POV: Identify Grid stability risks in advance using Notebooks in MS Fabric

Prepared by LSHCERU BG
Muskan Singhal

Objective

- ✓ To monitor and predict grid stress levels based on real-time and forecasted weather conditions, enabling proactive load balancing and outage prevention.
- ✓ To integrate weather forecasts into grid analytics for dynamic risk scoring and early warning of instability events.
- ✓ To develop predictive models that correlate weather anomalies (e.g., heatwaves, wind speed, storms) with grid performance metrics across regions.
- ✓ To visualize the impact of weather variables on grid reliability through interactive dashboards and alert systems.

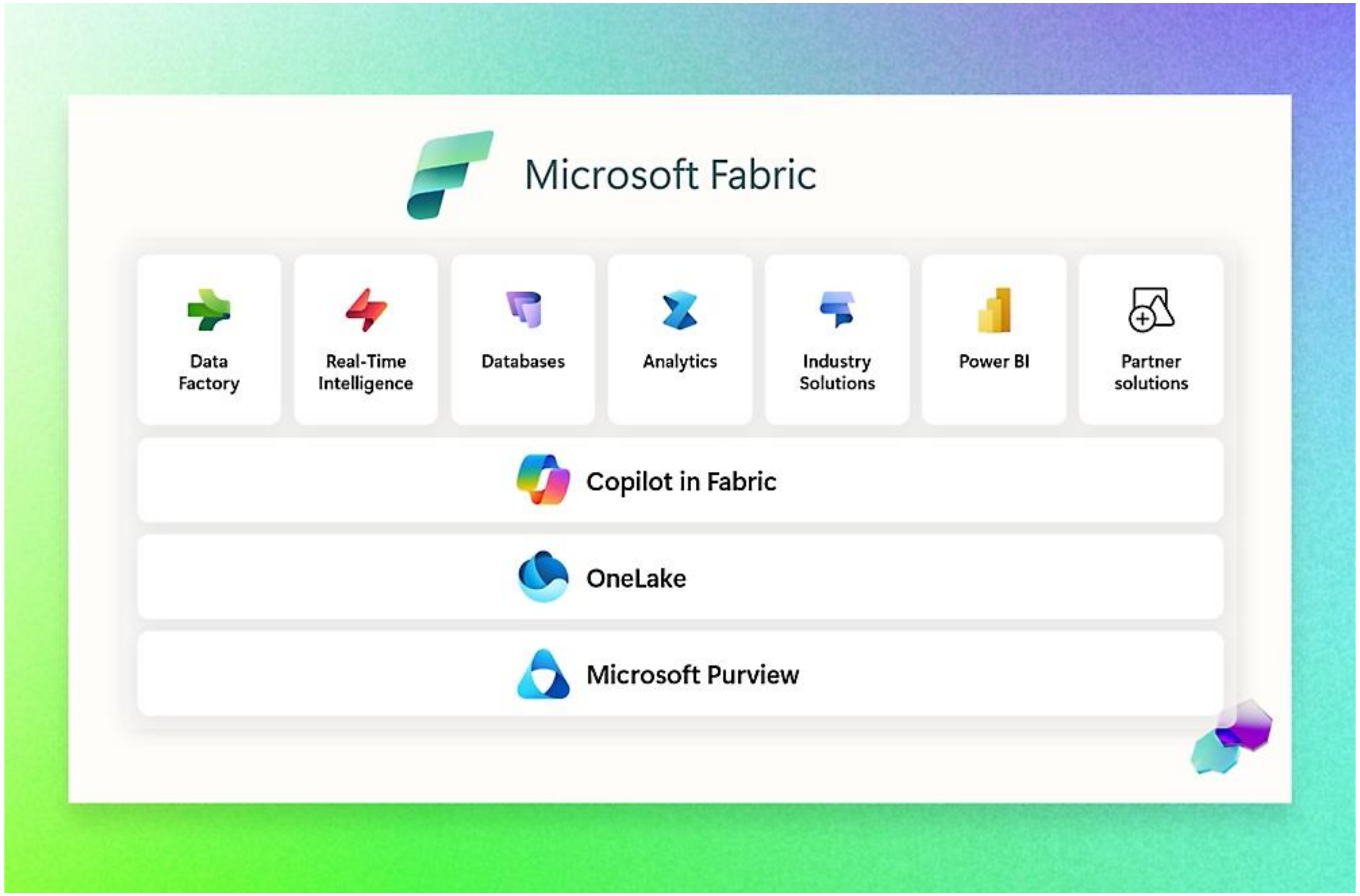
Overview of Grid stability risks

It’s about using **real-time and forecasted weather data** to anticipate conditions that could destabilize the power grid – before they happen. This enables utilities and grid operators to take **preventive actions** like load balancing, rerouting, or activating backup systems.

Weather Factors That Impact Grid Stability

Weather Element	Risk to Grid Infrastructure
High Temperature	Overheating of transformers and substations
Strong Winds	Damage to transmission lines and poles
Heavy Rain/Floods	Substation flooding, short circuits
Snow/Ice	Line sagging, equipment failure
Lightning	Surge damage, outages
Solar Radiation	Impacts solar generation forecasting

MicroSoft Fabric | Ecosystem



What is a Notebook in Microsoft Fabric?

The notebook serves as an interactive coding environment where users can use languages like Python (PySpark, Pandas) to perform tasks such as data ingestion, transformation, exploration, and analysis by loading data into dataframes for processing.

- A web-based interactive coding environment for data engineering and data science.
- Supports multiple languages: Python, R, Scala, and SQL.
- Ideal for data ingestion, transformation, visualization, and machine learning.
- Integrated with Lakehouse, pipelines, and other Fabric experiences.
- Offers rich visualizations, markdown support, and enterprise-grade security.

With a Fabric notebook, you can:

- Get started with zero set-up effort.
- Easily explore and process data with intuitive low-code experience.
- Keep data secure with built-in enterprise security features.
- Analyze data across raw formats (CSV, txt, JSON, etc.), processed file formats (parquet, Delta Lake, etc.), using powerful Spark capabilities.
- Be productive with enhanced authoring capabilities and built-in data visualization.

Core Features of Fabric Notebooks

Feature	Description
Multi-language Support	Write code in Python, Spark SQL, and markdown – all in one place.
Apache Spark Integration	Run distributed data processing jobs natively with Spark clusters.
Lakehouse Connectivity	Read/write directly to Lakehouse tables using shortcuts and workspace links.
Markdown Documentation	Add rich text, headers, and formatting to explain logic and annotate steps.
Built-in Visualizations	Use Python libraries (e.g., matplotlib, seaborn) or Spark display functions for charts.
Pipeline Integration	Trigger notebooks as activities in Fabric Data Factory pipelines.
Scheduler Support	Automate notebook runs with time-based triggers and security context control.
Security Context Awareness	Runs under user, pipeline owner, or scheduler identity – critical for governance.
Versioning & Export	Save, export, and track notebook changes for audit and collaboration.
Real-time Collaboration	Co-edit notebooks with team members in shared workspaces.

ETL Tools used in Microsoft Fabric

Lakehouse

- Served as the unified repository for raw, cleaned, and enriched weather data across multiple locations and timestamps.
- Structured Lakehouse into staging (raw API ingests) and final zones (transformed, validated datasets) for clear ETL separation.
- Seamlessly integrated with Spark Notebooks for Cross-Tool Compatibility.
- Provided fast, scalable access to weather data for Power BI dashboards and Spark transformations.

Spark Jobs

- Production grade batch processing jobs that need to run on schedule or be triggered by events.
- Can handle massive datasets & complex transformations using the full spark ecosystem.

Shortcuts

- Enabled seamless access to curated Lakehouse tables across multiple Fabric workspaces without duplicating data.
- Made curated datasets easily discoverable for Power BI reports without exposing raw ingestion zones.
- Enables cross-workspace access to curated datasets for scalable reuse and governance.

Notebooks

- Using PySpark, Scala , R, or SQL within Fabric Notebooks
- Gives complete programmatic control over Transformations.
- Ideal for complex business logic, machine learning preprocessing
- Exploit full power of distributed computing.
- Performs timestamp conversion, data cleaning, and enrichment across multi-location feeds.

Delta Tables

- Stores versioned, query-optimized weather data for downstream analytics and dashboards.

