End-to-End Cloud Data Pipeline for COVID-19 Analysis

1- Introduction.

In this project, we will demonstrate an end-to-end data engineering project based on data from the COVID-19 epidemic. We will use Azure Data Factory for data ingestion, Azure Data Lake Gen2 for storage, Azure Databricks for data transformation, Azure Synapse for modeling, and Power BI for visualization.

2- Project Planning and Definition.

In the following project we will use the following schema with the following order :

1. Data Ingestion (Azure Data Factory):

- Azure Data Factory serves as our data ingestion platform. It enables us to collect COVID-19 data from various sources, including government databases, APIs, and web scraping. Data Factory's data connectors and scheduling capabilities are invaluable for automated ingestion.

2. Data Storage (Azure Data Lake Gen2):

- Processed COVID-19 data finds its home in Azure Data Lake Gen2. This storage solution provides scalable, secure, and cost-effective storage, which is critical for accommodating the increasing volume of epidemic data.

3. Data Transformation (Azure Databricks):

- We utilize Azure Databricks for data transformation and processing tasks. Databricks clusters allow us to perform data cleansing, normalization, and feature engineering, preparing the COVID-19 data for analysis.

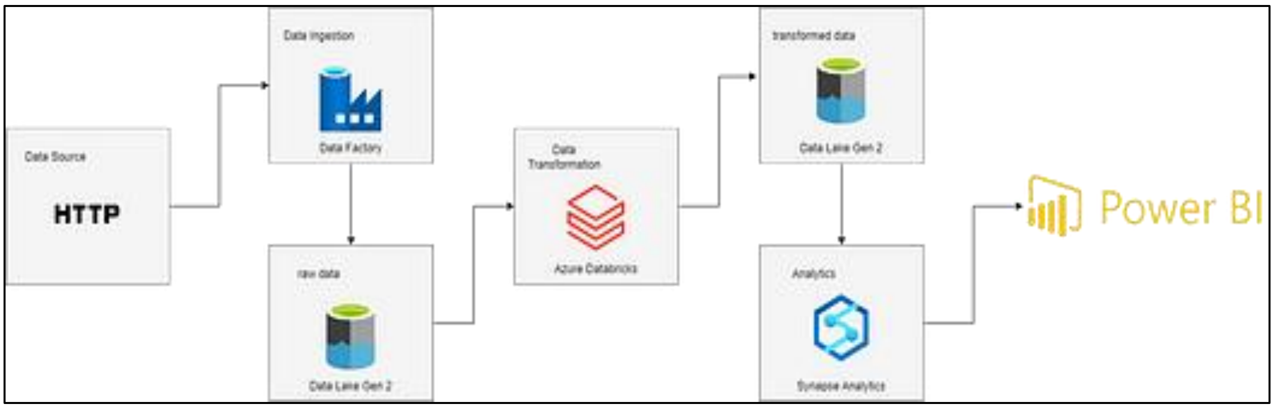
4. Data Modeling (Azure Synapse):

- Azure Synapse serves as our data modeling and analytics platform. We employ it to build data models, perform SQL-based queries, and conduct complex analytical tasks on the COVID-19 dataset.

5. Data Visualization (Power BI):

- Power BI is the tool of choice for data visualization. We create interactive dashboards and reports to present the COVID-19 data insights, enabling stakeholders to make informed decisions.

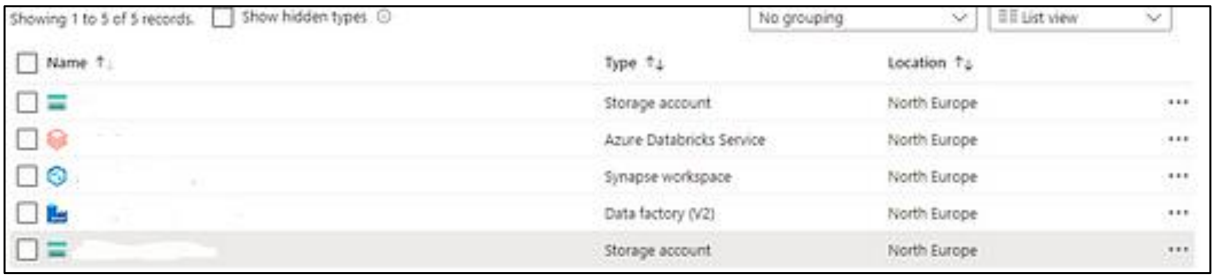
By orchestrating data flow, transformation, storage, modeling, and visualization using Azure services, we aim to provide actionable insights from this critical dataset.



Architectural overview

This architectural overview encapsulates our approach in this data engineering project, emphasizing the role of Azure services in processing and analyzing COVID-19 data. The architecture ensures that the data pipeline is efficient, secure, and capable of handling the evolving requirements of epidemic data analysis.

For this project, we initiated by establishing our resource group, within which we proceeded to create the essential resources.


Showing 1 to 5 of 5 records. Show hidden types

No grouping | List view

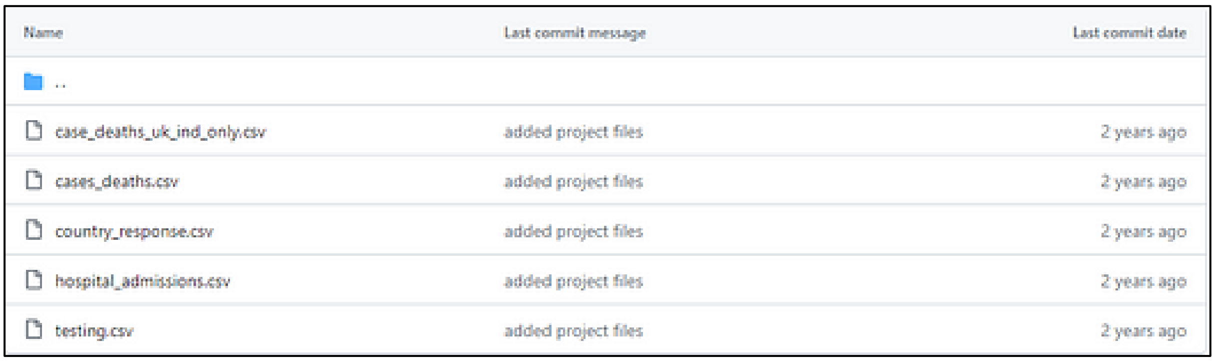
Name	Type	Location
[redacted]	Storage account	North Europe
[redacted]	Azure Databricks Service	North Europe
[redacted]	Synapse workspace	North Europe
[redacted]	Data factory (V2)	North Europe
[redacted]	Storage account	North Europe

