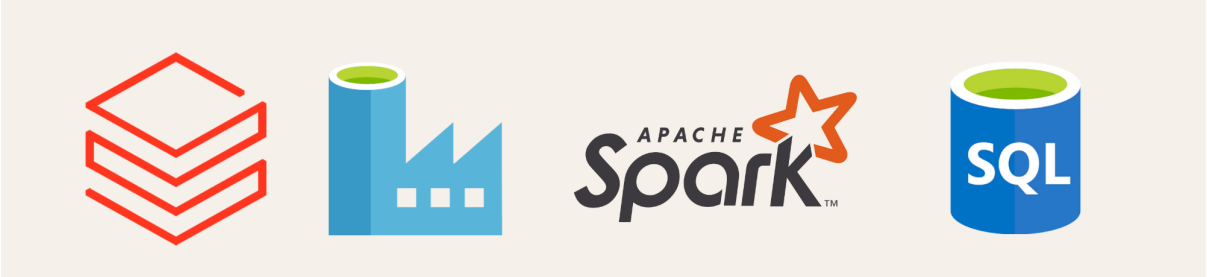
Azure Databricks and Data Factory Project (Formula1)

Seow Yong Tao

2023

Contents

1 Project Description

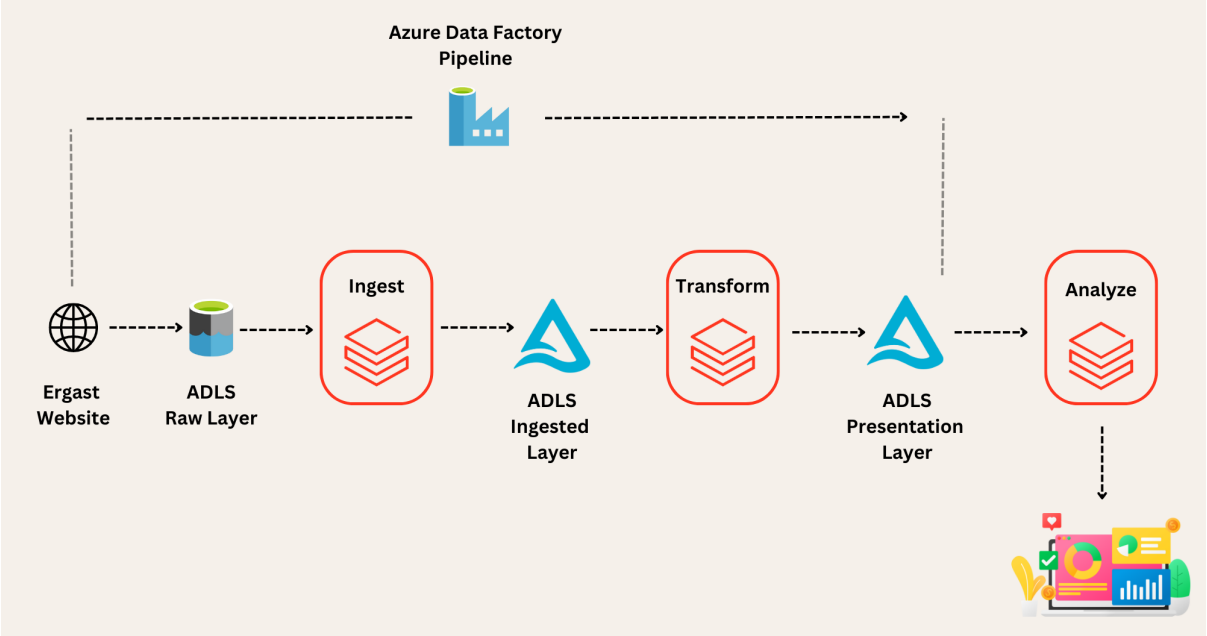


This project serves as a learning experience for me to delve into the world of Azure Databricks and Azure Data Factory and expand my skills in data engineering. The primary tools used in this project include Azure Databricks and Azure Data Factory, utilising SQL and Python (PySpark) programming languages.

The main objective of this project is to build a data pipeline in order to automate the ingestion of Formula 1 data (excel files) from the Ergast website (http://ergast.com/mrd/) and process it so that it is ready for analysis. The Ergast website updates the Formula 1 data on a weekly or biweekly basis, typically after each race. To accommodate this, a pipeline has been developed to execute on a weekly schedule.

While the primary emphasis of this project lies in the ingestion and processing of data, a simple dashboard has also been created to illustrate the concepts.

2 Overview



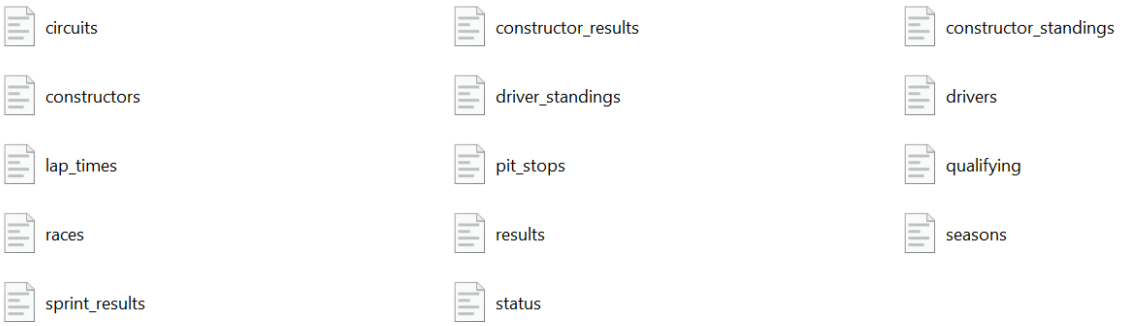
The architecture of this project is illustrated in the image above, showcasing the end-to-end data flow and processing.

The data sources are retrieved from the Ergast Website and then stored in the raw format within the Azure Data Lake Storage (ADLS) raw layer. In this initial stage, the data is in its unprocessed state, awaiting for further processing. The data is then processed and loaded into the ADLS ingested layer using Azure Databricks.

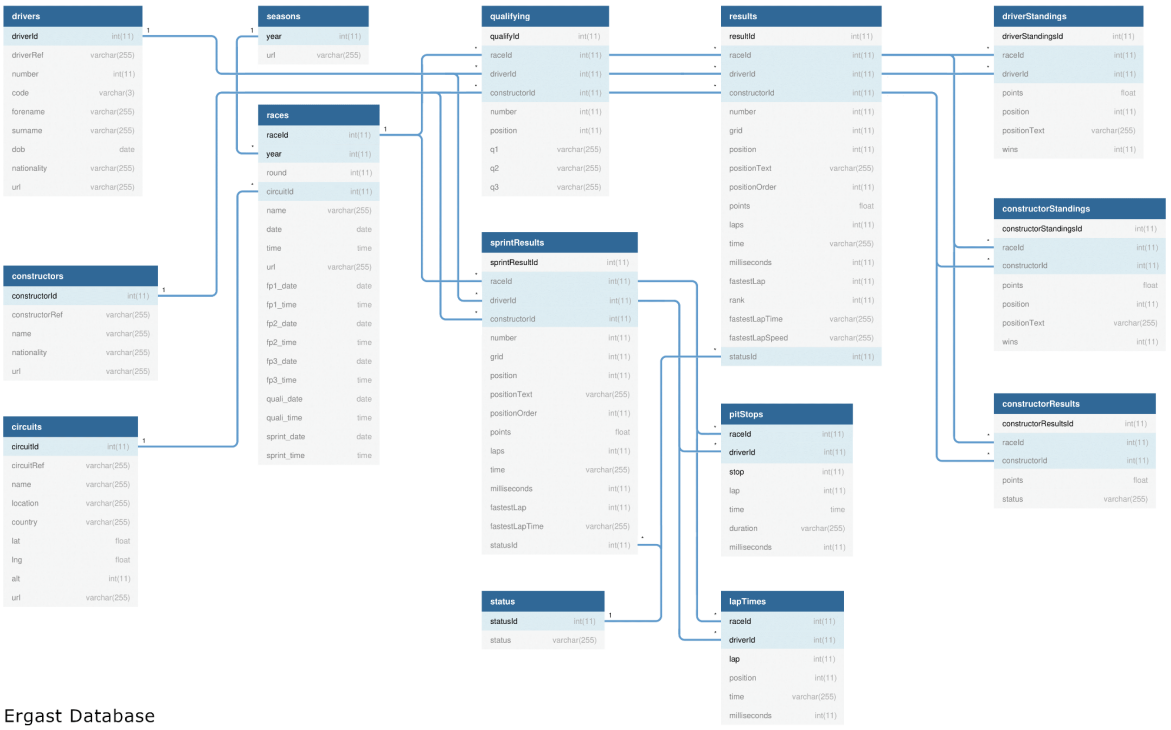
In the ADLS ingested layer, the data is prepped and optimised for analysis. Based on the analysis requirements, the data in the ADLS ingested layer will then be transformed or combined and loaded into the ADLS presentation layer. And lastly, a simple dashboard is built using the data sourced from the ADLS presentation layer.

Azure Data Factory is used to orchestrate and automate this end-to-end data pipeline. The pipeline is scheduled to run every Sunday at 10 pm, ensuring that the latest data is always available for analysis and visualisation.

