**AI-Assisted Data Platform Test Cases**

Version 1.0 | Generated: September 03, 2025

Combined scenarios: Emplifi Medallion Architecture & Semrush API (ADF + ADLS + SQL Server)

# 1. Introduction

This document compiles comprehensive test cases for two Azure data engineering scenarios using GitHub Copilot and Microsoft Copilot. Scenario 1 validates a Medallion architecture (Bronze/Silver/Gold) for Emplifi profile data across Databricks, ADLS Gen2, Synapse, and Power BI. Scenario 2 validates API-based ingestion from Semrush with Copilot-generated Azure Data Factory (ADF) pipelines, landing into ADLS and loading Microsoft SQL Server.

Each scenario includes a summary table and multiple detailed test cases with objectives, preconditions, data, steps, expected results, and pass/fail criteria. Scenarios begin on separate pages for clarity and execution planning.

# Scenario 1 – Emplifi Medallion Architecture

Validates end-to-end ingestion and transformation for the Emplifi dataset using Databricks, ADLS Gen2, Synapse, and Power BI.

## Test Case Summary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Title | Priority | Type | Owner | Traceability |
| S1-DBX-001 | Load Emplifi dataset (2000 rows) to DBFS | P1 | Functional (Positive) | QA | S1-1 |
| S1-VSC-002 | Copilot in VS Code generates Databricks PySpark & syncs | P1 | Functional (Positive) | Data Eng | S1-2, S1-3 |
| S1-ADLS-003 | Mount ADLS to Databricks | P1 | Functional (Positive) | Data Eng | S1-4 |
| S1-ARCH-004 | Create Medallion folders (bronze/silver/gold) | P1 | Functional (Positive) | Data Eng | S1-5 |
| S1-BRONZE-005 | Bronze: straight copy to parquet | P1 | Functional (Positive) | QA | S1-6 |
| S1-SILVER-006 | Silver: enforce non-null columns to parquet | P1 | Functional (Pos/Neg) | QA | S1-7 |
| S1-SYN-007 | Synapse external tables for Bronze & Silver | P1 | Functional (Positive) | DBA | S1-8, S1-9 |
| S1-PBI-008 | Power BI Copilot dashboard from Synapse table | P2 | Functional (Positive) | BI | S1-10 |
| S1-SYN-009 | Publish the app via Synapse/Power BI workspace | P2 | Functional (Positive) | BI | S1-11 |

## Detailed Test Cases

## S1-DBX-001 – Load Emplifi dataset (2000 rows) to DBFS

Objective: Verify 2000-row Emplifi dataset is uploaded and readable from DBFS.

Feature/Scope: Databricks, DBFS

Priority: P1

Type: Functional (Positive)

Pre-Conditions: Databricks cluster running; permissions to DBFS.

Test Data: File: emplifi\_profiles\_2000.csv (UTF-8, header). Target: dbfs:/FileStore/emplifi/landing/

Steps:

1. Upload CSV to DBFS using UI (Data > Add Data) or CLI/databricks fs cp.

2. List files at dbfs:/FileStore/emplifi/landing/ to confirm presence.

3. In a notebook, read with spark.read.csv(header=True, inferSchema=True).

4. Count rows and printSchema(); display 10 sample rows.

Expected Results:

• File exists at target path and is readable.

• Row count equals 2000.

• Schema inferred with expected columns (IDs, network, date, metrics).

Pass/Fail Criteria: Pass if read succeeds and row count == 2000.

## S1-VSC-002 – Copilot in VS Code generates Databricks PySpark & syncs

Objective: Validate Copilot authors clean PySpark to read DBFS and syncs via Git to Databricks.

Feature/Scope: VS Code, GitHub Copilot, Databricks Repos

Priority: P1

Type: Functional (Positive)

Pre-Conditions: Repo connected to Databricks Repos; developer permissions.

Test Data: Prompt: "Read Emplifi CSV from dbfs:/FileStore/emplifi/landing and write parquet to bronze."

Steps:

1. Open VS Code, create notebook/script (PySpark).

2. Invoke Copilot with the prompt; accept/refine generated code.

3. Commit & push; verify sync in Databricks Repos.

4. Execute notebook on cluster and capture output/logs.

Expected Results:

• Copilot generates syntactically correct code with comments.

