Tokyo 2021 Olympics - Data Analytics

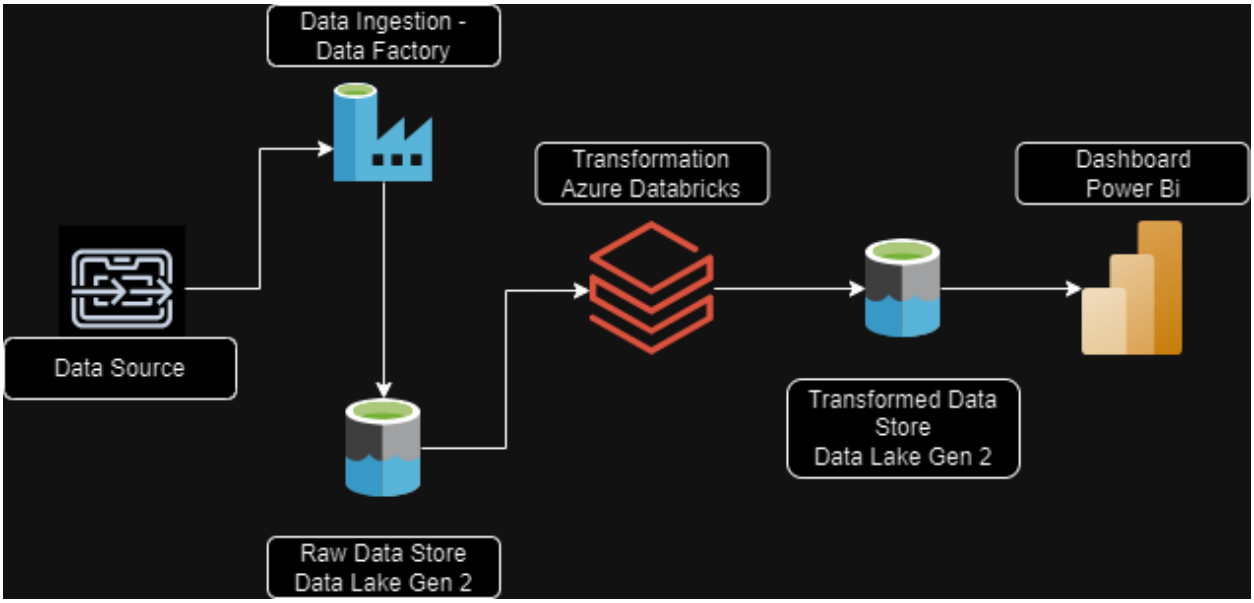
Data Engineering project using Azure

This document outlines a data engineering project that utilizes Microsoft Azure for comprehensive data analytics. The project is divided into several key steps:

1. **Data Download from Kaggle**: The primary data source for this project is Kaggle, a popular online platform that hosts various datasets. The initial step involves downloading the necessary data from this platform.
2. **Ingestion to Azure Data Factory**: The downloaded data is then ingested into Azure Data Factory, a cloud-based data integration service that allows the creation of data-driven workflows for orchestrating and automating data movement and data transformation.
3. **Storage of Raw Data in Data Lake Gen 2**: Once ingested, the raw data is stored in Azure Data Lake Gen 2. This service combines the scalability and cost benefits of object storage with the reliability and performance of the Big Data file system capabilities.
4. **ETL Processing in Azure Databricks**: The raw data undergoes Extract, Transform, Load (ETL) operations in Azure Databricks, utilizing PySpark for the processing.
5. **Storage of Transformed Data in Data Lake Gen 2**: The transformed data is then stored back in Azure Data Lake Gen 2 for further analysis.
6. **Visualization with Power BI**: Finally, the insights derived from the analysis are visualized using Power BI, a business analytics tool that delivers interactive visualizations with self-service business intelligence capabilities.

Each step in this roadmap not only showcases the capabilities of Azure's data engineering services but also mirrors the typical workflow of a data engineering project.

Architecture:

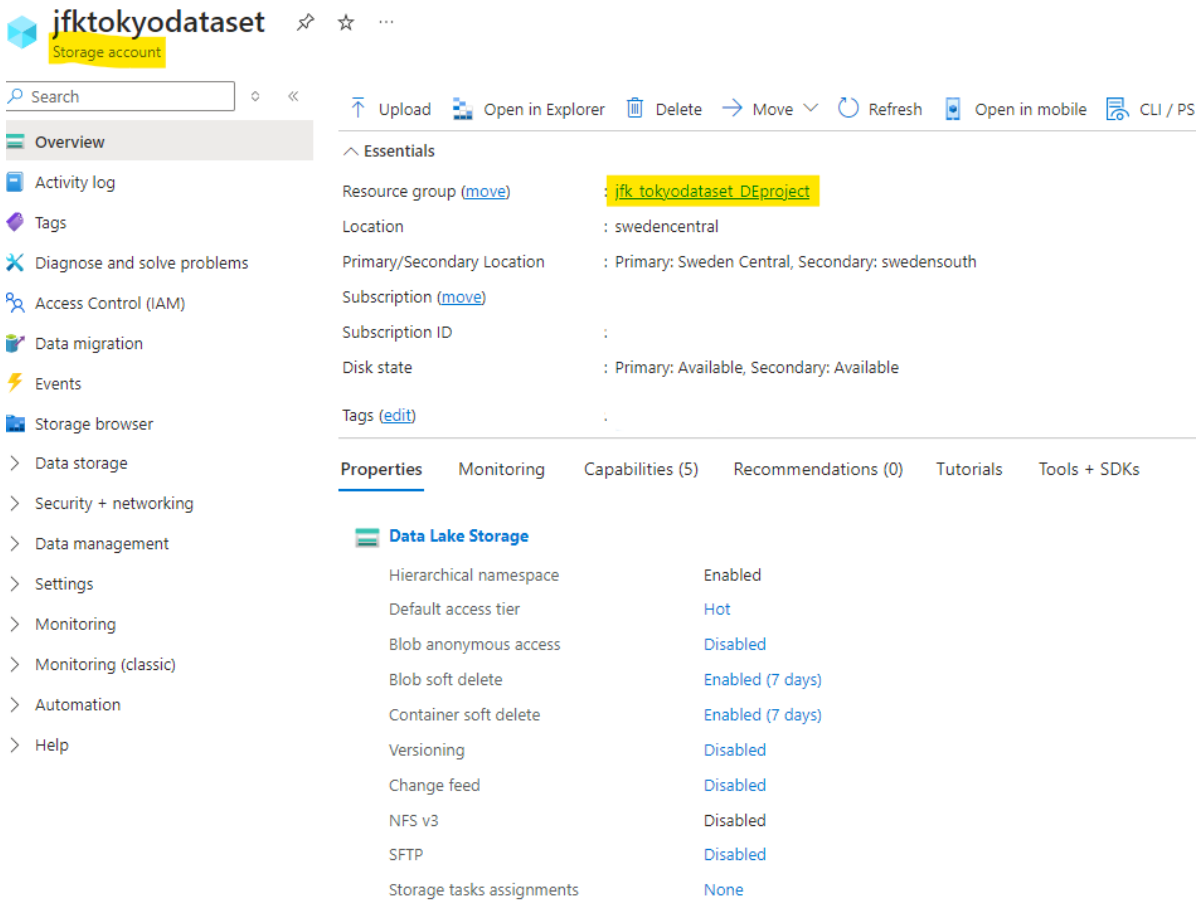


Data Source:

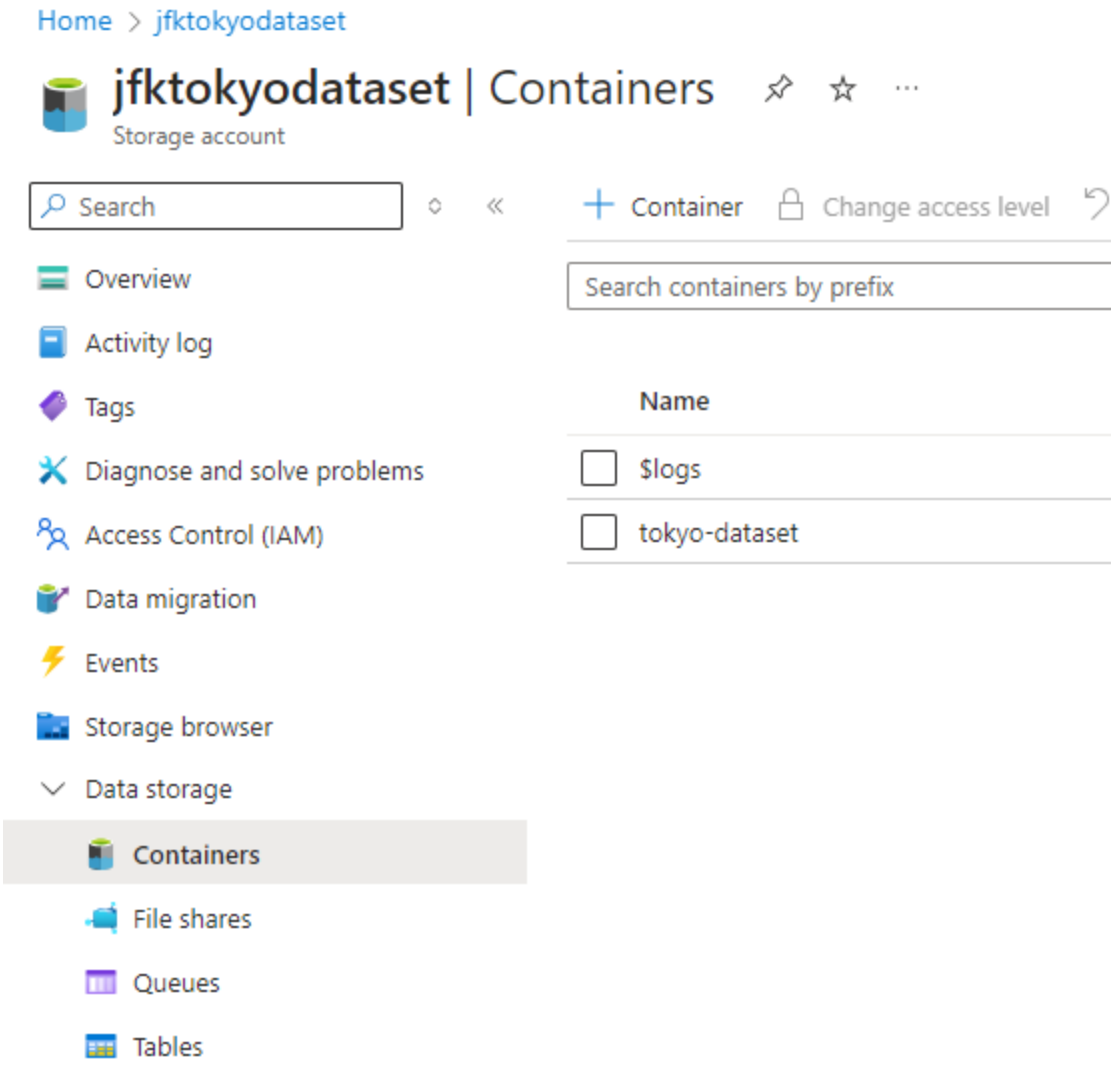
For this project, we're diving into the data pool at Kaggle. The 2021 Olympics in Tokyo dataset. You can grab it too, right here: https://www.kaggle.com/datasets/arjunprasadsarkhel/2021-olympics-in-tokyo.