3 Data Overview

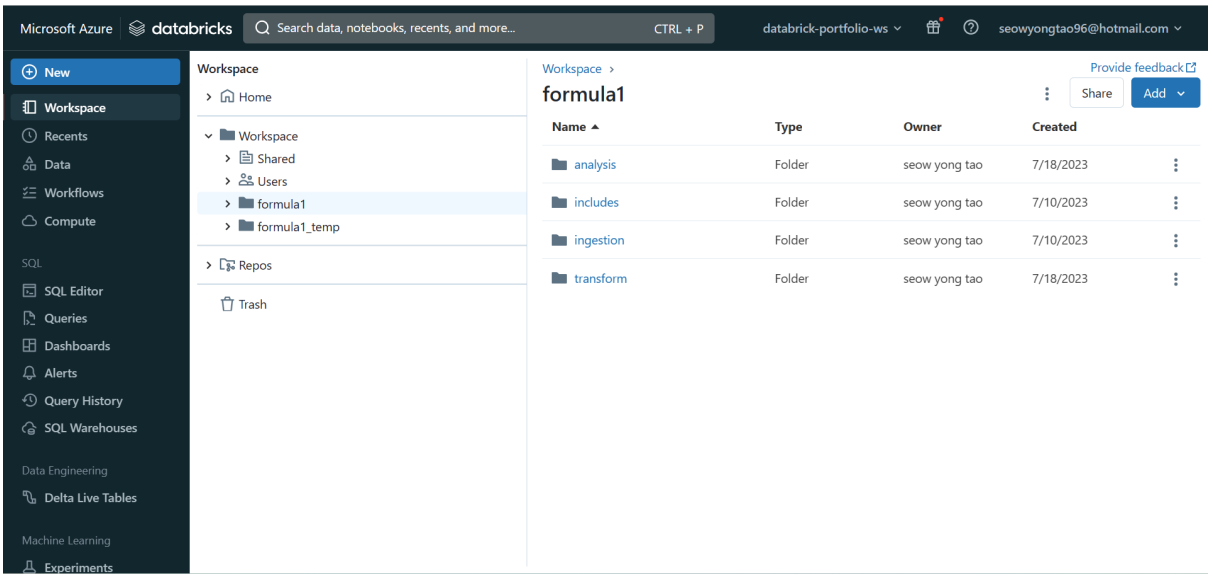


The project retrieves data from the Ergast website using the following link: http://ergast.com/downloads/f1db_csv.zip. Ergast provides a total of 14 CSV files, as shown in the image above. However, there are only 8 specific files used in this project: circuits, constructors, lap_times, races, pit_stops, results, drivers, and qualifying. Below is ER diagram obtained from Ergast website:



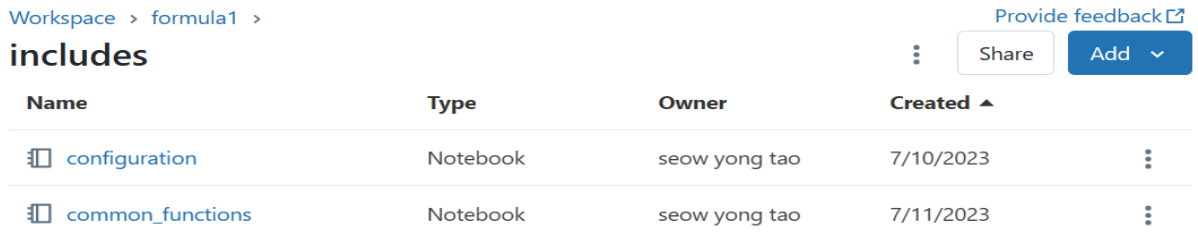
Ergast Database

4 Azure Databricks Notebooks



A workspace named “formula1” is created in the Databricks environment. Within this workspace, there are 4 folders which each contain notebooks for different purposes. In the following sections, each of the notebooks within the folders will be described.

4.1 includes



The “includes” folder contains notebooks that are used to save the common functions and common path links used throughout the project.

4.1.1. configuration

Description:

The “configuration” notebook is used to save the path links that are commonly used throughout the project.

Notebook’s Code:

```
raw_folder_path = "abfss://raw@formula1datalake722.dfs.core.windows.net"
processed_folder_path = "abfss://processed@formula1datalake722.dfs.core.windows.net"
presentation_folder_path = "abfss://presentation@formula1datalake722.dfs.core.windows.net"
```

4.1.2. common_functions

Description:

The “common_functions” notebook is used to save the common functions that are commonly used throughout the project.

Notebook’s Code:

```
from pyspark.sql.functions import current_timestamp

def add_ingestion_date(input_df):

    output_df = input_df.withColumn("ingestion_date", current_timestamp())

    return output_df
```

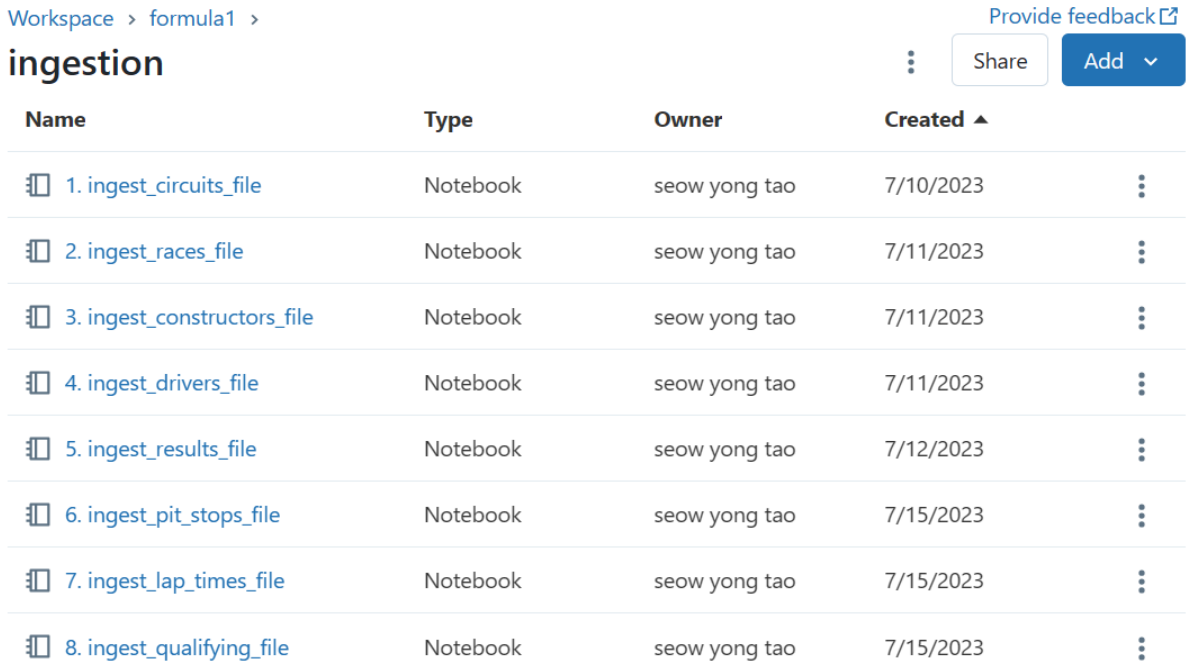
```
# put partition column to the end
def re_arrange_partition_column (input_df, partition_column):

    column_list = []

    for column_name in input_df.schema.names:
        if column_name != partition_column:
            column_list.append(column_name)

    column_list.append(partition_column)

    output_df = input_df.select(column_list)

    return output_df
```

```
def df_column_to_list(input_df, column_name):

    df_row_list = input_df.select(column_name)\
                          .distinct()\
                          .collect()

    column_value_list = [row[column_name] for row in df_row_list]

    return column_value_list
```

```
def overwrite_partition (input_df, db_name, table_name, partition_column):

    output_df = re_arrange_partition_column(input_df, partition_column)

    spark.conf.set("spark.sql.sources.partitionOverwriteMode", "dynamic")

    if (spark._jsparkSession.catalog().tableExists(f"{db_name}.{table_name}")):

        # spark expect the last column in the list to be the partition column, which is in our case "race_id"
        output_df.write.mode("overwrite").insertInto(f"{db_name}.{table_name}")

    else:

        output_df.write.mode("overwrite").partitionBy(partition_column).format("parquet").saveAsTable(f"{db_name}.{table_name}")
```

```
def merge_delta_data (input_df, db_name, table_name, folder_path, merge_condition, partition_column):

    spark.conf.set("spark.databricks.optimizer.dynamicPartitionPruning","true")

    from delta.tables import DeltaTable

    if (spark._jsparkSession.catalog().tableExists(f"{db_name}.{table_name}")):

        deltaTable = DeltaTable.forPath(spark, f"{folder_path}/{table_name}")
        deltaTable.alias("tgt").merge(input_df.alias("src"), merge_condition)\
                               .whenMatchedUpdateAll()\
                               .whenNotMatchedInsertAll()\
                               .execute()

    else:

        input_df.write.mode("overwrite").partitionBy(partition_column).format("delta").saveAsTable(f"{db_name}.{table_name}")
```

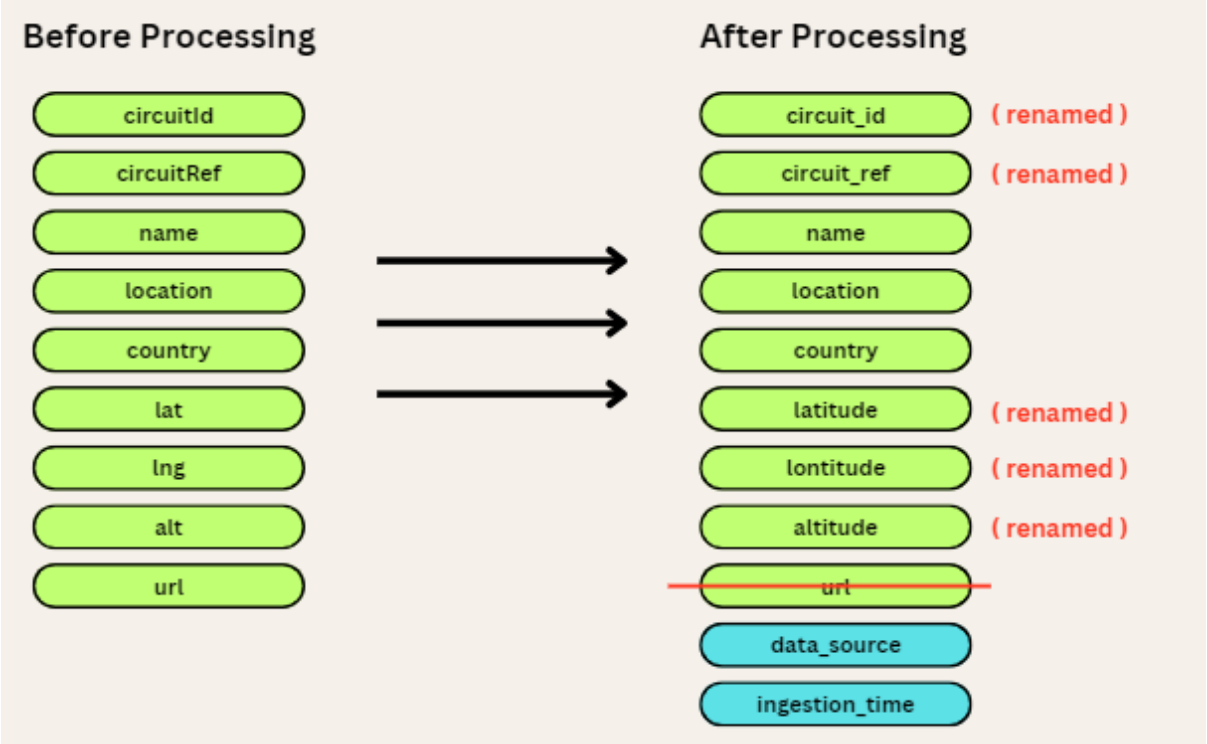
4.2 ingestion



The “ingestion” folder contains notebooks that are used to process different source files in the ADLS raw layer and load them into the ADLS ingested layer.