• Repo syncs; notebook appears in Databricks Repos.

• Notebook runs without errors and prints schema/sample.

Pass/Fail Criteria: Pass if code compiles, runs in Databricks, and meets prompt intent.

## S1-ADLS-003 – Mount ADLS to Databricks

Objective: Ensure ADLS Gen2 is mounted with correct permissions.

Feature/Scope: ADLS Gen2, Databricks

Priority: P1

Type: Functional (Positive)

Pre-Conditions: Storage account and containers exist; service principal or passthrough configured; secrets in Key Vault.

Test Data: Mount point: /mnt/datalake; Containers: bronze, silver, gold.

Steps:

1. Configure secrets (Key Vault-backed secret scope).

2. Run dbutils.fs.mount() or abfss direct access to set up mount(s).

3. List /mnt/datalake and subfolders to validate access.

4. Write and read a small test file to confirm R/W.

Expected Results:

• Mount(s) created without error; listings succeed.

• R/W test confirms correct ACLs and permissions.

Pass/Fail Criteria: Pass if listing and R/W succeed with no unauthorized errors.

## S1-ARCH-004 – Create Medallion folders (bronze/silver/gold)

Objective: Create folder structure via code and ensure idempotency.

Feature/Scope: ADLS Gen2, Databricks

Priority: P1

Type: Functional (Positive)

Pre-Conditions: ADLS mount available at /mnt/datalake.

Test Data: Paths: /mnt/datalake/bronze/emplifi/, /mnt/datalake/silver/emplifi/, /mnt/datalake/gold/emplifi/.

Steps:

1. Run helper function to create folders if absent.

2. Re-run function to ensure no failure on existing folders.

3. List paths to confirm existence.

Expected Results:

• Folders created on first run and no-op on second run (idempotent).

Pass/Fail Criteria: Pass if folders exist and re-run produces no errors.

## S1-BRONZE-005 – Bronze: straight copy to parquet

Objective: Persist raw Emplifi to Bronze parquet with minimal transformations.

Feature/Scope: Databricks, ADLS Gen2 (Bronze)

Priority: P1

Type: Functional (Positive)

Pre-Conditions: Medallion folders exist.

Test Data: Source: DBFS CSV; Target: abfss://bronze/.../emplifi/ (parquet, snappy).

Steps:

1. spark.read.csv(...).write.format('parquet').mode('overwrite').save(target)

2. List target to capture file count/size; record partitioning if applied.

3. Read back parquet; compare schema & row count.

Expected Results:

• Parquet files present in Bronze; schema matches raw; row count == 2000.

Pass/Fail Criteria: Pass if read-back count equals 2000 and schema equal.

## S1-SILVER-006 – Silver: enforce non-null columns to parquet

Objective: Filter out records with NULL in required columns and persist as Silver parquet.

Feature/Scope: Databricks, ADLS Gen2 (Silver)

Priority: P1

Type: Functional (Positive/Negative)

Pre-Conditions: Bronze exists with 2000 rows.

Test Data: Required non-null columns: profile\_id, network, date, engagement.

Steps:

1. Read Bronze parquet.

2. Compute null counts by required column; log metrics.

3. Filter rows where all required columns are non-null.

4. Write parquet to Silver path with mode='overwrite' and snappy compression.

5. Reread Silver and recompute null counts for validation.

6. Negative: artificially introduce NULL in a required column and rerun validation to assert failure detection.

Expected Results:

• Silver parquet exists; required columns have 0 NULLs.

• Data quality log shows before/after counts and dropped rows.

• Negative run surfaces validation failure with clear error log.

Pass/Fail Criteria: Pass if Silver has zero NULLs in required columns and DQ metrics are persisted.

## S1-SYN-007 – Synapse external tables for Bronze & Silver

Objective: Expose parquet datasets via Synapse serverless external tables.

Feature/Scope: Synapse serverless SQL, ADLS

Priority: P1

Type: Functional (Positive)

Pre-Conditions: Synapse workspace with storage credential; database & schema exist.

Test Data: External data source DS\_DATALAKE; file format FF\_PARQUET; Tables: ext\_emplifi\_bronze, ext\_emplifi\_silver.

