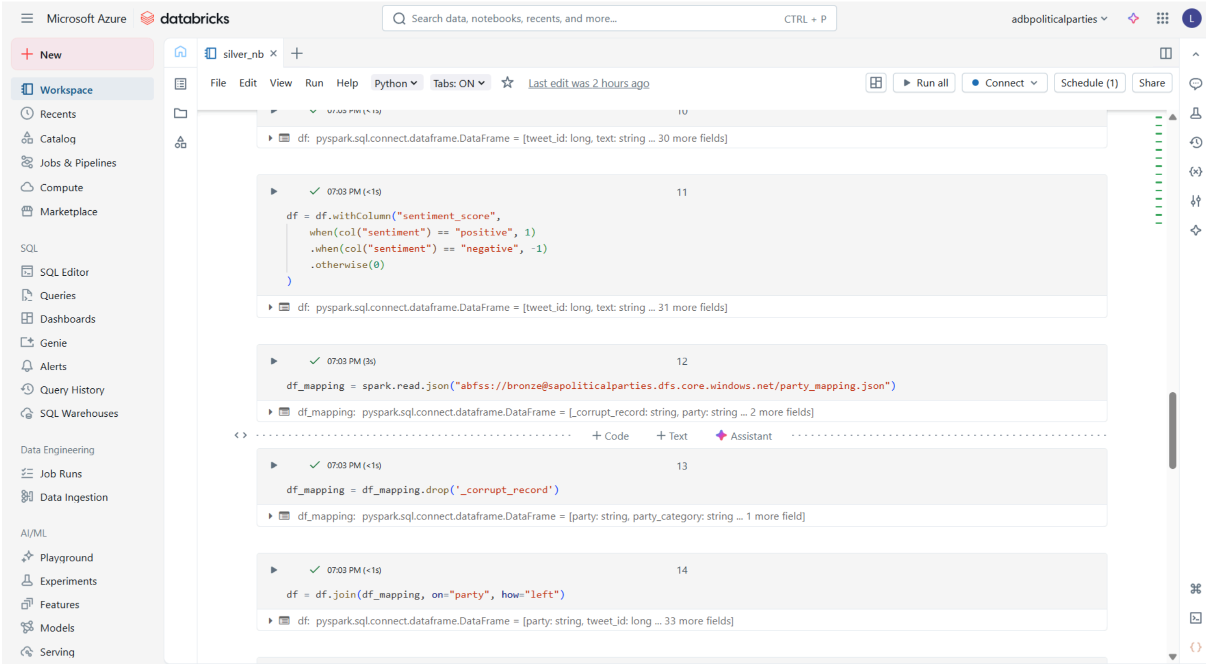


Then I performed a Schema validation, ensuring all the required columns are present and that their names match the defined specification, which includes data type enforcement.