4.2.1. ingest_circuits_file

Below image shows the transformation performed in this notebook:



Notebook's Code:

```
dbutils.widgets.text("p_data_source", "ergast")
v_data_source = dbutils.widgets.get("p_data_source")
```

```
%run "../includes/configuration"
```

```
%run "../includes/common_functions"
```

```
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, IntegerType

circuits_schema = StructType(fields=[
    StructField("circuitId", IntegerType(), False),
    StructField("circuitRef", StringType(), True),
    StructField("name", StringType(), True),
    StructField("location", StringType(), True),
    StructField("country", StringType(), True),
    StructField("lat", DoubleType(), True),
    StructField("lng", DoubleType(), True),
    StructField("alt", IntegerType(), True),
    StructField("url", StringType(), True)
])
```

```
circuits_df = spark.read\
                    .option("header", True)\
                    .schema(circuits_schema)\
                    .csv(f"{raw_folder_path}/circuits.csv")
```

```
from pyspark.sql.functions import col

circuits_selected_df = circuits_df.select(col("circuitID"),col("circuitRef"), col("name"),
col("location"), col("country"), col("lat"), col("lng"), col("alt"))
```

```
circuits_renamed_df = circuits_selected_df.withColumnRenamed("circuitID", "circuit_id")\
                                          .withColumnRenamed("circuitRef", "circuit_ref")\
                                          .withColumnRenamed("lat", "latitude")\
                                          .withColumnRenamed("lng", "longtitude")\
                                          .withColumnRenamed("alt", "altitude")
```

```
from pyspark.sql.functions import lit

circuits_column_added = circuits_renamed_df.withColumn("data_source", lit(v_data_source))
```

```
circuits_final_df = add_ingestion_date(circuits_column_added)
```

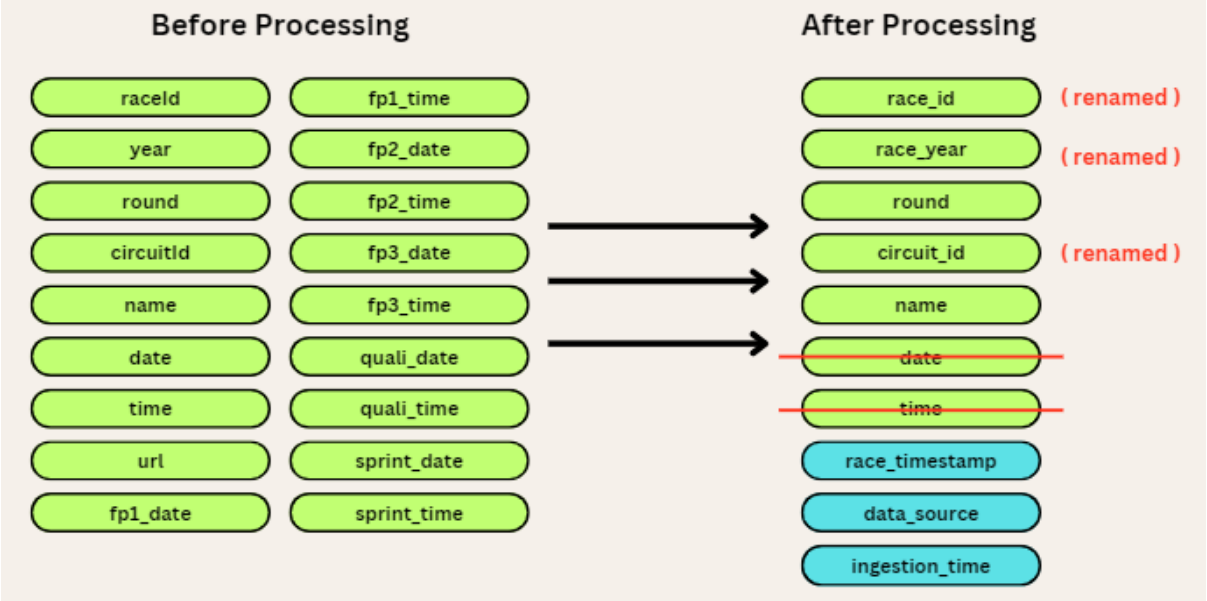
```
circuits_final_df.write.mode("overwrite").format("delta").saveAsTable("f1_processed.circuits")
```

```
dbutils.notebook.exit("Success")
```

```
Success
```

4.2.2. Ingest_races_file

Below image shows the transformation performed in this notebook:



Notebook's Code:

```
dbutils.widgets.text("p_data_source", "ergast")
v_data_source = dbutils.widgets.get("p_data_source")
```

```
%run "../includes/configuration"
```

```
%run "../includes/common_functions"
```

```
from pyspark.sql.types import StructType, StructField, IntegerType, DoubleType, StringType, TimestampType

races_schema = StructType(fields=[
    StructField("raceId", IntegerType(), False),
    StructField("year", IntegerType(), True),
    StructField("round", IntegerType(), True),
    StructField("circuitId", IntegerType(), False),
    StructField("name", StringType(), True),
    StructField("date", StringType(), True),
    StructField("time", StringType(), True),
    StructField("url", StringType(), True),
    StructField("fp1_date", StringType(), True),
    StructField("fp1_time", StringType(), True),
    StructField("fp2_date", StringType(), True),
    StructField("fp2_time", StringType(), True),
    StructField("fp3_date", StringType(), True),
    StructField("fp3_time", StringType(), True),
    StructField("quali_date", StringType(), True),
    StructField("quali_time", StringType(), True),
    StructField("sprint_date", StringType(), True),
    StructField("sprint_time", StringType(), True),
])
```

```
races_df = spark.read\
                .option("header", True)\
                .schema(races_schema)\
                .csv(f"{raw_folder_path}/races.csv")
```

```
from pyspark.sql.functions import col, concat, lit, to_timestamp, current_timestamp

races_add_race_timestamp_df = races_df.withColumn("race_timestamp", to_timestamp(concat(col("date"), lit(" "), col("time")),
'yyyy-MM-dd HH:mm:ss'))
```

```
races_add_ingestion_date_df = add_ingestion_date(races_add_race_timestamp_df)
```

```
from pyspark.sql.functions import col

races_selected_df = races_add_ingestion_date_df.select(col("raceId"), col("year"), col("round"), col("circuitId"),
col("name"),col("race_timestamp"), col("ingestion_date") )
```

```
races_column_added = races_selected_df.withColumn("data_source", lit(v_data_source))
```

```
races_final_df = races_column_added.withColumnRenamed("raceId", "race_id")\
                                   .withColumnRenamed("year", "race_year")\
                                   .withColumnRenamed("circuitId", "circuit_id")
```

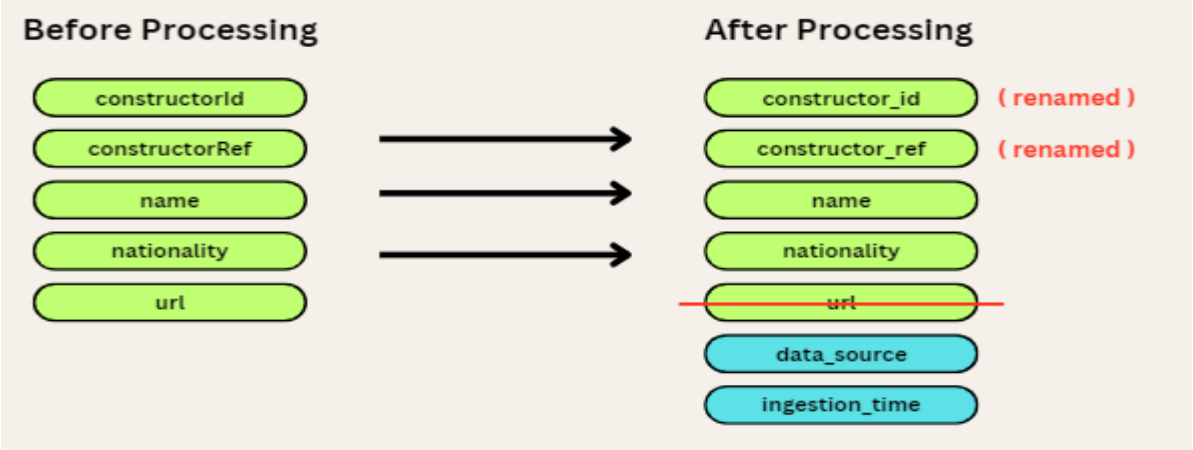
```
races_final_df.write.mode("overwrite").partitionBy('race_year').format("delta").saveAsTable("f1_processed.races")
```

```
dbutils.notebook.exit("Success")
```

```
Success
```

4.2.3. ingest_contructors_file

Below image shows the transformation performed in this notebook:



Notebook's Code:

```
dbutils.widgets.text("p_data_source", "ergast")
v_data_source = dbutils.widgets.get("p_data_source")
```

```
%run "../includes/configuration"
```

```
%run "../includes/common_functions"
```

```
constructors_schema = "constructorId INT, constructorRef STRING, name STRING, nationality STRING, url STRING"
```

```
constructors_df = spark.read \
                        .option("header", True)\
                        .schema(constructors_schema) \
                        .csv(f"{raw_folder_path}/constructors.csv")
```

```
from pyspark.sql.functions import col

constructors_dropped_df = constructors_df.drop(col("url"))
```

```
from pyspark.sql.functions import current_timestamp, lit

constructors_renamed_df = constructors_dropped_df\
                          .withColumnRenamed("constructorId", "constructor_id")\
                          .withColumnRenamed("constructorRef", "constructor_ref")
```

```
constructors_final_df = add_ingestion_date(constructors_renamed_df)\
                        .withColumn("data_source", lit(v_data_source))
```

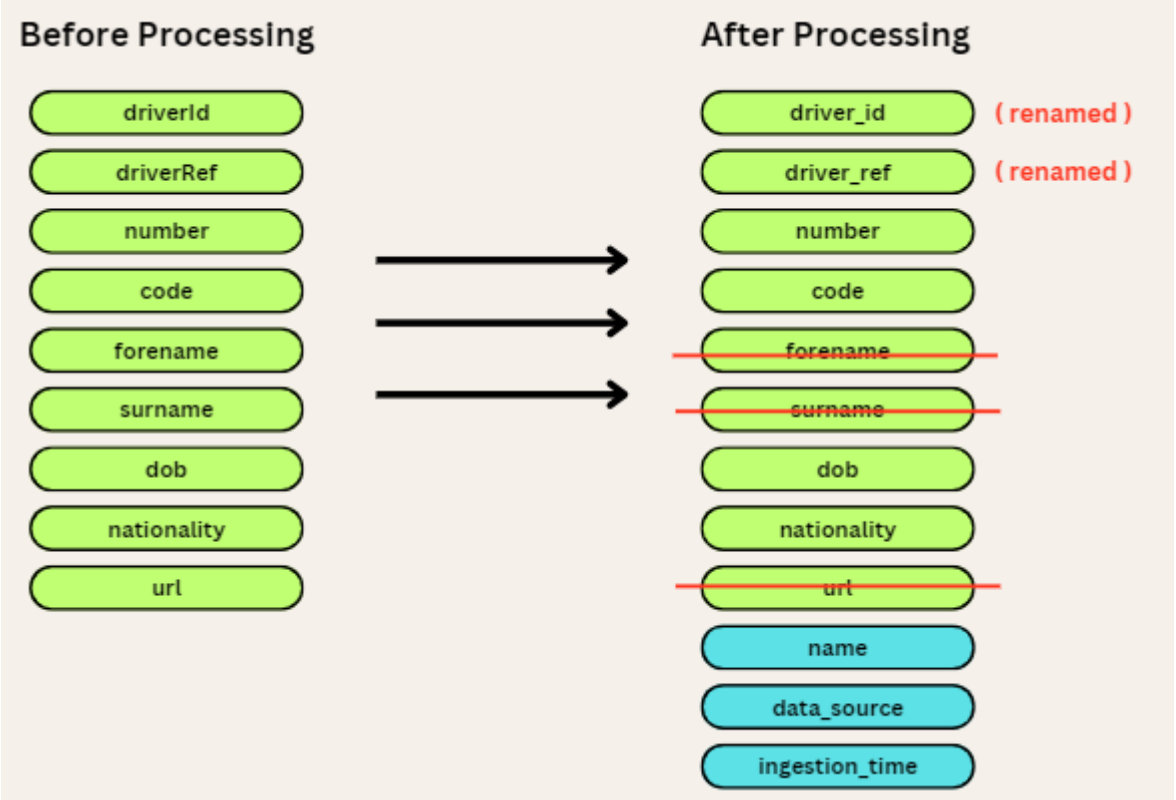
```
constructors_final_df.write.mode("overwrite").format("delta").saveAsTable("f1_processed.constructors")
```

```
dbutils.notebook.exit("Success")
```

```
Success
```

4.2.4. ingest_drivers_file

Below image shows the transformation performed in this notebook:



Notebook's Code:

```
dbutils.widgets.text("p_data_source", "ergast")
v_data_source = dbutils.widgets.get("p_data_source")
```

```
%run "../includes/configuration"
```

```
%run "../includes/common_functions"
```

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DateType

drivers_schema = StructType(fields=[
    StructField("driverId", IntegerType(), False),
    StructField("driverRef", StringType(), True),
    StructField("number", IntegerType(), True),
    StructField("code", StringType(), True),
    StructField("forename", StringType(), True),
    StructField("surname", StringType(), True),
    StructField("dob", DateType(), True),
    StructField("nationality", StringType(), True),
    StructField("url", StringType(), True),
])
```

```
drivers_df = spark.read\
                  .option("header", True)\
                  .schema(drivers_schema)\
                  .csv(f"{raw_folder_path}/drivers.csv")
```

```
from pyspark.sql.functions import current_timestamp, concat, col, lit

drivers_columns_added_df = add_ingestion_date(drivers_df)\
                           .withColumn("name", concat(col("forename"), lit(" "), col("surname")))\
                           .withColumn("data_srouce", lit(v_data_source))
```

```
drivers_renamed_df = drivers_columns_added_df.withColumnRenamed("driverId", "driver_id")\
                                             .withColumnRenamed("driverRef", "driver_ref")
```

```
drivers_final_df = drivers_renamed_df.drop(col("url"))
```

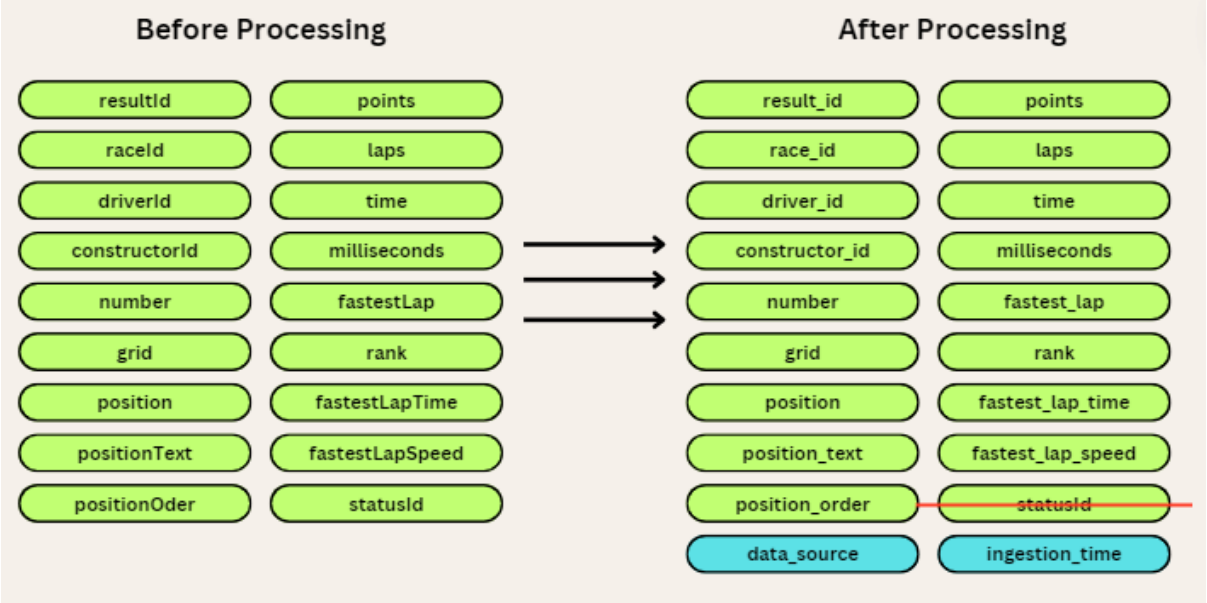
```
drivers_final_df.write.mode("overwrite").format("delta").saveAsTable("f1_processed.drivers")
```

```
dbutils.notebook.exit("Success")
```

```
Success
```

4.2.5. ingest_results_file

Below image shows the transformation performed in this notebook:



Notebook's Code:

```
dbutils.widgets.text("p_data_source", "ergast")
v_data_source = dbutils.widgets.get("p_data_source")
```

```
%run "../includes/configuration"
```

```
%run "../includes/common_functions"
```

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType

results_schema = StructType(fields=[
    StructField("resultId", IntegerType(), False),
    StructField("raceId", IntegerType(), True),
    StructField("driverId", IntegerType(), True),
    StructField("constructorId", IntegerType(), True),
    StructField("number", IntegerType(), True),
    StructField("grid", IntegerType(), True),
    StructField("position", IntegerType(), True),
    StructField("positionText", StringType(), True),
    StructField("positionOrder", IntegerType(), True),
    StructField("points", DoubleType(), True),
    StructField("laps", IntegerType(), True),
    StructField("time", StringType(), True),
    StructField("milliseconds", IntegerType(), True),
    StructField("fastestLap", IntegerType(), True),
    StructField("rank", IntegerType(), True),
    StructField("fastestLaptime", StringType(), True),
    StructField("fastestLapSpeed", StringType(), True),
    StructField("statusId", IntegerType(), True)
])
```

```
results_df = spark.read\
                  .option("header", True)\
                  .schema(results_schema)\
                  .csv(f"{raw_folder_path}/results.csv")
```

```
from pyspark.sql.functions import col

results_dropped_df = results_df.drop(col("statusId"))
```

```
results_renamed_df = results_dropped_df.withColumnRenamed("resultId", "result_id")\
                                       .withColumnRenamed("raceId", "race_id")\
                                       .withColumnRenamed("driverId", "driver_id")\
                                       .withColumnRenamed("constructorId", "constructor_id")\
                                       .withColumnRenamed("positionText", "position_text")\
                                       .withColumnRenamed("positionOrder", "position_order")\
                                       .withColumnRenamed("fastestLap", "fastest_lap")\
                                       .withColumnRenamed("fastestLapTime", "fastest_lap_time")\
                                       .withColumnRenamed("fastestLapSpeed", "fastest_lap_speed")
```

```
from pyspark.sql.functions import current_timestamp, lit

results_final_df = add_ingestion_date(results_renamed_df)\
                   .withColumn("data_source", lit(v_data_source))
```

```
results_deduped_df = results_final_df.dropDuplicates(["race_id", "driver_id"])
```