Resources

3- Data Collection and Ingestion.

We will extract the data from GitHub, which contains information about victims and cases infected by the COVID epidemic, as well as other related data.

the data will be look :

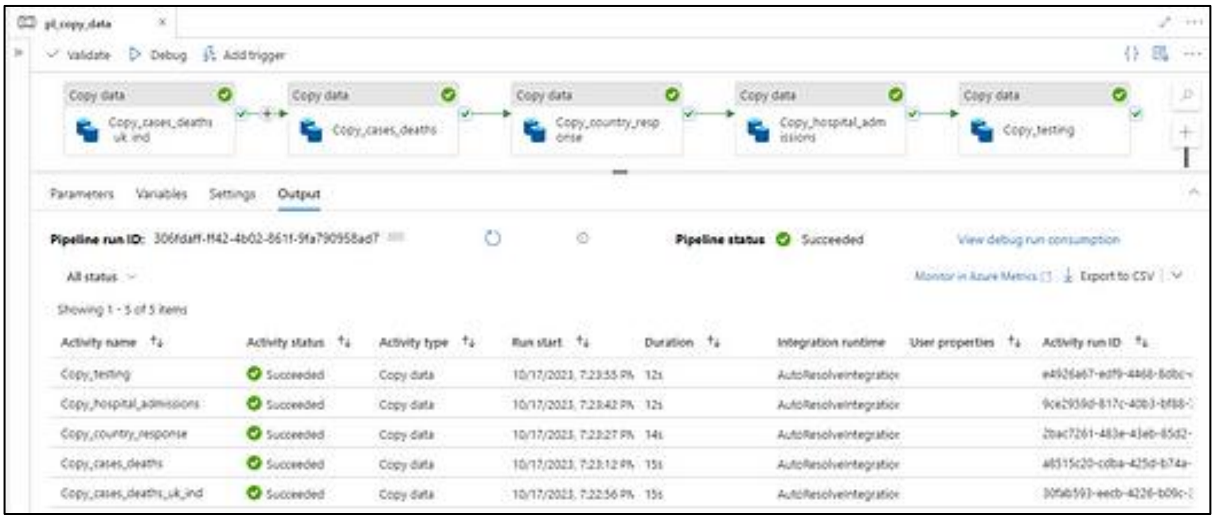


Name	Last commit message	Last commit date
..		
case_deaths_uk_ind_only.csv	added project files	2 years ago
cases_deaths.csv	added project files	2 years ago
country_response.csv	added project files	2 years ago
hospital_admissions.csv	added project files	2 years ago
testing.csv	added project files	2 years ago

Raw Data

In this phase, we harnessed the power of Azure Data Factory to seamlessly ingest data from the “GitHub” source into our Azure environment.

In the next picture we will see the pipeline where I made the ingestion :



Pipeline run ID: 306fdaff-f142-4b02-861f-9fa790958ad7

Pipeline status: Succeeded

All status

Showing 1 - 5 of 5 items

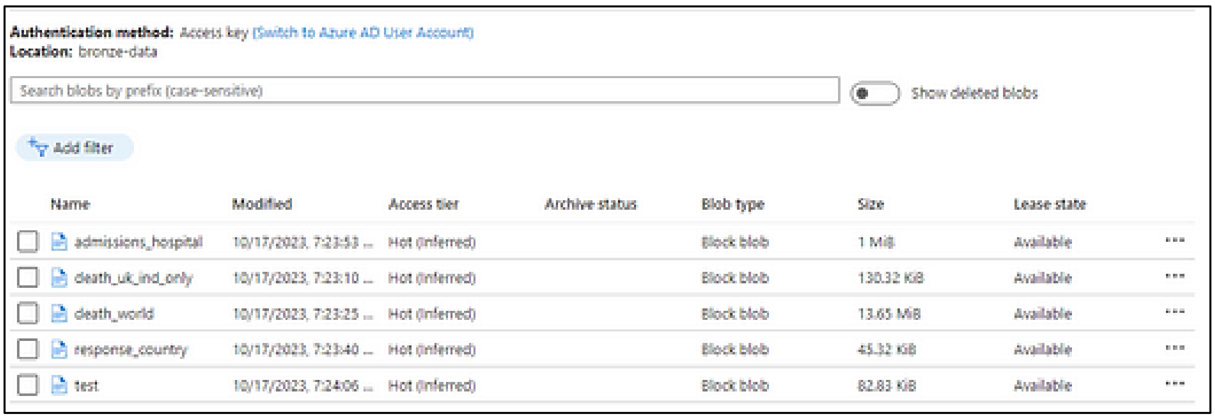
Activity name	Activity status	Activity type	Run start	Duration	Integration runtime	User properties	Activity run ID
Copy_testing	Succeeded	Copy data	10/17/2023, 7:23:55 PM	12s	AutoResolveIntegration		e4926a67-edf9-4468-8dbc-
Copy_hospital_admissions	Succeeded	Copy data	10/17/2023, 7:23:42 PM	12s	AutoResolveIntegration		9ce2939d-817c-40b3-bf88-
Copy_country_response	Succeeded	Copy data	10/17/2023, 7:23:27 PM	14s	AutoResolveIntegration		2bac7261-483e-43eb-85d2-
Copy_cases_deaths	Succeeded	Copy data	10/17/2023, 7:23:12 PM	15s	AutoResolveIntegration		a6515c20-cdba-425d-b74a-
Copy_cases_deaths_uk_ind	Succeeded	Copy data	10/17/2023, 7:22:56 PM	15s	AutoResolveIntegration		30fab593-eecb-4226-b09c-

