Data Preprocessing on Azure

Lab 3

I. Introduction

A modern ***lakehouse*** architecture combines the scalability of a data lake with the management features of a data warehouse. In a lakehouse, raw data lands in a *bronze* (raw) layer, is cleaned and validated into a *silver* layer, then aggregated/enriched into a *gold* (serving) layer. This pattern (often called the *medallion architecture*) ensures data is progressively refined for analytics and ML.

The diagram below illustrates a reference architecture: data is ingested (via batch/stream), stored in **Azure Data Lake Storage Gen2 (ADLS Gen2)**, transformed on **Azure Databricks** (using Delta Lake for ACID and schema enforcement), and served for BI or ML. Azure services like **Data Factory** (or Synapse Pipelines) handle ingestion, Databricks runs Spark transformations, and the **Delta** format unifies storage for queries and ML.

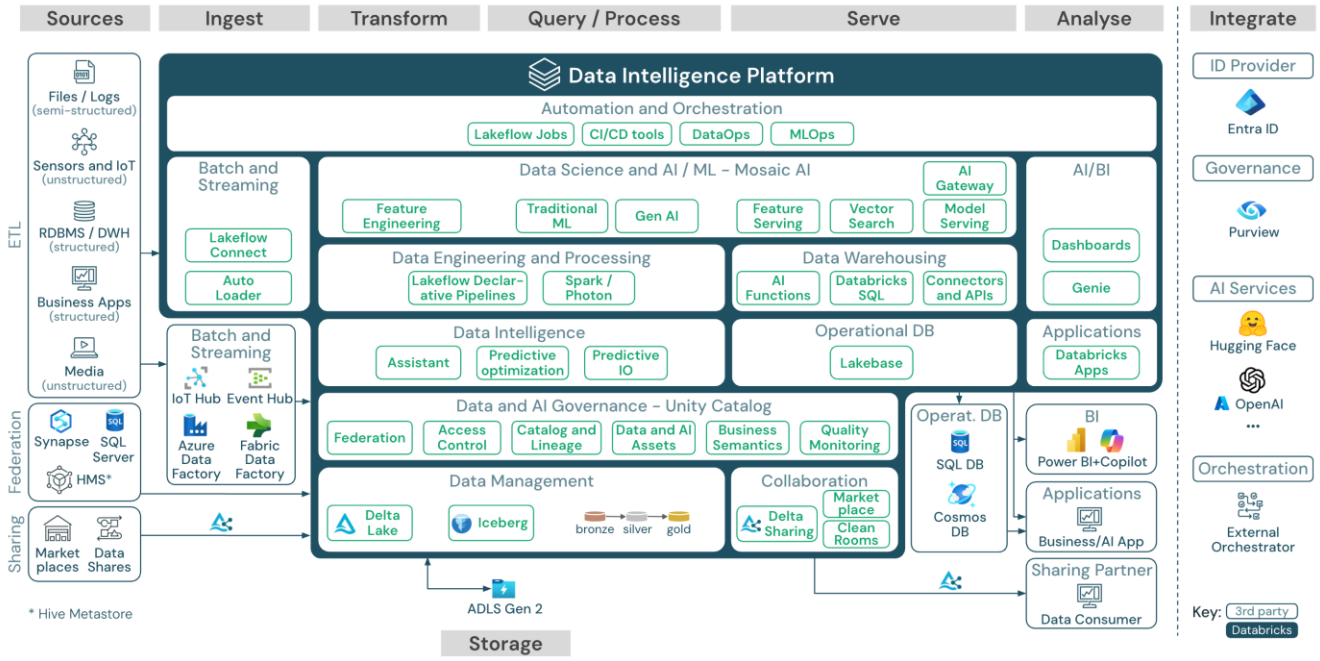

Figure 1. ***Azure Databricks Lakehouse reference architecture (ingest, storage, transform, serve).***

II. Prepare the Data Lake

II.1. Set up the storage and containers

a. Create an ADLS Gen2 Storage Account

- In the Azure Portal, **Create a Storage account**, in your resource group and in your region.
- Give a unique and meaningful name: `goodreadsreviews60XXXXXX`.
- **Enable *Hierarchical namespace* (under Advanced)**.
- Click **Review + create**, then **Create**.