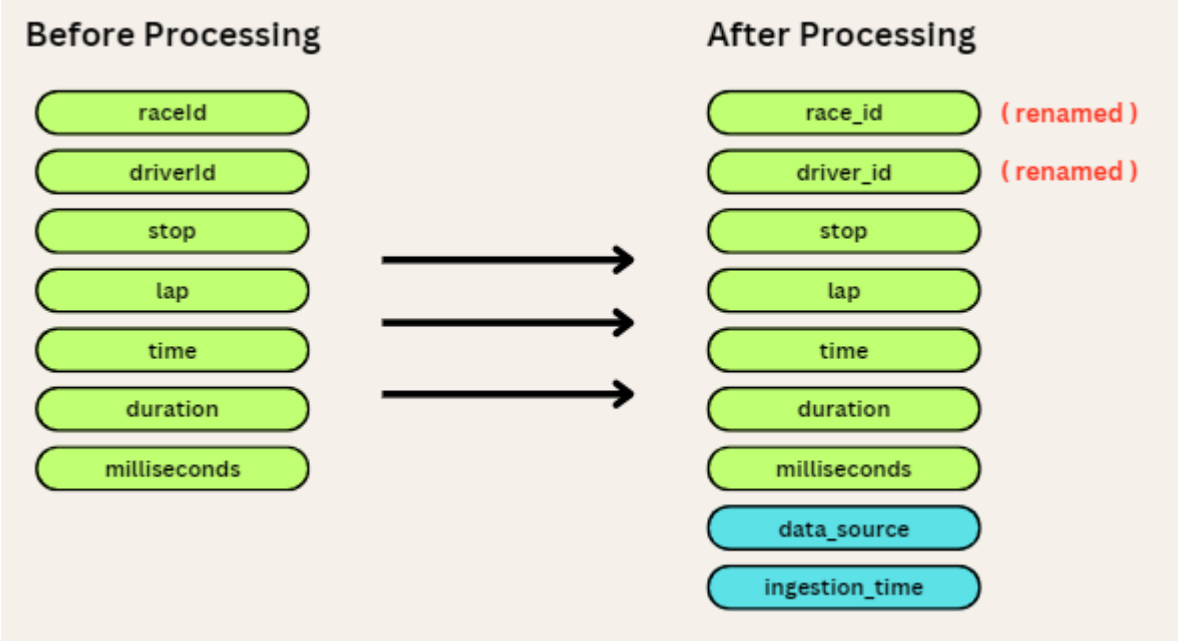
```
merge_condition = "tgt.result_id = src.result_id AND tgt.race_id = src.race_id"
merge_delta_data(results_deduped_df, "f1_processed", "results", processed_folder_path, merge_condition, "race_id")
```

```
dbutils.notebook.exit("Success")
```

```
Success
```

4.2.6. ingest_pit_stops_file

Below image shows the transformation performed in this notebook:



Notebook's Code:

```
dbutils.widgets.text("p_data_source", "ergast")
v_data_source = dbutils.widgets.get("p_data_source")
```

```
%run "../includes/configuration"
```

```
%run "../includes/common_functions"
```

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType

pit_stops_schema = StructType(fields=[
    StructField("raceId", IntegerType(), False),
    StructField("driverId", IntegerType(), True),
    StructField("stop", StringType(), True),
    StructField("lap", IntegerType(), True),
    StructField("time", StringType(), True),
    StructField("duration", StringType(), True),
    StructField("milliseconds", IntegerType(), True)
])
```

```
pit_stops_df = spark.read\
                    .option("header", True)\
                    .schema(pit_stops_schema)\
                    .csv(f"{raw_folder_path}/pit_stops.csv")
```

```
pit_stops_renamed_df = pit_stops_df.withColumnRenamed("raceId", "race_id")\
                                   .withColumnRenamed("driverId", "driver_id")
```

```
from pyspark.sql.functions import current_timestamp, lit

pit_stops_final_df = add_ingestion_date(pit_stops_renamed_df)\
                  .withColumn("data_source", lit(v_data_source))
```

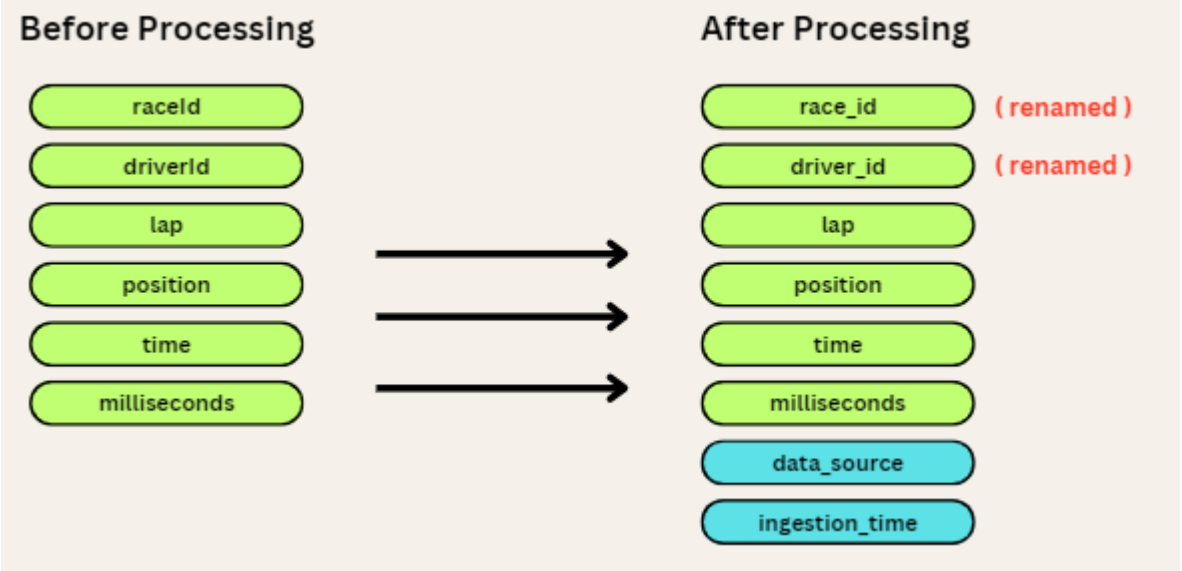
```
merge_condition = "tgt.race_id = src.race_id AND tgt.driver_id = src.driver_id AND tgt.stop = src.stop"
merge_delta_data(pit_stops_final_df, "f1_processed", "pit_stops", processed_folder_path, merge_condition, "race_id")
```

```
dbutils.notebook.exit("Success")
```

```
Success
```

4.2.7. ingest_lap_times_file

Below image shows the transformation performed in this notebook:



Notebook's Code:

```
dbutils.widgets.text("p_data_source", "ergast")
v_data_source = dbutils.widgets.get("p_data_source")
```

```
%run "../includes/configuration"
```

```
%run "../includes/common_functions"
```

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType

lap_times_schema = StructType(fields=[
    StructField("raceId", IntegerType(), False),
    StructField("driverId", IntegerType(), True),
    StructField("lap", IntegerType(), True),
    StructField("position", IntegerType(), True),
    StructField("time", StringType(), True),
    StructField("milliseconds", IntegerType(), True)
])
```

```
lap_times_df = spark.read\
                    .option("header", True)\
                    .schema(lap_times_schema)\
                    .csv(f"{raw_folder_path}/lap_times.csv")
```

```
lap_times_renamed_df = lap_times_df.withColumnRenamed("raceId", "race_id")\
                                   .withColumnRenamed("driverId", "driver_id")
```

```
from pyspark.sql.functions import current_timestamp, lit

lap_times_final_df = add_ingestion_date(lap_times_renamed_df)\
                     .withColumn("data_source", lit(v_data_source))
```

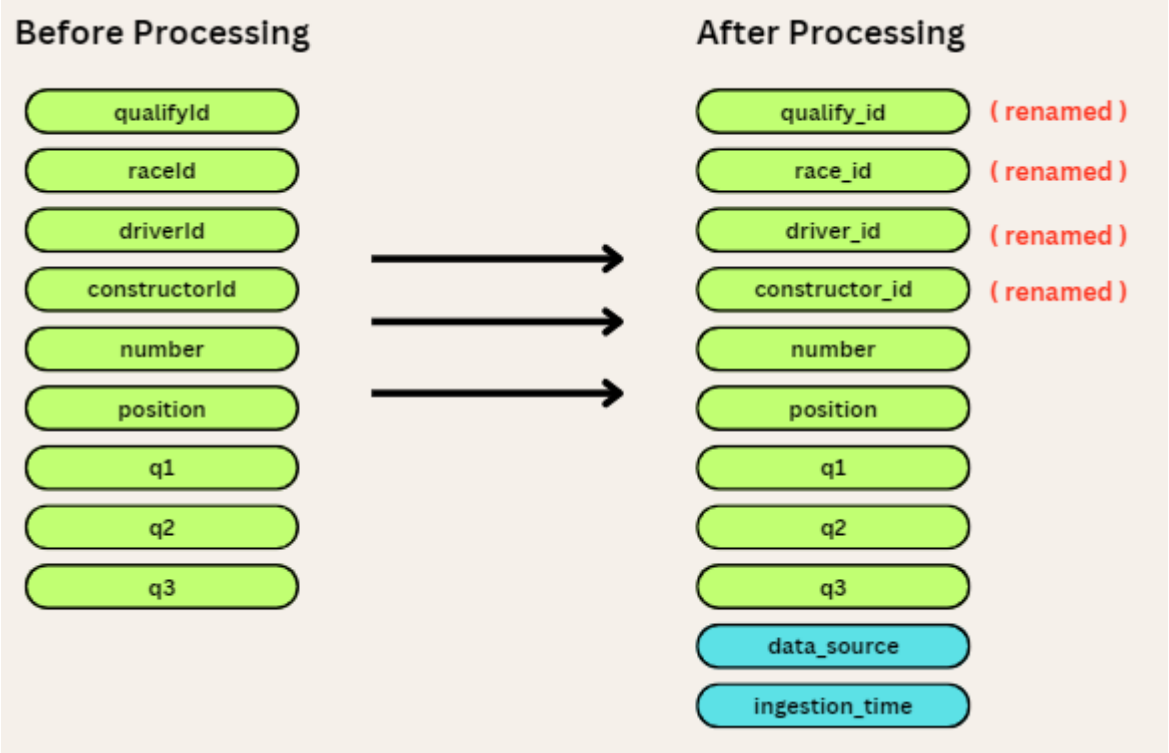
```
merge_condition = "tgt.race_id = src.race_id AND tgt.driver_id = src.driver_id AND tgt.lap = src.lap"
merge_delta_data(lap_times_final_df, "f1_processed", "lap_times", processed_folder_path, merge_condition, "race_id")
```

```
dbutils.notebook.exit("Success")
```

```
Success
```

