Here is the complete documentation for **Project 3: The Data Cleaner**, following the same "Architect-First" structure we used for Projects 1 and 2.

This project introduces you to **Apache Spark** (via **Azure Databricks**), the industry standard for big data processing. You will move from "Moving Data" (ADF) to "Transforming Data" (Code).

PROJECT 3: THE DATA CLEANER

Role: Data Engineer

Difficulty: Easy (Foundation)

Est. Time: 4-6 Hours

---

1. Executive Summary & Scenario

The Business Problem:

"CityRide," a taxi startup, wants to optimize their pricing model. The Data Science team needs historical trip data to train their Machine Learning models.

However, the raw data provided by the city is "dirty":

1. **Nulls:** Some trips have missing passenger counts.
2. **Outliers:** Some trips have negative distances (impossible!).
3. **Duplicates:** The same trip ID appears multiple times due to system glitches.
4. **Wrong Types:** The "Total Amount" column is stored as a String ("$15.50"), but models need a Number (15.50).

The Goal:

Build a "Silver Layer" pipeline that:

1. Ingests the **Raw (Bronze)** Parquet files.
2. Uses **PySpark** on Databricks to clean, deduplicate, and type-cast the data.
3. Saves the clean data back to the Data Lake as **Delta Tables** (Silver Layer).

The Senior-Level Requirement:

- **Code Reusability:** Do not hardcode paths. The notebook must accept parameters (e.g., input_path, output_path).
- **Data Quality Checks:** The pipeline must *count* how many bad records were dropped

and log this metric.

---

2. Architecture Overview

We are introducing a new compute engine: **Azure Databricks**.

The Stack:

- **Source:** NYC Taxi Trip Data (Public Dataset).
- **Storage:** ADLS Gen2 (Bronze/Silver Containers).
- **Compute:** Azure Databricks (Standard Cluster).
- **Language:** Python (PySpark).

---

3. Pre-Requisites (Data Setup)

You need raw data. Since the full NYC dataset is huge (billions of rows), we will use a sample.

Action:

1. Go to Azure Open Datasets - NYC Taxi or search for "NYC Taxi Trip Data" on Kaggle.
2. Download **Yellow Taxi Trip Records (Parquet or CSV)** for **January 2023** (just one month).
3. Upload this file to your ADLS Gen2 container: raw-bronze/nyc-taxi/2023-01.parquet.

---

4. The Design Phase (Interactive)

As the Architect, you must decide *how* to clean the data before writing code.

Challenge A: Handling Nulls

Scenario: 5% of trips have passenger_count = NULL.

Task: What do we do with these rows?

- *Option 1:* Drop them? (Lose 5% of data).
- *Option 2:* Fill with 0? (Technically wrong).
- *Option 3:* Fill with 1 (Mode imputation)?
- *Decision:* Which option is safest for a pricing model?

Challenge B: The "Negative Money" Problem

Scenario: Some rows have total_amount = -10.50. This usually means a refund or error.

Task: Should we convert them to positive (abs(-10.50)) or drop them?

- *Hint:* If it's a refund, the trip *did* happen, but the money flow is reversed. Does a pricing model care about refunds?

Challenge C: Deduplication Strategy

Scenario: You have two rows with the exact same trip_id.

- Row A: trip_id=101, amount=15.00, timestamp=10:00
- Row B: trip_id=101, amount=15.00, timestamp=10:05
  Task: How do you decide which one to keep?
- *Method:* usage of dropDuplicates(['trip_id']) vs. Window Functions to keep the *latest* timestamp.

🛑 STOP! Think about these logic problems.

✅ Architect's Solution (Reference)

Answer A: Null Handling

For pricing models, a trip with 0 passengers is impossible. However, dropping 5% is aggressive.

- *Senior Decision:* **Impute with Median/Mode (1).** Most taxi rides have 1 passenger. This preserves the row for distance/cost analysis.

Answer B: Negative Values

- *Senior Decision:* **Drop them.** Negative amounts often indicate cancelled transactions or disputes. Training a model on "cancelled" trips will confuse it. Filter: WHERE total_amount > 0.

Answer C: Deduplication

- *Senior Decision:* Simple dropDuplicates() is risky because you might keep the *older/wrong* version.
- *Better Approach:* Use a **Window Function** partitioned by trip_id and ordered by ingestion_timestamp DESC to keep row number 1 (the latest version). For this beginner project, dropDuplicates() is acceptable if we assume the data is static.

---

5. Implementation Guide (Step-by-Step)

Phase 1: Databricks Setup

1. **Create Resource:** Create an **Azure Databricks Service** (Standard Tier).