b. Set up Containers and Folders

- Once the **Storage Account** is deployed. Create a Container and call `lakehouse`.
- Open the container you just created. Add a directory[1] and call it `raw`.
- Inside raw, create three subfolders: `books, reviews, authors`.
- You will end up with paths like:
  - `/lakehouse/raw/books/`
  - `/lakehouse/raw/reviews/`
  - `/lakehouse/raw/authors/`

II.2. Stream the Data from Public URL to ADLS

- The dataset is hosted on a public HTTP/HTTPS link, we need to download and **pipe it directly into your storage account** without saving it locally into our machine (This is around 16 Gb after all.).
- But first, **Generate an SAS key** for your storage account with:
  - Allowed services: Blob, and files minimum.
  - Allowed resource types: Service, Container, and Object, minimum.
  - Allowed permissions: Read and Write, minimum.
  - Copy the key and save it somewhere safe, and nice… *it gets scared*.
- Use a VM to download and send the data:
  - Create a VM directly or Compute instance in Azure Machine Learning Workspace (Which I recommend).
    - *I would recommend creating a new Workspace Called* `GoodReads-Reviews-Analysis-60XXXXX`.
    - Create a Compute instance and choose `Standard_E4ds_v4`.
  - Start a the terminal
    - Use `wget` to download the data from the following links [2].

Books	https://mcauleylab.ucsd.edu/public_datasets/gdrive/goodreads/goodreads_books.json.gz
Reviews	https://mcauleylab.ucsd.edu/public_datasets/gdrive/goodreads/goodreads_reviews_dedup.json.gz

[1] Directory is another name for a "F*older*". Put down the torches and pitchforks, no need to panic… yet.

[2] Goodreads Datasets.

Authors	https://mcauleylab.ucsd.edu/public_datasets/gdrive/goodreads/goodreads_book_authors.json.gz

- Check that they are downloaded using `ls`.
- Use `gzip -d` to unzip the files one by one.
- Use `azcopy` to upload the files to their appropriate folders>For instance :

```
azcopy copy \
  "./goodreads_book_authors.json" \

"https://yourstorageaccount.blob.core.windows.net/lakehouse/raw/authors/goodreads_book_authors
.json?YourSAS" \
  --overwrite=true
```

- *Do this for the three files!*

Now your ADLS Gen2 has a clear "raw" zone for each dataset. Next, we'll ingest and convert this raw JSON into optimized Parquet/Delta format in a **processed** zone.

III. Ingest & Transform Raw Data with Azure Data Factory

We now move to the next stage of our ETL pipeline. The raw JSON files are stored in our Data Lake 🌊:

- /lakehouse/raw/books/
- /lakehouse/raw/reviews/
- /lakehouse/raw/authors/

This is the bronze layer. Our task now is to transform them into a faster, more structured format (Parquet) and store them in a new silver layer. For this, we use **Azure Data Factory (ADF)**. ADF gives us a visual interface to design data pipelines without writing code. In this step, we'll connect ADF to our storage account, create datasets, and build a pipeline that copies the JSON data as-is into Parquet.We intentionally keep this step simple and defer column selection or schema cleaning to Databricks (Step V).

III.1. Setting Up Azure Data Factory

a. Create an Azure Data Factory (ADF) Instance

- In **Azure Portal** → click **Create a resource** (top left menu).
- Search for **Data Factory** 🏭 → click **Create**.
- Select the correct **Subscription**.
- Choose the correct **Resource Group**.
- Enter a Name: e.g., goodreads-data-factory-60XXXXXX.
- Choose a **Region** close to you.
- Choose **V2**.
- On the Git configuration tab → leave defaults (no Git required for now).
- On the Networking tab → leave defaults (public endpoint enabled).

- On the Advanced tab → leave defaults.
- Click **Review + Create** → then **Create**.

b. Create Linked Services (Storage connections)

In this part, we are setting up a **Linked Service** in ADF, which acts like a secure bridge between ADF and our storage account.