Setting up the environment:

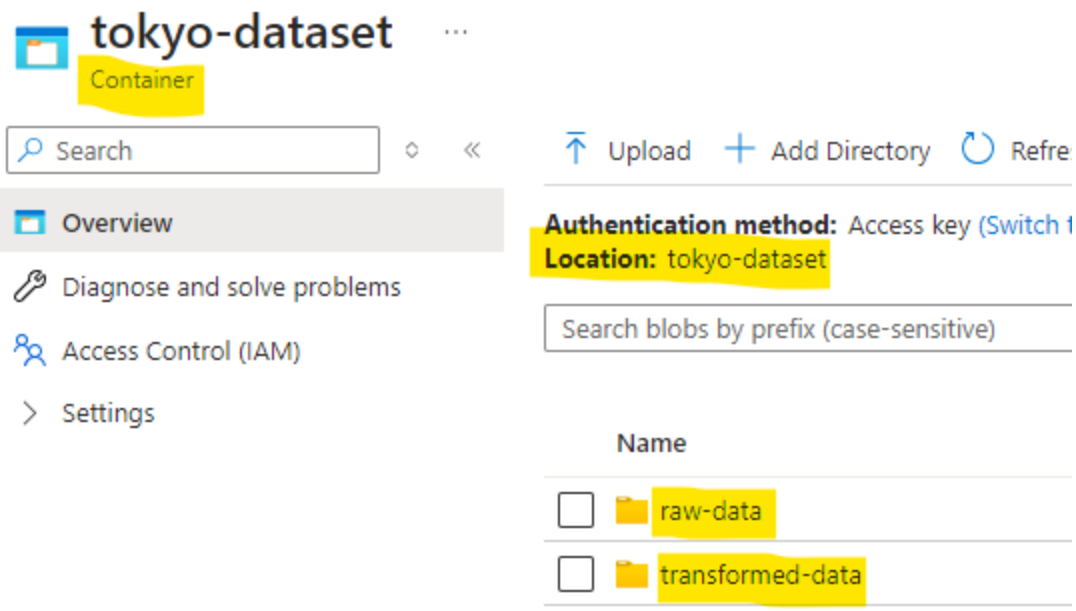
1. First we create a Storage account and a new resource group:



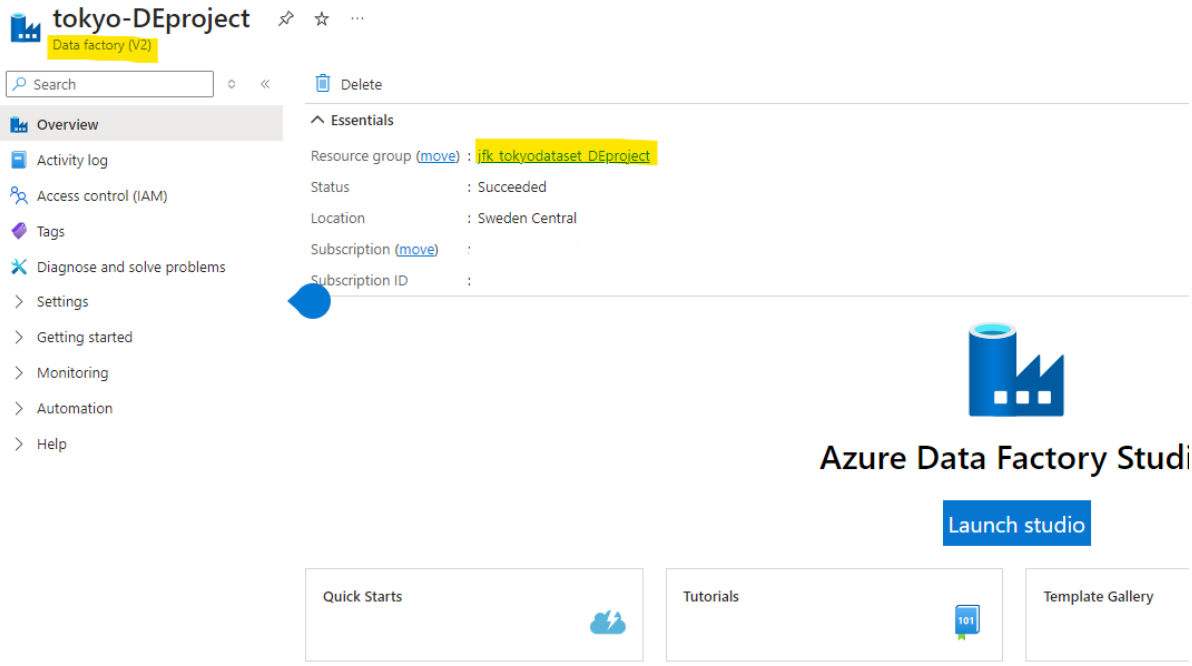
2. Next we create a container for the project under ‘Data Storage - Containers’



