

Smart Manufacturing Data Analytics Platform – End-to-End DevOps & DataOps

1. Problem Statement

AutoForge Ltd., a global automotive manufacturer, struggles with:

- Delayed analytics due to siloed legacy systems.
- Frequent deployment errors causing production data delays.
- Lack of real-time monitoring for IoT-based predictive maintenance.
- Inefficient model retraining due to absence of drift monitoring.
- Manual, error-prone deployment processes without rollback capability.

Creating Data Flow

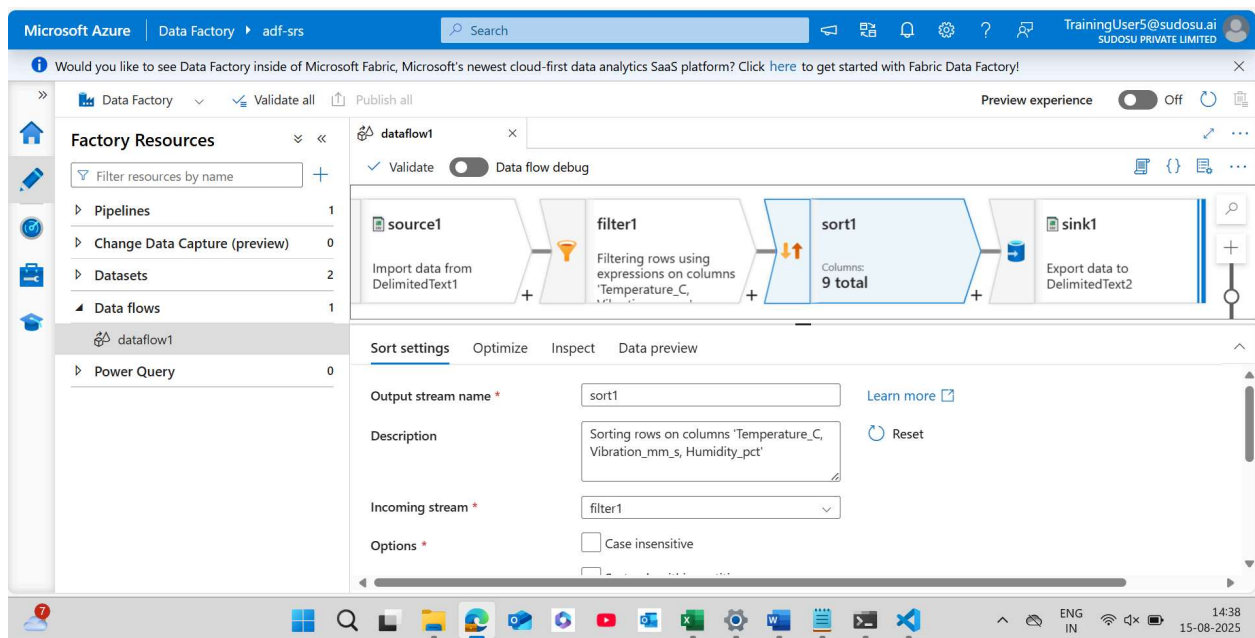
Source (CSV in Blob) → Sort → Filter → Sink (CSV/Parquet in Blob)

The screenshot displays the Microsoft Azure Data Factory portal interface. The top navigation bar shows the URL <https://adf.azure.com/en/authoring/dataflow/dataflow1?factory=%2Fsubscriptions%2F800168c9-2f62-4abb-a3e5-4372039d5366%2FresourceGroups%2F...> and the user profile 'TrainingUser5@sudoai'. The left sidebar shows the 'Factory Resources' tree with 'Data flows' selected, listing 'dataflow1'. The main canvas displays a data flow named 'dataflow1' with four components: 'source1' (Import data from DelimitedText1), 'filter1' (Columns: 9 total), 'sort1' (Sorting rows on columns 'Temperature_C', 'Vibration_mm_s'), and 'sink1' (Export data to DelimitedText2). Below the canvas, the 'Filter settings' tab is active for 'filter1', showing the 'Filter on' expression: `toDouble(Temperature_C) > 68 && toDouble(Vibration_mm_s) > 1.9`. The bottom status bar shows the time '14:38' and date '15-08-2025'.

In theory, a data flow is the logical movement of data from a source system, through one or more transformation steps, to a target (sink) system.