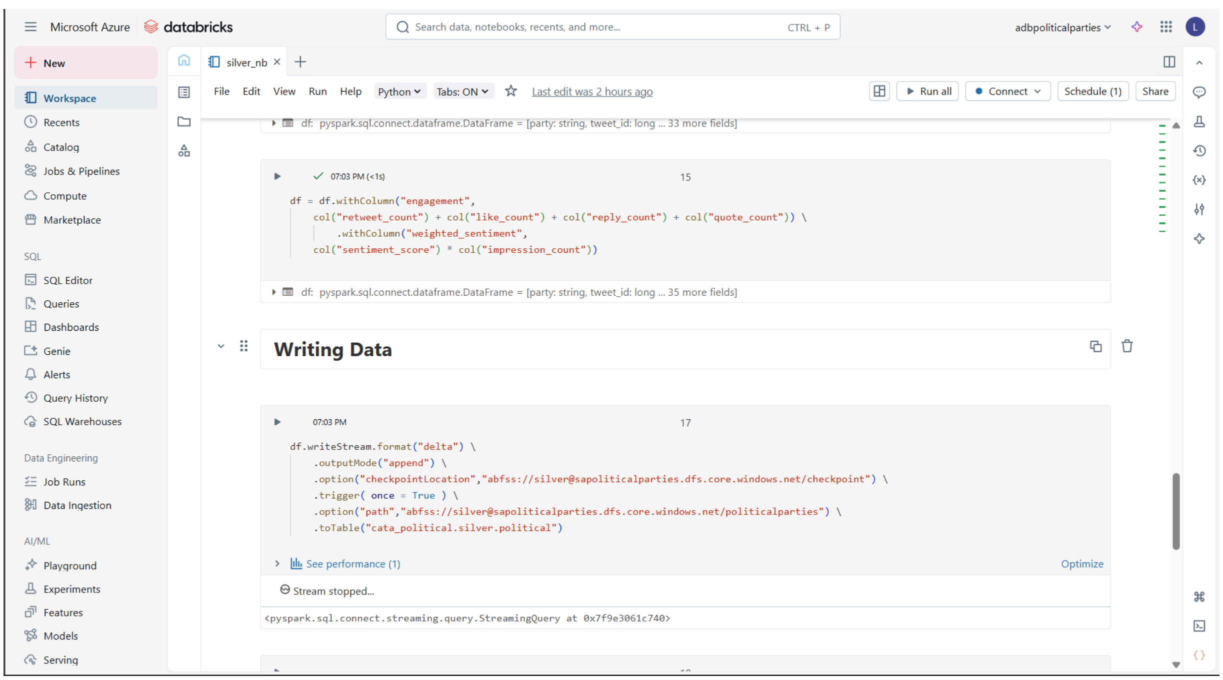




Then, a simple transformation created a few columns with the help of the created_at column using the withColumn function.



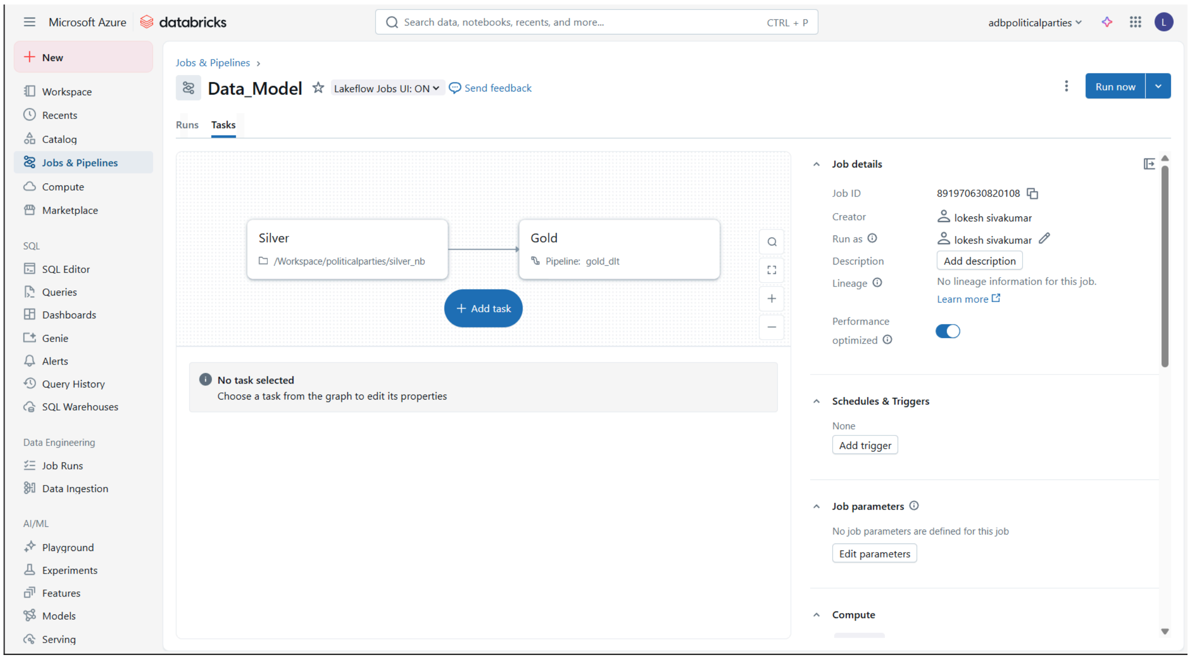
Followed by a mapping process, with the help of a dedicated file that stores the political parties' information alone in a JSON format.