3. Next we want to create two directories inside the created container ‘tokyo-dataset’

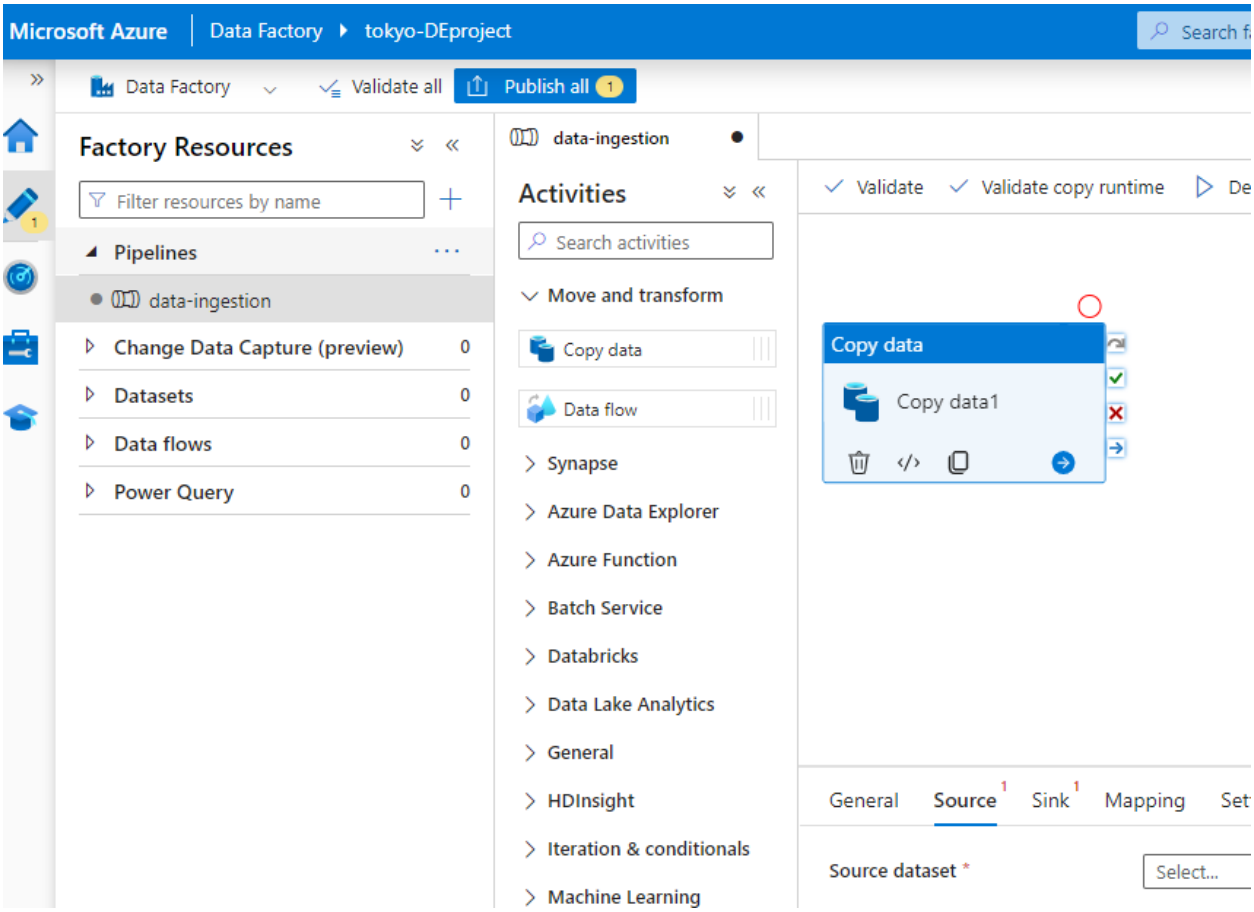


4. Then we create Azure Data Factory using our same resource group 'jfk_tokyodataset_DEproject'

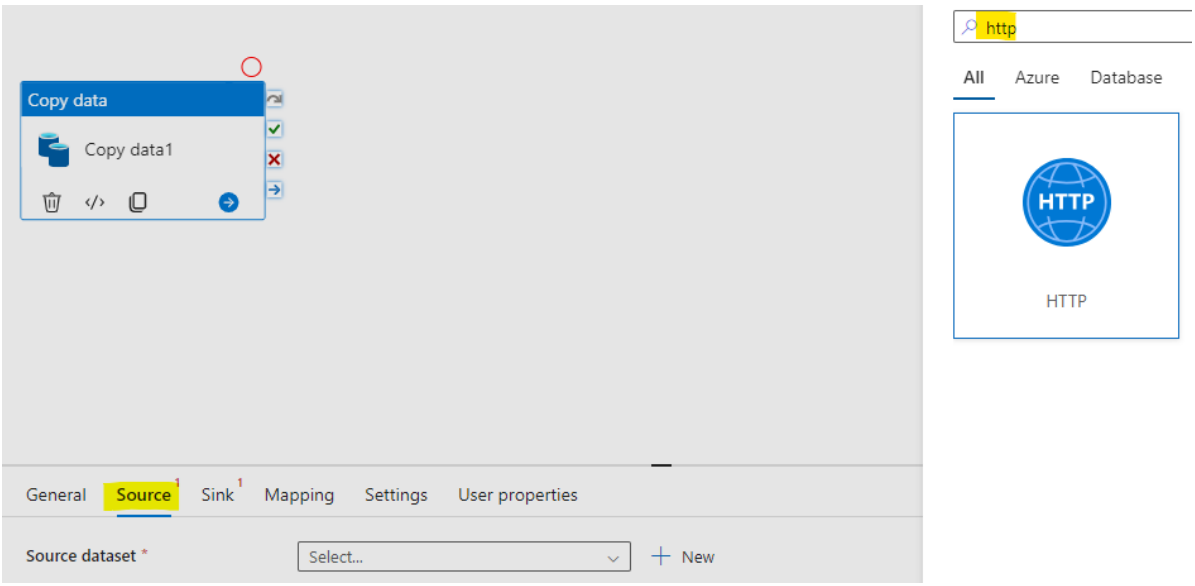


Data Factory

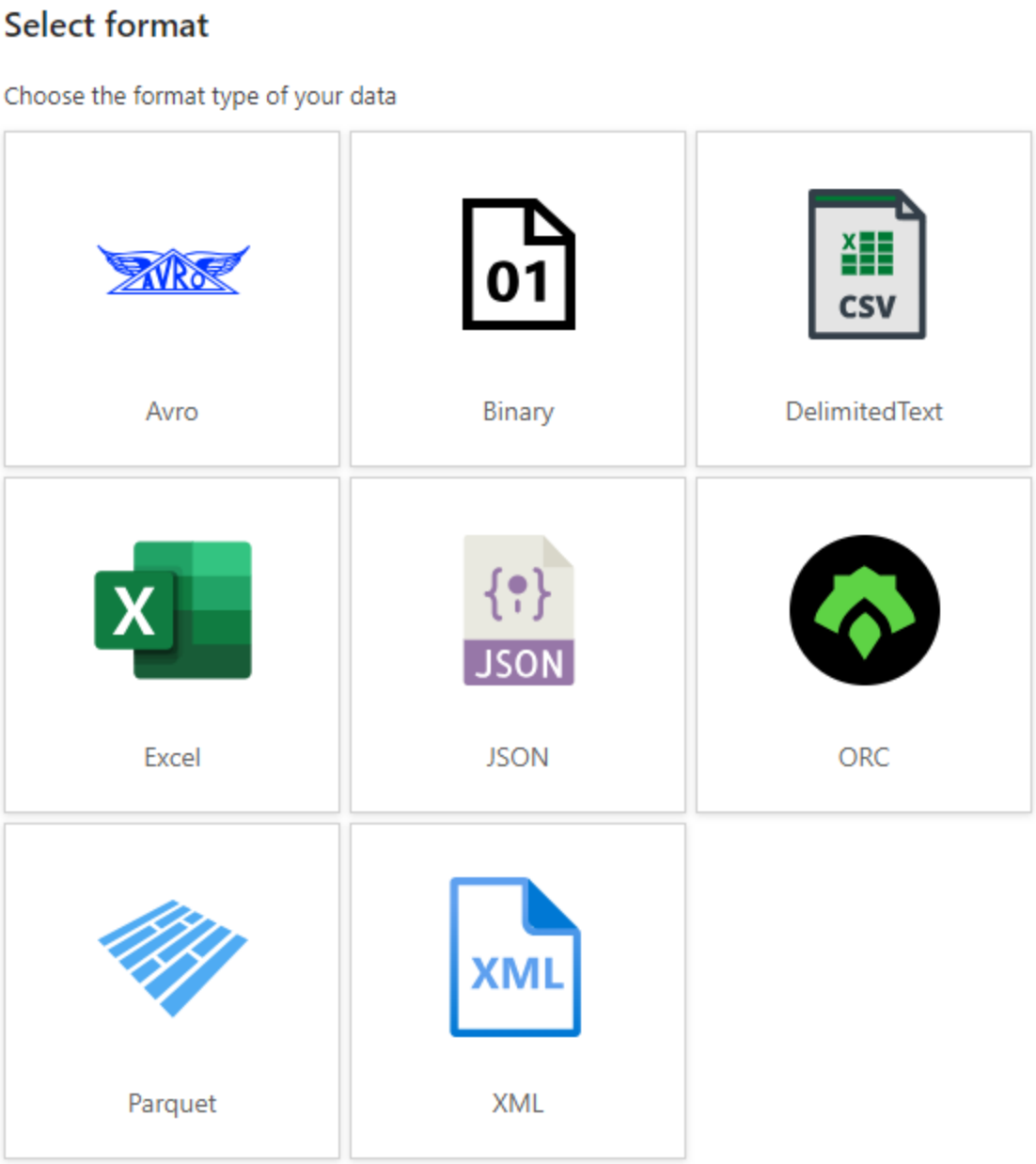
Next we are going to create a pipeline for the ‘data ingestion’



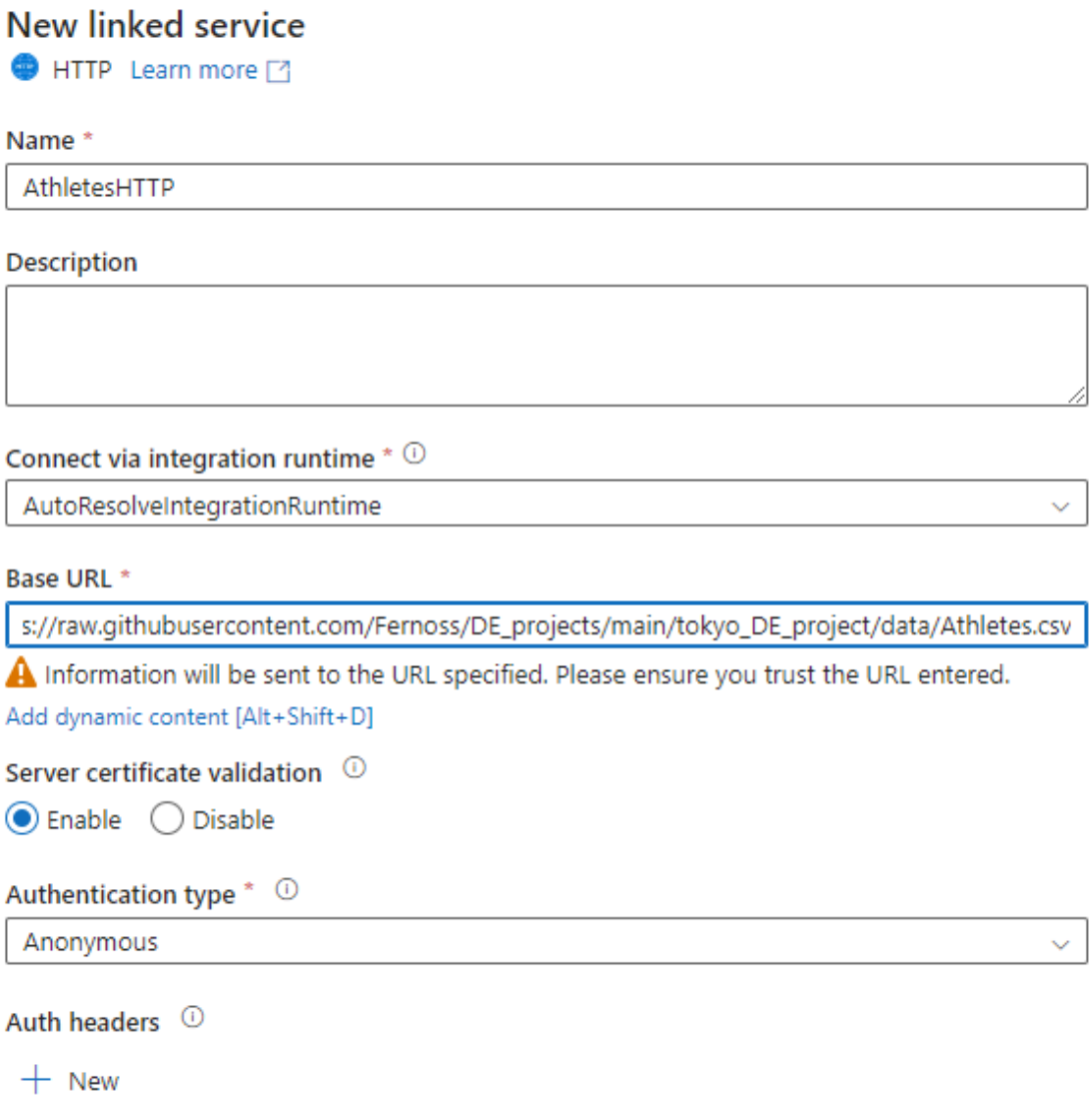
1. We utilize HTTP-method to get the raw data from my GitHub-repository - where I have stored as *.csv files from Kaggle. that was provided earlier.



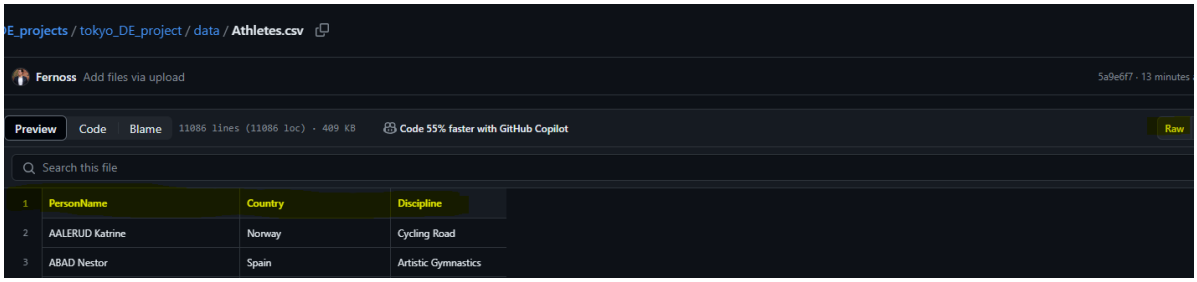
and as the files are in *.csv format, we choose ‘Delimeted Text’

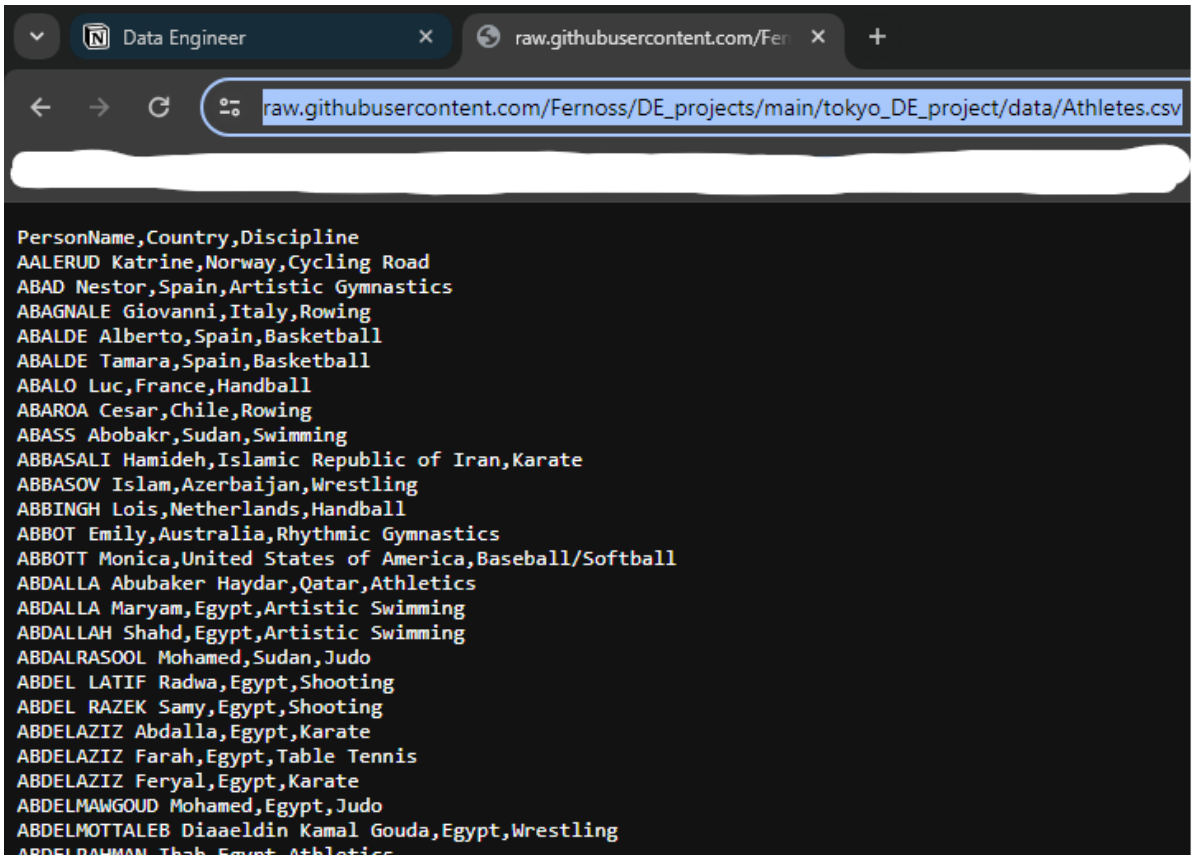


Afterwards we need to create a Linked service - so basically the URL for the ‘RAW data’



Here we define the URL for the raw version of the *.csv in GitHub repository. Authentication type is set to ‘Anonymous’ because the repository is set to ‘Public’.