the pipeline of copy all data

We commence by establishing our source dataset, configured as an HTTP source. This step encompasses the setup of the Linked service and the definition of the BASE URL for all our files. In the case of the sink datasets, we select Delimited Text (CSV) format, and for the Linked service, we designate the destination container as “raw-data” within our Storage account. Since the data remains unaltered and requires no transformations, the primary task revolves around copying the files. We apply minor modifications to the file names to improve their clarity and comprehensibility.

4- Data Storage.

In this phase, we ensure that our data is meticulously organized within our Azure Data Lake Storage. We have designated a specific container, which we refer to as “bronze-data,” for this purpose. This staging area acts as the initial repository for our raw data files, providing a secure and organized location for the data to reside.



Authentication method: Access key (Switch to Azure AD User Account)
Location: bronze-data

Search blobs by prefix (case-sensitive)

Show deleted blobs

Add filter

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
admissions_hospital	10/17/2023, 7:23:53 ...	Hot (Inferred)		Block blob	1 MiB	Available
death_uk_ind_only	10/17/2023, 7:23:10 ...	Hot (Inferred)		Block blob	130.32 KiB	Available
death_world	10/17/2023, 7:23:25 ...	Hot (Inferred)		Block blob	13.65 MiB	Available
response_country	10/17/2023, 7:23:40 ...	Hot (Inferred)		Block blob	45.32 KiB	Available
test	10/17/2023, 7:24:06 ...	Hot (Inferred)		Block blob	82.83 KiB	Available

Raw-data in ADLS-Gen2

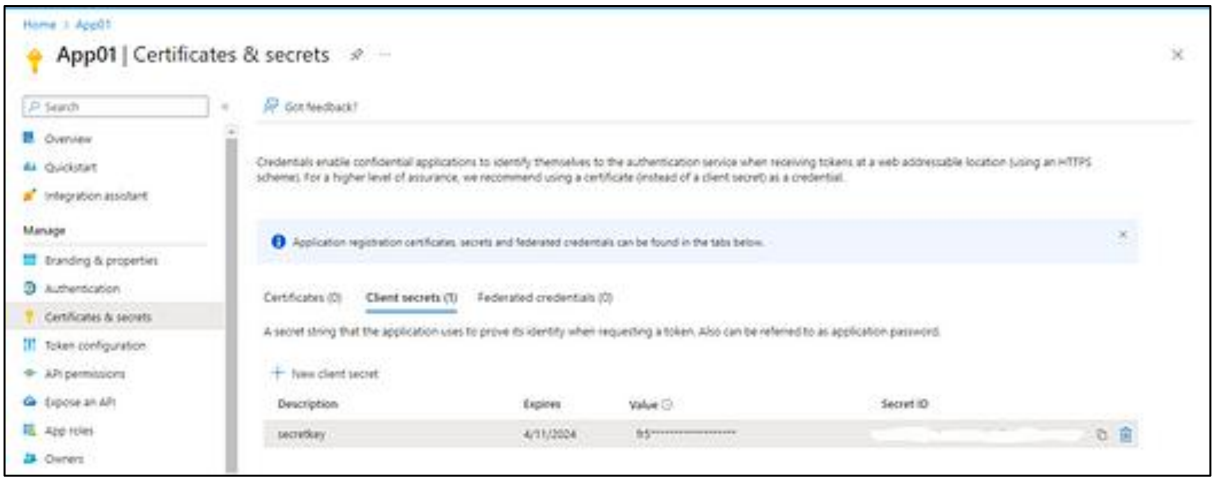
5- Data Transformation.

In the data transformation phase, we leverage Azure Databricks with a notebook environment to write and execute our Spark code. To initiate this process, we've provisioned Azure Databricks by creating the required resources and compute instances. However, before delving into coding, it is crucial to establish a secure connection between Azure Databricks and the Azure Storage accounts that house our raw data.

To achieve this, we have developed an application using the 'App registrations' resource provided by Azure. Within this application, we've generated an 'Application (client) ID' and a 'Directory (tenant) ID.' For clarity, this application is named 'App01.'

Subsequently, within the 'Certificates & secrets' section of application management, we have generated a client secret key (as illustrated in the figure below). This secret key plays a pivotal role in maintaining a secure and robust connection between Azure Databricks and the Azure Storage accounts, enabling seamless data transformation and processing.

Finally, we must configure role assignments for 'Storage Blob Data Contributor.' This assignment allows Azure Databricks to read, write, and delete access to Azure Storage blob containers and data, facilitating efficient data management and processing."



Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Application registration certificates, secrets and federated credentials can be found in the tabs below.