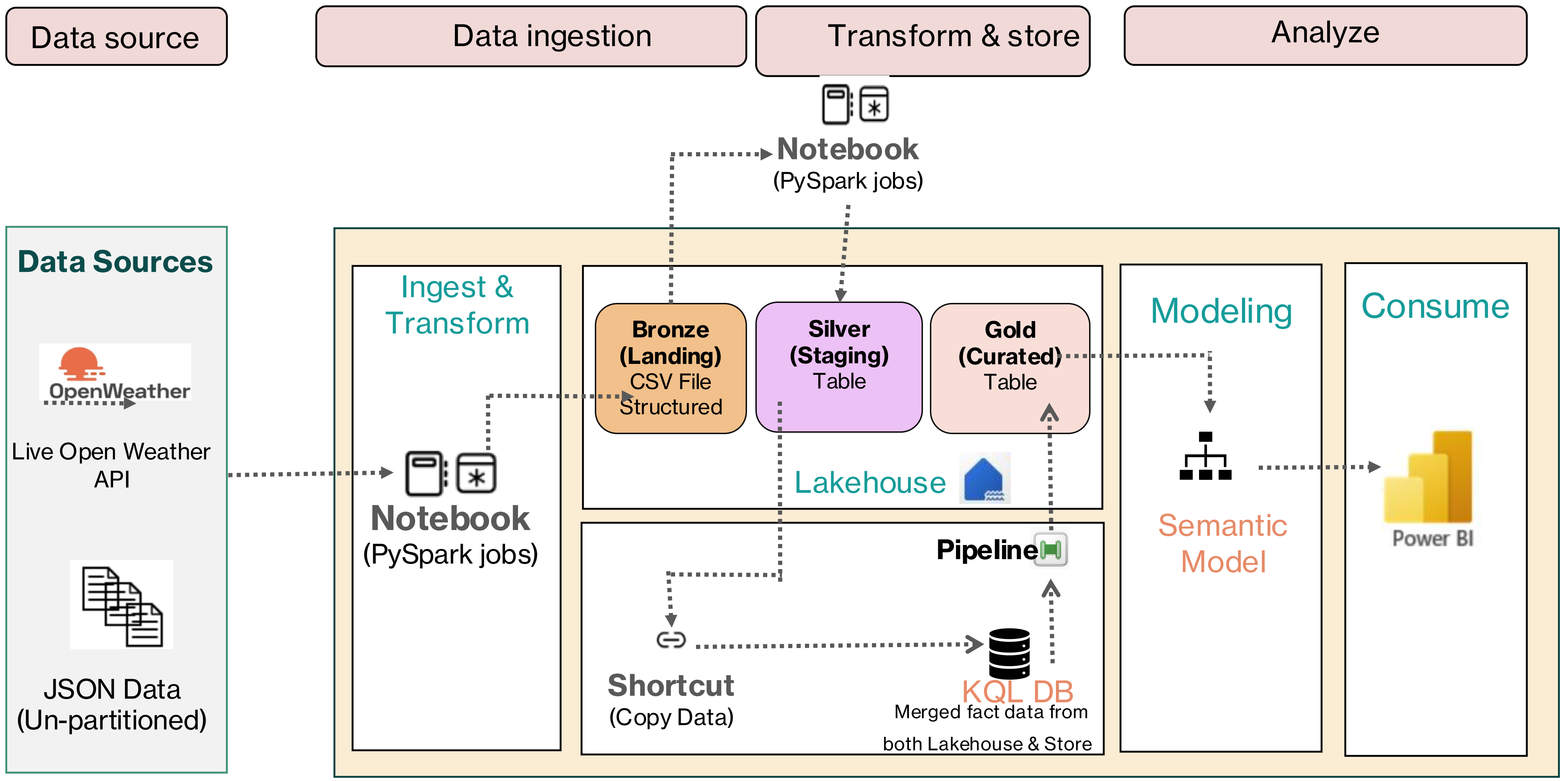
Real-time Analytics(KQL)

- For streaming data scenarios which need continuous ingestion & transformation.
- KQL provides powerful real-time processing capabilities

Pipeline

- **Source Connection**: Connects to diverse data sources (e.g., REST APIs, Azure Blob, SQL DBs, OneLake shortcuts).
- **Activities**: Includes Copy Data, Data Flow, Notebook, Spark Job, and custom scripts.
- **Triggers**: Supports manual, scheduled, and event-based triggers for pipeline execution.
- **Parameters**: Enables dynamic pipeline behavior using runtime parameters for flexibility.

Architecture in Microsoft Fabric

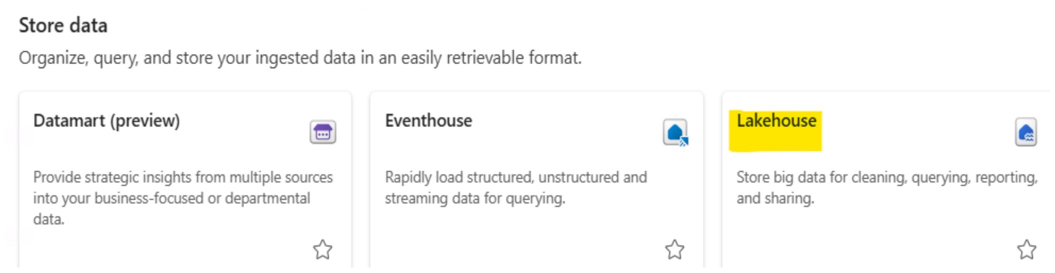


Create Notebooks-

1. Create a Lakehouse

You create a Lakehouse first to serve as a centralized, structured data storage for notebooks to access & manipulate. Notebooks are then used to query this data in the Lakehouse, performing tasks like data ingestion, transformation, analysis, and developing machine learning models. The Lakehouse provides a unified environment, allowing notebooks to interact directly with data without complex data movement, making data exploration and development efficient.

Workspace-> New Item-> Lakehouse



Store data

Organize, query, and store your ingested data in an easily retrievable format.

Datamart (preview)

Provide strategic insights from multiple sources into your business-focused or departmental data.

Eventhouse

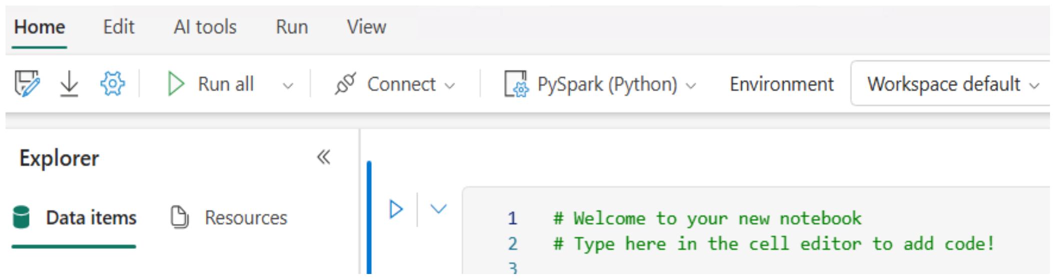
Rapidly load structured, unstructured and streaming data for querying.

Lakehouse

Store big data for cleaning, querying, reporting, and sharing.

2. Create a new notebook - You can easily create a new notebook from the Fabric Data Engineering homepage, the workspace New option, under workspace.

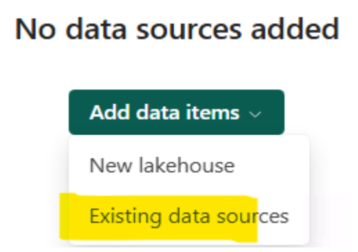
Workspace-> New Item-> Notebook



```
# Welcome to your new notebook
# Type here in the cell editor to add code!
```

3. Connect Lakehouse and notebooks

Fabric notebooks support close interactions with Lakehouse; you can easily add a new or existing Lakehouse from the Lakehouse explorer. This connection allow users to build data pipelines, train models, and gain insights from large datasets without needing to import or copy the data. So, We connect a Lakehouse to a notebook to programmatically access and manipulate data stored in the Lakehouse for data engineering, analytics, and AI workloads.



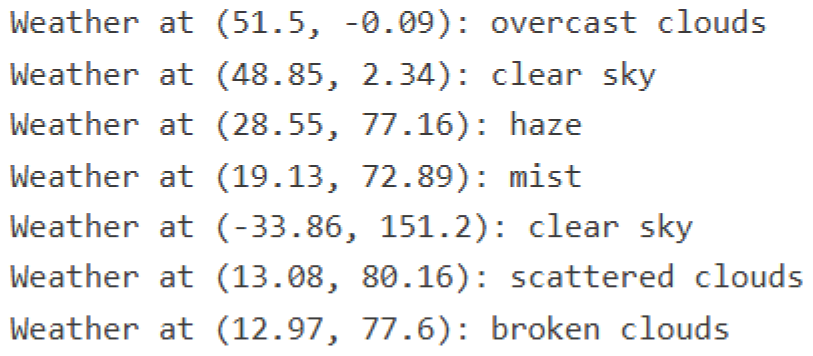
Add Data items-> Existing data sources-> select your Lakehouse-> connect

4. Query your PySpark/Spark SQL in Notebook

How I utilized Notebooks for Weather Data Use Case-

Step 1. Calling API with Parameters in Notebook using PySpark

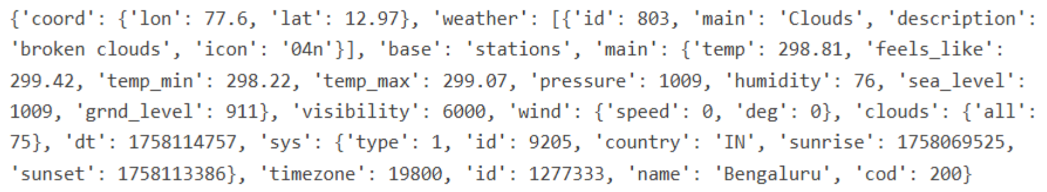
In 1st cell of notebook, I making a request to call API by passing API URL with passing few parameters [longitude & latitude] to fetch the data of all available locations available in API Data.