Important to note: Make sure to tick ‘First row as a header’

We can check with ‘Preview data’ that the link works and we get the wanted data:

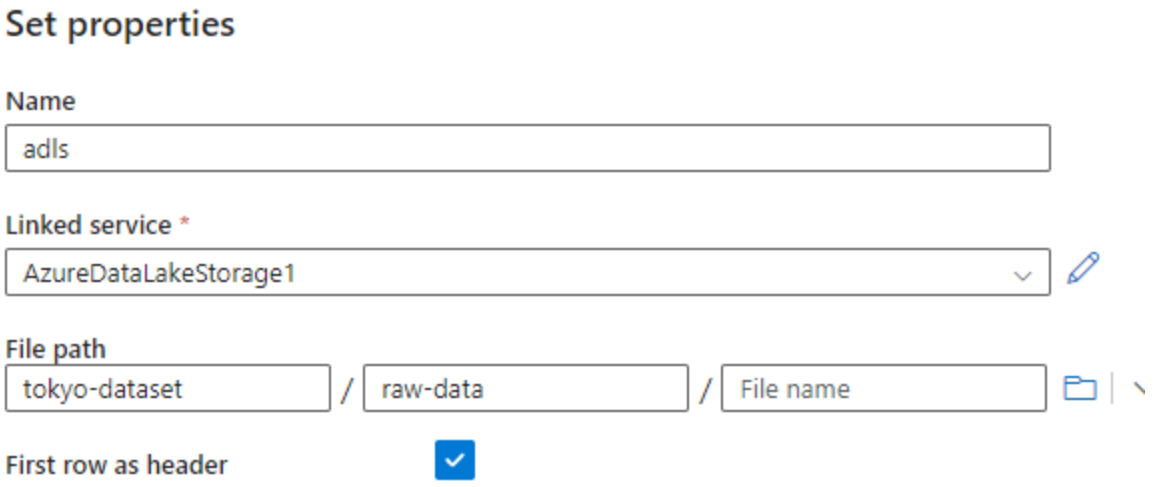
Preview data

Linked service: AthletesHTTP

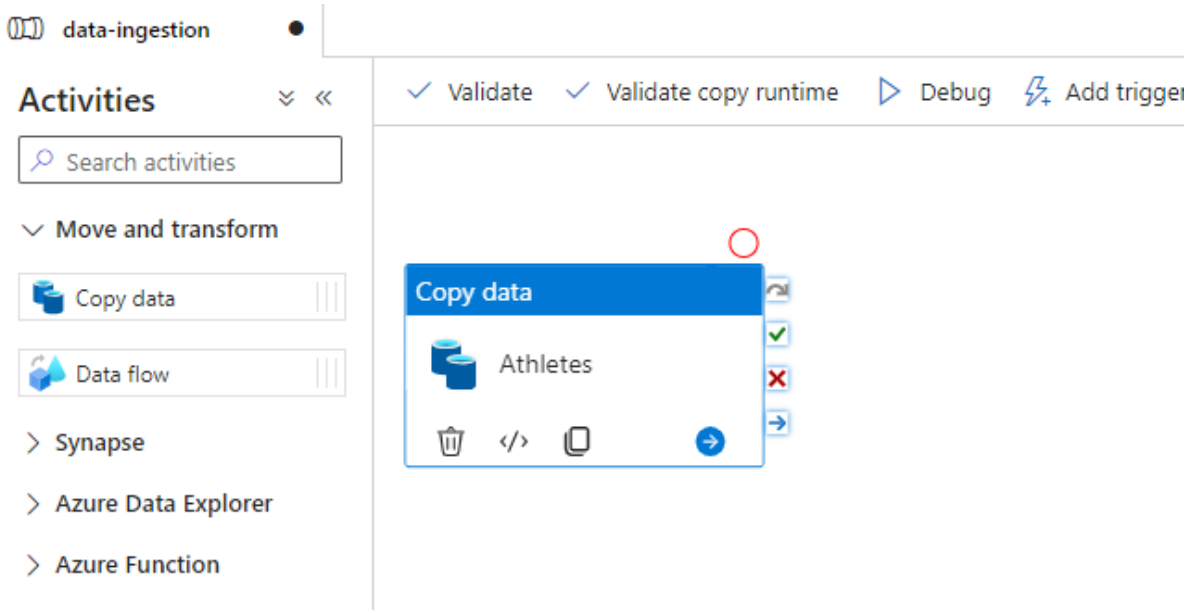
Object:

	PersonName	Country	Discipline
1	AALERUD Katrine	Norway	Cycling Road
2	ABAD Nestor	Spain	Artistic Gymnastics
3	ABAGNALE Giovanni	Italy	Rowing
4	ABALDE Alberto	Spain	Basketball

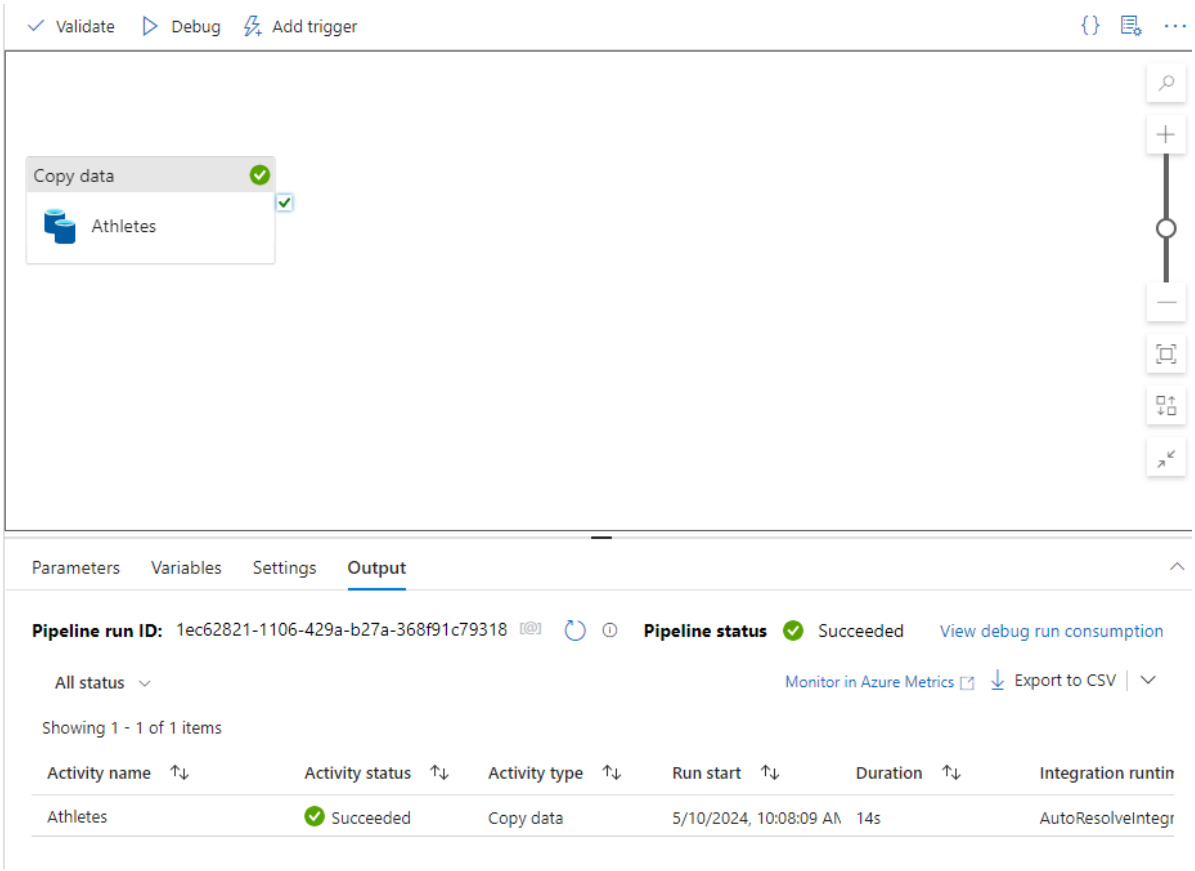
Once the linked services for the source is done, next we create the ‘Sink’ which is for the Data Lake Storage Gen2 - Raw data directory.



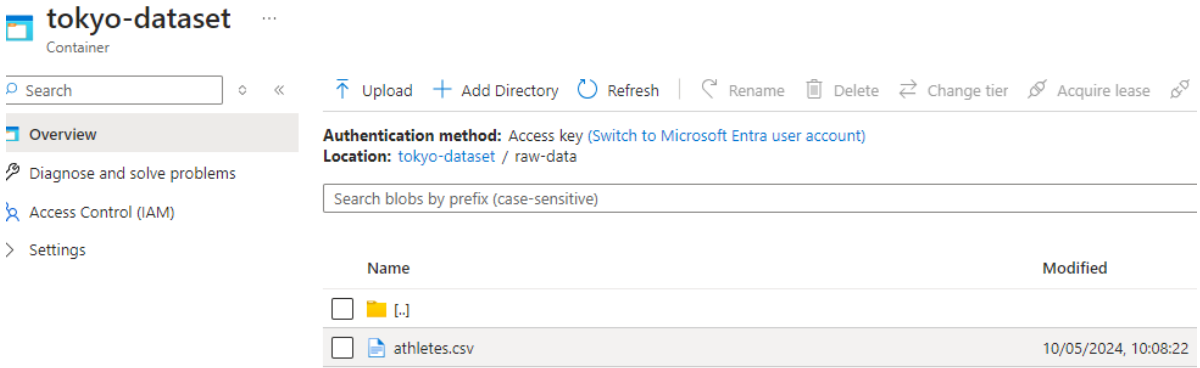
Once these steps are done, we can click ‘Validate’ on the top to see if any errors occur.



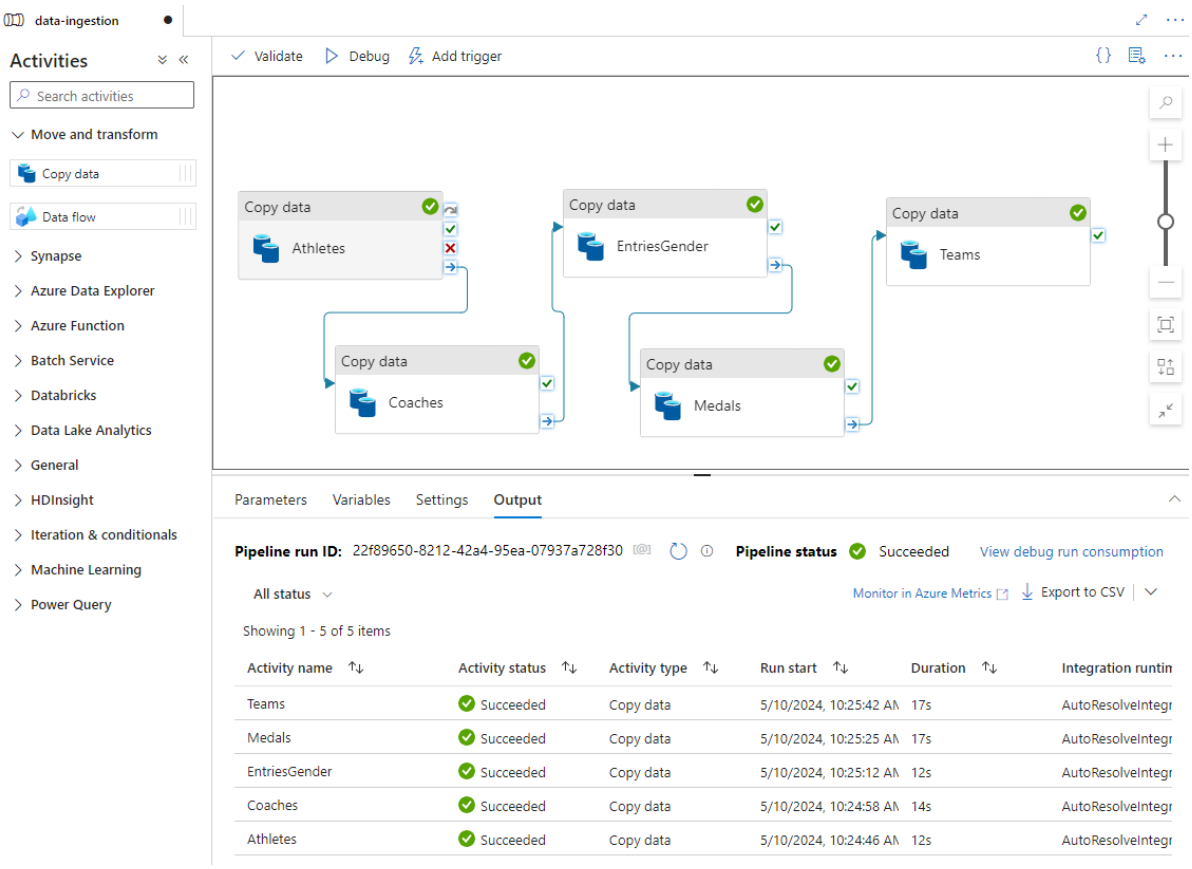
Afterwards we run it manually by pressing ‘Debug’ to see if the pipeline works