Steps:

1. Create external data source and file format if absent.

2. Create external tables pointing to Bronze and Silver folder paths.

3. Run SELECT COUNT(\*) against both tables.

4. Validate sample records reflect null cleanup from Bronze→Silver.

Expected Results:

• External tables created successfully.

• Bronze count == 2000; Silver count ≤ 2000 depending on NULL drops.

• Queries return within acceptable latency for small datasets.

Pass/Fail Criteria: Pass if objects create and counts align as expected.

## S1-PBI-008 – Power BI Copilot dashboard from Synapse table

Objective: Use Copilot in Power BI Desktop to generate visuals and DAX from Synapse external table.

Feature/Scope: Power BI Desktop, Synapse

Priority: P2

Type: Functional (Positive)

Pre-Conditions: Connectivity to Synapse; Copilot enabled in PBI Desktop.

Test Data: Dataset: ext\_emplifi\_silver; Measures: engagement total, YoY % change by network.

Steps:

1. Connect PBI Desktop to Synapse external table.

2. Use Copilot to author visuals: daily engagement by network; highlight anomalies.

3. Validate generated DAX against manual calculation for a sample slice.

4. Save PBIX and export to workspace (if applicable).

Expected Results:

• Visuals and DAX generated; results match manual baseline within tolerance.

• Report saves and can refresh successfully.

Pass/Fail Criteria: Pass if visuals/DAX are correct and refresh works.

## S1-SYN-009 – Publish the app via Synapse/Power BI workspace

Objective: Publish the Power BI report as an app and surface in Synapse workspace hub.

Feature/Scope: Power BI Service, Synapse workspace integration

Priority: P2

Type: Functional (Positive)

Pre-Conditions: Report present in workspace; publish permissions granted.

Test Data: App audience list; workspace linked in Synapse Develop > Power BI.

Steps:

1. From Power BI Service, create/update App containing the report.

2. From Synapse, verify Power BI artifact appears under Develop.

3. Share app to target users and test access.

Expected Results:

• App published; target users can view with correct permissions.

• Synapse displays linked artifact successfully.

Pass/Fail Criteria: Pass if app is accessible and visible via Synapse integration.

## Requirements Traceability

|  |  |  |
| --- | --- | --- |
| Scenario Step | Description | Test IDs |
| S1-1 | Dataset to DBFS | S1-DBX-001 |
| S1-2 | VS Code reads from DBFS | S1-VSC-002 |
| S1-3 | Copilot creates & syncs Python to Databricks | S1-VSC-002 |
| S1-4 | ADLS is mounted with Databricks | S1-ADLS-003 |
| S1-5 | Create Medallion folders in ADLS | S1-ARCH-004 |
| S1-6 | Bronze has straight copy parquet | S1-BRONZE-005 |
| S1-7 | Silver has non-null columns parquet | S1-SILVER-006 |
| S1-8 | Synapse mounted/reads ADLS | S1-SYN-007 |
| S1-9 | Create external tables in Synapse | S1-SYN-007 |
| S1-10 | Power BI Desktop dashboard via Copilot | S1-PBI-008 |
| S1-11 | Publish the app in Synapse | S1-SYN-009 |

# Scenario 2 – Semrush API with ADF, ADLS & SQL Server

Validates Copilot-driven ingestion and orchestration for Semrush data, landing to ADLS and loading SQL Server via ADF.

## Test Case Summary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Title | Priority | Type | Owner | Traceability |
| S2-API-001 | Semrush API pull via Copilot-generated Python | P1 | Functional (Pos/Neg) | QA | S2-1 |
| S2-ADF-002 | Copilot creates ADF pipeline template | P1 | Functional (Positive) | QA | S2-2 |
| S2-DBX-003 | Copilot creates Python & syncs to Databricks | P1 | Functional (Positive) | Data Eng | S2-3 |
| S2-ADF-004 | Copilot creates ADF JSON; create pipelines from JSON | P1 | Functional (Positive) | QA | S2-4, S2-5 |
| S2-ADLS-005 | ADLS mounted/validated for landing | P2 | Functional (Positive) | Data Eng | S2-5, S2-6 |
| S2-ADF-006 | ADF creates folder & parquet in ADLS | P1 | Functional (Positive) | QA | S2-6 |
| S2-SQL-007 | ADF creates and loads SQL Server tables | P1 | Functional (Pos/Neg) | DBA | S2-7 |
| S2-AI-008 | End-to-end prompting experience in VS Code | P2 | Non-Functional (Process) | DevOps | S2-8 |

## Detailed Test Cases

## S2-API-001 – Semrush API pull via Copilot-generated Python

Objective: Ingest Semrush data using Python authored by Copilot, with pagination, retries, and secure secrets.