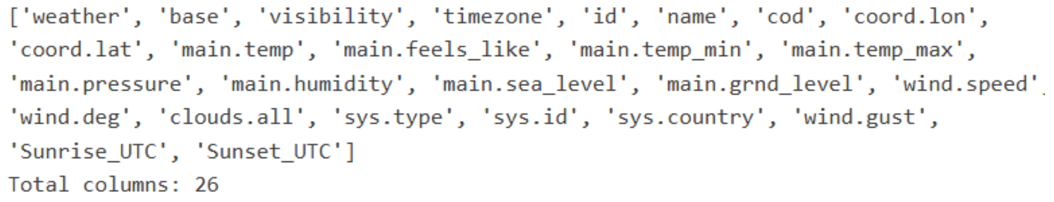
```
Weather at (51.5, -0.09): overcast clouds
Weather at (48.85, 2.34): clear sky
Weather at (28.55, 77.16): haze
Weather at (19.13, 72.89): mist
Weather at (-33.86, 151.2): clear sky
Weather at (13.08, 80.16): scattered clouds
Weather at (12.97, 77.6): broken clouds
```

Step 2. Checking no. of columns with column names

In 2nd block, I am checking the format of API data whether it is in JSON/Parquet. Also, I used Python Print code to get all the columns and length of columns available in this data.



```
{'coord': {'lon': 77.6, 'lat': 12.97}, 'weather': [{'id': 803, 'main': 'Clouds', 'description':
'broken clouds', 'icon': '04n'}], 'base': 'stations', 'main': {'temp': 298.81, 'feels_like':
299.42, 'temp_min': 298.22, 'temp_max': 299.07, 'pressure': 1009, 'humidity': 76, 'sea_level':
1009, 'grnd_level': 911}, 'visibility': 6000, 'wind': {'speed': 0, 'deg': 0}, 'clouds': {'all':
75}, 'dt': 1758114757, 'sys': {'type': 1, 'id': 9205, 'country': 'IN', 'sunrise': 1758069525,
'sunset': 1758113386}, 'timezone': 19800, 'id': 1277333, 'name': 'Bengaluru', 'cod': 200}
```

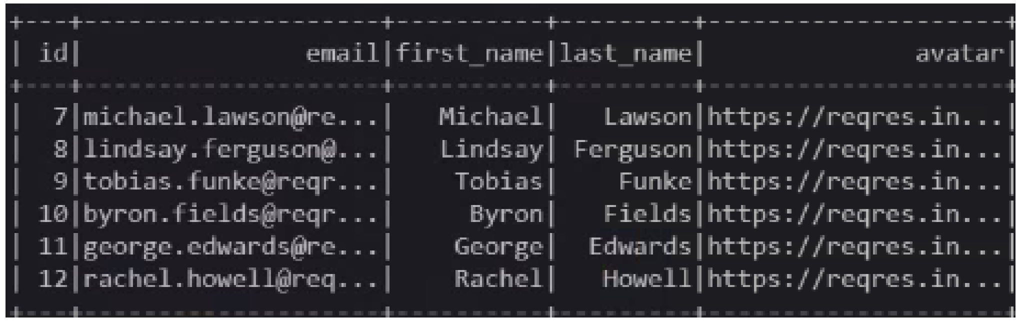


```
['weather', 'base', 'visibility', 'timezone', 'id', 'name', 'cod', 'coord.lon',
'coord.lat', 'main.temp', 'main.feels_like', 'main.temp_min', 'main.temp_max',
'main.pressure', 'main.humidity', 'main.sea_level', 'main.grnd_level', 'wind.speed',
'wind.deg', 'clouds.all', 'sys.type', 'sys.id', 'sys.country', 'wind.gust',
'Sunrise_UTC', 'Sunset_UTC']
Total columns: 26
```

Step 3. Transform the JSON response to a DataFrame using Pandas

In 3rd block, I import Pandas library & created DataFrame because the data we extracted from API is Semi-Structured which we stored in a raw output as a JSON or binary file. So, we can convert the raw output into structured format we can easily store it in Delta Table. Also, I added static column which shows status like Current.

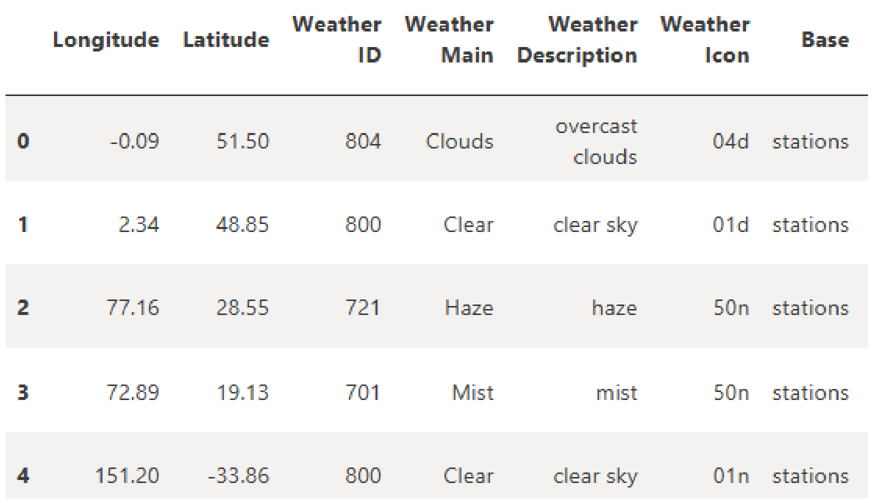
Now the data is shown in a row column format but still not in a fine table.



```
+---+--------------------+----------+---------+--------------------+
| id|               email|first_name|last_name|              avatar|
+---+--------------------+----------+---------+--------------------+
|  7|michael.lawson@re...|   Michael|   Lawson|https://reqres.in...|
|  8|lindsay.ferguson@...|   Lindsay| Ferguson|https://reqres.in...|
|  9|tobias.funke@reqr...|    Tobias|    Funke|https://reqres.in...|
| 10|byron.fields@reqr...|     Byron|   Fields|https://reqres.in...|
| 11|george.edwards@re...|    George|  Edwards|https://reqres.in...|
| 12|rachel.howell@req...|    Rachel|   Howell|https://reqres.in...|
+---+--------------------+----------+---------+--------------------+
```

Step 4. Write the Data Frame to a Delta Table

In 4th block, I format the above output in a fine table which shows the data properly.



	Longitude	Latitude	Weather ID	Weather Main	Weather Description	Weather Icon	Base
0	-0.09	51.50	804	Clouds	overcast clouds	04d	stations
1	2.34	48.85	800	Clear	clear sky	01d	stations
2	77.16	28.55	721	Haze	haze	50n	stations
3	72.89	19.13	701	Mist	mist	50n	stations
4	151.20	-33.86	800	Clear	clear sky	01n	stations

Step 5. Filter the table data

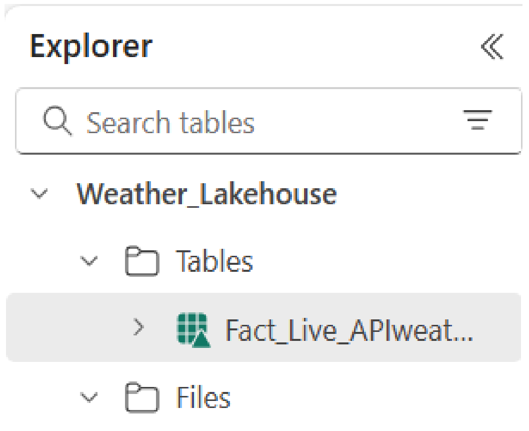
Format the unsupported column names like space, brackets & correct the format of sunrise, sunset date time data which we were getting in Unix Timestamp earlier.

Step 6. Export Data Frame into CSV format

Step 7. Save that csv file in my Lakehouse

Step 8. Loading table into existing Lakehouse

Step 9. We can refresh & check in Lakehouse, the table is there.



How I schedule a refresh in notebook-

To fetch the fresh data daily from a live API using a **Microsoft Fabric notebook**, you can automate the process using a combination of **notebook scheduling.** If you don't schedule a refresh run, it will give you the old data of that day when you manually run the code of notebook.

Step 1. In my notebook, I don't want to run each code cell in schedule run because it has checks also like total columns, total rows, show tables. To exclude them from daily run I will freeze those cells.