- Wait for deployment → click Go to resource to open ADF Studio.
- Once Open go to → *Manage* (The icon with the mechanical key on the left)
  → ***Linked services***
  → ***New.***
- Choose **Azure Data Lake Storage Gen2**,
  - Give it name: `GoodreadsLink60107583`.
  - Authenticate using an account key.
  - point to your storage account `goodreadsreviews60XXXXXX` by choosing the the right subscription and storage account.
- Test connection first then **Create**.

III.2. Create the Datasets

Here, we will connect ADF to our Data Lake so it can read the raw JSON files and write them into a more efficient Parquet format. This step is essential because JSON is flexible but slow to query at scale, while Parquet is optimized for analytics and machine learning — making it the right format for the next stages of our pipeline.

a. The Source (JSON) Datasets

- In ADF Studio, click **Author** (pencil symbol).
- Under Datasets → + New dataset.
- Select Azure Data Lake Storage Gen2.
- Choose **JSON** as the format.
- Configure the dataset:
  - Name: `raw_authors`
  - **Linked service** → pick the one you created for `goodreadsreviews60XXXXXX`.
  - **File path** → click on the folder icon and navigate to your file.

*You can also navigate to the folders (e.g. authors) and use wildcards like *.json for all JSON files.*

  - **Click Ok.**
  - **Then test connection, and preview data.**
- Repeat for the other datasets: `raw_books`, and `raw_reviews`.

b. The Sink (Parquet) Datasets

- In **ADF Studio**, click **Author** (pencil symbol).
- Under Datasets → + New dataset.

- Choose Azure Data Lake Storage Gen2.
- Select **Parquet** as the format.
    - Choose **JSON** as the format.
    - Configure the dataset:
        - Name: `authors_parquet`
    - **Linked service** → pick the one you created for `goodreadsreviews60XXXXXX`.
    - **File path**
        - Container: `lakehouse`
        - Directory: `processed/books/`
        - Give it a clear name, e.g. `books_parquet`
    - **Import Schema: None** (*We don't have data for the moment ey !*).
    - Click Ok, then Click Test connection → To linked service (should succeed).
- **Repeat for the other datasets: `raw_authors`, and `raw_reviews`.**
- **Publish! Publish! Publish!**

What we have done so far — creating linked services, datasets, and pipelines — is temporary until it is published. If you close your browser or laptop before publishing, all these objects will disappear.

When working in ADF, always click Publish All before leaving. Otherwise, your datasets and pipelines will disappear when you close your session.

III.3. Copy JSON to Parquet Using Pipelines

Now that both source (JSON) and sink (Parquet) datasets are defined, we can build pipelines to move and transform the data. Each pipeline will copy data from one raw folder to its corresponding processed folder. We will not do complex mapping at this stage — we simply copy the JSON as-is. Schema cleanup will be done later in Databricks.

a. Create a Pipeline and add an Activity

- In ADF Studio → click **Author** (pencil symbol).
- Under Pipelines → + New pipeline.
- Give it a clear name, e.g., pl_books_json_to_parquet.
- From the Activities pane (on the right) → drag Copy Data (under Move and Transform) onto the canvas.
- Name the activity clearly, e.g., copy_books.

b. Configure the Source

- Click on the Source tab.
- Select the JSON dataset you created earlier (raw_books).
- Preview the data if needed.

c. Configure the Sink

- Open the Sink tab.
- Select the Parquet dataset (books_parquet).
- In Copy behavior, choose Preserve hierarchy so folder structure remains consistent.

👉 Important: Go modify the Mapping tab. To make sure all necessary information are included. For, instance, author_id. You can also get rid of Popular Shelves and Series... Or at least be careful with those columns as they might generate errors. So flatten them.

---

d. Debug & Publish

- Click Debug (on top) to test-run the pipeline.
- If successful, data from /raw/books/ is copied into /processed/books/ as Parquet.
- Once tested, click Publish All to save.

e. Repeat for the other datasets:

- pl_reviews_json_to_parquet → source raw_reviews → sink reviews_parquet.
- pl_authors_json_to_parquet → source raw_authors → sink authors_parquet.

💡 One trick to making this easier is to clone the pipeline for the other datasets and just change the dataset references.

Note on Reviews Dataset
The reviews dataset may contain **malformed JSON rows**.
If the pipeline fails for reviews, do not panic.
This is expected due to bad records (e.g., unescaped quotes in review text).
👉 We will handle this gracefully later in Databricks, which can read and clean malformed JSON with Spark's permissive mode.

