The accompanying document demonstrates my implementation using a Databricks notebook file, as it is easiest to visualise all I have done. In addition, I have included a modular and parameterised **wind\_turbine\_pipeline.py** script, which orchestrates the entire pipeline as a DAG. This script not only executes the pipeline end-to-end but also provides summary statistics and runs machine learning models. By offering a structured and reusable approach, it simplifies testing and verification while being more suitable for production environments.

A screenshot of a computer program

Description automatically generated

**Bronze**

I have implemented the data pipeline using a **Medallion Architecture**.

The process begins with ingesting raw data into a bronze layer, where I perform data quality checks. Errors are flagged explicitly (Null values found in critical fields), while soft warning messages are logged for further review.

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Silver**

From the bronze layer, the data transitions to a silver layer for further processing. In this stage:

The data is merged with existing tables, so that only the newly uploaded data is ingested. This can be done by attaching a load date using a workflow orchestration tool like ADF or Airflow.

Aggregations such as calculating the average wind speed, wind direction, and power output per wind turbine are performed and used to populate null values most accurately.

A screen shot of a computer code

Description automatically generated

Duplicate records are removed.

Rows with values outside the interquartile range (IQR) bounds are excluded.

A computer screen shot of text

Description automatically generated

**Gold**

Finally, the processed data is moved to the gold layer, where it is structured into a star schema for efficient querying and analytics. The schema includes a fact\_turbine\_metrics table, alongside dimension tables like dim\_turbine (I have made up some dummy data here) and dim\_date.

A screenshot of a computer

Description automatically generated

This warehouse enables comprehensive reporting, including daily aggregated turbine statistics. A table called daily\_summary is made to show average summary statistics of a turbine per day. Additionally, I created a daily\_anomalies table, which flags turbines exhibiting anomalous data for a given day. These could be stored in one table as well, but to have granularity for the purposes of this exercise I have kept them separate.

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

Given that the implementation is built on Databricks using Apache Spark, I utilized Delta Tables to manage data in each layer. This ensures robust versioning and facilitates tracking of slowly changing dimensions. For example, if a turbine's manufacturer changes, its historical records can still be traced through Delta versioning. The use of delta tables also allows us to revert to previous versions if errors occur that need to be fixed. Also, this task is done in one notebook however a prod scale implementation should have this done in separate jobs and orchestrated using a tool like ADF or Databricks workflows

**Machine Learning**

To extend the functionality, I developed two simple machine learning models:

A Linear Regression model to predict power output based on wind speed and wind direction.

A screenshot of a computer

Description automatically generated

A Logistic Regression model to classify anomalies using wind speed and wind direction as predictors.

A screenshot of a computer

Description automatically generated

**Assumptions and Scalability Enhancements:**

The provided solution assumes the dataset fits into memory across layers (bronze, silver, and gold). In real-world scenarios with larger datasets, scalability adjustments would be necessary. For instance:

In the silver layer, I would filter the bronze data by date to ingest only the last 24 hours and merge it with existing records in the silver layer. Or use a workflow orchestration tool like ADF to add a ‘load\_date’ column and only process today’s rows and merge that into silver. This incremental approach would extend to the gold layer, where only the most recent records from the silver layer are processed and merged into the fact\_turbine\_metrics table.

Similarly, the daily summary and daily anomalies tables would be updated incrementally to calculate metrics and anomalies only for the latest data, likely using an append method.

I am also assuming that the schema for the raw data DOES NOT change and have thus explicitly defined the schema.

By designing the pipeline with these optimizations, the solution ensures scalability while maintaining high performance and data accuracy.