Certificates (0) Client secrets (1) Federated credentials (0)

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

+ New client secret

Description	Expires	Value	Secret ID
secretkey	4/11/2024	fr5******************	[redacted]

the secret Key

So now, we have commenced writing our Spark code in the notebook, beginning with the establishment of the connection between the container in the storage accounts and our notebook.

```
configs = {"fs.azure.account.auth.type": "OAuth",
"fs.azure.account.oauth.provider.type":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
"fs.azure.account.oauth2.client.id": "client ID",
"fs.azure.account.oauth2.client.secret": 'Secret Key',
"fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/Directory
(tenant) ID/oauth2/token"}
```

```
#raw data = bronze data
dbutils.fs.mount(
source = "abfss://bronze-data@<Storage Account name>.dfs.core.windows.net", #
contrainer@storageacc
mount_point = "/mnt/bronze-data",
extra_configs = configs)


#transformed data
dbutils.fs.mount(
source = "abfss://transformed-data@<Storage Account name>.dfs.core.windows.net", #
contrainer@storageacc
mount_point = "/mnt/transformed-data",
extra_configs = configs)
```

After this, we can list the files and directories in the 'bronze data' container by using the following command:

```
%fs
ls "/mnt/raw-data"
```

After that, we should configure our Spark session using the command: Spark. Now, we can begin our actual data work by first reading the data:

```
#read the data
admissions_hospital = spark.read.format("csv").option("header","true").load("/mnt/covid-
data/admissions_hospital")
death_uk_ind_only = spark.read.format("csv").option("header","true").load("/mnt/covid-
data/death_uk_ind_only")
death_world = spark.read.format("csv").option("header","true").load("/mnt/covid-
data/death_world")
response_country = spark.read.format("csv").option("header","true").load("/mnt/covid-
data/response_country")
test = spark.read.format("csv").option("header","true").load("/mnt/covid-data/test")
```

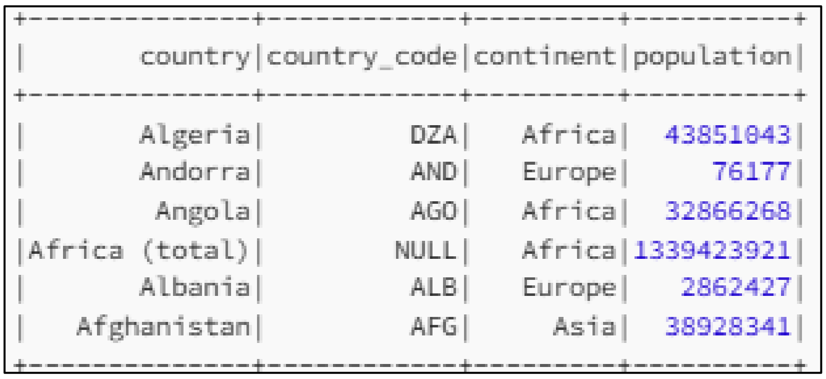
We can utilize various PySpark SQL DataFrame methods, such as .show() and .printSchema(), to view and gain a better understanding of the data.

After reviewing the data and considering our requirements, we have decided to extract two dimension tables from our dataset. The first one, named ‘dim_country,’ will contain information about countries, including details such as country code, continent, and population. this table will be extracted from our ‘death_world’ file.

```
#let's process and transform the death world data
death_world.createOrReplaceTempView("raw_death_world") #to query the files
dim_country = spark.sql("""select distinct country
                        , country_code
                        , continent
                        , population
                    from raw_death_world
                    """)
```

The data in the ‘dim_country’ table will be structured as follows:

dim_country.show(6)



```
+--------------+------------+---------+----------+
|       country|country_code|continent|population|
+--------------+------------+---------+----------+
|       Algeria|         DZA|   Africa|  43851043|
|       Andorra|         AND|   Europe|     76177|
|        Angola|         AGO|   Africa|  32866268|
|Africa (total)|        NULL|   Africa|1339423921|
|       Albania|         ALB|   Europe|   2862427|
|   Afghanistan|         AFG|     Asia|  38928341|
+--------------+------------+---------+----------+
```

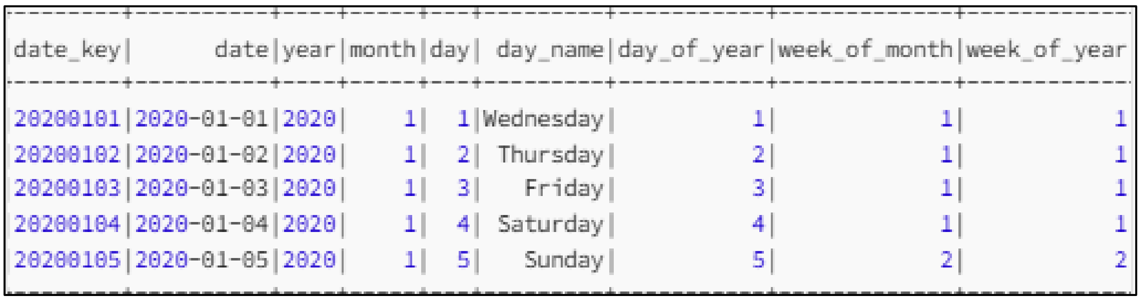
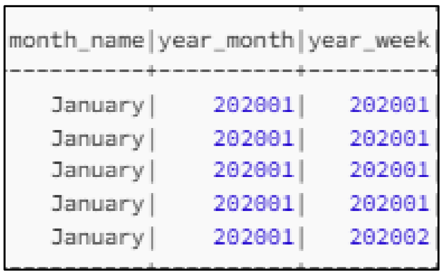
only showing top 6 rows

The second dimension table that we will use is ‘dim_date,’ which contains information structured as follows: date_key, date, year, month, day, day_name, day_of_year, week_of_month, week_of_year, month_name, year_month, and year_week.