III.4. At the End of This Step

After all three pipelines run, your Data Lake will contain a **processed zone** with:

- /processed/books/ → Parquet files
- /processed/reviews/ → (to be fixed in Databricks)
- /processed/authors/ → Parquet files

You have now completed the **bronze → silver** step: raw JSON data has been ingested and stored in Parquet, ready for analytics, joining, and Lakehouse exploration.

Here's the **rewritten Step V — Build the Lakehouse with Databricks**, keeping Databricks for curation and **including the corrected reviews flow** (Spark **PERMISSIVE** read → clean → write to silver). I've kept the same style and level of detail, and I **explain each code block** right after it.

IV. Build the Lakehouse with Databricks

We have now transformed our raw JSON files for **reviews, books** and **authors** to Parquet (silver) using ADF. The **reviews** dataset can contain malformed JSON rows; Databricks can handle this gracefully. In this stage, we finalize the **Lakehouse layer**: we connect Databricks to ADLS Gen2, **clean and convert reviews** into Parquet (silver), **drop the role field** from authors, **register Delta tables**, and **curate** a gold table for analytics and ML.

Databricks is a collaborative data engineering and analytics platform built around **Apache Spark**, an open-source distributed computing engine. Spark is designed to process and analyze very large datasets efficiently by breaking computations into smaller tasks and executing them in parallel across multiple machines.

Using Databricks, data scientists and engineers can work in a unified environment that supports Python, SQL, R, and Scala, with tight integration to Azure Data Lake and other services. It provides a **lakehouse architecture**, combining the scalability of data lakes with the structure of data warehouses. This makes it ideal for:

- Reading and transforming large datasets stored in Azure Data Lake.
- Cleaning, joining, and enriching data before advanced analytics or machine learning.
- Creating structured layers (Raw → Silver → Gold) to support BI and AI workloads.

In this lab, we use Databricks and Spark primarily to **clean and curate data coming from ADF pipelines**, **join multiple datasets** (books, authors, reviews), and **publish a single curated Gold table** for downstream feature engineering and visualization in Fabric or Power BI.

The main actions we will take are:

- Connect Databricks to ADLS Gen2 using the storage account key and the abfss:// protocol to enable direct reads and writes to your lakehouse container.
- Load silver Parquet datasets for books and authors that were prepared earlier using Azure Data Factory.
- Fix and clean the reviews dataset by reading the raw JSON with Spark in PERMISSIVE mode, dropping malformed records, selecting the essential fields, and saving the clean data to the processed/ (silver) zone as Parquet.
- Drop unnecessary fields (for example, the role column in authors) to simplify the schema before building the gold layer.
- Register Delta tables (books, authors, reviews) to make the datasets easily accessible for SQL queries and future analytics.
- Curate the Gold layer by joining reviews, books, and authors to produce a single analytics-ready Delta table curated_reviews.

- Publish the curated dataset to the gold/ zone and register it as a Delta table for Power BI or feature engineering in upcoming steps.
- (Optional) Automate this pipeline by creating a Databricks Job, allowing quick re-execution after new ingestions.

IV.1. Create an Azure Databricks Workspace

- In **Azure Portal** → click **Create a resource**.
  - Search for **Azure Databricks** → click **Create**.
  - **Subscription** → same one you used earlier.
    - **Resource Group** → same one as ADF.
    - **Workspace name** → e.g. `goodreads-dbx-60XXXXXX`.
    - **Region** → same as your Data Lake,
      but the Qatar region is crowded. I had to go Southeast Asia....
    - Leave defaults for networking and advanced.
  - Click **Review + Create** → then **Create**.

IV.2. Create a Cluster

- Open your new Databricks workspace → click Launch Workspace.
- Go to Compute → + Create cluster.
- Cluster settings:
  - Name: `goodreads-cluster-60XXXXXX`.
  - Runtime: select a Long-Term Support (LTS) Databricks Runtime, e.g. Databricks Runtime 16.4 LTS (with Spark + ML).
    This ensures stability and avoids issues with preview versions.
  - Worker type:
    - Choose `Standard_D3_v2` or something similar in memory.