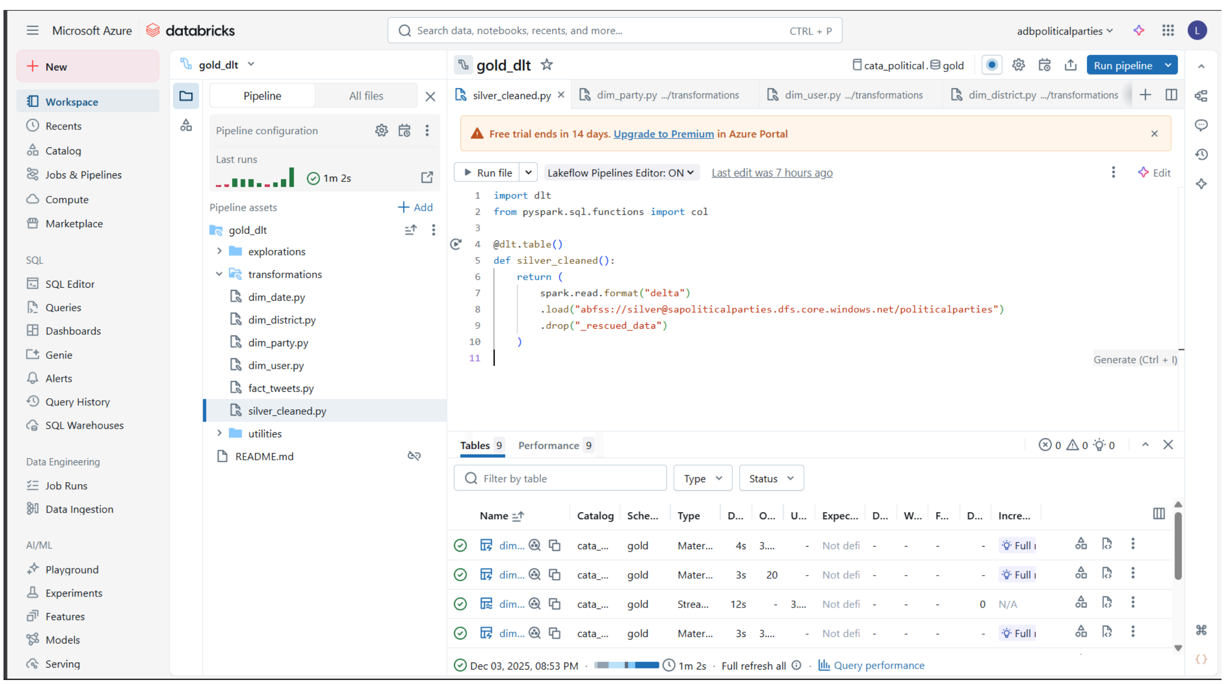
Then I stored it as a Delta Table in the Silver container.

Gold layer ETL

This layer contains the final optimised data for our downstream purpose.