This table will include data for the date range from January 1, 2020, to December 30, 2022. The table will contain 1,095 records.

the table will look like that:

dim_date.show(5)

```
+--------+----------+----+-----+---+---------+-----------+-------------+------------
|date_key|      date|year|month|day| day_name|day_of_year|week_of_month|week_of_year
+--------+----------+----+-----+---+---------+-----------+-------------+------------
|20200101|2020-01-01|2020|    1|  1|Wednesday|          1|            1|           1
|20200102|2020-01-02|2020|    1|  2| Thursday|          2|            1|           1
|20200103|2020-01-03|2020|    1|  3|   Friday|          3|            1|           1
|20200104|2020-01-04|2020|    1|  4| Saturday|          4|            1|           1
|20200105|2020-01-05|2020|    1|  5|   Sunday|          5|            2|           2
+--------+----------+----+-----+---+---------+-----------+-------------+------------

month_name|year_month|year_week
----------+----------+---------
   January|    202001|   202001
   January|    202001|   202001
   January|    202001|   202001
   January|    202001|   202001
   January|    202001|   202002
```

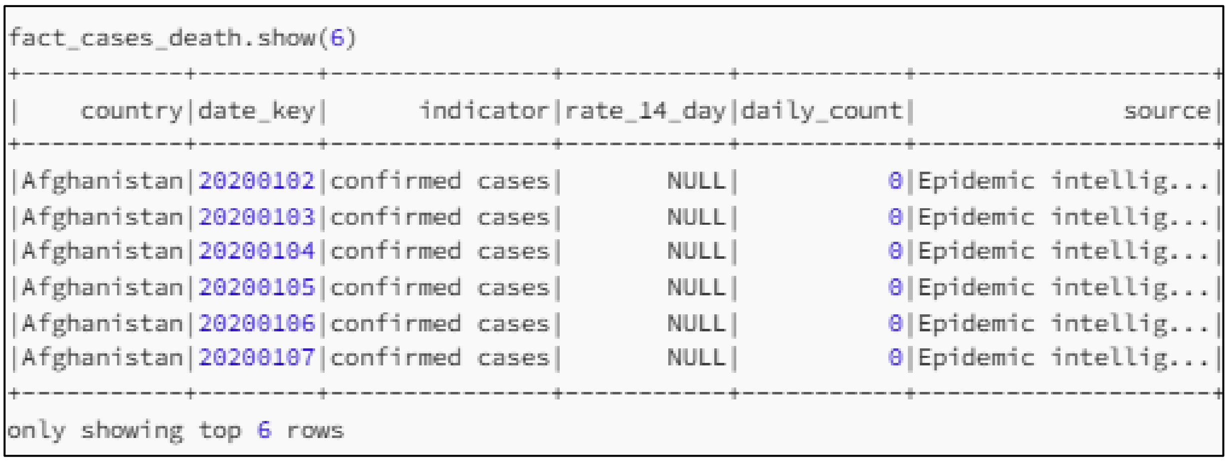
only showing top 5 rows.

For the fact tables, we’re going to extract three fact tables: fact_cases_death, fact_response_country, and fact_admissions_hospital.

fact_cases_death will contain measures related to countries, date, indicator, rate_14_day, daily_count, and source. Regarding the date column, we need to change the format to facilitate our analytics on Azure Synapse, making it more convenient.

```
#let's go now for the fact table of cases death
death_world.createOrReplaceTempView("death_world") #pour que on puisse faire des requetes
fact_cases_death = spark.sql("""select country, date_format(date,'yyyyMMdd' ) as date_key,
                    indicator, rate_14_day, daily_count, source
                    from death_world
                  """)
```

So, in it, the data looks like this :



```
fact_cases_death.show(6)
+-----------+--------+---------------+-----------+-----------+--------------------+
|    country|date_key|      indicator|rate_14_day|daily_count|              source|
+-----------+--------+---------------+-----------+-----------+--------------------+
|Afghanistan|20200102|confirmed cases|       NULL|          0|Epidemic intellig...|
|Afghanistan|20200103|confirmed cases|       NULL|          0|Epidemic intellig...|
|Afghanistan|20200104|confirmed cases|       NULL|          0|Epidemic intellig...|
|Afghanistan|20200105|confirmed cases|       NULL|          0|Epidemic intellig...|
|Afghanistan|20200106|confirmed cases|       NULL|          0|Epidemic intellig...|
|Afghanistan|20200107|confirmed cases|       NULL|          0|Epidemic intellig...|
+-----------+--------+---------------+-----------+-----------+--------------------+
only showing top 6 rows
```

fact_response_countrywill contain measures related to Country, Response_measure, change, date_start, and date_end. Regarding the date_end and date_start columns, we also need to change the format to facilitate our analytics on Azure Synapse, making it more convenient.

```
response_country.createOrReplaceTempView("response_country")
fact_response_country = spark.sql("""select Country
                            , Response_measure
                            , change
                            , date_format(date_start,'yyyyMMdd' ) as date_start
                            , date_format(date_end,'yyyyMMdd' ) as date_end
                        from response_country
                      """)
```

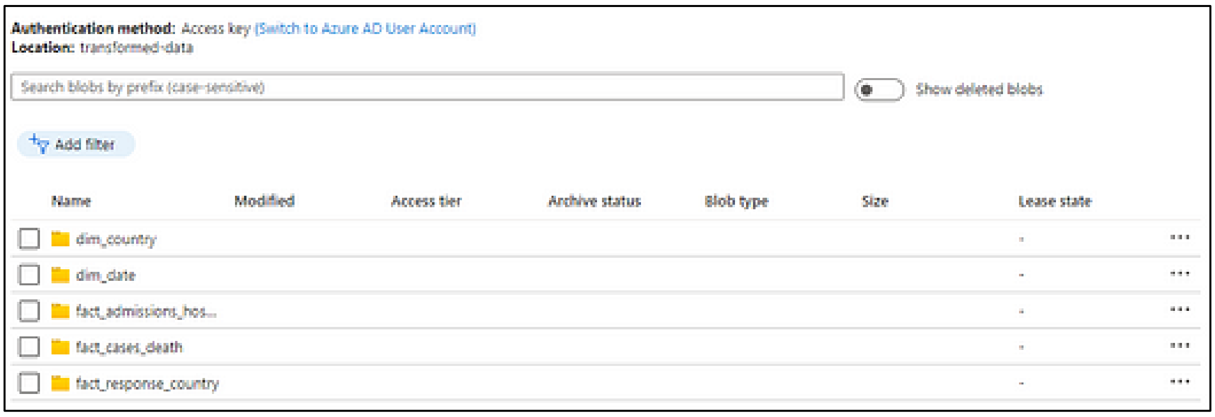
fact_admissions_hosptalwill contain measures related to Country, indicator, date, value, source, and URL.

```
admissions_hospital.createOrReplaceTempView("admissions_hospital")
fact_admissions_hosptal = spark.sql("""select country
                            , indicator
                            , date_format(date,'yyyyMMdd' ) AS date_key
                            , value
                            , source
                            , url
                        from admissions_hospital
                      """)
```