4.2.8. ingest_qualifying_file

Below image shows the transformation performed in this notebook:



Notebook's Code:

```
dbutils.widgets.text("p_data_source", "ergast")
v_data_source = dbutils.widgets.get("p_data_source")
```

```
%run "../includes/configuration"
```

```
%run "../includes/common_functions"
```

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType

qualifying_schema = StructType(fields=[
    StructField("qualifyId", IntegerType(), False),
    StructField("raceId", IntegerType(), True),
    StructField("driverId", IntegerType(), True),
    StructField("constructorId", IntegerType(), True),
    StructField("number", IntegerType(), True),
    StructField("position", IntegerType(), True),
    StructField("q1", StringType(), True),
    StructField("q2", StringType(), True),
    StructField("q3", StringType(), True)
])
```

```
qualifying_df = spark.read\
                    .option("header", True)\
                    .schema(qualifying_schema)\
                    .csv(f"{raw_folder_path}/qualifying.csv")
```

```
qualifying_renamed_df = qualifying_df.withColumnRenamed("qualifyId", "qualify_id")\
                                     .withColumnRenamed("raceId", "race_id")\
                                     .withColumnRenamed("driverId", "driver_id")\
                                     .withColumnRenamed("constructorId", "constructor_id")
```

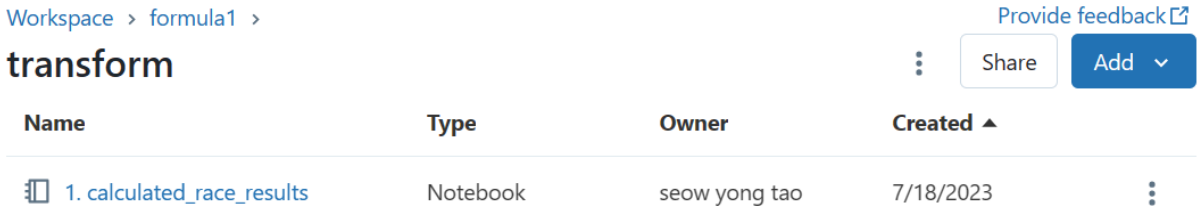
```
from pyspark.sql.functions import current_timestamp, lit

qualifying_final_df = add_ingestion_date(qualifying_renamed_df)\
                      .withColumn("data_source", lit(v_data_source))
```

```
merge_condition = "tgt.race_id = src.race_id AND tgt.qualify_id = src.qualify_id"
merge_delta_data(qualifying_final_df, "f1_processed", "qualifying", processed_folder_path, merge_condition, "race_id")
```

```
dbutils.notebook.exit("Success")
```

```
Success
```

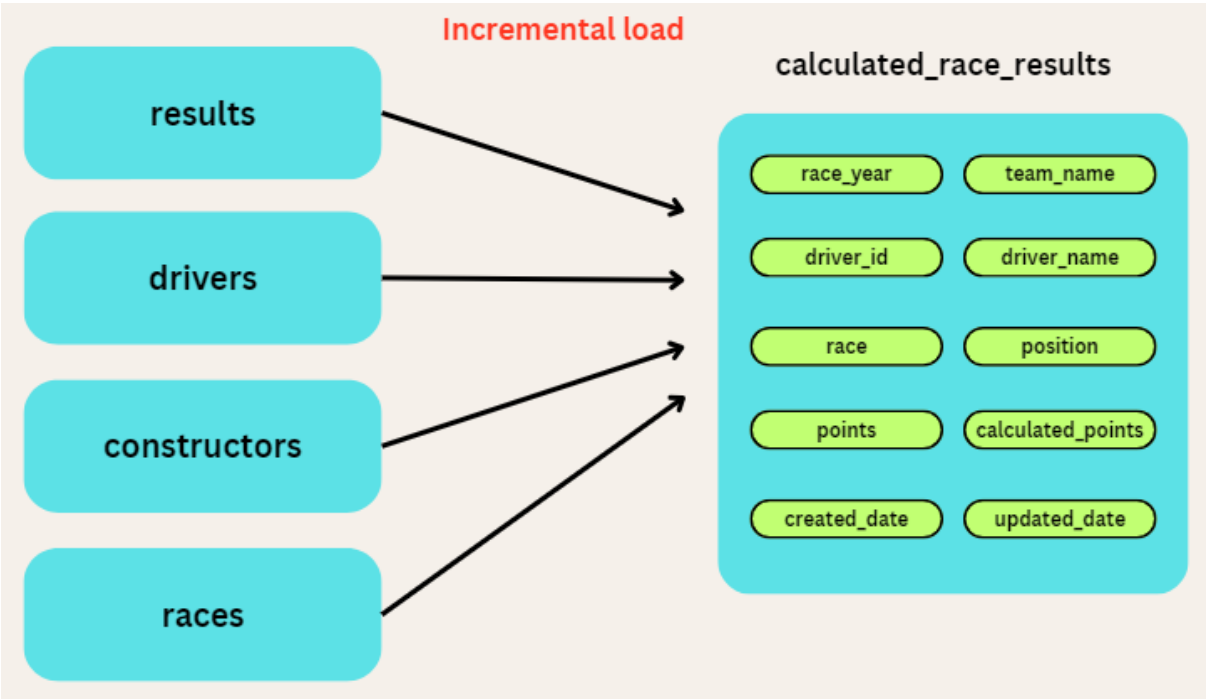
4.3 transform



The "transform" folder serves as a repository for notebooks responsible for transforming and combining data from various sources within the ADLS ingested layer and loading them into the ADLS presentation layer. These notebooks are instrumental in preparing the data for further analysis and visualisation. As shown in the image above, a notebook named “calculated_race_results” was created to demo the concept.

4.3.1. calculated_race_results

Below image shows the transformation performed in this notebook:



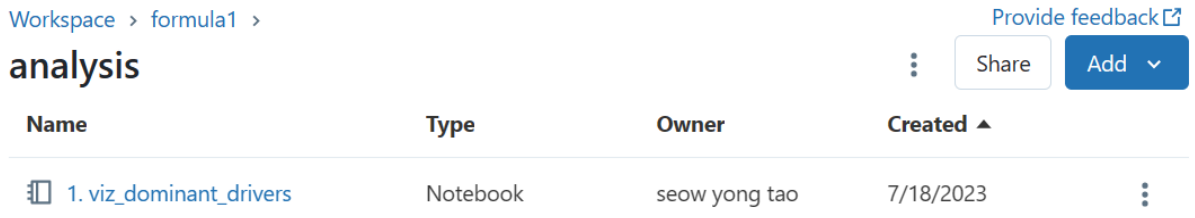
Notebook’s Code:

```
spark.sql(
    f"""
        CREATE TABLE IF NOT EXISTS f1_presentation.calculated_race_results
        (
        race_year INT,
        team_name STRING,
        driver_id INT,
        driver_name STRING,
        race_id INT,
        position INT,
        points INT,
        calculated_points INT,
        created_date TIMESTAMP,
        updated_date TIMESTAMP
        )
        USING DELTA
    """
)
```

```
spark.sql(
    f"""
        CREATE OR REPLACE TEMP VIEW race_result_updated
        AS
        SELECT races.race_year,
               constructors.name AS team_name,
               drivers.driver_id,
               drivers.name AS driver_name,
               races.race_id,
               results.position,
               results.points,
               11 - results.position AS calculated_points
          FROM f1_processed.results
          JOIN f1_processed.drivers ON (results.driver_id == drivers.driver_id)
          JOIN f1_processed.constructors ON (results.constructor_id == constructors.constructor_id)
          JOIN f1_processed.races ON (results.race_id == races.race_id)
         WHERE results.position <= 10
    """
)
```

```
spark.sql(
    f"""
        MERGE INTO f1_presentation.calculated_race_results tgt
        USING race_result_updated upd
        ON (tgt.driver_id = upd.driver_id AND tgt.race_id = upd.race_id)
        WHEN MATCHED THEN
        UPDATE SET tgt.position = upd.position,
                   tgt.points = upd.points,
                   tgt.calculated_points = upd.calculated_points,
                   tgt.updated_date = current_timestamp
        WHEN NOT MATCHED
        THEN INSERT (race_year, team_name, driver_id, driver_name, race_id, position, points, calculated_points,
created_date)
        VALUES  (race_year, team_name, driver_id, driver_name, race_id, position, points, calculated_points,
current_timestamp)
    """
)
```

4.4 analysis



The "analysis" folder serves as a repository for notebooks responsible for performing analysis or creating the visualisations using the data in the ADLS presentation layer.

4.4.1. Viz_dominant_drivers

This notebook is used to create the dashboard using the data created in the transform stage.

Notebook's Code:

```
CREATE OR REPLACE TEMP VIEW v_dominant_drivers
AS
SELECT driver_name,
       count(1) AS total_races,
       SUM(calculated_points) AS total_points,
       AVG(calculated_points) AS avg_points,
       RANK() OVER(ORDER BY AVG(calculated_points) DESC) driver_rank
  FROM f1_presentation.calculated_race_results
GROUP BY driver_name
HAVING count(1) >= 50
ORDER BY avg_points DESC
```

OK

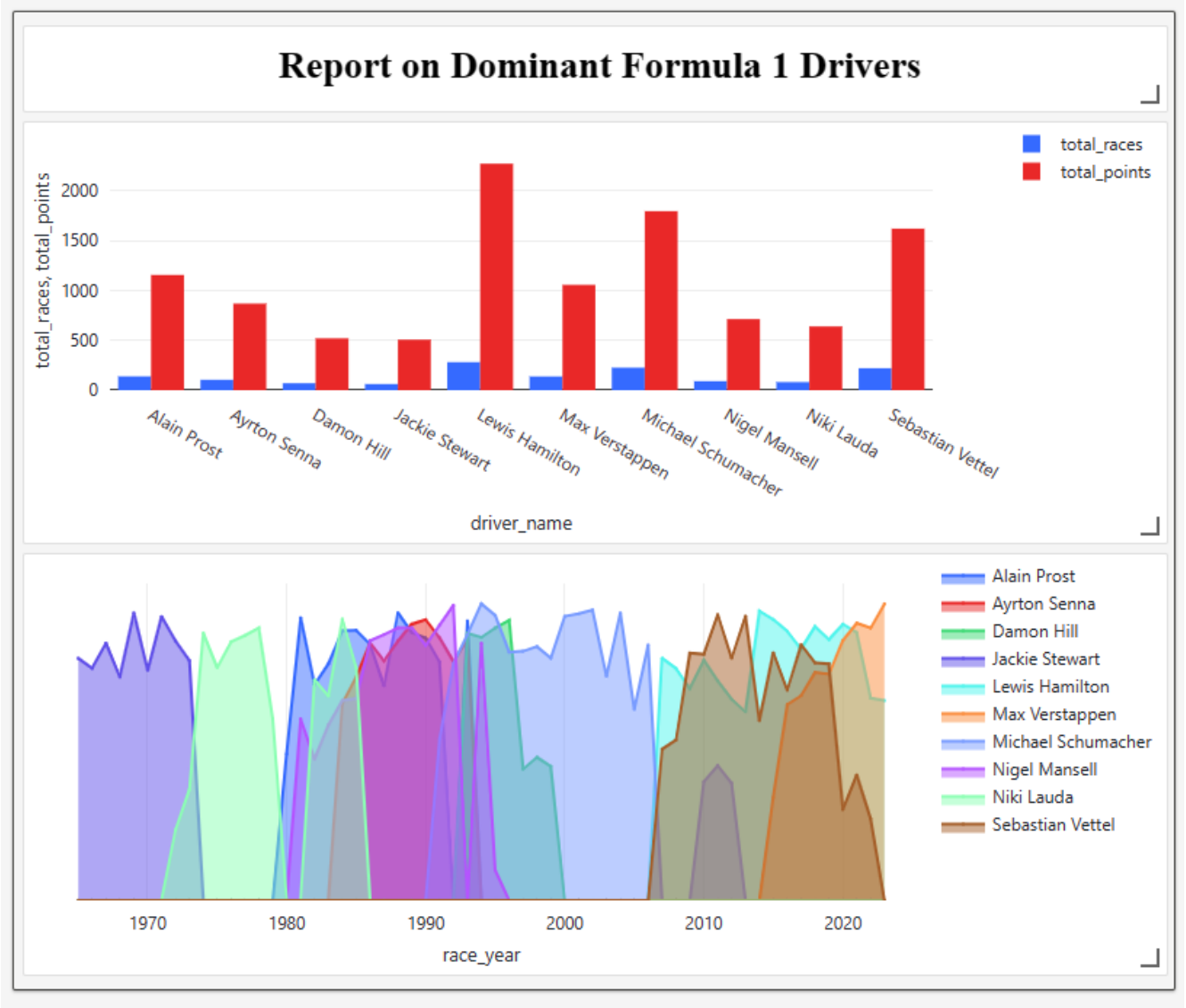
```
SELECT race_year,
       driver_name,
       count(1) AS total_races,
       SUM(calculated_points) AS total_points,
       AVG(calculated_points) AS avg_points
  FROM f1_presentation.calculated_race_results
 WHERE driver_name IN (SELECT driver_name from v_dominant_drivers where driver_rank <= 10)
GROUP BY driver_name, race_year
ORDER BY race_year, avg_points DESC
```

Table | Area Chart | Bar Chart

	race_year	driver_name	total_races	total_points	avg_points
1	1965	Jackie Stewart	7	56	8
2	1966	Jackie Stewart	3	23	7.666666666666667
3	1967	Jackie Stewart	2	17	8.5
4	1968	Jackie Stewart	8	59	7.375
5	1969	Jackie Stewart	8	76	9.5
6	1970	Jackie Stewart	5	38	7.6
7	1971	Jackie Stewart	8	75	9.375

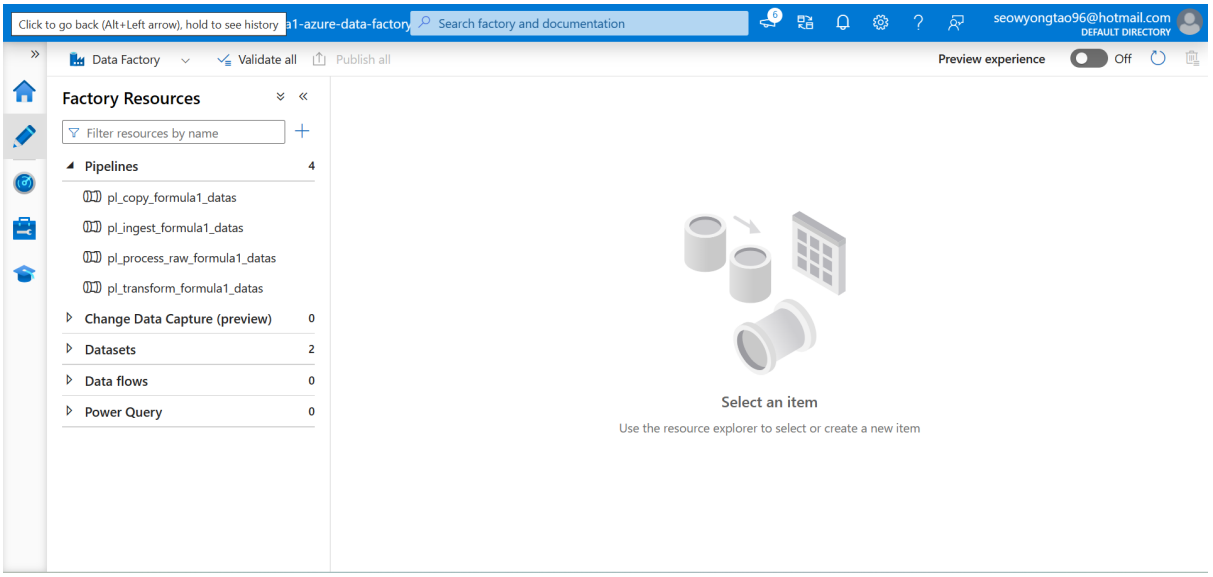
126 rows

Dashboard:


Report on Dominant Formula 1 Drivers
total_races
total_points
total_races, total_points
2000
1500
1000
500
0
Alain Prost
Ayrton Senna
Damon Hill
Jackie Stewart
Lewis Hamilton
Max Verstappen
Michael Schumacher
Nigel Mansell
Niki Lauda
Sebastian Vettel
driver_name