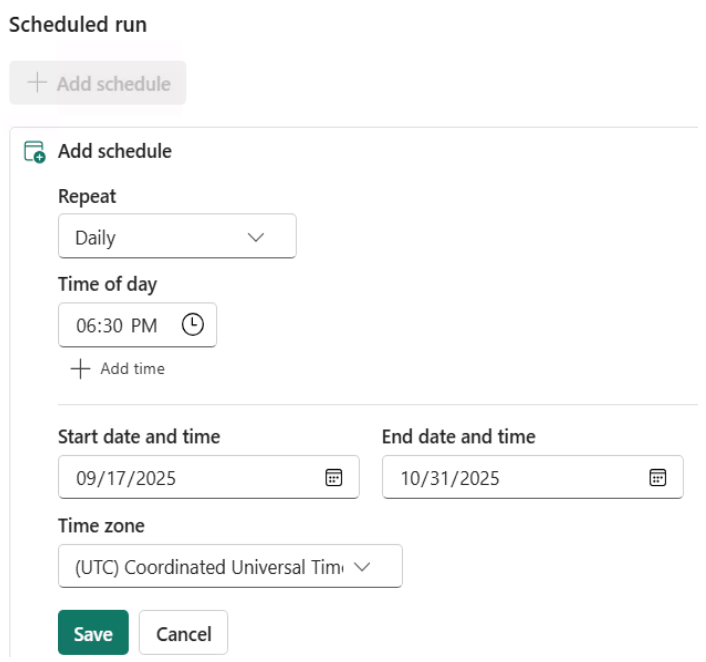
Step 2. Go to 'Run' in top & choose schedule and give details when you want to run the refresh of data.

Step 3. I select the schedule as daily once in a day, time of 6:30PM because it is present in UTC which is 12AM in IST and start from 17th Sept till 31st Oct.

Step 4. Save the schedule and add, it will show like in image. It will show the next schedule run also.

Step 5. Now you will see my data will refresh every day & you can see the recent runs also mentioning manual run or schedule run details.

Note: I checked the data by scheduled run in every 7 min & found it is placing the file in Lakehouse every time when it ran. To reduce this, I used to overwrite query to refresh the same file instead of creating new.



Scheduled run

Add schedule

Add schedule

Repeat: Daily

Time of day: 06:30 PM

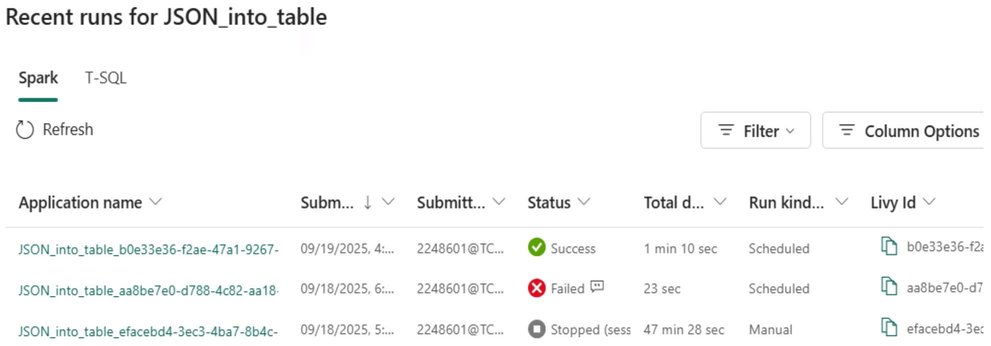
Add time

Start date and time: 09/17/2025

End date and time: 10/31/2025

Time zone: (UTC) Coordinated Universal Tim

Save | Cancel

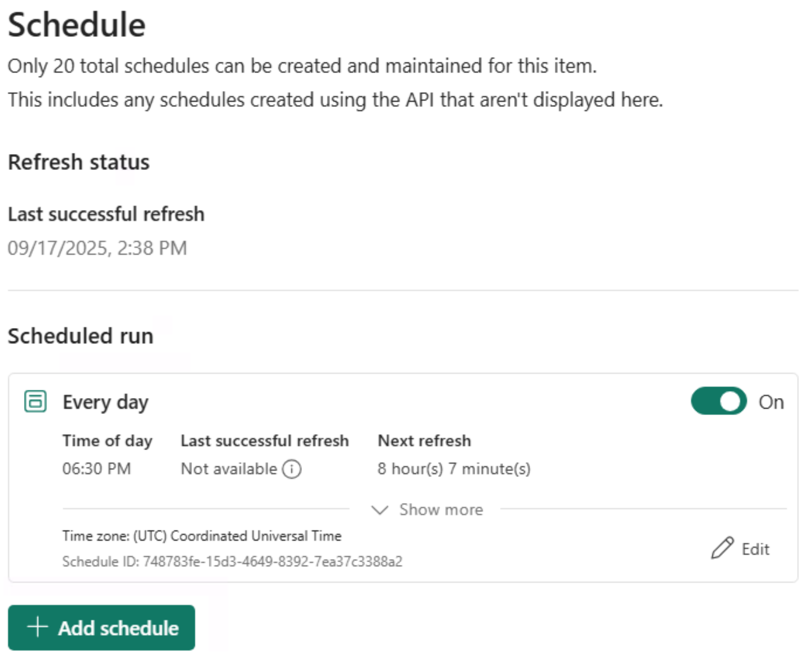


Recent runs for JSON_into_table

Spark | T-SQL

Refresh | Filter | Column Options

Application name	Subm...	Submitt...	Status	Total d...	Run kind...	Livy Id
JSON_into_table_b0e33e36-f2ae-47a1-9267-	09/19/2025, 4:...	2248601@TC...	✓ Success	1 min 10 sec	Scheduled	b0e33e36-f2
JSON_into_table_aa8be7e0-d788-4c82-aa18-	09/18/2025, 6:...	2248601@TC...	✗ Failed	23 sec	Scheduled	aa8be7e0-d7
JSON_into_table_efacebd4-3ec3-4ba7-8b4c-	09/18/2025, 5:...	2248601@TC...	Stopped (sess	47 min 28 sec	Manual	efacebd4-3e



Schedule

Only 20 total schedules can be created and maintained for this item.
This includes any schedules created using the API that aren't displayed here.

Refresh status

Last successful refresh
09/17/2025, 2:38 PM

Scheduled run

Every day — On

Time of day: 06:30 PM
Last successful refresh: Not available
Next refresh: 8 hour(s) 7 minute(s)

Show more

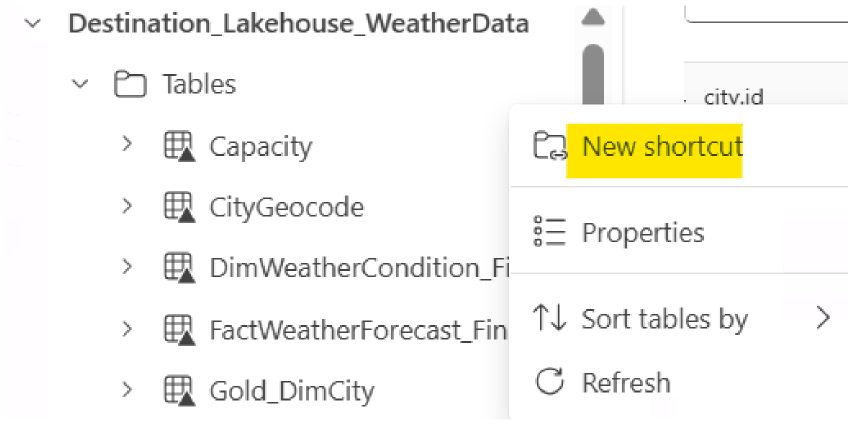
Time zone: (UTC) Coordinated Universal Time
Schedule ID: 748783fe-15d3-4649-8392-7ea37c3388a2

Edit

Add schedule

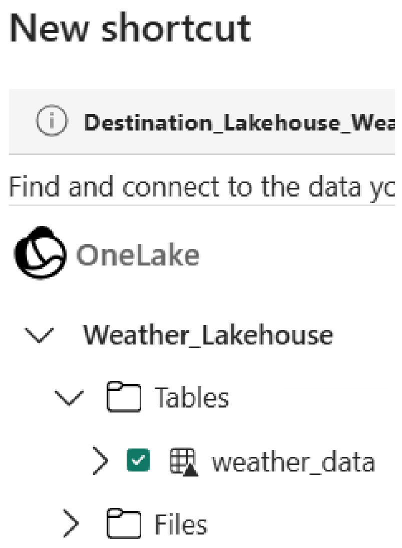
Shortcut: How I pull data from source Lakehouse & Copy to destination Lakehouse

Step 1. Inside destination Lakehouse, in tables click on ellipse and create shortcut.



Step 2. Choose OneLake & select your source Lakehouse.

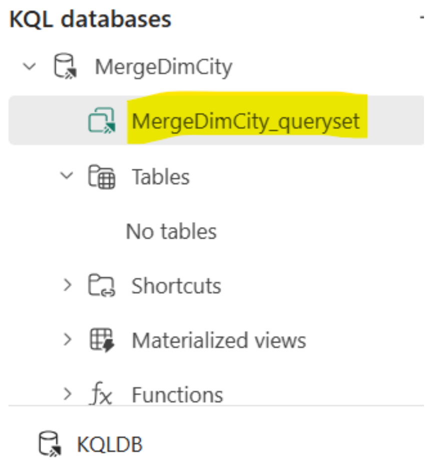
Step 3. Choose the table/file you want to copy and hit next.