Now that we know the dimensions table and the fact table, we should write these tables into our container named ‘transformed data’

```
#Write our files
dim_date.write.format("com.databricks.spark.csv").option("header","true").option("delimiter", ",").mode("overwrite").save("/mnt/transformed-data/dim_date")
dim_country.write.format("com.databricks.spark.csv").option("header","true").option("delimiter", ",").mode("overwrite").save("/mnt/transformed-data/dim_country")
fact_cases_death.write.format("com.databricks.spark.csv").option("header","true").option("delimiter", ",").mode("overwrite").save("/mnt/transformed-data/fact_cases_death")
fact_response_country.write.format("com.databricks.spark.csv").option("header","true").option("delimiter", ",").mode("overwrite").save("/mnt/transformed-data/fact_response_country")
fact_admissions_hosptal.write.format("com.databricks.spark.csv").option("header","true").option("delimiter", ",").mode("overwrite").save("/mnt/transformed-data/fact_admissions_hosptal")
```

So far, so good. Now we can locate our files in our sink container named ‘transformed-data’:



Authentication method: Access key (Switch to Azure AD User Account)
Location: transformed-data

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
dim_country						-
dim_date						-
fact_admissions_hos...						-
fact_cases_death						-
fact_response_country						-

Transformed-data Container

6- Data Modeling.

Now that we have our data in the tables, we will proceed to load it into the Lake Database in Azure Synapse Analytics, enabling us to create our models.

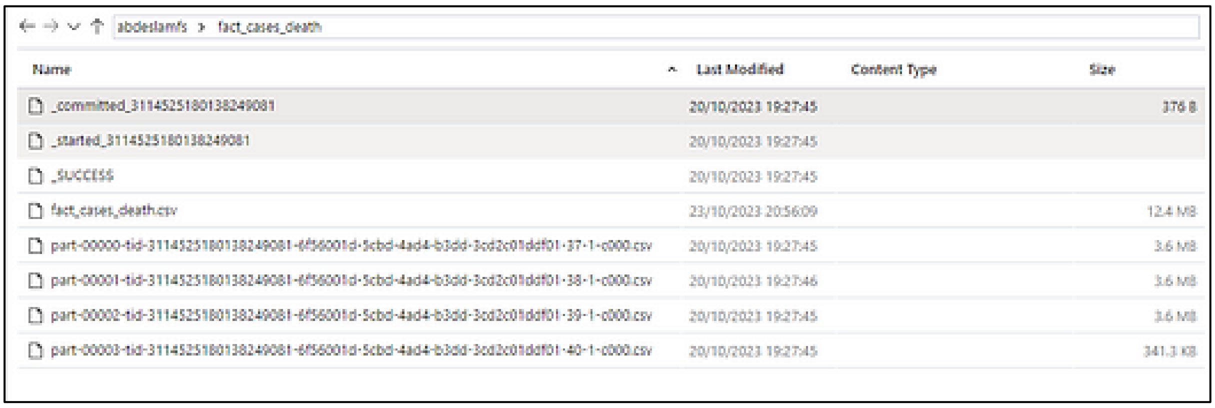
First, we need to set up our Azure Synapse workspace. By creating our Synapse Studio, we also create another Storage Account: Azure Data Lake Storage Gen2 (ADLS Gen2).

To use Azure Synapse for working with this data, we should copy the files from the 'Transformed-data' container into our ADLS Gen2. For this purpose, we will utilize a pipeline containing a copy activity from our source with the linked service: AzureBlobStorage, to our destination with the linked service: Default Storage account for our Synapse workspace (ADLS Gen2).

Another tip: to copy all the files in the 'transformed-data' container, rather than one file at a time, we can utilize the 'Wildcard File Path' option with the input as 'transformed-data/*'.

Now, in the data part of the Synapse workspace, we add a Lake Database named 'CovidDB.' Following this, we create external tables from the data lake. To do this, we specify the External table name (which will be the same as 'dim_country,' 'dim_date,' etc.), the Linked service (which will be 'ADLS Gen2'), and the Input file or folder. This input will specify the path to the files.

In the phase of creating our tables in the Lake database, we have recently discovered that the files in the "fact_cases_death" folder have been duplicated four times due to their size, as demonstrated in the picture below.