If you face an error of resources not available or VM not available. Pick bigger machines, but reduce the number of nodes in the min and max.

    - Choose min 2 and max 4.
  - Click Create Cluster.
- Open your new Databricks workspace → click **Launch Workspace**.

Note on Cost Control: Databricks clusters keep running (and billing) as long as they are active.
- Always Terminate your cluster from the Compute tab when you finish the lab.
- You can also enable Auto Termination (e.g., 30 minutes of inactivity) when creating the cluster, so it shuts down automatically if students forget.

IV.3. Use Databricks and Spark for Data Manipulation

a. Connect to ADLS Gen2 (Storage Account)

We'll access ADLS Gen2 via the **ABFSS** protocol. In a new notebook, set the storage account key in Spark config (temporary for the session).

```
# Replace with your actual storage account name and key
spark.conf.set(
    "fs.azure.account.key.goodreadsreviews60XXXXXX.dfs.core.windows.net",
    "<your-account-key>"
)
```

This code configures Spark to authenticate to your Storage Account so that **abfss://** paths are readable/writable. After this, you can read and write with URLs like:

```
abfss://lakehouse@goodreadsreviews60XXXXXX.dfs.core.windows.net/processed/books/.
```

Here's a rewritten and properly documented section you can use directly in the lab document:

b. Load Silver Parquet (Books & Authors)

In this step, we load the books and authors datasets that were ingested and transformed earlier using **Azure Data Factory**. These datasets are stored in the **silver layer** of the data lake in Parquet format.

They will be the foundation for the gold layer that we will build later.

```
# Load the books dataset from the silver layer
books = spark.read.parquet(
    "abfss://lakehouse@goodreadsreviews60XXXXXX.dfs.core.windows.net/processed/books/"
)

# Load the authors dataset from the silver layer
authors = spark.read.parquet(
    "abfss://lakehouse@goodreadsreviews60XXXXXX.dfs.core.windows.net/processed/authors/"
)
# Display the first few records to confirm the data was loaded correctly
books.show(5)
authors.show(5)
# Display the columns and their data types to verify the schema
books.printSchema()
authors.printSchema()
```

Explanation of the code:

- spark.read.parquet(...) loads the Parquet files stored in the silver layer into Spark DataFrames.
- books and authors are stored in the processed/ directory of the lakehouse container.
- .show(5) prints the first five rows of each dataset to help you quickly verify that the data was loaded correctly.
- .printSchema() prints the structure of the DataFrame, showing column names and data types. This step is important to confirm that the schema matches expectations before proceeding with curation and joins.

Got it — since the Parquet is the direct output of the pipeline and not cleaned yet, we'll **read the Parquet, identify "problematic" rows, drop them, and save a cleaned version** (and optionally keep a rejects file for audit).

---

c. Read reviews (Parquet) and profile

In this step, we load the Parquet from the silver layer and quickly profile it to understand basic quality issues before cleaning.

```
from pyspark.sql.functions import col, length, trim, count, when

# Read raw (uncleaned) reviews from the silver layer
reviews = spark.read.parquet(
    "abfss://lakehouse@goodreadsreviews60XXXXXX.dfs.core.windows.net/processed/reviews/"
)

# Peek at rows and schema
reviews.show(5, truncate=False)
reviews.printSchema()

# Basic profiling: counts and potential issues
total_rows = reviews.count()
null_review_id = reviews.filter(col("review_id").isNull()).count()
null_book_id   = reviews.filter(col("book_id").isNull()).count()
null_user_id   = reviews.filter(col("user_id").isNull()).count()
null_rating    = reviews.filter(col("rating").isNull()).count()
empty_text     = reviews.filter( (col("review_text").isNull()) | (trim(col("review_text")) ==
"") ).count()

print(f"Total rows: {total_rows}")
print(f"NULL review_id: {null_review_id}, NULL book_id: {null_book_id}, NULL user_id:
{null_user_id}, NULL rating: {null_rating}")
```

```
print(f"Empty/NULL review_text: {empty_text}")
```

What this does

- Loads the Parquet written by the pipeline (uncleaned).
- Displays a sample and schema to confirm columns and types.
- Computes quick quality counts for critical fields to guide cleaning rules.