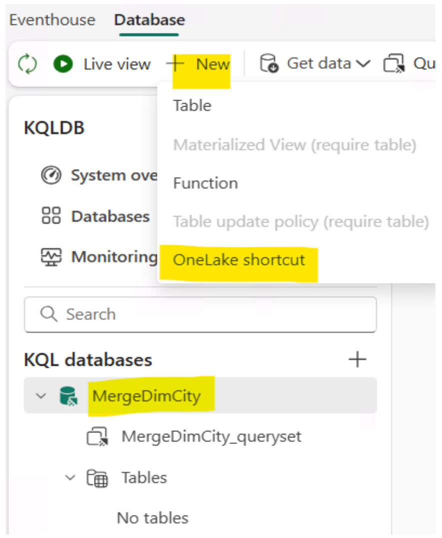


How I created KQL Database in MS Fabric

Event house: Create or use an existing Event house – this is the container for your KQL databases.

- **KQL Database**: In the Event house Explorer, select **New database** under KQL Databases. Choose between:
  - **New database** (default)
  - **New shortcut database** (follower)
- Once your KQL database is created, an attached **query environment** appears.
- Select the **KQLdatabasename_queryset** to open the editor.
- Start writing KQL queries to explore your data.





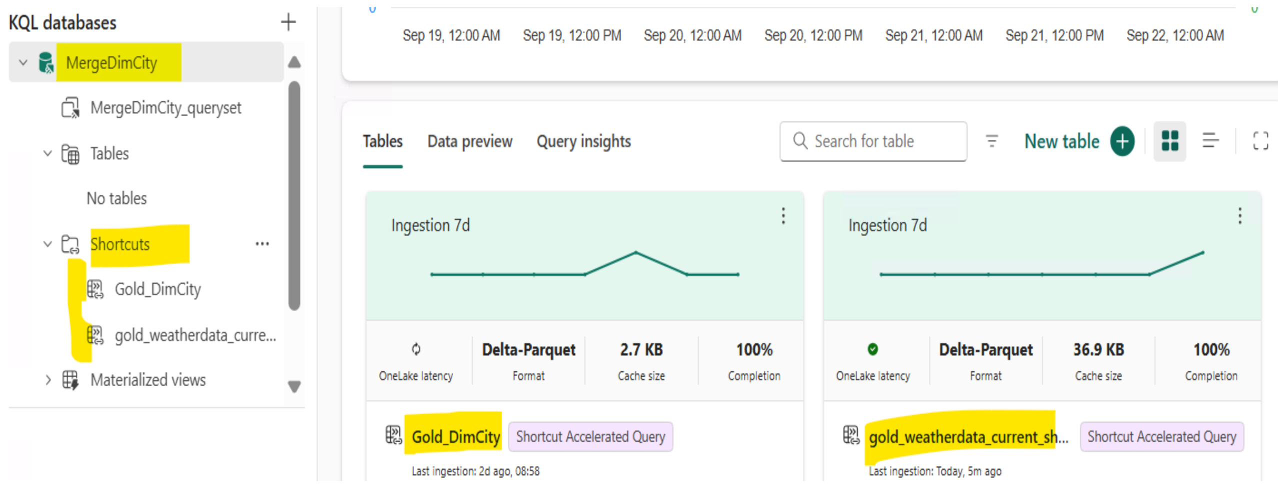
How to Query Lakehouse Table from KQL Database

1. Attach the Lakehouse to your KQL Database- Before querying, ensure your Lakehouse is attached to the KQL database:

- Go to your KQL database in Fabric.
- Click **"+ New"** → **"OneLake Shortcut"** → Choose OneLake.
- Select the Lakehouse that contains your table & select tables which you want to query in KQL.

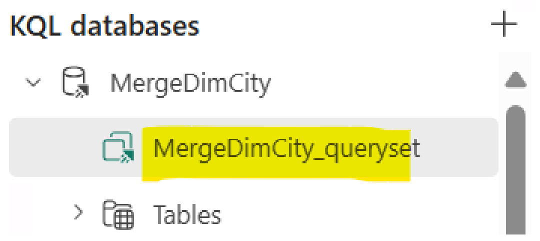
This creates a logical link so KQL can access the Delta tables stored in the Lakehouse.

2. Explore Available Tables - Once attached, you can preview available tables as shown in image.



How I Merge the data using KQL in Real-Time Analytics

1. In KQL DB which you created is having in built query set file. Open it to run query in KQL.



2. I firstly gave my table name in query & performed left join operation to get specific columns from another table.

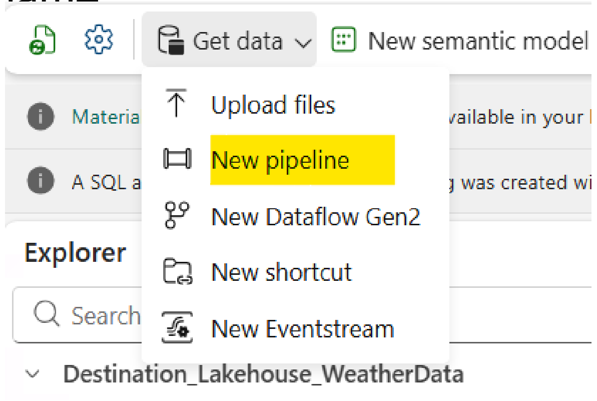
```
external_table('Table A')
| join kind=inner     (
    external_table('Table B')
    | project column_names
) on $left.common_column == $right.common_column
 | project column_names
```

3. I merged another table which is fact tables of both source Lakehouse & destination Lakehouse using union to add all required columns of both fact data..

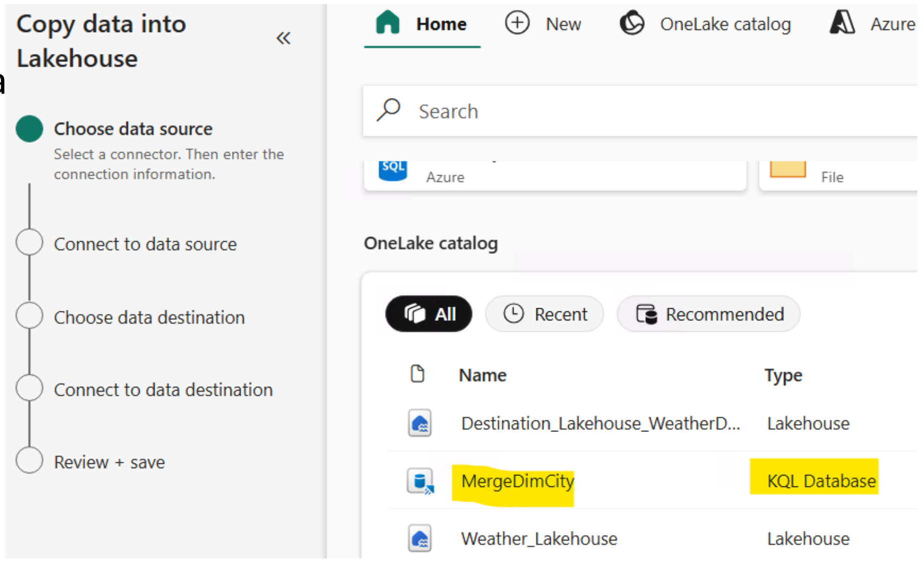
```
Union(
'Table A'
| project   column_names
),
(
    'Table B'
    | project column_names
)
```

How I loaded **KQL Data Tables** in Target Lakehouse using **Real-Time Analytics Pipeline**

1. In workspace go to destination Lakehouse > Get Data > New Pipeline>Give name



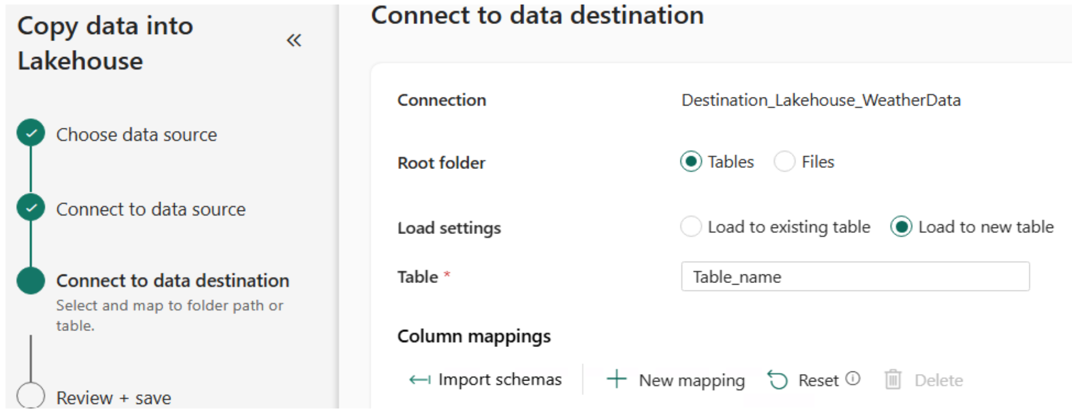
2. Select data


Copy data into Lakehouse

- Choose data source
  Select a connector. Then enter the connection information.
- Connect to data source
- Choose data destination
- Connect to data destination
- Review + save

Name	Type
Destination_Lakehouse_WeatherD...	Lakehouse
MergeDimCity	KQL Database
Weather_Lakehouse	Lakehouse