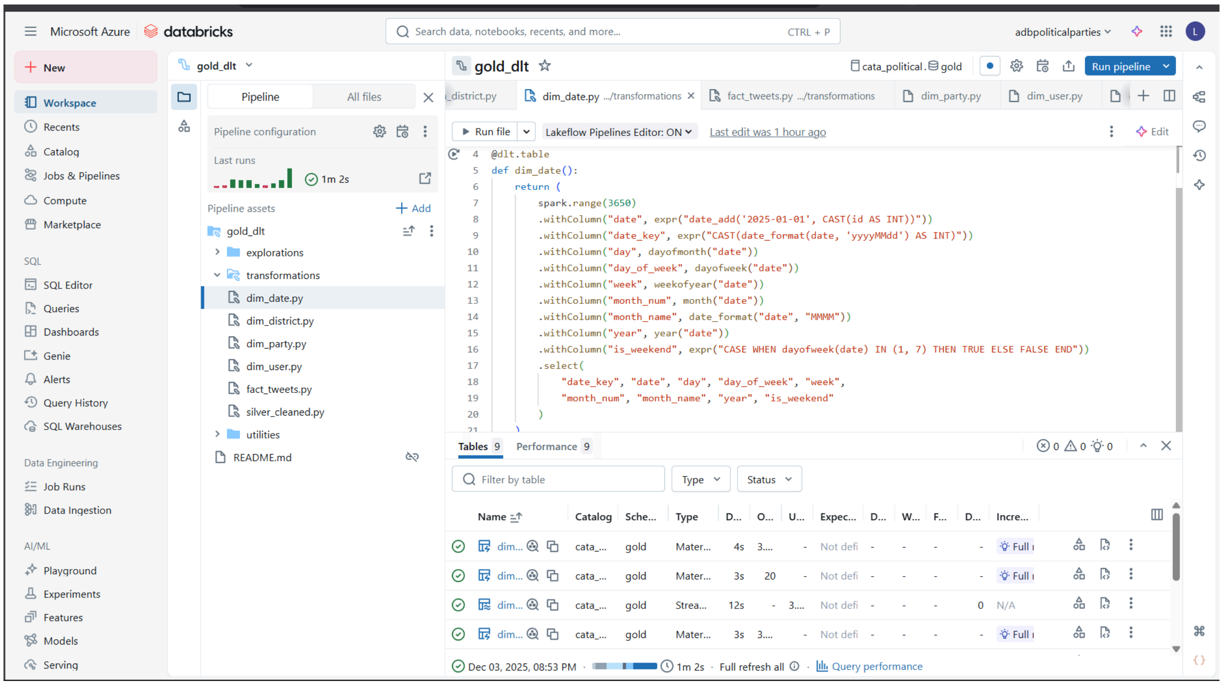
Gold layer ETL Pipeline



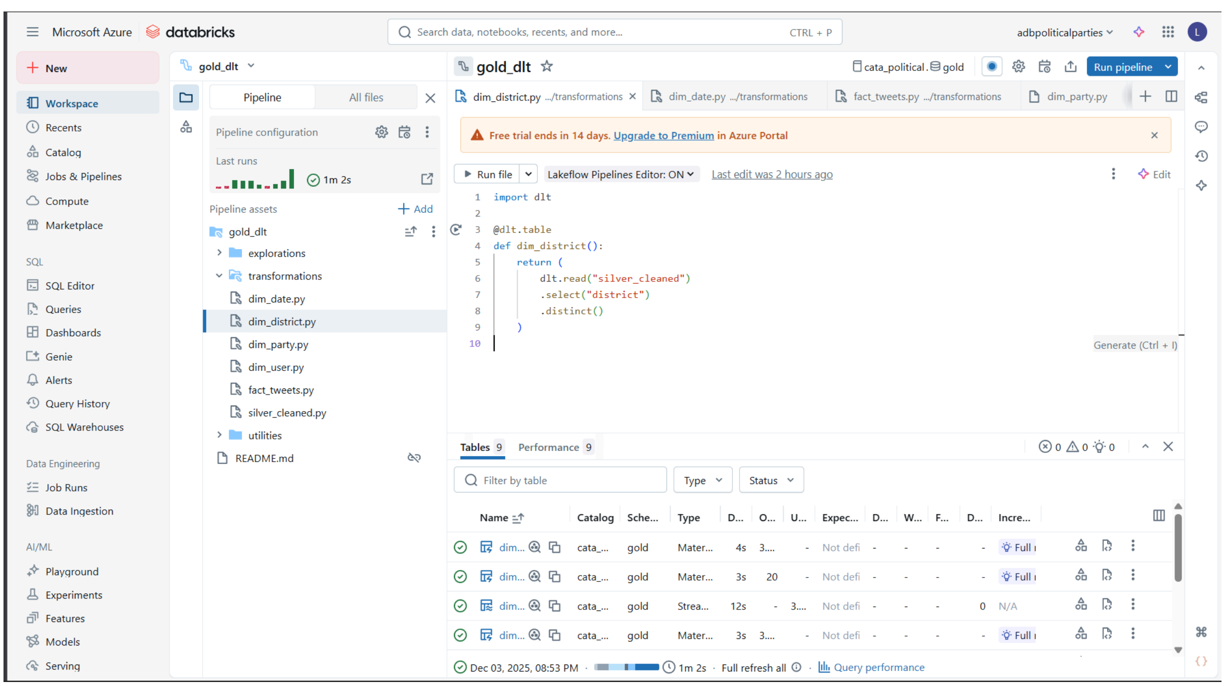
SilverCleaned