In your case, the data flows from the source dataset in Blob Storage, undergoes sorting and filtering transformations to arrange and refine the records, and is then written to the sink location for further use.

- Sort: You're arranging rows based on one or more columns.(Ascending)
- Filter: You're keeping only rows that match a condition (for example, Temperature_C > 70 AND Vibration_mm_s > 2.0).



Running a pipeline from a data flow after applying transformations to the notebook means that the pipeline orchestrates the process of fetching raw data from the source, executing the defined data flow logic (such as sorting and filtering), and then triggering the notebook to perform additional processing or analytics. The data flow handles structured ETL tasks, while the notebook adds custom code-based operations, creating a seamless end-to-end data processing workflow.

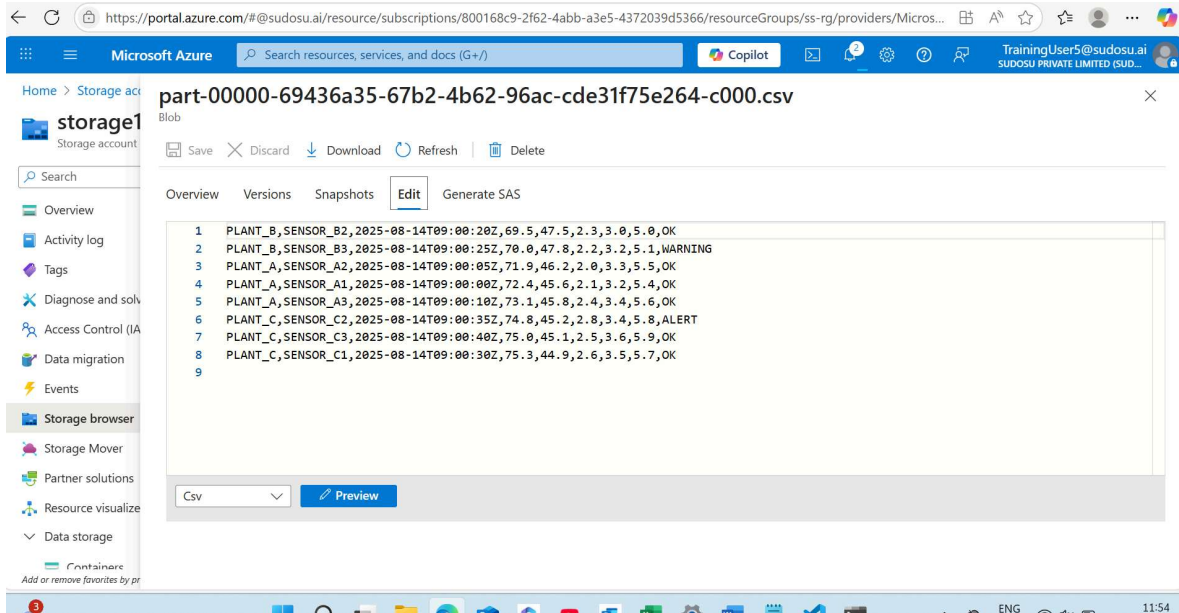
The screenshot shows the 'All pipeline runs' page for 'pipeline1 - Activity runs' in the Microsoft Azure Data Factory portal. The left sidebar contains navigation options: Dashboards, Runs, Pipeline runs (selected), Trigger runs, Change Data Capture (previous), Runtimes & sessions, Integration runtimes, Data flow debug, Notifications, and Alerts & metrics. The main content area displays a visual representation of the pipeline with a 'Data flow' activity (Data flow1) and a 'Notebook' activity (Notebook1). Below this, the 'Activity runs' section shows a table of pipeline runs. The table has columns for Activity name, Activity status, Activity type, Run start, Duration, and Integration runtime. Two runs are listed: Notebook1 (In progress) and Data flow1 (Succeeded).