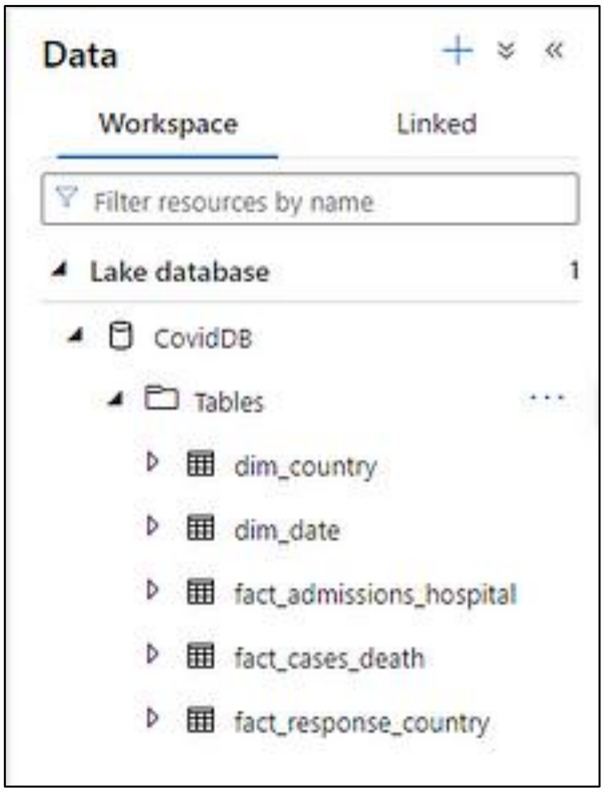
abdeslamfs > fact_cases_death

Name	Last Modified	Content Type	Size
_committed_3114525180138249081	20/10/2023 19:27:45		376 B
_started_3114525180138249081	20/10/2023 19:27:45		
_SUCCESS	20/10/2023 19:27:45		
fact_cases_death.csv	23/10/2023 20:56:09		12.4 MB
part-00000-tid-3114525180138249081-6f56001d-5cbd-4ad4-b3dd-3cd2c01ddf01-37-1-c000.csv	20/10/2023 19:27:45		3.6 MB
part-00001-tid-3114525180138249081-6f56001d-5cbd-4ad4-b3dd-3cd2c01ddf01-38-1-c000.csv	20/10/2023 19:27:46		3.6 MB
part-00002-tid-3114525180138249081-6f56001d-5cbd-4ad4-b3dd-3cd2c01ddf01-39-1-c000.csv	20/10/2023 19:27:45		3.6 MB
part-00003-tid-3114525180138249081-6f56001d-5cbd-4ad4-b3dd-3cd2c01ddf01-40-1-c000.csv	20/10/2023 19:27:45		341.3 KB

folder fact_cases_death

We will now implement a different pipeline to consolidate the files within the "fact_cases_death" folder. This new pipeline will consist of a single activity: data copying. In this pipeline, we will use the wildcard path option directly targeting our "fact_cases_death" folder. Additionally, we will modify the sink settings by choosing the "merge files" copy behavior.

This will make it easier for us to add our tables, as illustrated in the picture below:



Data

Workspace | Linked

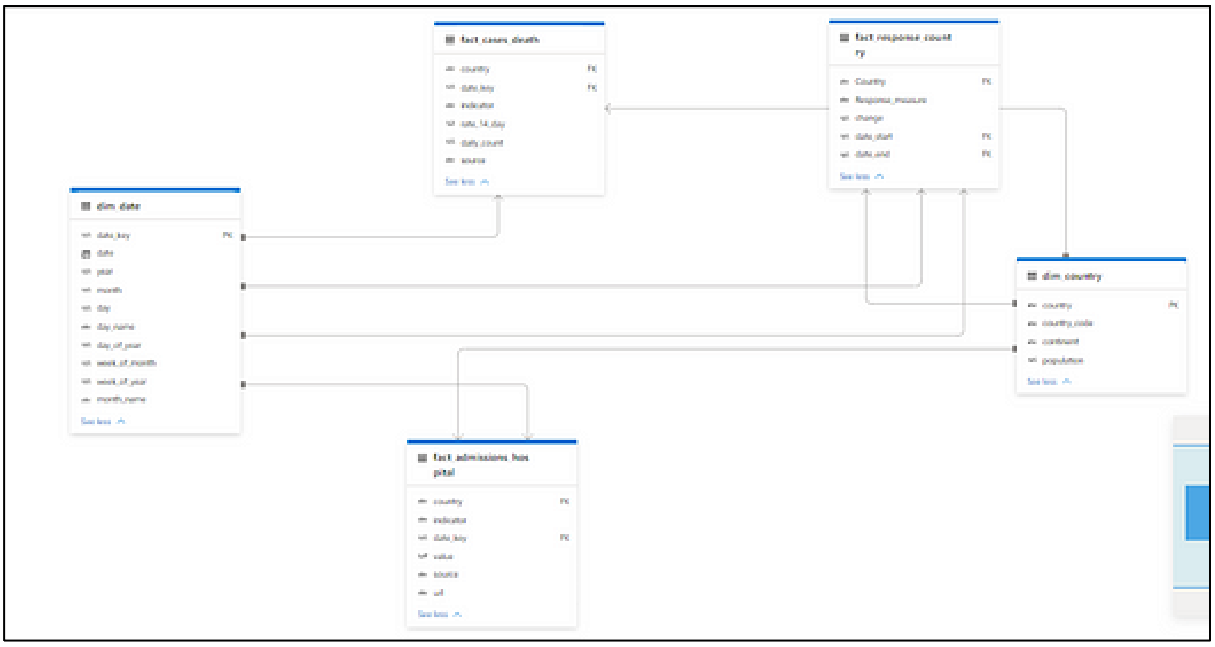
Filter resources by name

- Lake database 1
  - CovidDB
    - Tables
      - dim_country
      - dim_date
      - fact_admissions_hospital
      - fact_cases_death
      - fact_response_country

the Database CovidDB

We will now establish relationships between the tables. A relationship is a connection that defines how the data in these tables should be associated or correlated.

We chose the "To table" option for the fact tables, as these tables serve as the parent tables for the dimension tables.