3. Paste the KQL Query & hit next.


Copy data into Lakehouse

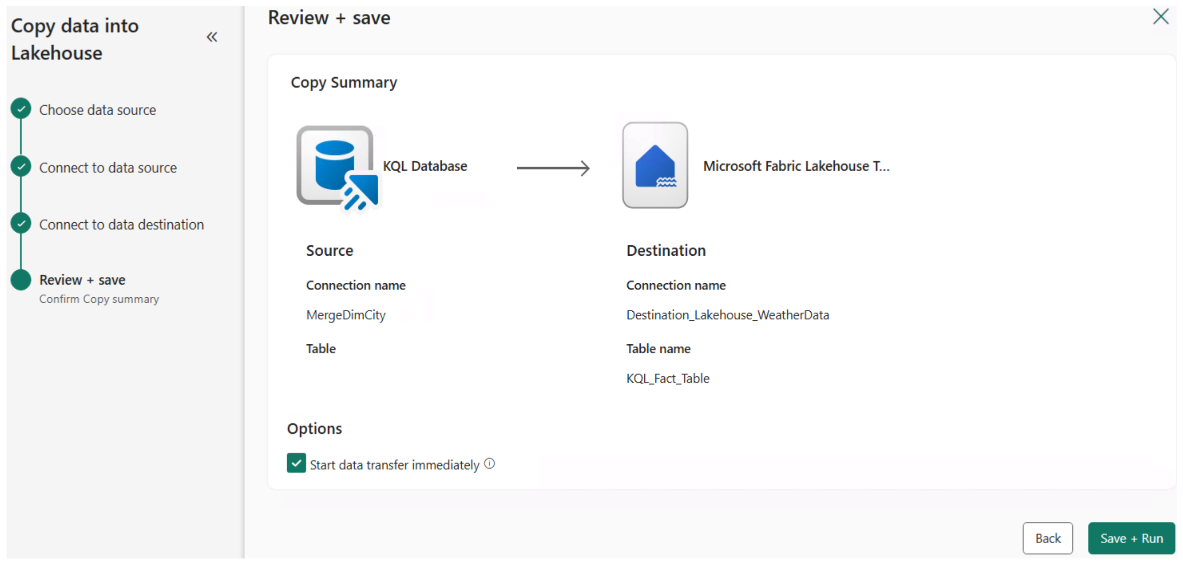
- Choose data source
- Connect to data source
- Connect to data destination
  Select and map to folder path or table.
- Review + save

Connect to data destination

Connection	Destination_Lakehouse_WeatherData
Root folder	[✓ Tables] Files
Load settings	Load to existing table [✓ Load to new table]
Table *	Table_name

Column mappings

4. Select destination lakehouse & give new table name & hit next.


Copy data into Lakehouse

- Choose data source
- Connect to data source
- Connect to data destination
- Review + save
  Confirm Copy summary

Review + save

Copy Summary

KQL Database → Microsoft Fabric Lakehouse T...

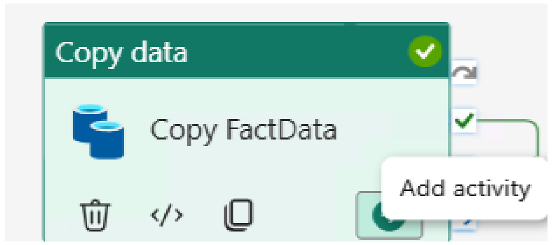
Source	Destination
Connection name	Connection name
MergeDimCity	Destination_Lakehouse_WeatherData
Table	Table name
	KQL_Fact_Table

Options

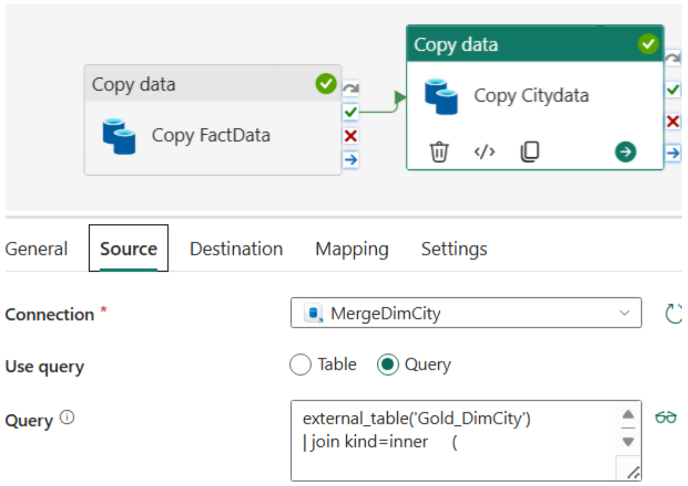
[✓] Start data transfer immediately

5. I have another query also in KQL which I need to load in table form in destination Lakehouse.

6. Open same pipeline and click on add activity on existing copy data activity block.



7. Click on Copy Data Activity & enter Source as KQL DB & paste that query.



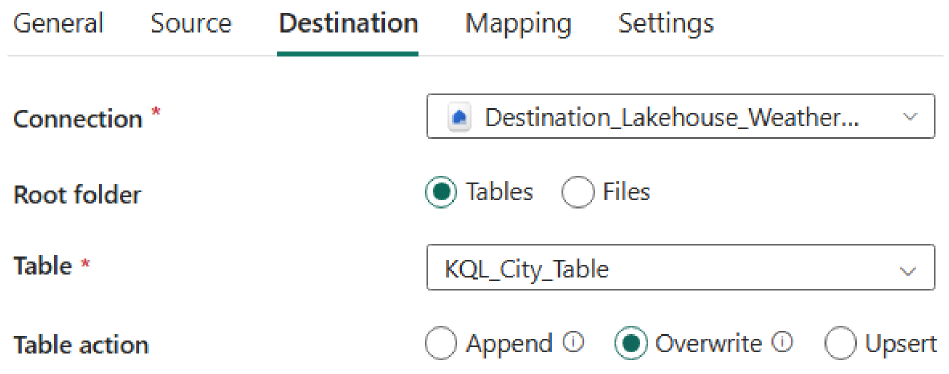
Connection * MergeDimCity

Use query: Table [✓ Query]

Query

```
external_table('Gold_DimCity')
| join kind=inner    (
```

8. Give destination details and give new name in table.



Connection * Destination_Lakehouse_Weather...

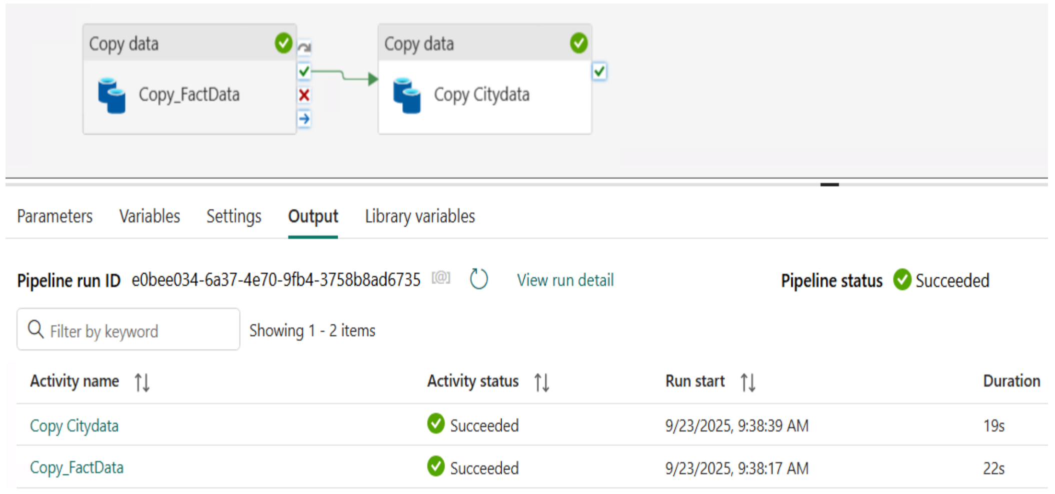
Root folder: [✓ Tables] Files

Table * KQL_City_Table

Table action: Append [✓ Overwrite] Upsert

9. Validate & run the pipeline.

10. You can see my tables are loaded in lakehouse after pipeline ran.



Pipeline run ID e0bee034-6a37-4e70-9fb4-3758b8ad6735

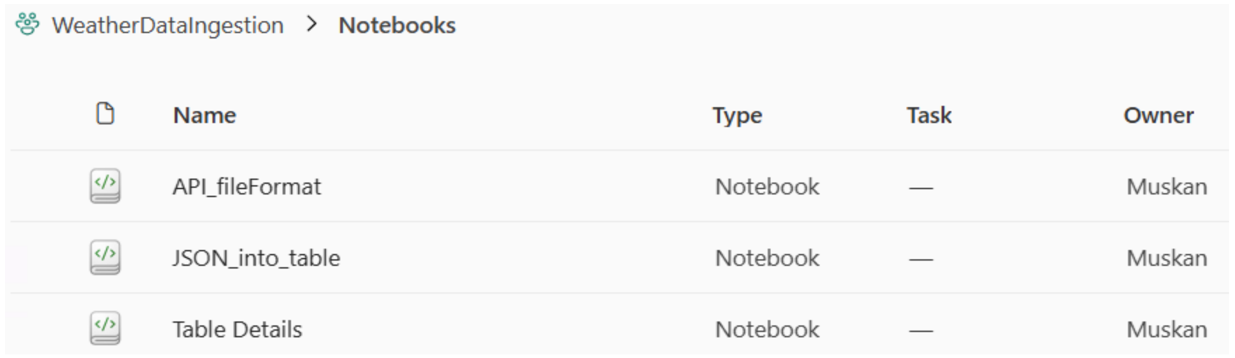
Pipeline status Succeeded

Showing 1 - 2 items

Activity name	Activity status	Run start	Duration
Copy Citydata	Succeeded	9/23/2025, 9:38:39 AM	19s
Copy_FactData	Succeeded	9/23/2025, 9:38:17 AM	22s