Activity name	Activity status	Activity type	Run start	Duration	Integration runtime
Notebook1	In progress	Notebook	8/15/2025, 2:22:16 PM	18s	
Data flow1	Succeeded	Data flow	8/15/2025, 2:18:57 PM	3m 19s	AutoResolveIntegrationRuntime (East US)

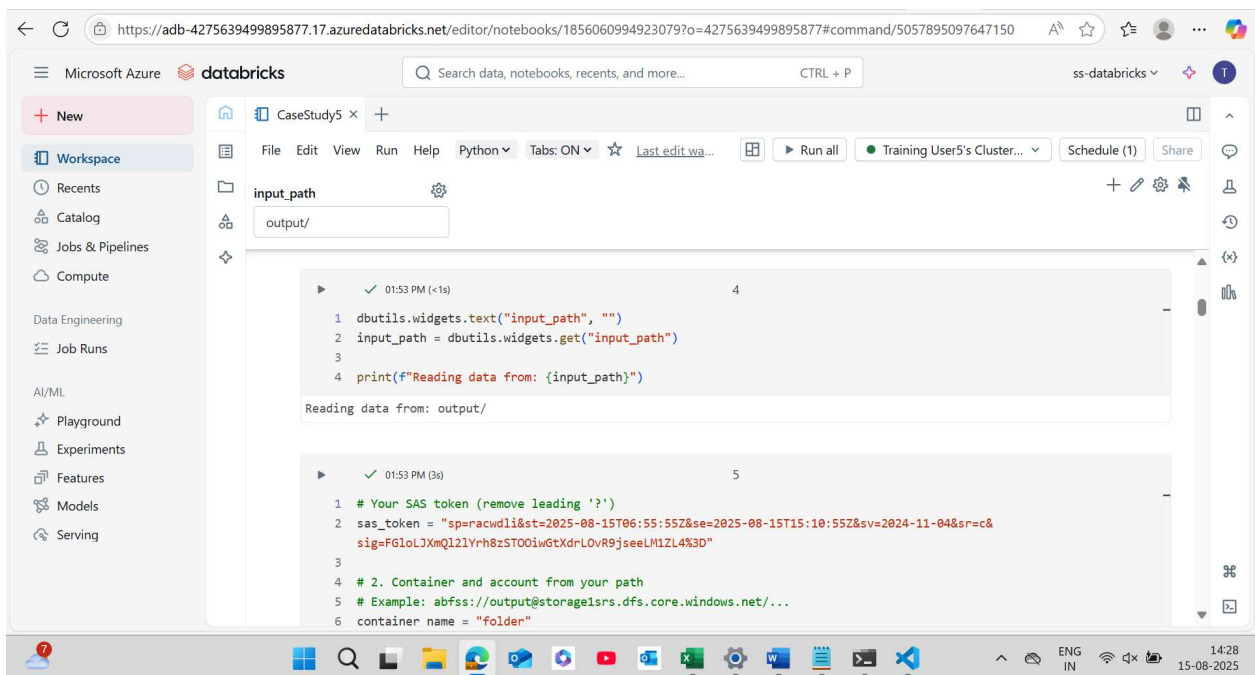
The csv file in a Sink blob container

The screenshot shows the 'storage1srs' Storage account in the Microsoft Azure Storage browser. The left sidebar contains navigation options: Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser (selected), Storage Mover, Partner solutions, Resource visualizer, and Data storage. The main content area displays the 'Blob containers' section for the 'storage1srs' account. It shows a list of blob containers: \$logs, data, folder, and View all. The 'data' container is selected, and its contents are displayed in a table. The table has columns for Name, Last modified, Access tier, and Blob type. Two items are listed: '_SUCCESS' and 'part-00000...'. Both items are of type 'Block blob' and have an 'Access tier' of 'Hot (Inferred)'. The '_SUCCESS' file is a CSV file.

Name	Last modified	Access tier	Blob type
_SUCCESS	15/8/2025, 11:42:15 am	Hot (Inferred)	Block blob
part-00000...	15/8/2025, 11:42:15 am	Hot (Inferred)	Block blob



Notebook- the input_path from the output files after running the dataflow



Converted the data to silver and gold (used Pydeequ)

Microsoft Azure | Search resources, services, and docs (G+)

Home > Storage accounts > storage1srs

storage1srs | Storage browser

Search

Overview
Activity log
Tags
Diagnose and solve problems
Access Control (IAM)
Data migration
Events
Storage browser
Storage Mover
Partner solutions
Resource visualizer
Data storage

storage1srs

Favorites
Recently viewed
Blob containers
\$logs
data
folder
View all
File shares
Queues
Tables