Data Modeling

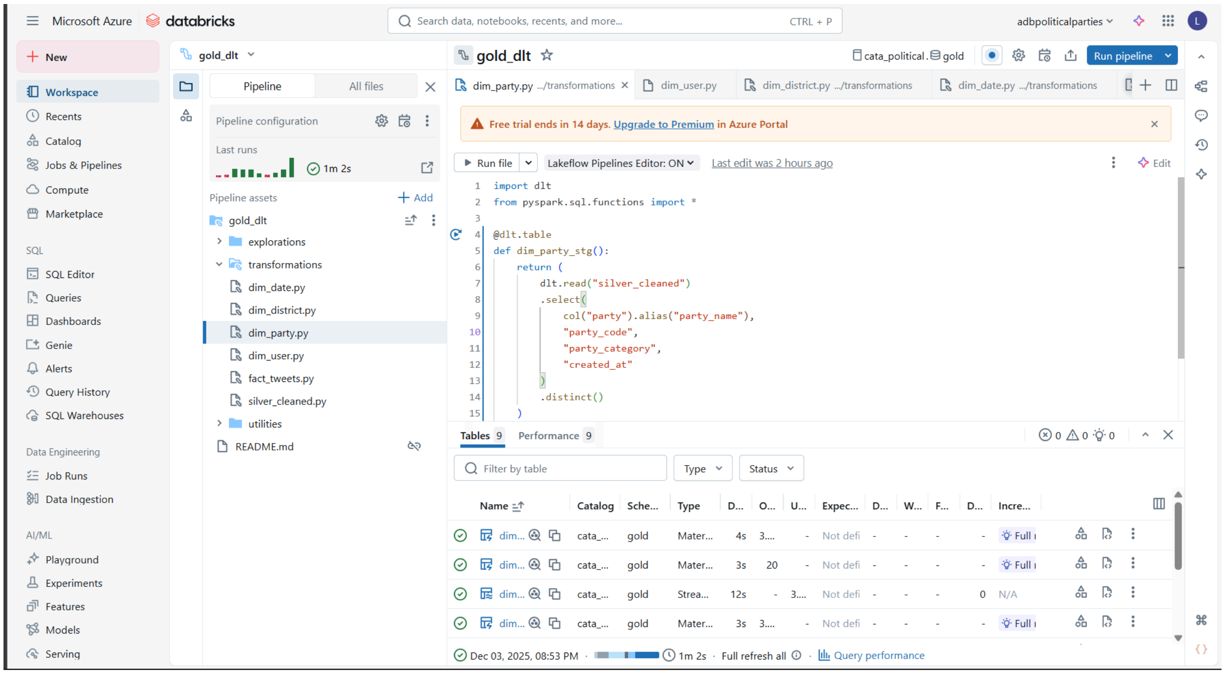
Here, I have transformed the data into a star schema format.