Feature/Scope: VS Code, GitHub Copilot, Python runtime

Priority: P1

Type: Functional (Positive/Negative)

Pre-Conditions: Semrush API key stored in Key Vault/secret scope; outbound network allowed.

Test Data: Endpoint: Semrush API (e.g., domain\_analytics). Params: domain, date range; Secrets: SEMRUSH\_API\_KEY.

Steps:

1. Prompt Copilot to generate Python for Semrush API call with retry/backoff (HTTP 429/5xx).

2. Implement secrets retrieval from env/Key Vault; ensure no keys are hard-coded.

3. Run script; persist raw JSON to local or DBFS staging.

4. Negative: Mock/simulate 429 & 500 responses to validate retries and error classification.

Expected Results:

• Successful data retrieval and persisted raw JSON.

• Retries/backoff observed on throttling; structured error logs created.

• No secrets printed in console/logs.

Pass/Fail Criteria: Pass if data retrieved and error handling meets policy without exposing secrets.

## S2-ADF-002 – Copilot creates ADF pipeline template

Objective: Use Copilot in ADF Studio to scaffold a parameterized pipeline template (Copy/Notebook, schedule, failure policy).

Feature/Scope: Azure Data Factory

Priority: P1

Type: Functional (Positive)

Pre-Conditions: ADF Studio accessible; contributor rights; linked services placeholders available.

Test Data: Template name: tpl\_semrush\_ingest; Parameters: startDate, endDate, sinkPath; Retry: 3 w/ exponential backoff.

Steps:

1. Open ADF Studio > Copilot panel; enter prompt defining activities, parameters, retry, and logs.

2. Review generated pipeline; confirm activities, parameters, and failure path.

3. Publish and run a test instance with a narrow date range.

Expected Results:

• Template created with requested objects and policies.

• Test run completes; run logs show parameter values and retry policy.

Pass/Fail Criteria: Pass if pipeline is generated per prompt and executes successfully.

## S2-DBX-003 – Copilot creates Python & syncs to Databricks

Objective: Author transformation code in VS Code and sync via Git to Databricks for execution.

Feature/Scope: VS Code, GitHub Copilot, Databricks Repos

Priority: P1

Type: Functional (Positive)

Pre-Conditions: Databricks Repos linked to Git repo; cluster available.

Test Data: Transform: normalize JSON → columns, cast types, dedupe by (domain,date).

Steps:

1. Use Copilot to draft PySpark code to parse Semrush JSON and produce typed DataFrame.

2. Commit & push; verify sync to Databricks Repos.

3. Run notebook; ensure unit assertions on schema & uniqueness pass.

Expected Results:

• Code compiles; schema matches design; deduplication removes duplicates.

• Artifacts saved (e.g., tests/logs).

Pass/Fail Criteria: Pass if tests green and sample run yields expected parquet.

## S2-ADF-004 – Copilot creates ADF JSON; create pipelines from JSON

Objective: Generate ADF JSON with Copilot and deploy to create pipelines (Import/ARM).

Feature/Scope: ADF, Git, ARM/Import

Priority: P1

Type: Functional (Positive)

Pre-Conditions: Service connection or permissions to import JSON/ARM; repo contains JSON.

Test Data: JSON includes linkedServices (ADLS, SQL, Key Vault), datasets, pipelines, triggers.

Steps:

1. Prompt Copilot in VS Code to produce ADF JSON scaffolding with parameters and linked services.

2. Import JSON into ADF (Manage > ARM templates or Import dialog) and publish.

3. Run pipeline and confirm resources (datasets/linked services/activities) function.

Expected Results:

• ADF resources appear as defined in JSON; run succeeds without manual edits.