Microsoft Fabric | Ingest Weather API Data

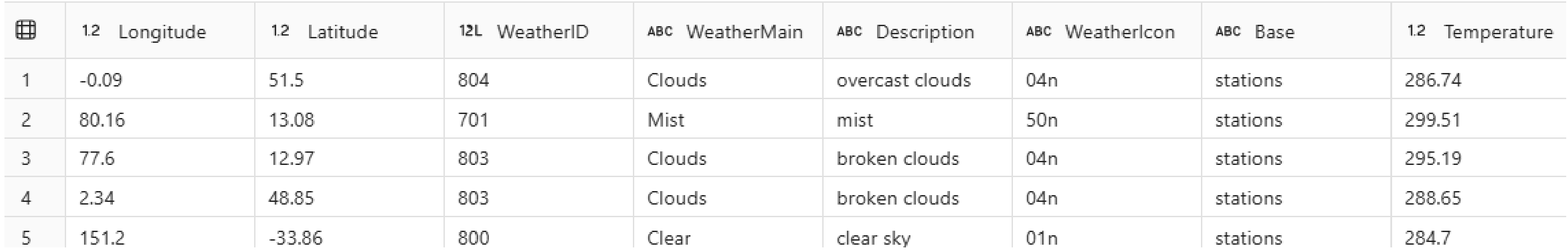
Created Notebooks for data ingestion



WeatherDataIngestion > Notebooks

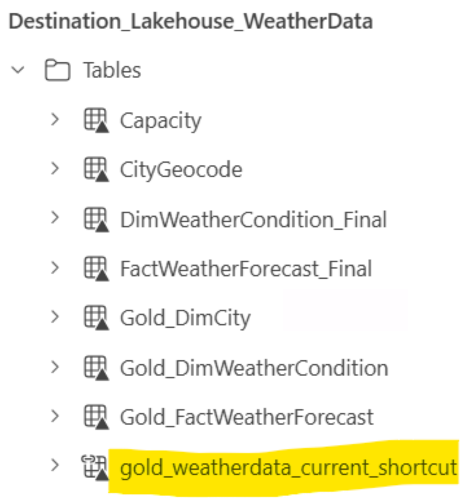
Name	Type	Task	Owner
API_fileFormat	Notebook	—	Muskan
JSON_into_table	Notebook	—	Muskan
Table Details	Notebook	—	Muskan

Ingested in Lakehouse using Pyspark



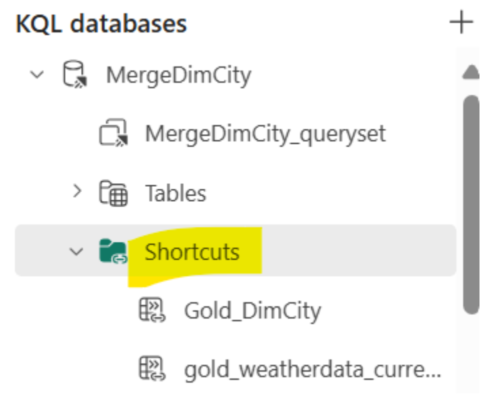
	Longitude	Latitude	WeatherID	WeatherMain	Description	WeatherIcon	Base	Temperature
1	-0.09	51.5	804	Clouds	overcast clouds	04n	stations	286.74
2	80.16	13.08	701	Mist	mist	50n	stations	299.51
3	77.6	12.97	803	Clouds	broken clouds	04n	stations	295.19
4	2.34	48.85	803	Clouds	broken clouds	04n	stations	288.65
5	151.2	-33.86	800	Clear	clear sky	01n	stations	284.7

Created shortcut to copy table in silver Lakehouse

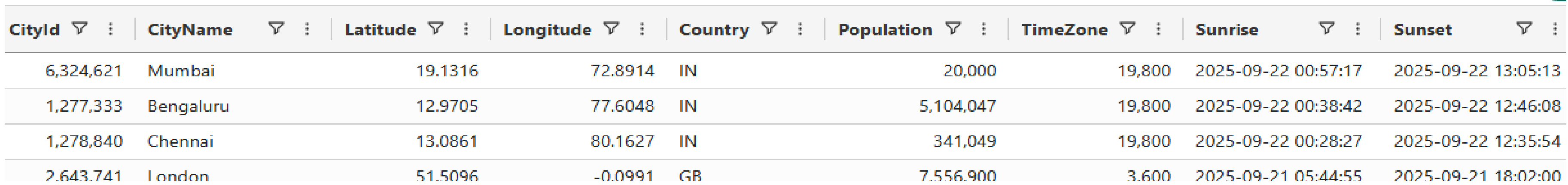


Destination_Lakehouse_WeatherData
- Tables
  - Capacity
  - CityGeocode
  - DimWeatherCondition_Final
  - FactWeatherForecast_Final
  - Gold_DimCity
  - Gold_DimWeatherCondition
  - Gold_FactWeatherForecast
  - gold_weatherdata_current_shortcut

Created shortcut in KQL DB to load lakehouse tables

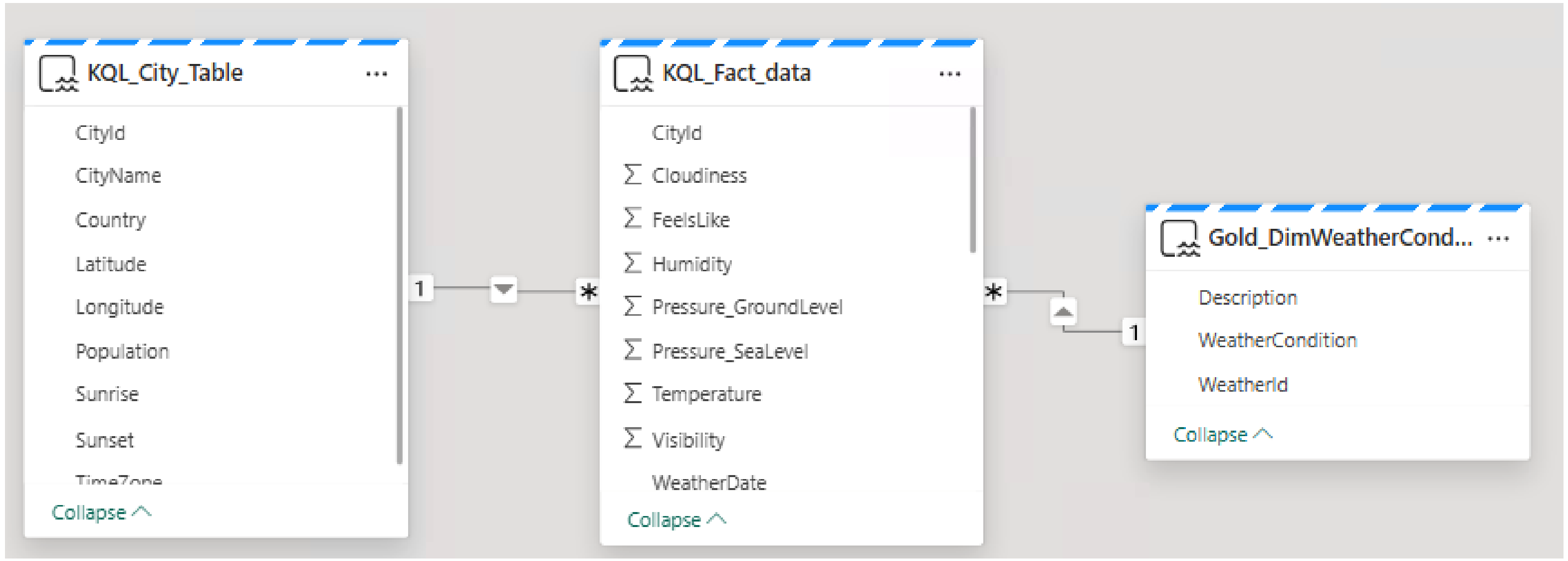

KQL databases
- MergeDimCity
  - MergeDimCity_queryset
  - Tables
  - Shortcuts
    - Gold_DimCity
    - gold_weatherdata_curre...