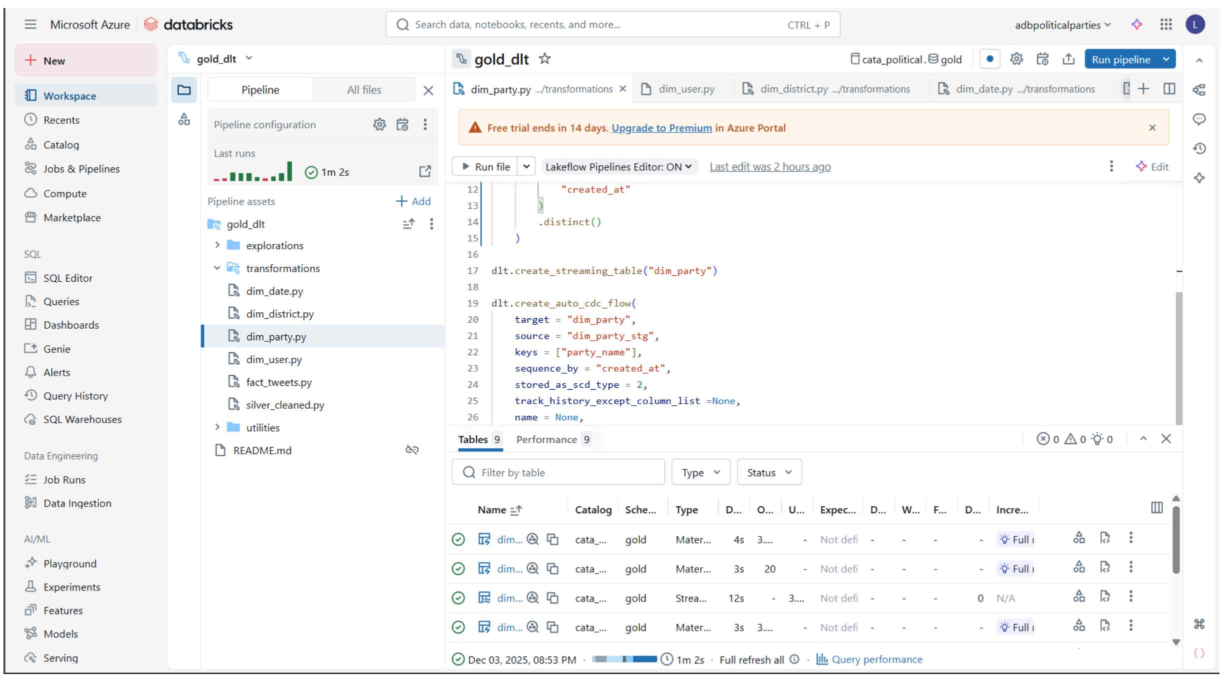
DimDate



DimDistrict



DimParty

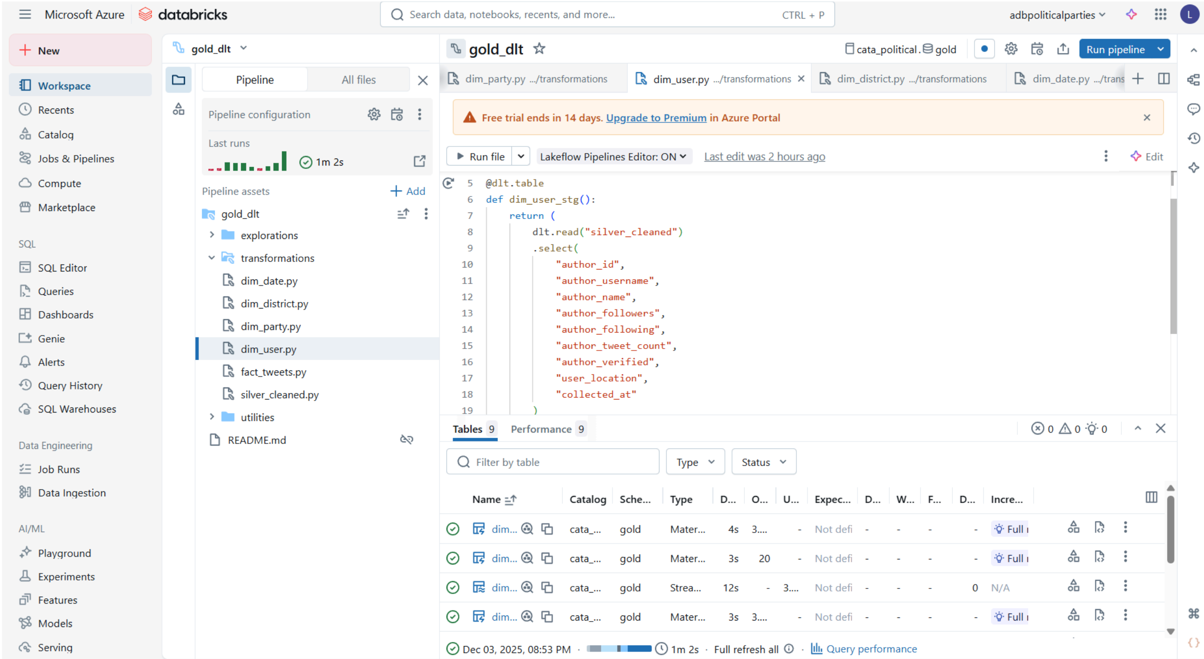


DimParty SCD

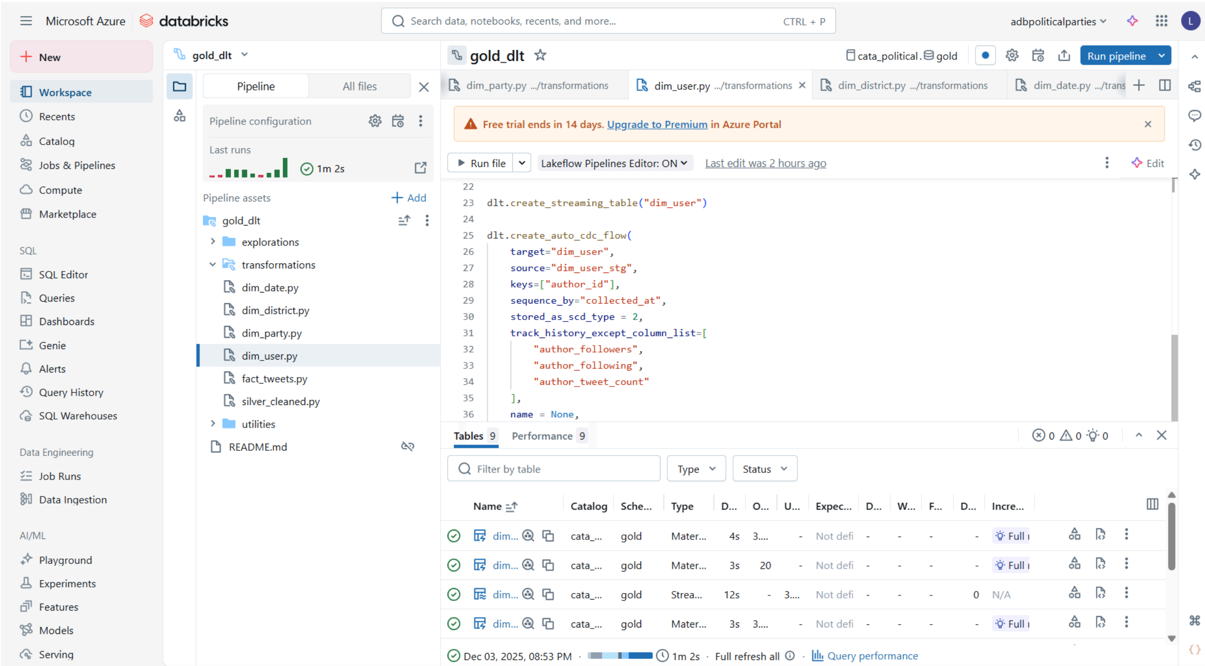
Star Schema Structure

Fact Table: I have created a central Fact Table containing metrics and foreign keys.