d. Clean reviews by removing problematic rows

We apply pragmatic deletion rules to keep downstream joins and features stable. Feel free to adjust thresholds to your dataset.

```
from pyspark.sql.functions import col, trim, length

# Start from the existing Parquet-loaded DataFrame
# (Assumes you already did: reviews = spark.read.parquet(".../processed/reviews/"))
df = reviews

# 1) Drop rows missing critical keys
df = df.filter(
    col("review_id").isNotNull() &
    col("book_id").isNotNull() &
    col("user_id").isNotNull()
)

# 2) Enforce rating to be integer in [1..5]
df = df.withColumn("rating_int", col("rating").cast("int"))
df = df.filter(
    col("rating_int").isNotNull() &
    (col("rating_int") >= 1) &
    (col("rating_int") <= 5)
)

# 3) Normalize text; drop empty or ultra-short reviews (<10 chars after trim)
df = df.withColumn("review_text", trim(col("review_text")))
df = df.filter(
    col("review_text").isNotNull() &
    (length(col("review_text")) >= 10)
)

# 4) De-duplicate by review_id (keep arbitrary first; refine if you have timestamps)
df = df.dropDuplicates(["review_id"])
```

```
# 5) Select final shape
reviews_clean = df.select(
    "review_id",
    "book_id",
    "user_id",
    col("rating_int").alias("rating"),
    "review_text"
)
```

What this does

- Starts from the **uncleaned Parquet reviews** already produced by the pipeline.
- **Step 1:** Drops rows with missing critical keys (review_id, book_id, user_id) to ensure future joins and analytics won't break.
- **Step 2:** Casts rating to integer and keeps only values between 1 and 5 to enforce valid rating semantics.
- **Step 3:** Trims review_text and removes empty or very short reviews (under 10 characters), which are usually noise.
- **Step 4:** Removes duplicates based on review_id to avoid double counting.
- **Step 5:** Selects only the essential columns to build a clean and consistent reviews_clean dataset that will be used in the gold layer.

e. Persist Cleaned Reviews

In this step, we save the cleaned reviews_clean dataset back to the silver layer so that downstream steps (such as gold layer curation and Fabric feature engineering) use the validated, cleaned version.

```
# Write the cleaned reviews back to the silver layer (overwrite)
reviews_clean.write.mode("overwrite").parquet(
    "abfss://lakehouse@goodreadsreviews60XXXXXX.dfs.core.windows.net/processed/reviews/"
)

# Sanity check: re-read from disk and inspect schema and a few rows
reviews_verified = spark.read.parquet(
    "abfss://lakehouse@goodreadsreviews60XXXXXX.dfs.core.windows.net/processed/reviews/"
)
reviews_verified.printSchema()
reviews_verified.show(5, truncate=False)

print(f"Written cleaned rows: {reviews_verified.count()}")
```

What this does

- Saves the cleaned reviews_clean DataFrame back to the **processed/silver path**, overwriting the uncleaned pipeline output.
- Ensures all downstream steps will work on a clean and consistent dataset.
- Re-reads the Parquet file to validate that the structure and sample content are correct.
- Confirms the row count after cleaning.

Optional: If needed in the future, you can add separate "rejects" write step for audit. For now, this lab focuses on the cleaned dataset only.

Notes

- We're intentionally **deleting** problematic records instead of trying to repair them automatically to keep the lab simple and reliable.
- If your dataset includes a timestamp (e.g., date_added), you can refine the de-duplication step by keeping the **latest** review per review_id.
- All **heavy feature work** (casts, imputations, advanced NLP) will happen later in the Fabric/feature-engineering stage; here we're only ensuring structural consistency and basic data hygiene.

Homework Part I

- **Curate the Gold Table**
  - Join the books, authors, book_authors (bridge), and reviews_clean datasets into a single curated DataFrame.
  - Use the existing in-memory DataFrames (no reloading).
  - Join on book_id and author_id to bring all information together.
  - Keep **only these columns**:
    - review_id
    - book_id
    - title
    - author_id
    - name
    - user_id
    - rating
    - review_text
    - language
    - n_votes
    - date_added
  - Print the schema and display a few rows to verify your result.
- **Persist and Register the Gold Table**
  - Save your curated DataFrame as a **Delta table** and make it available for SQL queries.

