**Case Study**

**Background**

A retail analytics company, **DataMartX**, relies on Python + PySpark to process millions of daily transactions.  
Problems they faced:

* Data quality issues sneaking into production.
* Schema drift causing downstream failures.
* Slow manual rollback process.

They decided to:

1. Run **Deequ tests** in Python (via PyDeequ) during CI.
2. Block deployments if tests fail.
3. Auto-rollback failed releases.
4. Visualize **data drift** in Grafana.

**Business Objectives**

* Prevent low-quality data from being deployed.
* Automate rollback to minimize downtime.
* Detect drift proactively and make it visible to stakeholders.

**Architecture Flow**

1. **Pull Request** triggers **GitHub Actions**.
2. Actions run:
   * Python unit tests.
   * **PyDeequ** quality checks on staging data.
3. If checks fail → stop CI (no deploy).
4. If production deploy fails → auto-rollback.
5. Drift metrics → stored in Delta → visualized in Grafana.

**1. Gate CI on Deequ Tests (Python)**

**PyDeequ Quality Check:**

from pydeequ.checks import Check, CheckLevel

from pydeequ.verification import VerificationSuite

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("DataQualityChecks").getOrCreate()

# Load sample data

data = spark.read.csv("sample\_sales\_data.csv", header=True, inferSchema=True)

# Define quality rules

check = Check(spark, CheckLevel.Error, "Data quality checks") \

.hasSize(lambda x: x >= 10000) \

.isComplete("transaction\_id") \

.isNonNegative("sales\_amount")

# Run verification

result = VerificationSuite(spark) \

.onData(data) \

.addCheck(check) \

.run()

if result.status != "Success":

print("Data quality checks failed. Blocking deployment.")

exit(1)

**GitHub Actions YAML:**

jobs:

quality-check:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- uses: actions/setup-python@v4

with:

python-version: 3.9

- name: Install dependencies

run: pip install pyspark pydeequ

- name: Run data quality checks

run: python quality\_checks.py

**2. Auto-Rollback on Fail**

**Python Rollback Script:**

import os

import subprocess

LAST\_GOOD\_RELEASE = "s3://my-bucket/last\_good\_release.tar.gz"

def rollback():

print("Deployment failed. Rolling back...")

subprocess.run(["aws", "s3", "cp", LAST\_GOOD\_RELEASE, "."])

subprocess.run(["tar", "-xzf", "last\_good\_release.tar.gz"])

subprocess.run(["bash", "deploy.sh", "stable"])

if \_\_name\_\_ == "\_\_main\_\_":

rollback()

This script can be triggered from CI/CD if a deployment job returns a non-zero exit code.

**3. Visualise Drift (Python)**

**Drift Detection Using KS Test:**

from scipy.stats import ks\_2samp

import pandas as pd

from datetime import date

# Load data

baseline = pd.read\_parquet("baseline\_sales.parquet")

new\_data = pd.read\_parquet("sales\_today.parquet")

# KS test

stat, p\_value = ks\_2samp(baseline['sales\_amount'], new\_data['sales\_amount'])

drift\_detected = p\_value < 0.05

# Store results in Delta (Databricks example)

from delta import DeltaTable

from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

df = spark.createDataFrame([{

"date": str(date.today()),

"column": "sales\_amount",

"drift\_detected": drift\_detected,

"p\_value": p\_value

}])

df.write.format("delta").mode("append").save("/mnt/drift\_metrics")

**Grafana Dashboard:**

* Connect Grafana to Databricks SQL endpoint.
* Create panels showing:
  + Drift detected over time.
  + Columns with highest drift frequency.
  + Recent rollback events.

**Outcomes**

* **Zero bad data deployments** after rollout.
* **Rollback time reduced from 3 hours → 10 minutes**.
* Real-time drift visibility improved trust in analytics.

**Lessons Learned**

* Python + PyDeequ works well in CI/CD pipelines.
* Automated rollback is essential for production stability.
* Drift visualization is a preventive control, not just a reactive one.