Alain Prost
Ayrton Senna
Damon Hill
Jackie Stewart
Lewis Hamilton
Max Verstappen
Michael Schumacher
Nigel Mansell
Niki Lauda
Sebastian Vettel
1970
1980
1990
2000
2010
2020
race_year

5 Azure Data Factory

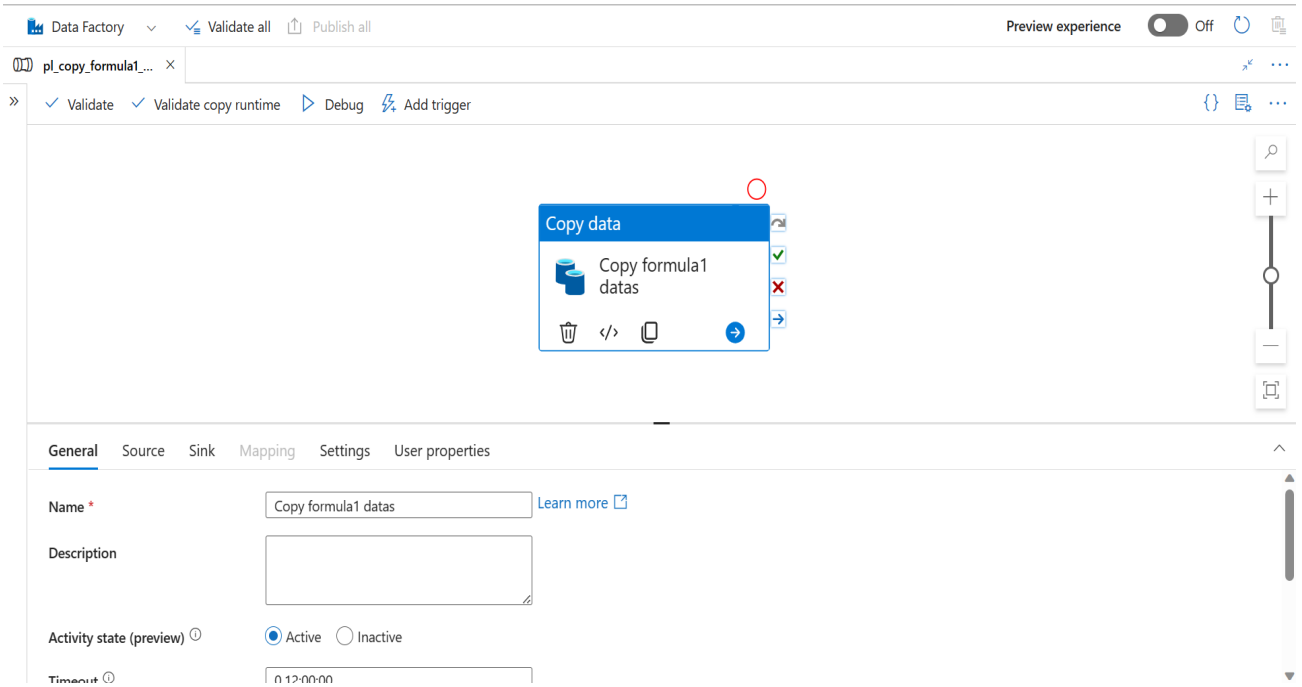
5.1. Pipelines



There are 4 pipelines created as shown in the image above. In the following sections, the details about each of the pipelines will be described and shown.

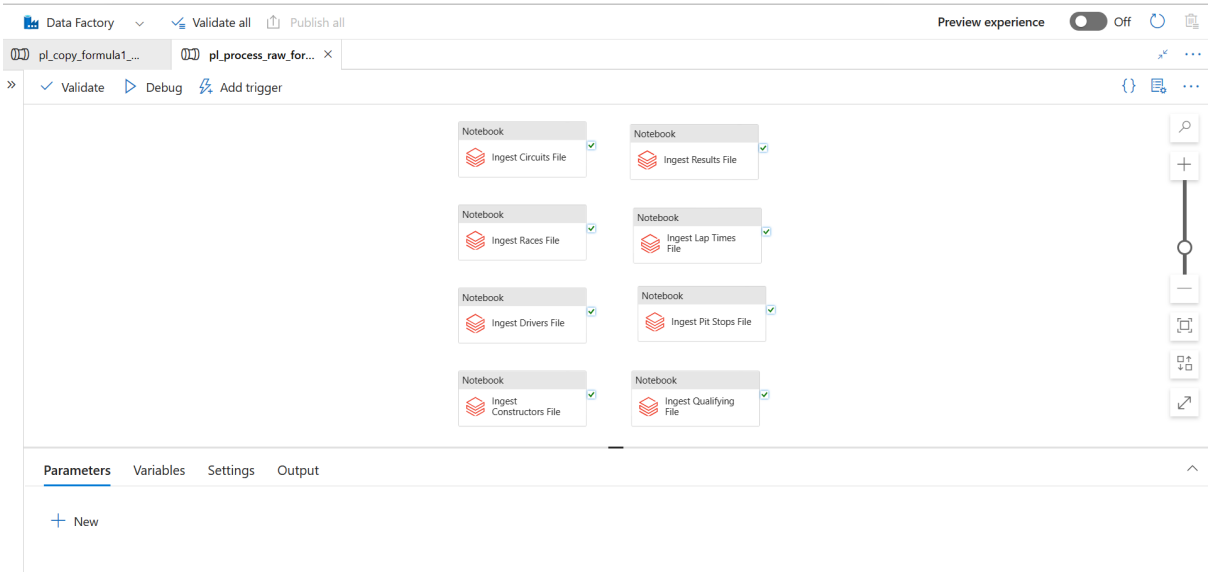
5.1.1. pl_copy_formula_datas

This pipeline is used to copy the data from the link provided by Ergast website: http://ergast.com/downloads/f1db_csv.zip and load it into the ADLS raw layer.



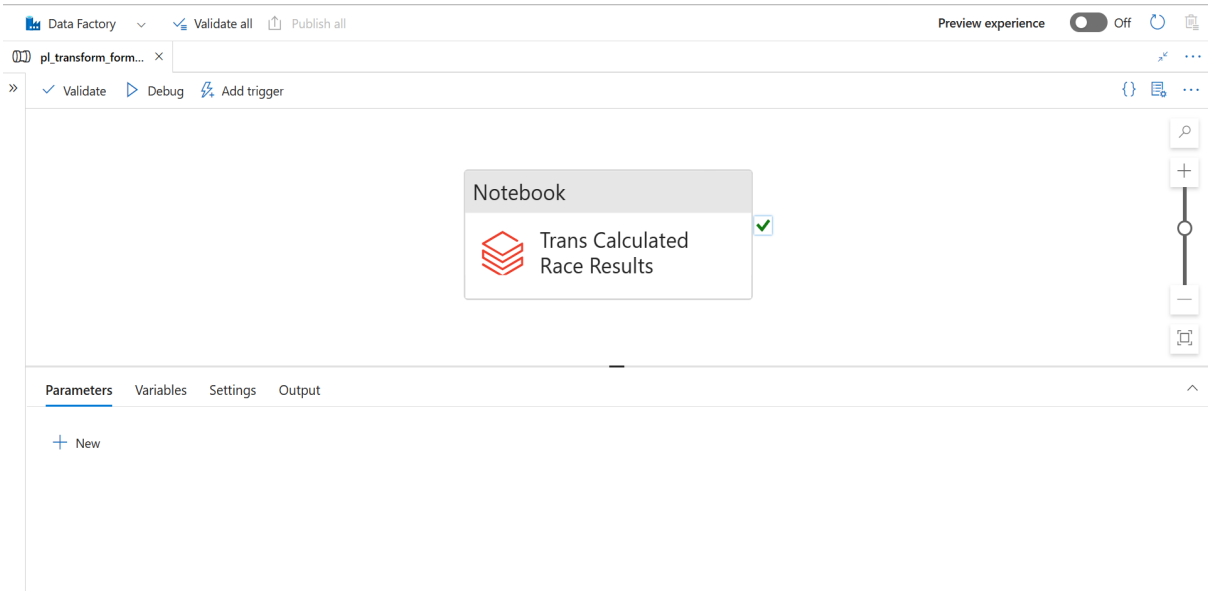
5.1.2. pl_process_raw_formula1_datas

This pipeline is used to execute those databrick notebooks that are responsible for processing different source files in the ADLS raw layer.



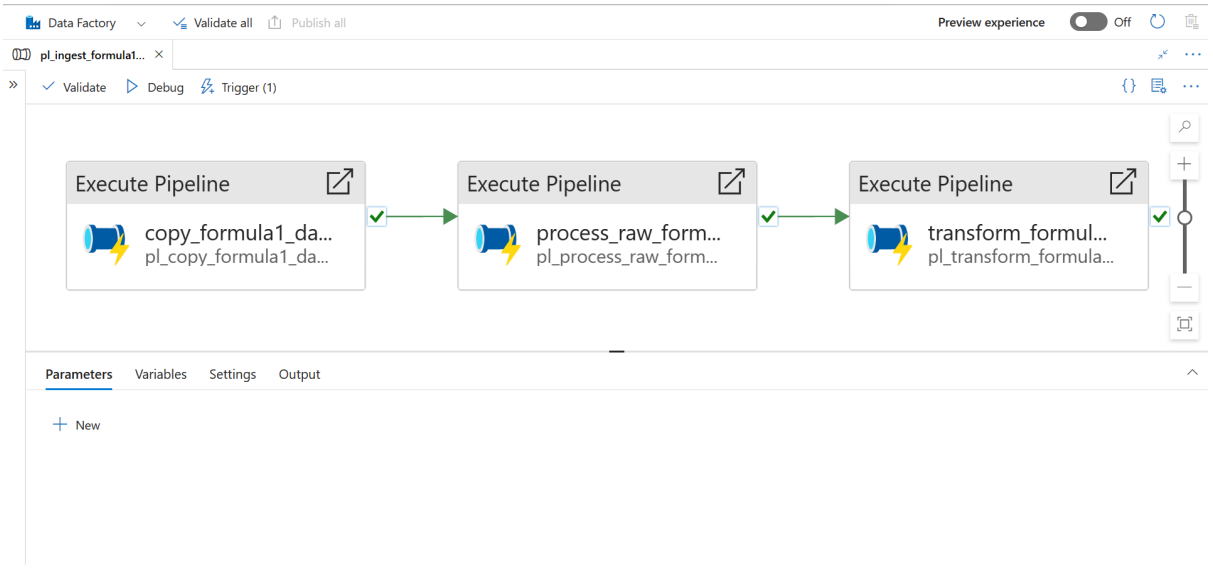
5.1.3. pl_transform_formula1_datas

This pipeline is used to execute those notebooks that are responsible for transforming and combining data from various sources within the ADLS ingested layer.

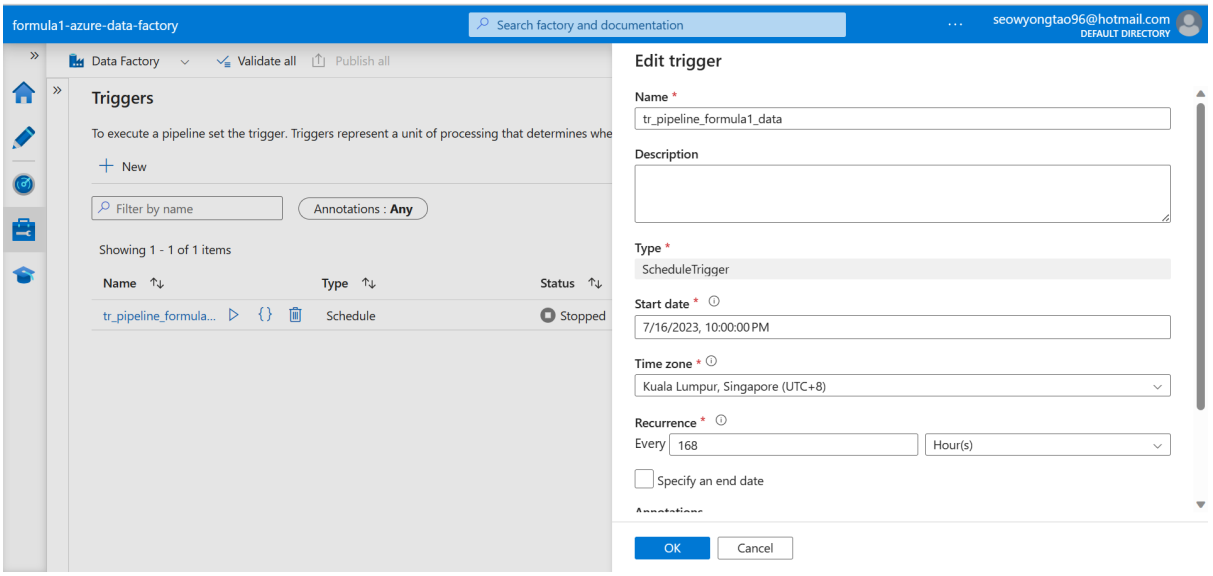


5.1.4. pl_ingest_fomula1_datas

This pipeline is created to execute all the pipelines created in a sequence manner.



5.2 Trigger



A trigger named “tr_pipeline_formula1_data” is created and applied on pipeline “pl_ingest_fomula1_datas”, so that it will be executed automatically every week.