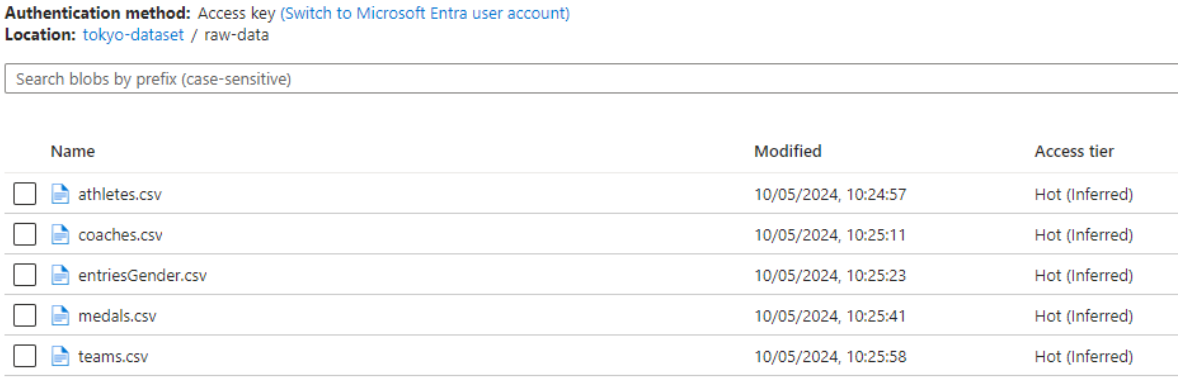


Go to the container and see if the ‘atheletes.csv’ file has been created in the directory



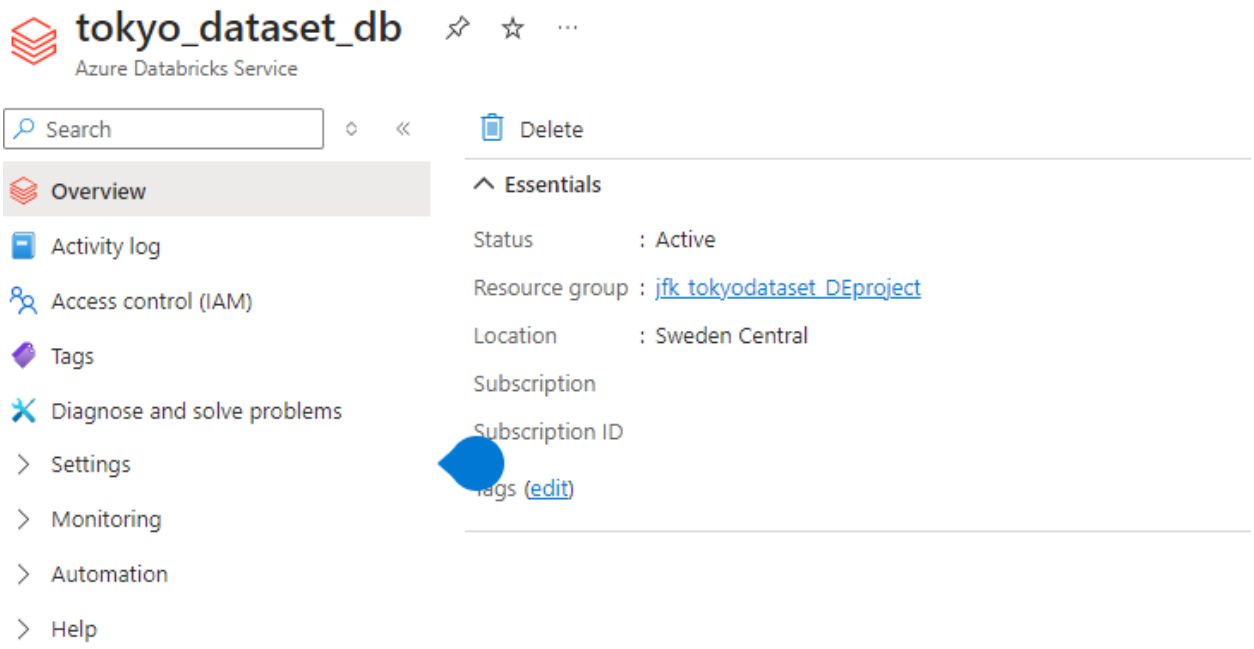
2. Once happy with the step 1, next is to repeat step 1 for the other *.csv files to get the rest of the tables. Afterwards verify that the pipeline is operational and working.



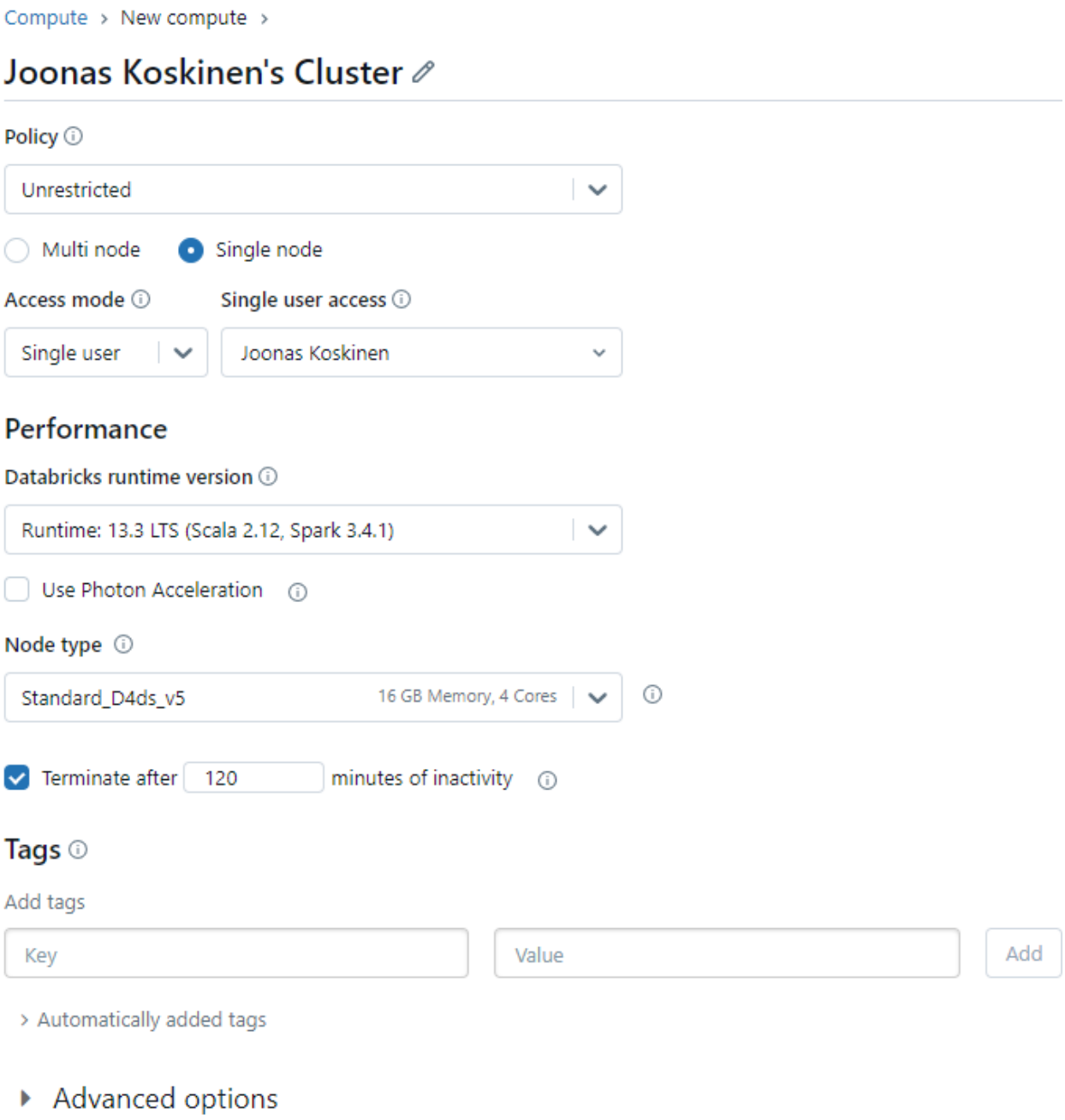


Azure Databricks

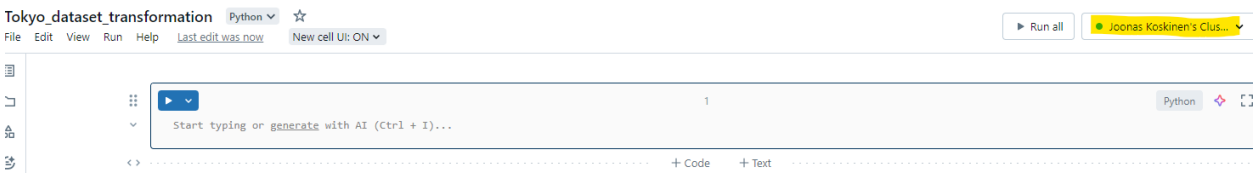
Next we setup a new Databricks service for our resource group



Then we lunch Databricks and create a compute in our cluster, so that we can run Spark jobs for the data transformation.