• Parameters flow through correctly to activities.

Pass/Fail Criteria: Pass if deployment succeeds and test run completes successfully.

## S2-ADLS-005 – ADLS mounted/validated for landing

Objective: Validate ADLS path for Semrush landing and temp areas (if Databricks used).

Feature/Scope: ADLS Gen2, Databricks (optional)

Priority: P2

Type: Functional (Positive)

Pre-Conditions: Storage account exists; mount configured or abfss direct access.

Test Data: Container: seo; Paths: /seo/semrush/yyyy=YYYY/mm=MM/dd=DD/.

Steps:

1. List container paths; create a temporary folder and write a small file.

2. Remove temp folder to validate delete permissions.

Expected Results:

• Read/write/delete operations succeed within the container.

Pass/Fail Criteria: Pass if R/W/D operations succeed without permission errors.

## S2-ADF-006 – ADF creates folder & parquet in ADLS

Objective: Run pipeline to land Semrush data as parquet into partitioned ADLS folders.

Feature/Scope: ADF, ADLS

Priority: P1

Type: Functional (Positive)

Pre-Conditions: Linked services configured; sink container exists.

Test Data: Sink path parameter: sinkPath; Partitioning: yyyy/mm/dd.

Steps:

1. Trigger pipeline with date parameters; observe activities and output.

2. Verify folders created and parquet files landed.

3. Read parquet (Databricks or serverless) to validate schema.

Expected Results:

• Partitioned folders created; parquet files present with expected schema.

• No partial files left on failure.

Pass/Fail Criteria: Pass if files exist and schema validates.

## S2-SQL-007 – ADF creates and loads SQL Server tables

Objective: Use ADF to create table if not exists and load/merge Semrush data.

Feature/Scope: ADF, SQL Server

Priority: P1

Type: Functional (Positive/Negative)

Pre-Conditions: SQL Server reachable; credentials via Key Vault; target schema granted.

Test Data: Target: dbo.SEMRUSH\_METRICS (typed columns, PK on (domain,date)).

Steps:

1. Configure pre-copy script to CREATE TABLE IF NOT EXISTS.

2. Execute copy/Upsert activity to load data.

3. Validate counts and constraints; verify primary key uniqueness.

4. Negative: introduce mismatched data type to confirm failure and rollback behavior.

Expected Results:

• Table created idempotently; rows loaded; PK constraints enforced.

• On error, transaction rolls back or failure leaves no partial/duplicate rows; error logged.

Pass/Fail Criteria: Pass if counts match, constraints hold, and error path is clean.

## S2-AI-008 – End-to-end prompting experience in VS Code

Objective: Assess repeatability and time-to-first-success (TTFS) using prompt-driven development.

Feature/Scope: VS Code, GitHub Copilot

Priority: P2

Type: Non-Functional (Process)

Pre-Conditions: Prompt catalog prepared; two engineers available for A/B run.

Test Data: Prompt playbook (markdown) in repo with expected artifacts.

Steps:

1. Execute the workflow using the prompt playbook; measure TTFS.

2. A second engineer repeats; compare deltas and number of corrections.

3. Capture feedback and issues for prompt refinement.

Expected Results:

• TTFS within target (e.g., < 2 hours); low variance across engineers.

• Catalog of effective prompts established.

Pass/Fail Criteria: Pass if TTFS and variance thresholds met.

## Requirements Traceability

|  |  |  |
| --- | --- | --- |
| Scenario Step | Description | Test IDs |
| S2-1 | Semrush API via Copilot | S2-API-001 |
| S2-2 | ADF template via Copilot | S2-ADF-002 |
| S2-3 | Copilot creates Python & syncs to Databricks | S2-DBX-003 |
| S2-4 | Copilot creates JSON for ADF | S2-ADF-004 |
| S2-5 | ADF pipelines from JSON | S2-ADF-004 |
| S2-6 | ADF creates folder & parquet in ADLS | S2-ADF-006 |
| S2-7 | ADF creates tables in SQL Server | S2-SQL-007 |
| S2-8 | Prompt-driven workflow in VS Code | S2-AI-008 |