- Write it to the gold zone in your lakehouse.
- Register it as a Delta table named curated_reviews.
- Run a simple SQL query to confirm it's registered correctly.

Homework Part I

V. Use Microsoft Fabric for Data Preprocessing

In this step, you will create your own **workspace in Microsoft Fabric** and connect it to the curated Delta table stored in **Azure Data Lake Storage (ADLS Gen2)**. This step is required because a workspace provides the environment where you will build and transform your dataset.

Note: You do **not** need to create any capacity in the Azure portal. There is one created for you.
If no capacity is available, you can start the **Fabric Trial** directly from the Fabric portal.
If the trial option is disabled, contact your tenant administrator.

V.1. Set up the Workspace

a. Create a Fabric Workspace

- Go to https://app.fabric.microsoft.com.
- If prompted, **start the Fabric Trial**:
  - Click your **profile icon** (top right).
  - Select **Start Fabric trial**.
- In the left panel, click **Workspaces → New workspace**.
- Name your workspace, for example: `goodreads-ws-60XXXXXX`
- Leave default settings and click **Create**.
- Once created, confirm it is linked to capacity:
  - If you started a trial, it is linked automatically.
  - If not, contact your professor to assign capacity *(He will be mad, bring something sweet or nice to appease him. Maybe and Oyster Perpetual? Or a Speedmaster Snoppy?)*.
- Inside your new workspace, click **New → Dataflow Gen2**.
- Choose **Add tables** to open the Power Query Editor.
- In the **Get Data** panel, select **Azure Data Lake Storage Gen2**.
- When asked for connection details:
  - URL: `https://<yourstorageaccount>.dfs.core.windows.net`
  - Authentication kind: Access key
  - Enter your storage account access key and click Connect (*You know where to get one*).

After the connection succeeds, close the navigator window (we only needed authentication). This is just a test.

b. Load the Delta Table

- Back in Power Query, click **Get Data** again → select **Blank Query**.
- In the query editor window, click **Advanced Editor** and replace the contents with:

```
let
  Source = AzureStorage.DataLake(
    "https://<yourstorageaccount>.dfs.core.windows.net/lakehouse/gold/curated_reviews/",
    [HierarchicalNavigation=true]
  ),
  DeltaTable = DeltaLake.Table(Source)
in
  DeltaTable
```

- **What this code does**
  This code connects Microsoft Fabric to your dataset stored in Azure Data Lake Storage (ADLS) and loads it as a Delta table that Power Query can understand.
  - **AzureStorage.DataLake(...)** – connects to your Azure storage account using its HTTPS address.
  - The link points to the folder that contains your processed data: lakehouse/gold/curated_reviews/.
  - The option [HierarchicalNavigation=true] lets Power Query explore folders and subfolders, just like a file explorer.
  - **DeltaLake.Table(Source)** – tells Power Query to read the folder as a **Delta table** instead of just individual files.
  - It automatically finds the Delta log (_delta_log folder) and combines all the Parquet files into one clean table.
  - This gives you structured data with columns such as review_id, book_id, title, and rating.
  - **in DeltaTable** – returns the final table so you can see it in Power Query, transform it, and later load it into your Warehouse or Datamart.

  In short, this code securely connects Fabric to your curated dataset in Azure and rebuilds it as a ready-to-use table for cleaning, analysis, and visualization.
- Click **Done**.

Power Query will load the folder as a **Delta table** and display columns such as: review_id, book_id, title, author_id, name, user_id, rating, review_text.

V.2. Clean the data

a. Adjust Data Types

- Before saving the table, make sure that every column has the correct data type and clean values. This step ensures consistency and prevents type or formatting errors later when you join, aggregate, or visualize the data.

- **Example: Casting review_id to Text**
  - In the **Power Query Editor**, select the column **review_id**.
  - From the top menu, click **Transform → Data Type → Whole Number**.
  - Confirm the change when prompted.

This ensures that all review IDs are treated as text, not numbers.
It's good practice because IDs are identifiers, not quantities.