Once compute has been created, we now open a new notebook to start working on the code. Also important to make sure that your recently created compute is selected



Because I don’t have in this Azure sandbox admin rights - I have used SAS-token as a connector for the mount, in order to access Azure Data Lake Gen2. And tested it to show me first 10 rows:

```
dbutils.fs.mount(
  source = "wasbs://tokyo-dataset@jfktokyodataset.blob.core.windows.net",
  mount_point = "/mnt/tokyoDb",
  extra_configs = {"fs.azure.sas.tokyo-dataset.jfktokyodataset.blob.core.windows.net": "sv=2022-11-02&s
  sig=dhNCW2RO5wOEFD5zCf6uWARevuIh3rshR3B40w8T%2FkE%3D"}
)
```

12:48 PM (1s)

```
df = spark.read.csv("/mnt/tokyoDb/raw-data/athletes.csv", header=True)
```

▸ (1) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [PersonName: string, Country: string ... 1 more field]

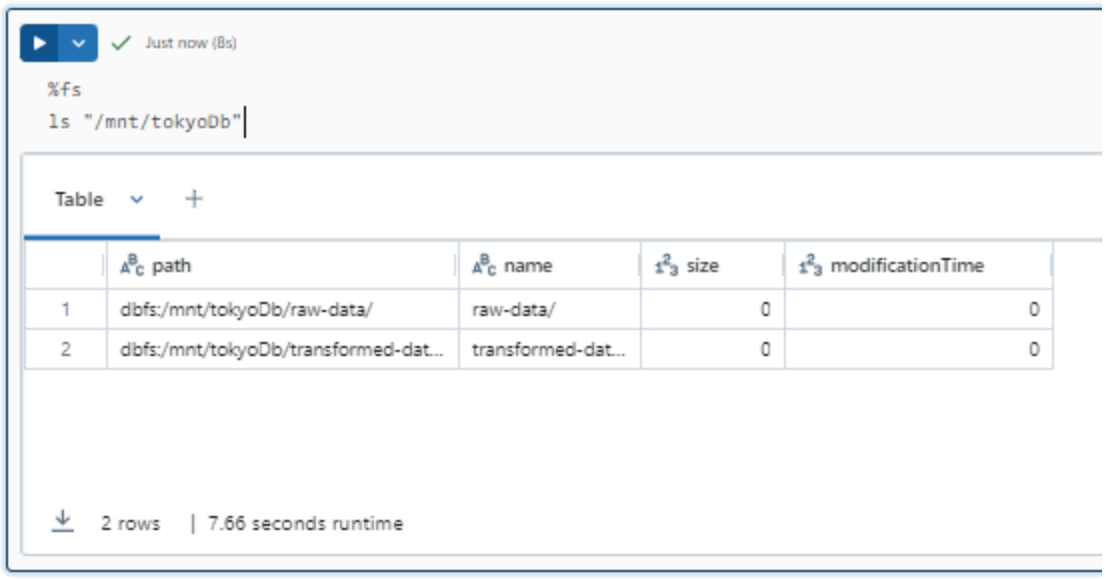
12:48 PM (1s)

```
df.show(10)
```

▸ (1) Spark Jobs

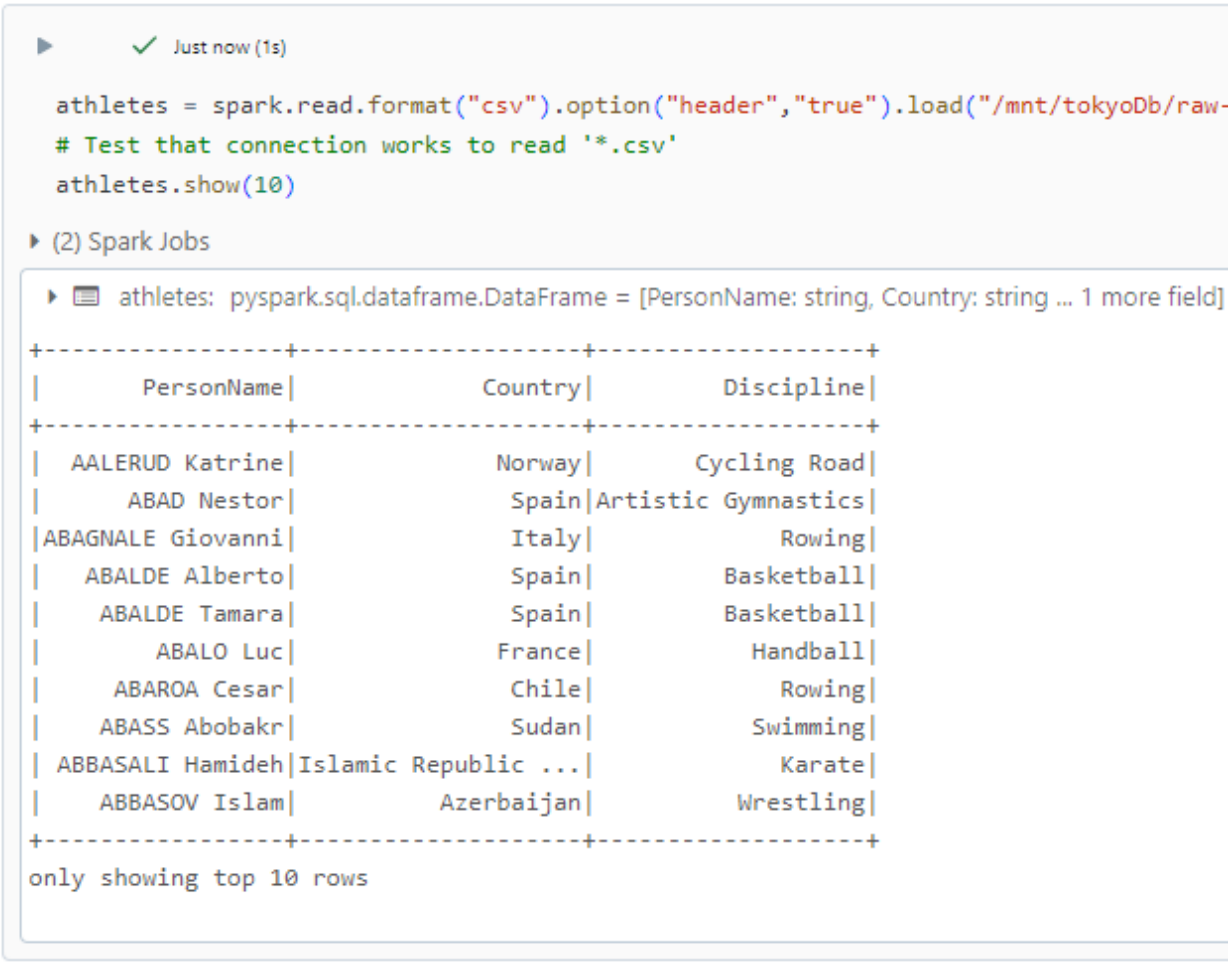
```
+-----------------+--------------------+-------------------+
|       PersonName|             Country|         Discipline|
+-----------------+--------------------+-------------------+
|  AALERUD Katrine|              Norway|       Cycling Road|
|      ABAD Nestor|               Spain|Artistic Gymnastics|
|ABAGNALE Giovanni|               Italy|             Rowing|
|   ABALDE Alberto|               Spain|         Basketball|
|    ABALDE Tamara|               Spain|         Basketball|
|        ABALO Luc|              France|           Handball|
|     ABAROA Cesar|               Chile|             Rowing|
|    ABASS Abobakr|               Sudan|           Swimming|
| ABBASALI Hamideh|Islamic Republic ...|             Karate|
|    ABBASOV Islam|          Azerbaijan|          Wrestling|
+-----------------+--------------------+-------------------+
only showing top 10 rows
```

Also we can check with ‘%fs ls ”/mnt/tokyoDb” to see the directory content

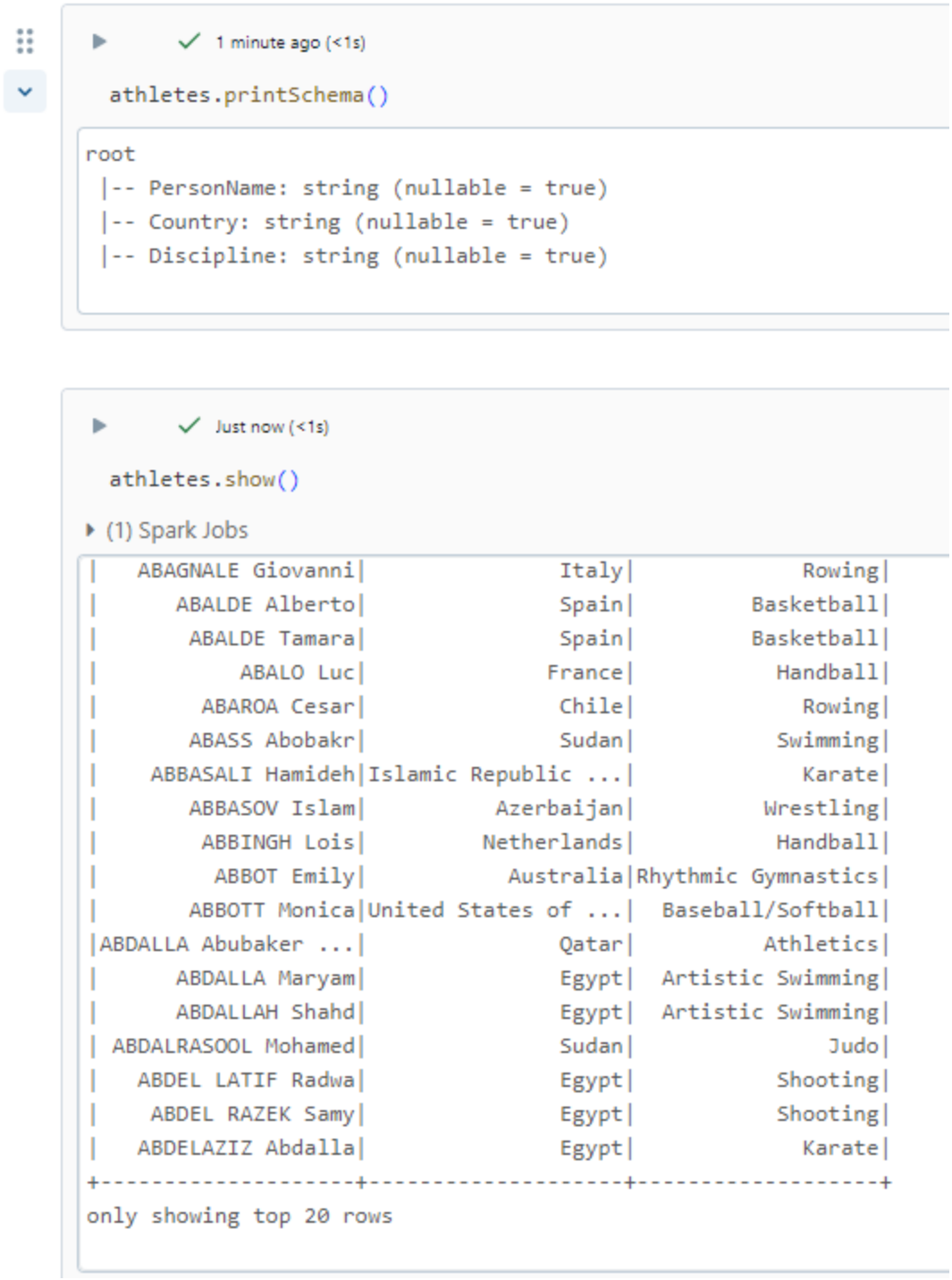


Then we start creating the paths for the '*.csv' files to be used in Spark. We use '.option("header","true")' to have the first row as headers.