Merged fact data using KQL & load in lakehouse



CityId	CityName	Latitude	Longitude	Country	Population	TimeZone	Sunrise	Sunset
6,324,621	Mumbai	19.1316	72.8914	IN	20,000	19,800	2025-09-22 00:57:17	2025-09-22 13:05:13
1,277,333	Bengaluru	12.9705	77.6048	IN	5,104,047	19,800	2025-09-22 00:38:42	2025-09-22 12:46:08
1,278,840	Chennai	13.0861	80.1627	IN	341,049	19,800	2025-09-22 00:28:27	2025-09-22 12:35:54
2,643,741	London	51.5096	-0.0991	GB	7,556,900	3,600	2025-09-21 05:44:55	2025-09-21 18:02:00

Created Semantic Model using Gold tables



KPI for Grid Stability :

KPI	Formula	Thresholds/Insights
Wind Power Potential Index (WPPI)	wind.speed / cut_in_speed (assume cut-in = 3 m/s)	<1 = no power, 1–3 = good generation
Cloudiness Index (CI)	Avg Cloudiness /100	➢ 0.7 → output drop
Irradiance Proxy (IrrProxy)	(1 - CI) × 1000 W/m^2	Used to forecast solar generation

KPI for Predictive Maintenance:

KPI	Formula	Thresholds/Insights
Temperature Stress Index (TSI)	(avg temp - 25) / 25	High → overheating of inverters/panels
Icing Risk Index (IRI)	If (-5 ≤ temp ≤ 2) AND humidity > 80% AND precipitation > 0 then 1 else 0	1 = Icing risk (turbine imbalance/panel loss)
Humidity Stress Index (HSI)	humidity / 100	➢ 0.8 → electrical/corrosion risk
Grid Risk Score	Total number of risky hours/ Total number of hours	0.0 - 0.3 -- low Risk 0.3 - 0.6 -- Medium Risk 0.6 - 1 – High Risk

Power BI semantic model :

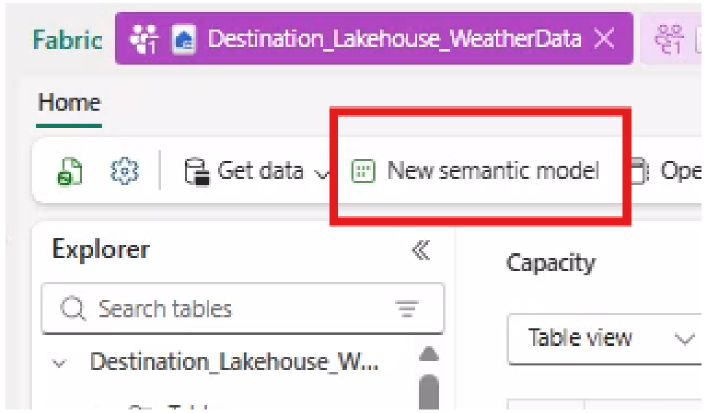
A **semantic model** in Microsoft Fabric is a logical representation of an analytical domain, designed to facilitate deeper analysis and reporting. It typically follows a star schema, with facts representing the domain and dimensions allowing for detailed analysis, filtering, and calculations.

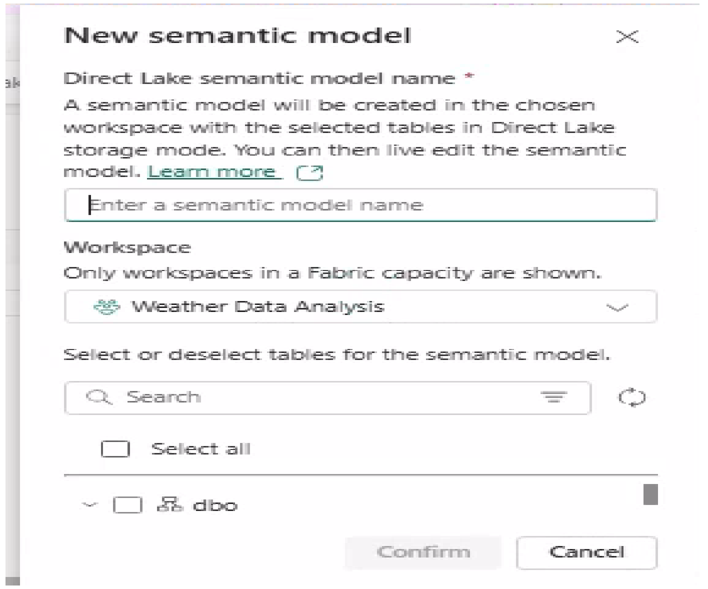
1. Open the lakehouse and select **New Power BI semantic model** from the ribbon.

2. Enter a name for the new semantic model, select a workspace to save it in, and pick the tables to include. Then select **Confirm**.

3. The new Power BI semantic model can be edited in the workspace, where you can add relationships, measures, rename tables and columns, choose how values are displayed in report visuals, and much more.

4. To edit the Power BI semantic model later, select **Open data model** from the semantic model context menu




New semantic model

Direct Lake semantic model name *

A semantic model will be created in the chosen workspace with the selected tables in Direct Lake storage mode. You can then live edit the semantic model. Learn more

Enter a semantic model name

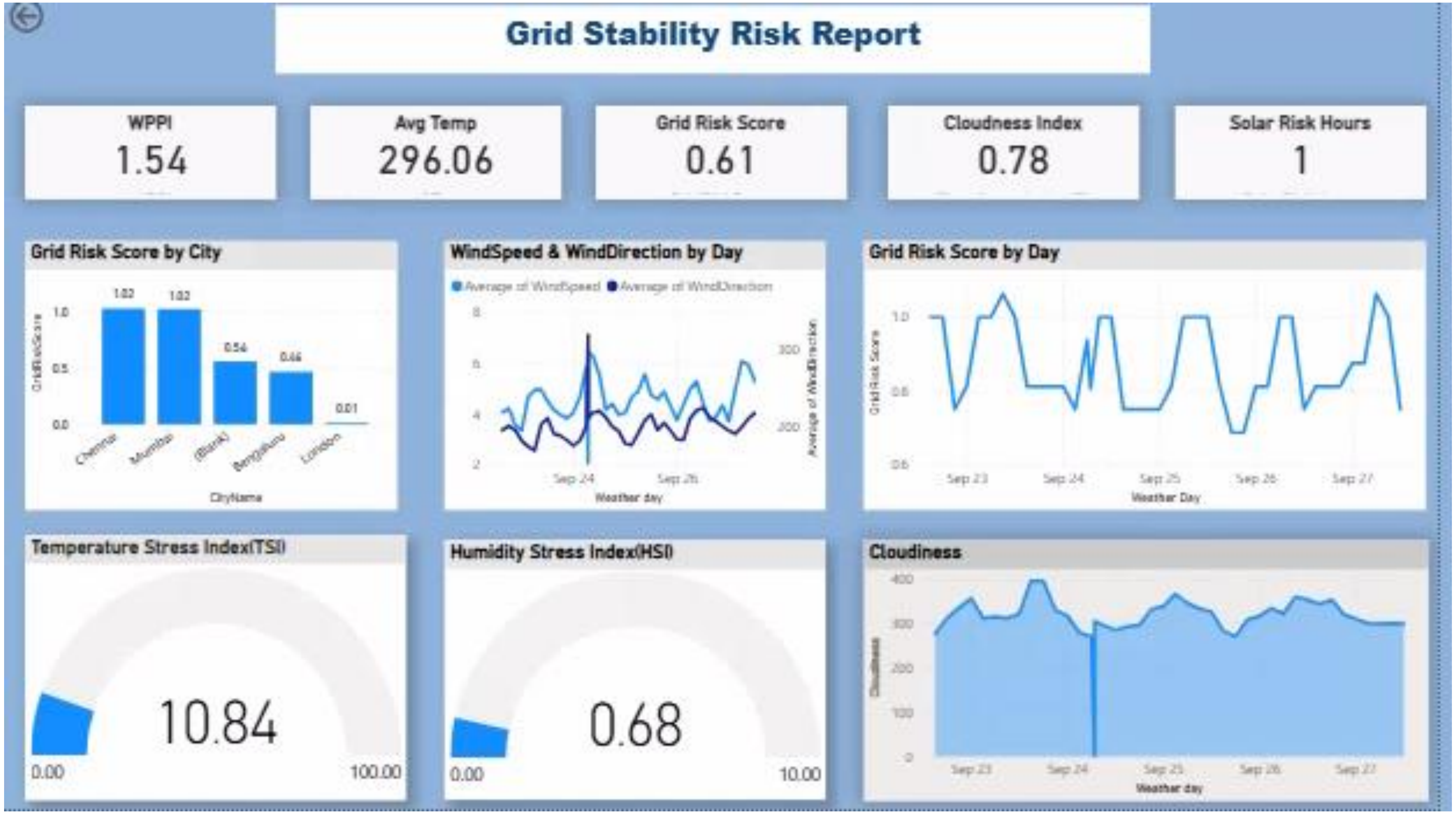
Workspace

Only workspaces in a Fabric capacity are shown.

Weather Data Analysis

Select or deselect tables for the semantic model.

Power BI Report :



Grid Stability Risk Report

WPPI 1.54

Avg Temp 296.06

Grid Risk Score 0.61

Cloudness Index 0.78

Solar Risk Hours 1