Blob containers > folder > output

Authentication method: Access key (Switch to Microsoft Entra user account)

Search blobs by prefix (case-sensitive)

Only show active blobs

Showing all 6 items

Name	Last modified	Access tier	Blob type
iot_gold			
iot_silver			
_SUCCESS (2)	15/8/2025, 12:44:38 pm	Hot (Inferred)	Block blob
iot_gold	15/8/2025, 1:01:26 pm	Hot (Inferred)	Block blob
iot_silver	15/8/2025, 1:01:46 pm	Hot (Inferred)	Block blob
part-00000...	15/8/2025, 12:44:38 pm	Hot (Inferred)	Block blob

The Job Run- Succeeded

Microsoft Azure | databricks

Search data, notebooks, recents, and more... CTRL + P

ss-databricks

+ New

Workspace
Recents
Catalog
Jobs & Pipelines
Compute

Data Engineering
Job Runs

AI/ML
Playground
Experiments
Features
Models
Serving

Runs > Run 134315866142389 >

notebook-task run Lakeflow Jobs UI: ON

Succeeded

Output

```
import json
from pydeequ.verification import VerificationResult

# Convert to Spark DF
result_df = VerificationResult.checkResultsAsDataFrame(spark, result)

# Collect to Python list of dicts
result_list = [row.asDict() for row in result_df.collect()]
result_dict = {"result": result_list}

# Write single JSON file to DBFS (using local path /dbfs/)
```

Details

Job ID	347367885912175
Job run ID	134315866142389
Task run ID	548201609431941
Run as	Training User5
Started	Aug 15, 2025, 02:16 PM
Ended	Aug 15, 2025, 02:17 PM
Duration	59s
Execution time	59s
Queue duration	-
Status	Succeeded

View run events View run libraries

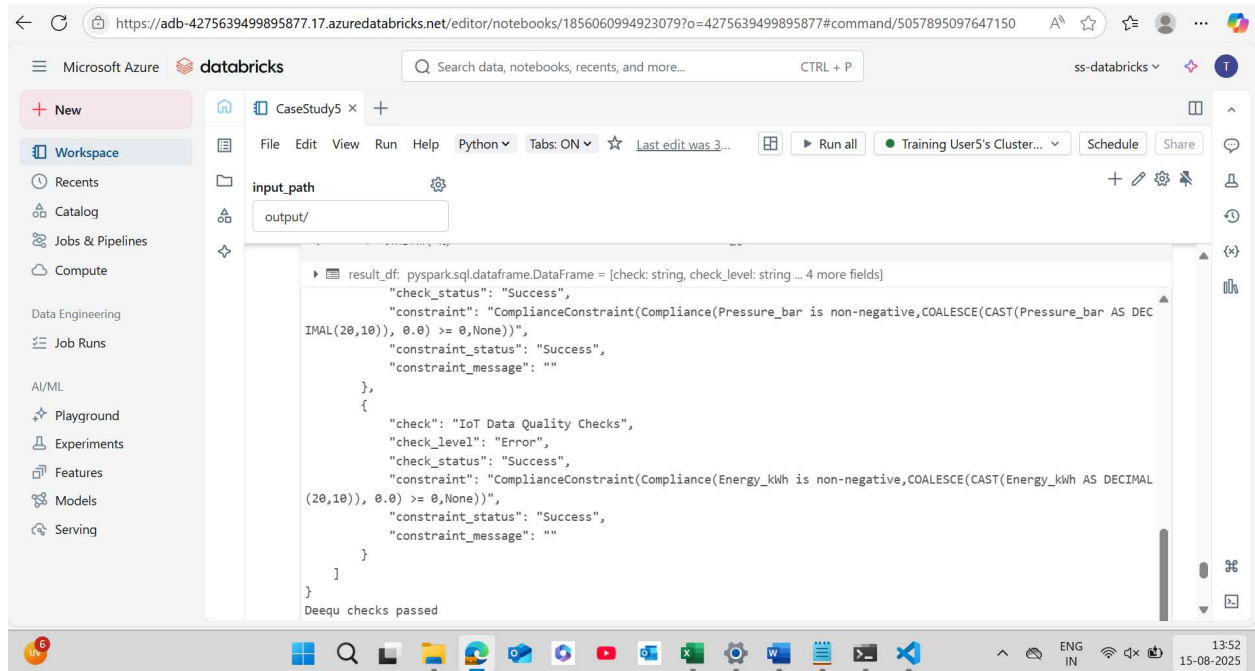
The **Deequ** checks passed

Deequ in the GitHub Actions workflow is being used as an automated data quality testing framework.