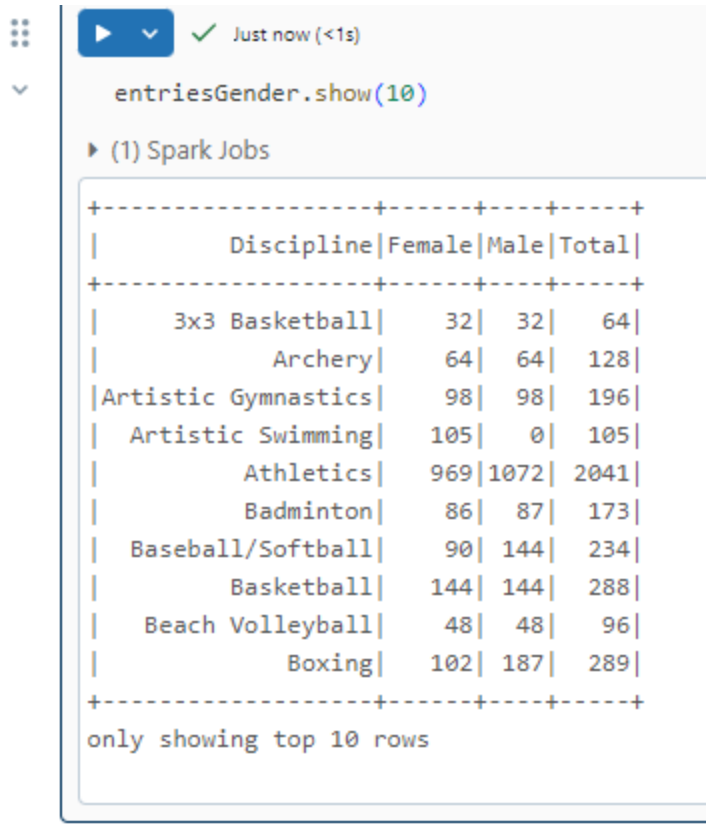
Creating variables for the *.csv read

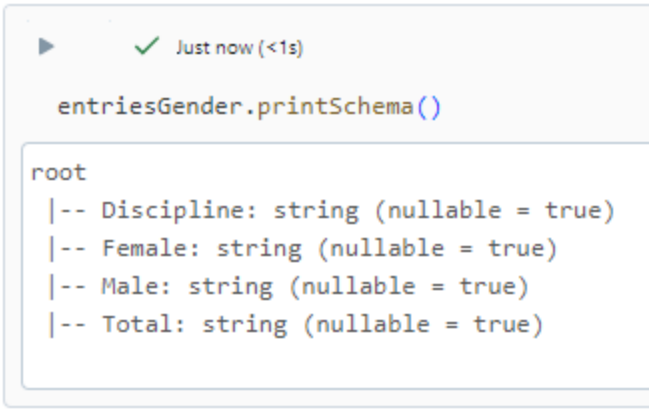


Next we start on cleaning the data and to check tables schemas to see any irregularities, incorrect data types etc. For example we can use ‘tableName.printSchema()’ to see information about the table.



As we can see in the previous, everything seems okay in terms of data types and the ones represented in it's table. But upon looking at 'entriesGender' -table we can see, that columns 'Female', 'Male' and 'Total' are in strings, not integers.





So we need to transform and replace ‘String’-data type as ‘Integers’. This was to showcase how to transform it manually or when seeing a wrong data type. On the next sessions will be shown an ‘easier’ way.

Transforming 'entriesGender' data types into integer, replacing string.

Just now (<1s)

```
entriesGender = entriesGender.withColumn("Female",col("Female").cast(IntegerType()))\
    .withColumn("Male",col("Male").cast(IntegerType()))\
    .withColumn("Total",col("Total").cast(IntegerType()))
```

entriesGender: pyspark.sql.dataframe.DataFrame = [Discipline: string, Female: integer ... 2 more fields]

Just now (<1s)

```
# Testing to see if transformation was successful
entriesGender.printSchema()
```

```
root
 |-- Discipline: string (nullable = true)
 |-- Female: integer (nullable = true)
 |-- Male: integer (nullable = true)
 |-- Total: integer (nullable = true)
```

So we can add here when creating the variable for Spark to read file *.csv another 'option tag'.

```
athletes = spark.read.format("csv").option("header","true").load("/mnt/tokyoDb/raw-data/athletes.csv")
coaches = spark.read.format("csv").option("header","true").load("/mnt/tokyoDb/raw-data/coaches.csv")
entriesGender = spark.read.format("csv").option("header","true").load("/mnt/tokyoDb/raw-data/entriesGender.csv")
medals = spark.read.format("csv").option("header","true").load("/mnt/tokyoDb/raw-data/medals.csv")
teams = spark.read.format("csv").option("header","true").load("/mnt/tokyoDb/raw-data/teams.csv")
# Test that connection works to read '*.csv'
```

01:57 PM (2s) 9

```
athletes = spark.read.format("csv").option("header","true").option("inferSchema","true").load("/mnt/tokyoDb/raw-data/athletes.csv")
coaches = spark.read.format("csv").option("header","true").option("inferSchema","true").load("/mnt/tokyoDb/raw-data/coaches.csv")
entriesGender = spark.read.format("csv").option("header","true").option("inferSchema","true").load("/mnt/tokyoDb/raw-data/entriesGender.csv")
medals = spark.read.format("csv").option("header","true").option("inferSchema","true").load("/mnt/tokyoDb/raw-data/medals.csv")
teams = spark.read.format("csv").option("header","true").option("inferSchema","true").load("/mnt/tokyoDb/raw-data/teams.csv")
# Test that connection works to read '*.csv'
```

So we have added '.option("inferSchema","true")' that lets Spark to check the Schema and interpret the table, assigning data types automatically.

Checking to see if .option("inferSchema","true") worked with *.medals.csv file

Just now (<1s)

```
medals.printSchema()
```

```
root
 |-- Rank: integer (nullable = true)
 |-- Team_Country: string (nullable = true)
 |-- Gold: integer (nullable = true)
 |-- Silver: integer (nullable = true)
 |-- Bronze: integer (nullable = true)
 |-- Total: integer (nullable = true)
 |-- Rank by Total: integer (nullable = true)
```

Now to work with the data itself, for example to query with Spark to look for top countries with gold medals descending order.

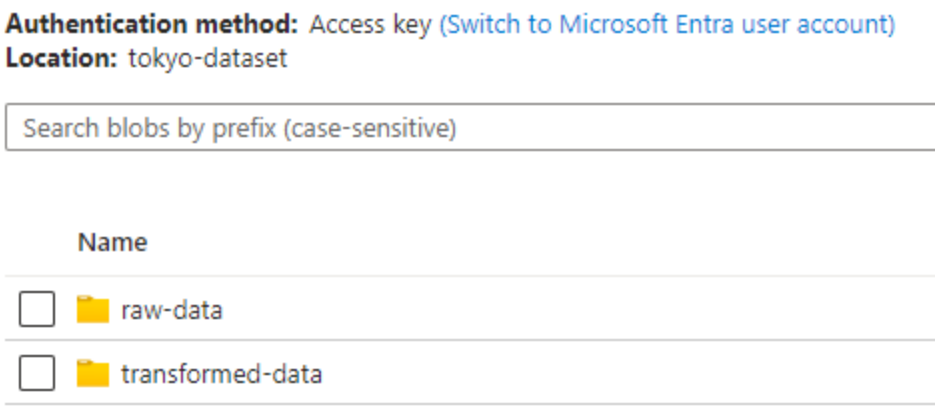
Just now (1s) 21

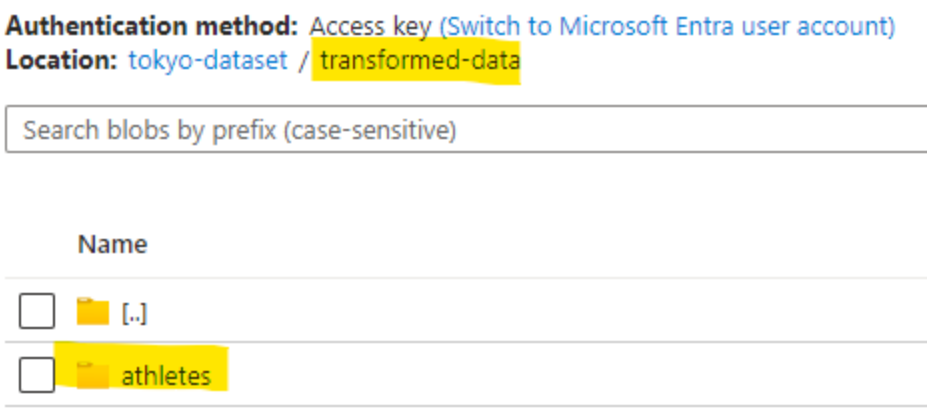
```
# Find the top countries with the highest number of gold medals
top_gold_medal_countries = medals.orderBy("Gold", ascending=False).select("Team_Country","Gold").show()
```

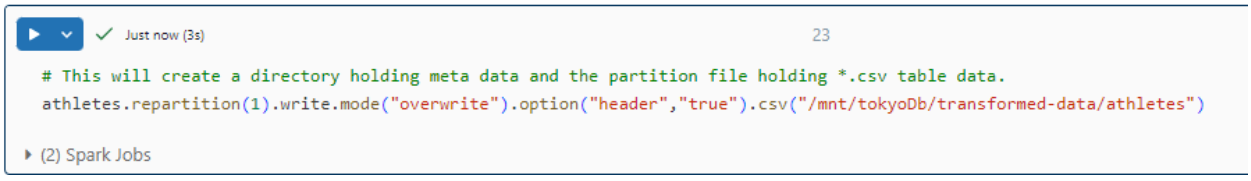
(1) Spark Jobs

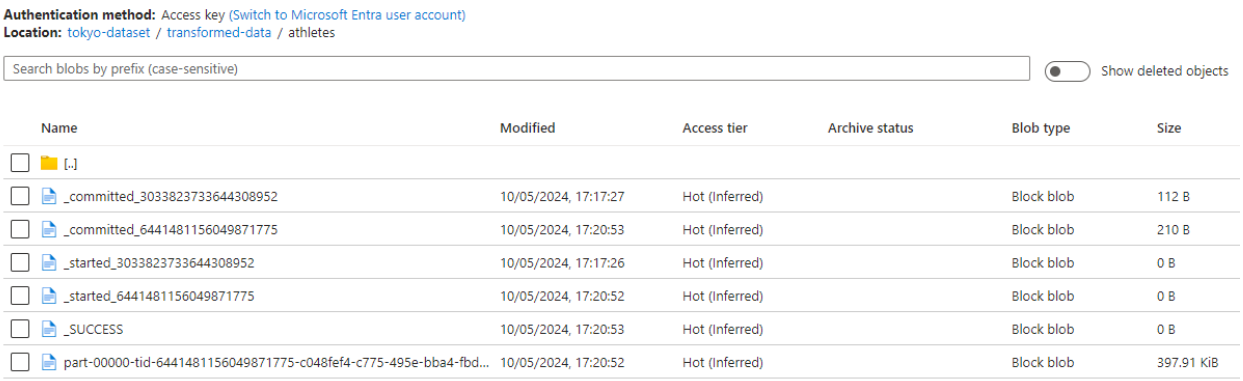
```
+--------------------+----+
|        Team_Country|Gold|
+--------------------+----+
|United States of ...|  39|
|People's Republic...|  38|
|               Japan|  27|
|       Great Britain|  22|
|                ROC|  20|
|          Australia|  17|
|         Netherlands|  10|
|              France|  10|
|             Germany|  10|
|               Italy|  10|
|              Canada|   7|
|              Brazil|   7|
|         New Zealand|   7|
|                Cuba|   7|
|             Hungary|   6|
|   Republic of Korea|   6|
|              Poland|   4|
|      Czech Republic|   4|
```

Next we write the transformed/cleaned files and store them back into Azure Data Lake Gen2 to the same container but ‘transformed-data’-directory. This creates in Spark a partition directory holding meta data and the *.csv file holding the table data. Using ‘Pandas’ you can create a single file of *.csv.