Homework Part II

- Repeat similar transformations for all other columns and perform the following additional cleaning steps.
  Your grade will depend on how complete, consistent, and logically sound your cleaning is.
- **Set Correct Data Types for these columns**
  - `book_id`
  - `author_id`
  - `user_id`
  - `rating`
  - `n_votes`
  - `title, name, review_text, language`
  - `date_added`

Homework Part II

---

b. Handle Missing or Invalid Values

- In the **Power Query Editor**, select the column **rating**.
- On the top menu, click **Remove Rows → Remove Blank Rows**.

Homework Part II

- Repeat similar transformations for all other interesting columns, for instance:
  - **rating,**
  - **book_id,**
  - **review_text**
- Drop reviews with extremely short review_text (less than ~10 characters). *You might want to create a new column called* ***`review_length`***.
- Check that date_added values are valid dates; remove rows with invalid or future dates.
- Replace missing n_votes with **0** and missing language with "Unknown".

Homework Part II

---

c. Trim and Standardize Text

- Use **Transform → Format → Trim** on text columns (title, name, review_text).
- Use **Transform → Format → Capitalize Each Word** for title or name to standardize formatting.

V.3. Aggregations and Enrichment

Use the Power Query **Group By** to build some derived features:

- Average rating per BookID.
- Number of reviews per BookID.
- Average rating per AuthorName.
- Word count statistics on reviews.

V.4. Publish your results

- Click **Next → Publish**.
- When asked for a destination, select your **Warehouse** or **Datamart (Preview)** and name the table: curated_reviews
- Wait for the refresh to complete, then open your Warehouse/Datamart and preview the table to verify data.

Your grade for this step will **depend on the quality of your data transformations** in Fabric. Specifically, you will be evaluated on:

- The **cleaning steps** you applied (handling nulls, invalid values, trimming text, casting types, dropping irrelevant columns).

- The **clarity and consistency of the resulting columns** (proper naming, appropriate types, meaningful derived features).

At the end of this step, your publication might not work. Save your steps, and the SQL code for them, add it to the GitHub for this project and comment it.

If you manage to publish and save the processed data in Fabric. You are cool, and I award thee a 100% in this lab (upon completion), and in that of Tutorial 5 as well.

You need to tell me how you did it.

Hey you! Yes, you…
So, you have spent hours trying to publish, and ChatGPT is driving you crazy… All of this for that sweet 100%… is it *worth* it? Is this all what life is about?
Go outside, touch grass, feel the wind in your face, eat a bug[3].

Now, come back here and finish the lab the right way.

VI. Cleaning the Data in Databricks

Since transformations in Fabric could not be completed, continue the preprocessing directly in **Databricks**. Create a new notebook to clean and prepare the curated dataset for the **Gold layer**.

- **Create a new notebook** in Databricks (Python).
  - Load the curated dataset from the **silver layer** or from your **curated_reviews** table.
  - Use Spark to read the Parquet or Delta files.
- **Clean the data**:
  - Perform a complete cleaning process similar to what was intended in Fabric, *such as* (this is not exhaustive):
    - Remove rows where rating, review_text, book_id, or author_id are missing.
    - Remove duplicates by review_id or by (user_id, book_id).
    - Normalize text fields by trimming spaces, lowercasing where appropriate, and removing malformed characters.
    - Drop reviews with very short text.
    - Correct data types — ensure rating is numeric, IDs are text, and dates are properly formatted.
    - Drop unused or irrelevant columns.
  - Verify that all numeric columns contain valid values and fall within expected ranges.
- **Feature preparation**
  - Enhance the dataset with additional features:
    - Compute review length in words .

[3] I am legally obliged to tell you here that "Do not eat a bug!" If you or your loved are thinking of doing so, please, seek help. You really need it.

        - Aggregate by book_id to calculate average rating and number of reviews like you did in Fabric.
- **Save results to the Gold layer**
    - Save the cleaned and enriched dataset in Delta format under the **Gold** path named `features_v1`.
- Verify output
    - Reload the saved dataset and inspect its schema.
    - Check the record count and sample rows to confirm that cleaning and feature creation were successful.

The grade for this section will depend on the **thoroughness of cleaning**, the **quality of generated features**, and the **absence of data leakage** in the prepared dataset.
You are still able to get the bonus if you amaze me.