Here's the flow in your case:

1. Databricks Notebook Execution – The GitHub Actions workflow triggers a Databricks notebook that runs PyDeequ checks (e.g., `.isNonNegative()`, `.hasSize()`, etc.) on IoT data. These checks validate that the dataset meets business and technical quality rules.
2. Result Generation – PyDeequ outputs the verification results (pass/fail for each check) in a structured JSON format and saves it to DBFS (`/FileStore/.../deequ_result.json`).
3. Report Actions – The GitHub Actions job downloads this JSON file from DBFS, prints it to the log, and uses `jq` to parse the `check_status` field.
 - If `check_status` is "Success", the workflow continues.
 - If `check_status` is "Error" or any failure, the workflow exits with a non-zero status, marking the Action as failed.
4. Purpose – This setup turns Deequ into an automated quality gate:
 - Ensures data is validated before merging code or deploying pipelines.
 - Detects anomalies or rule violations early in the development cycle.

It's basically unit testing for data—but running inside Databricks and controlled from GitHub Actions.



Github/Workflow

The report actions section of the GitHub workflow is responsible for retrieving and validating the results of the Databricks job run.

After the notebook finishes running on Databricks (where data quality checks are executed), this step downloads the generated JSON report from DBFS (Databricks File System) to the GitHub Actions runner. It then prints the report contents for visibility in the Actions log and uses tools like jq to parse the JSON, checking the check_status field.

If the status indicates failure, the workflow exits with an error code, ensuring that any pull request or manual run fails when data quality issues are detected. This enforces automated data validation and acts as a gatekeeper before merging or deploying changes.

← CaseStudy5

CaseStudy5 #5

Re-run all jobs

Summary

Jobs

- databricks-quality-check

Run details

Usage

Workflow file

databricks-quality-check

succeeded now in 1m 15s

Search logs

- Set up job 1s
- Checkout 1s
- Trigger Databricks Job 1m 6s

```
1 ▶ Run databricks/run-notebook@v0
9 Notebook run has status RUNNING. URL: ***/?o=4275639499895877#job/347367885912175/run/134315866142389
10 Notebook run has status RUNNING. URL: ***/?o=4275639499895877#job/347367885912175/run/134315866142389
11 Notebook run has status RUNNING. URL: ***/?o=4275639499895877#job/347367885912175/run/134315866142389
12 Notebook run has status RUNNING. URL: ***/?o=4275639499895877#job/347367885912175/run/134315866142389
13 Notebook run has status RUNNING. URL: ***/?o=4275639499895877#job/347367885912175/run/134315866142389
14 Notebook run has status RUNNING. URL: ***/?o=4275639499895877#job/347367885912175/run/134315866142389
15 Notebook run has status RUNNING. URL: ***/?o=4275639499895877#job/347367885912175/run/134315866142389
```

← CaseStudy5

CaseStudy5 #5

Re-run all jobs

Summary

Jobs

- databricks-quality-check

Run details

Usage

Workflow file

databricks-quality-check

succeeded 48 minutes ago in 1m 15s

Search logs

- Download and Print Deequ Report 5s

```
75     "constraint_message": ""
76   },
77   {
78     "check": "IoT Data Quality Checks",
79     "check_level": "Error",
80     "check_status": "Success",
81     "constraint": "ComplianceConstraint(Compliance(Pressure_bar is non-negative,COALESCE(CAST(Pressure_bar AS
82     DECIMAL(20,10)), 0.0) >= 0,None))",
83     "constraint_status": "Success",
84     "constraint_message": ""
85   },
86   {
87     "check": "IoT Data Quality Checks",
88     "check_level": "Error",
89     "check_status": "Success",
90     "constraint": "ComplianceConstraint(Compliance(Energy_kwh is non-negative,COALESCE(CAST(Energy_kwh AS
91     DECIMAL(20,10)), 0.0) >= 0,None))",
92     "constraint_status": "Success",
93     "constraint_message": ""
94   }
95 }
Deequ checks passed
```