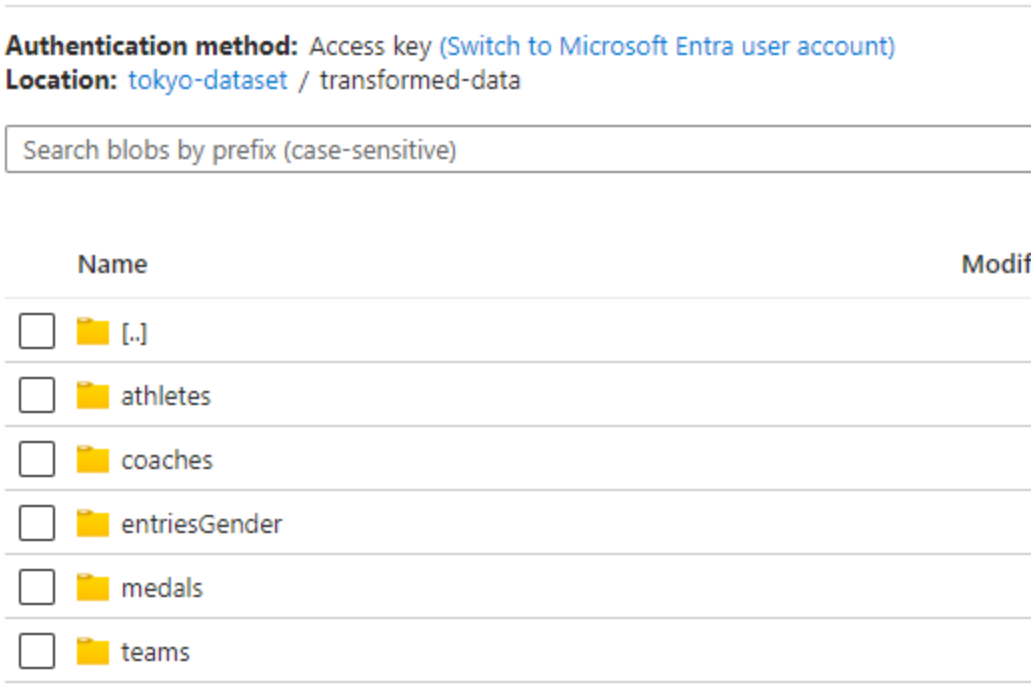






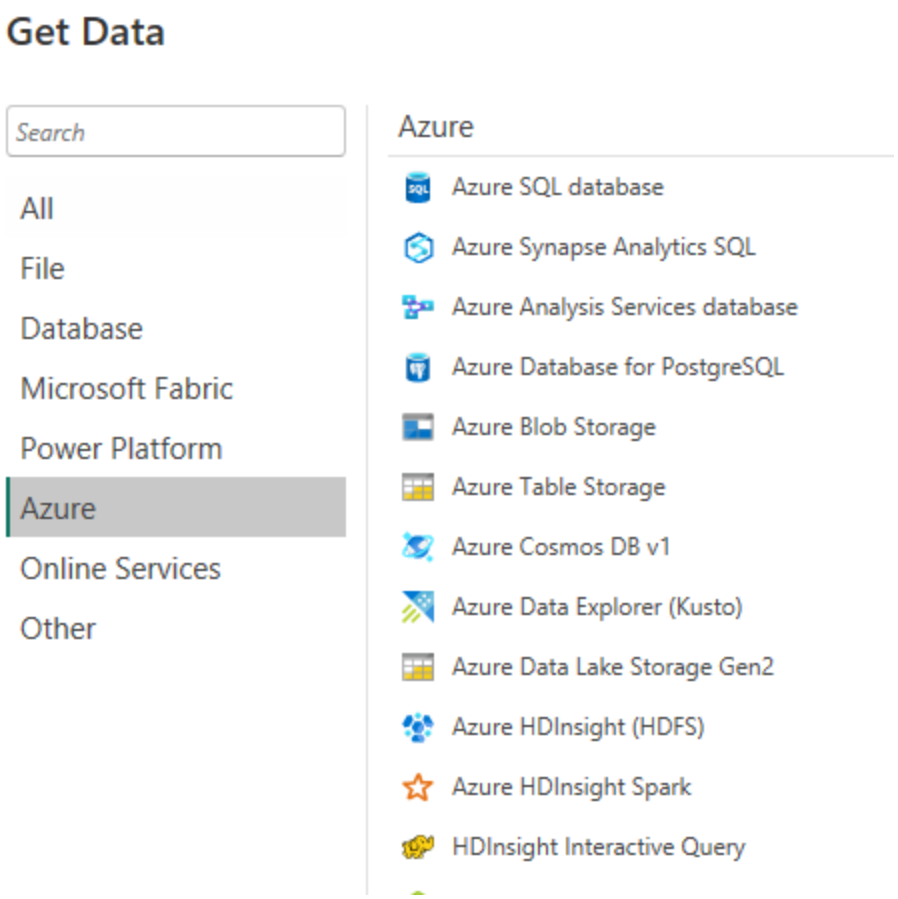
We use repartition(1) and mode(”overwrite”), so that no error occurs when running Spark jobs - would otherwise give error of “this already exists”.

Repeat this for the remaining tables.

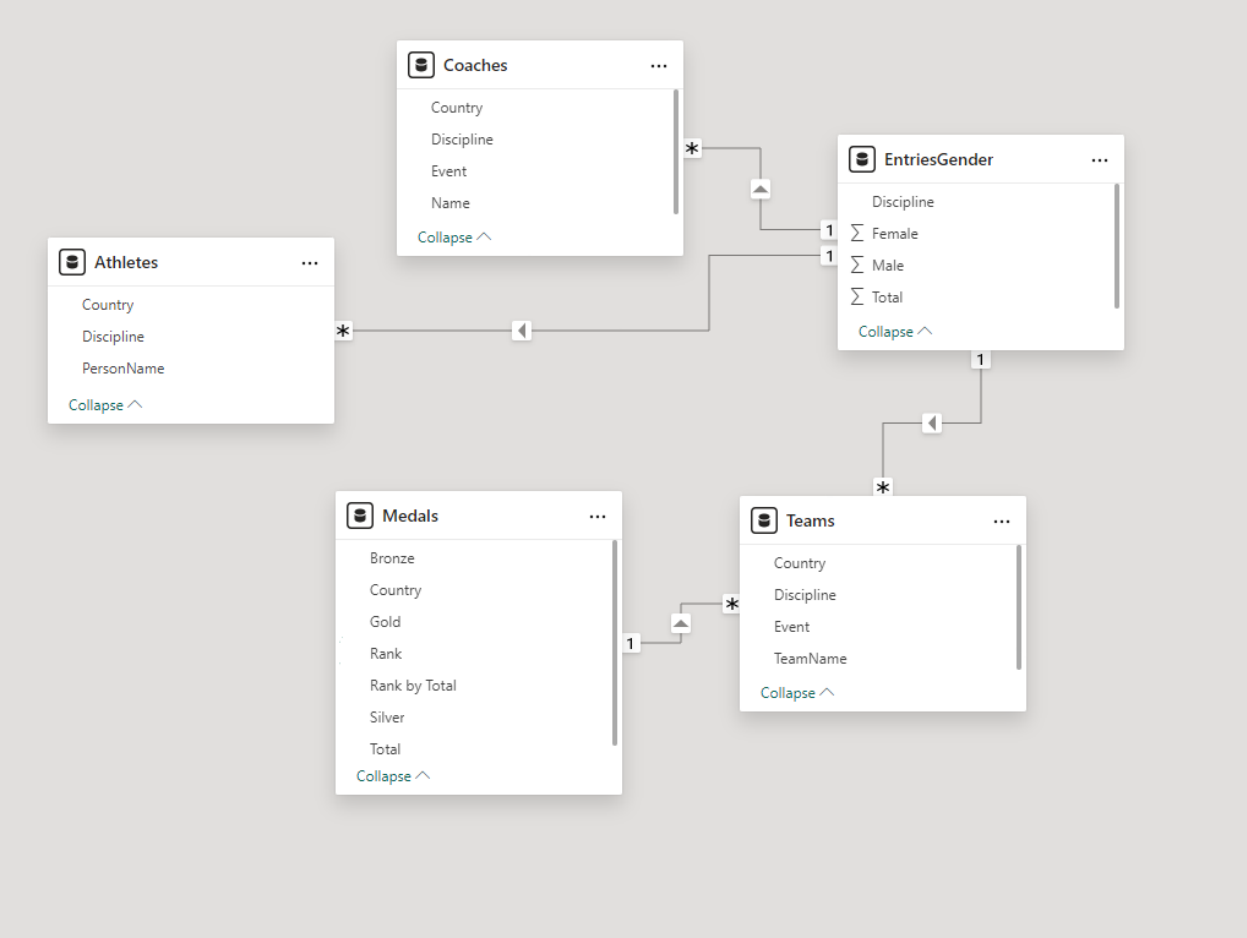


Power BI

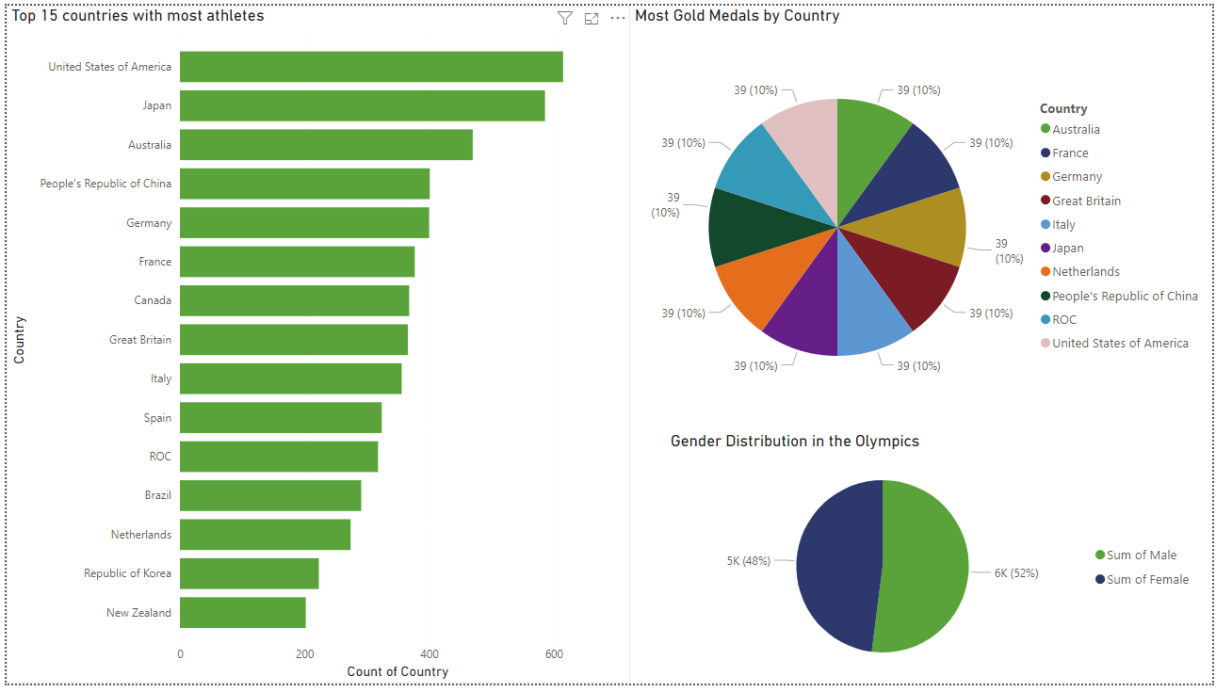
Next we open ‘Power BI Desktop’ and create a connection to our container on Azure via Azure Data Lake Storage Gen2



Then once loaded the data, we can have a look at the data model



Then we can play around with the dashboard and make some analysis based on the data and build a dashboard to visualize it


Top 15 countries with most athletes
United States of America
Japan
Australia
People's Republic of China
Germany
France
Canada
Great Britain
Italy
Spain
ROC
Brazil
Netherlands
Republic of Korea
New Zealand
Country
0
200
400
600
Count of Country
Most Gold Medals by Country
39 (10%)
Country
Australia
France
Germany
Great Britain
Italy
Japan
Netherlands
People's Republic of China
ROC
United States of America
Gender Distribution in the Olympics
5K (48%)
6K (52%)
Sum of Male
Sum of Female