2. **Launch Workspace:** Click "Launch Workspace".
3. **Cluster:**
    - Go to **Compute** -> **Create Compute**.
    - Name: cluster-project3.
    - Databricks Runtime: **12.2 LTS** (Standard).
    - Node Type: **Standard_DS3_v2** (Smallest/Cheapest).
    - *Important:* Set "Terminate after" to **20 minutes** (Save money!).

Phase 2: Connecting to Data Lake (Mounting)

Note: The modern way is "Unity Catalog" or "Direct Access," but "Mounting" is the classic method for beginners.

1. **Create a Secret Scope:** (Advanced, skipped for now. We will use Access Keys directly for simplicity, though not production-safe).
2. **Create a Notebook:** Name it 01_Ingest_Clean.
3. **Run this Code (Cell 1):** Connect to ADLS.

```
Python
# Configuration
storage_account_name = "adlsquickmartdev" # REPLACE ME
storage_account_key = "YOUR_ACCESS_KEY_HERE" # REPLACE ME (Get from Azure Portal > Storage > Access Keys)
container_name = "raw-bronze"

# Mount Config
spark.conf.set(
  f"fs.azure.account.key.{storage_account_name}.dfs.core.windows.net",
  storage_account_key
)

# Path to file
file_path = f"abfss://{container_name}@{storage_account_name}.dfs.core.windows.net/nyc-taxi/2023-01.parquet"
```

Phase 3: The Transformation (PySpark)

Step 3.1: Read Data (Bronze)

```
Python
```

```
# Read Parquet
df_raw = spark.read.parquet(file_path)

# Show schema (check for String types that should be Double)
df_raw.printSchema()
display(df_raw.limit(5))
```

Step 3.2: Cleaning Logic (Silver)

Python

```
from pyspark.sql.functions import col, lit, when

# 1. Handle Nulls (Impute Passenger Count with 1)
df_cleaned = df_raw.fillna(1, subset=['passenger_count'])

# 2. Filter Outliers (Negative Money & Distance)
df_cleaned = df_cleaned.filter(
    (col("total_amount") > 0) &
    (col("trip_distance") > 0)
)

# 3. Fix Data Types (Example: Casting string to double)
df_cleaned = df_cleaned.withColumn("total_amount", col("total_amount").cast("double"))

# 4. Remove Duplicates
df_cleaned = df_cleaned.dropDuplicates(['tpep_pickup_datetime', 'tpep_dropoff_datetime',
'PULocationID', 'DOLocationID'])

# Validation: Count rows dropped
original_count = df_raw.count()
final_count = df_cleaned.count()
print(f"Rows Dropped: {original_count - final_count}")
```

Step 3.3: Write Data (Silver)

Python

```
# Write to "Silver" container as Delta Table
output_path = f"abfss://silver-cleaned@{storage_account_name}.dfs.core.windows.net/nyc-taxi/"

df_cleaned.write.format("delta").mode("overwrite").save(output_path)
print("✅ Silver Data Written Successfully")
```

---

6. Key Learnings & Takeaways

- **PySpark DataFrame API:** You learned filter, withColumn, fillna, and dropDuplicates—the bread and butter of Data Engineering.
- **Lazy Evaluation:** Spark didn't actually *do* anything until you called .count() or .write(). This is how it optimizes big data plans.
- **Delta Lake:** You wrote the output as format("delta"). This creates a transaction log (_delta_log), allowing you to "Time Travel" later (we will use this in Project 5).

---

7. Project Defense (Thesis Questions)

Q1. The "Small File" Problem

"If your source data consisted of 10,000 tiny CSV files (1KB each), Spark would be very slow reading them. Why? And what command in Spark can you use to 'Compact' them into fewer, larger files when writing the output?"

Q2. Partitioning on Write

"We just wrote the data as one big blob. If analysts always query by VendorID, how should we modify the .write() command to optimize for that?"

Q3. Cluster Sizing

"You used a Single Node cluster. If the dataset was 10 TB, what kind of cluster would you need? Would you add more RAM or more Cores for a transformation that is 'Shuffle Heavy' (like Deduplication)?"

Q4. Hardcoded Keys

*"We pasted the Storage Key in the notebook. This is a security violation. How do we use **Databricks Secrets** (Azure Key Vault backed scope) to fix this?"*

Q5. Why Parquet/Delta?

"Why didn't we save the cleaned data as CSV? Give 2 reasons why Parquet/Delta is better for performance than CSV."

Reply with your answers to complete Project 3.

Once finished, you have completed **Phase 1 (The Foundation)**!

We will then proceed to **Phase 2 (The Builder)**, starting with **Project 4: The Modern Data Warehouse**, where things get significantly more complex (SCD Type 2, Star Schemas).