Dimension Table: The data was normalised into several independent Dimension Tables, namely: DimDate, DimDistrict, DimParty and DimUser.



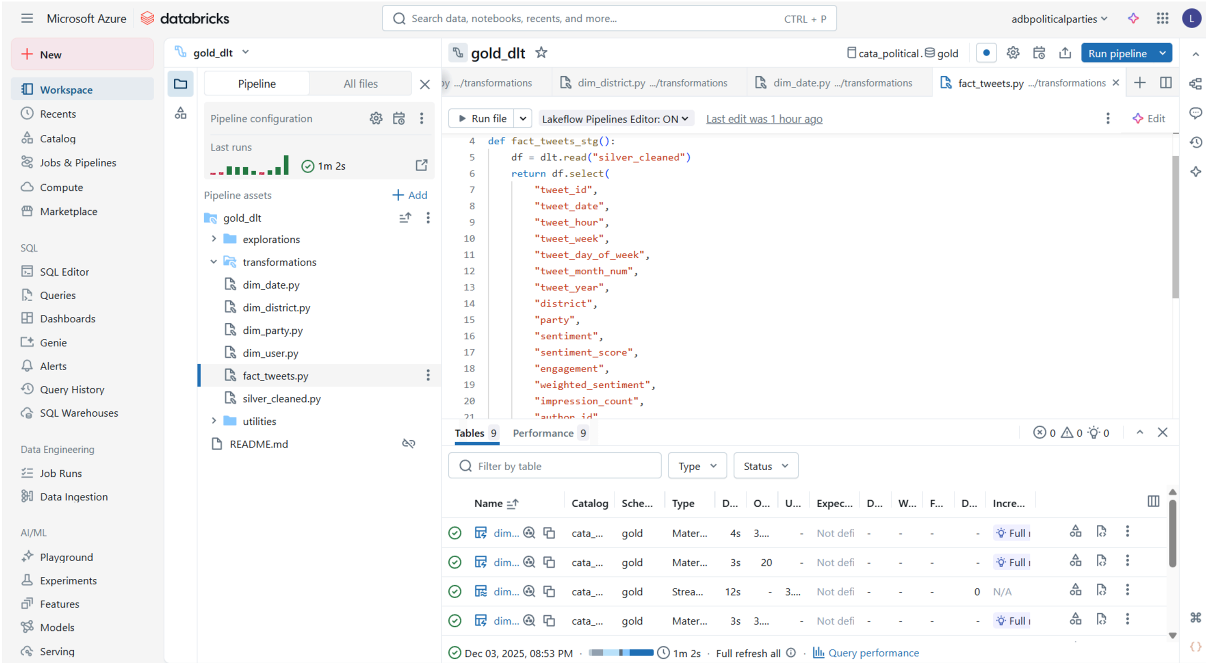
DimUser



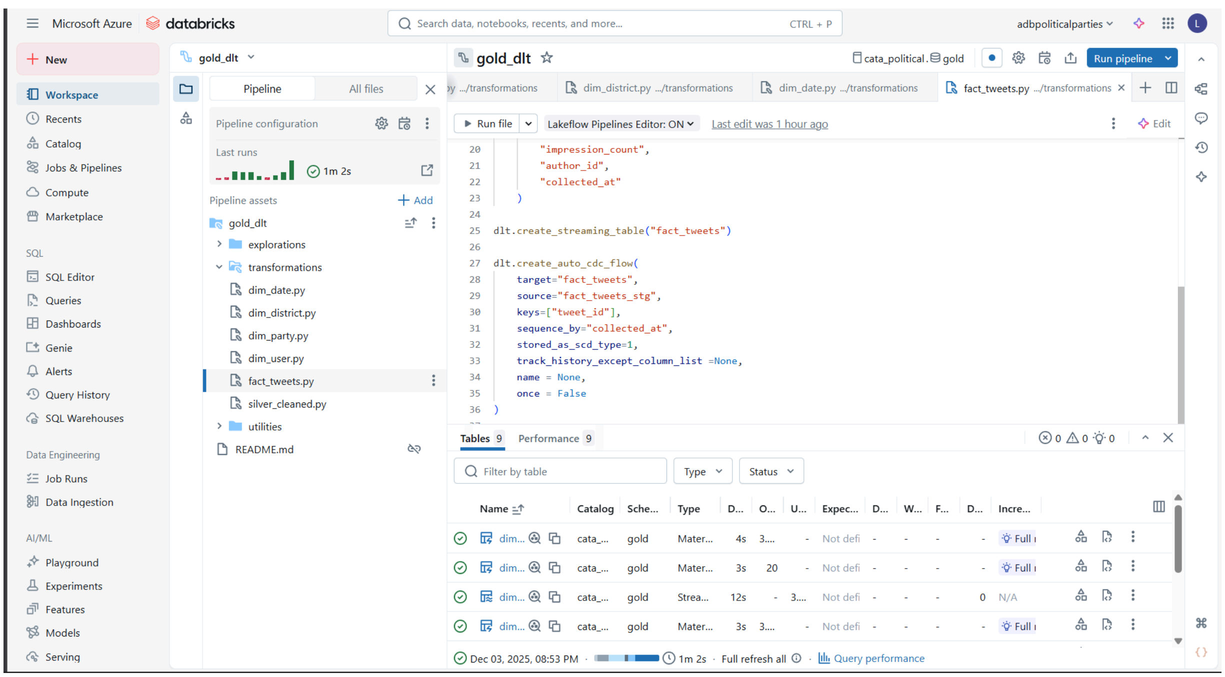
DimUser SCD

Slowly Changing Dimensions

It is essential for tracking changes to dimensional attributes over time. This ensures that historical reports reflect the data as it was at the time of the event, providing accurate historical analysis for our downstream team.



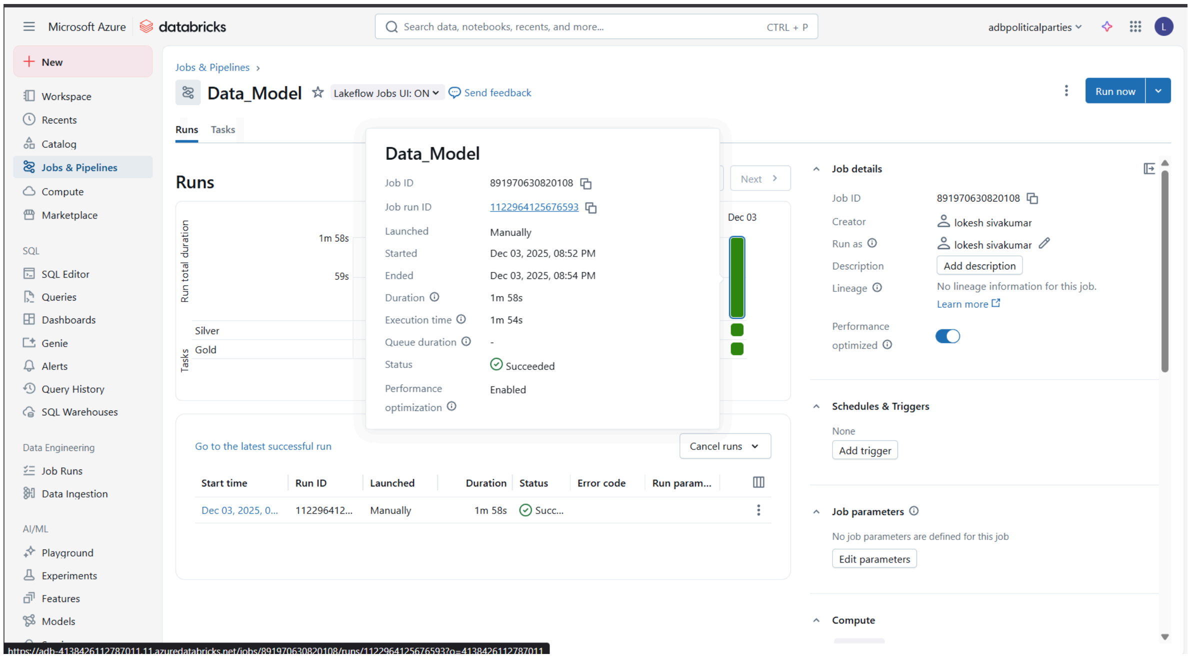
FactTweets



FactTweets SCD

Delta Live Tables

It is instrumental in managing the complexity of both the star schema creation and the automated implementation of the SCD logic.



Gold Pipeline

Pipeline Testing

I have tested this pipeline with the incremental data set, and it successfully proved its effectiveness by incorporating the new data without any errors.

Serving the Data

Once the data reaches the Gold layer, it is ready to be served to the downstream process team, which includes the Data Analyst. With this, I have ensured that our pipeline is industry-level and production-ready.

Challenges and Solutions

- Web Scraping Limitation → Used rotating headers.
- Data Inconsistency → Standardise formats using Pandas and Pyspark.
- Schema Evolution → Implemented Delta Lake for versioning.

Conclusion

I built this scalable, cloud-based, enterprise-ready pipeline from scratch, directly facing and solving real-world challenges, and now I am ready to deliver a production-level pipeline with real business insights.