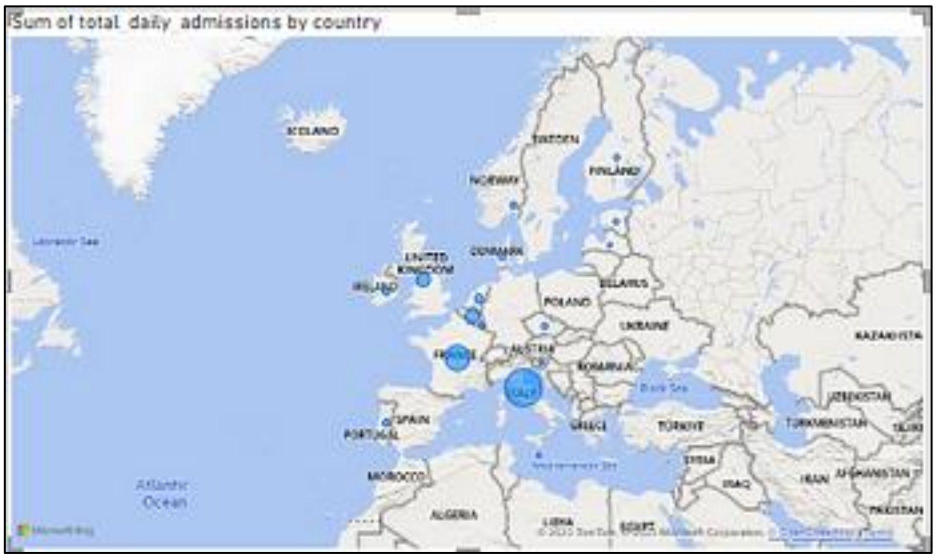
Data model

7- Data Visualization.

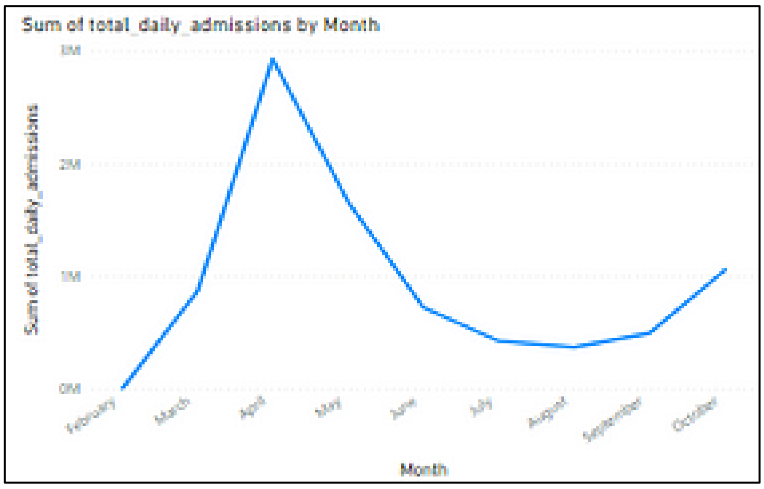
In this section, we will leverage data visualization to gain valuable insights from the dataset. The following measures will be visualized:

Total Daily Hospital Admissions per Country Over time:

- To comprehend the daily hospital admissions on a country level, we execute the following query. It calculates the sum of daily admissions, providing a pivotal metric to monitor the impact of events like COVID-19.



the daily hospital admissions in Mars



total daily admissions by month

for See the changing in time, we will use The “Play Axis” in our presentation, allowing the creation of a time series animation to illustrate how variations between countries change over time.

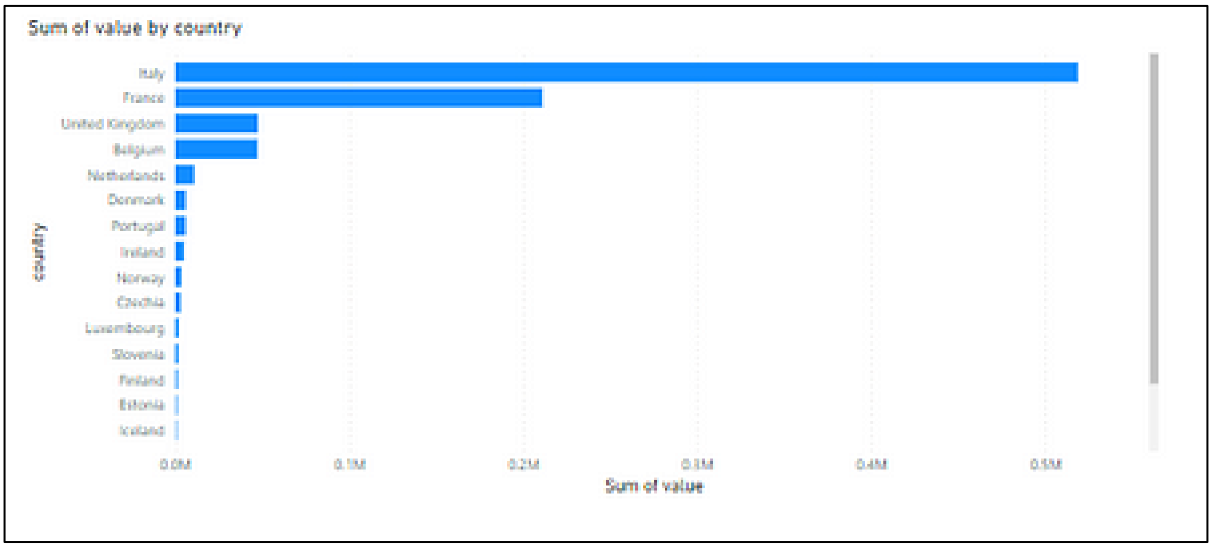
```
-- the query of Total Daily Hospital Admissions per Country Over time
SELECT
  fd.date,
  dc.country,
  SUM(fh.value) AS total_daily_admissions
FROM [CovidDB].[dbo].fact_admissions_hospital fh
JOIN [CovidDB].[dbo].dim_date fd ON fh.date_key = fd.date_key
JOIN [CovidDB].[dbo].dim_country dc ON fh.country = dc.country
GROUP BY fd.date, dc.country
ORDER BY fd.date, dc.country;
```

Trends Over Time:

- Understanding trends over time is crucial for assessing the evolution of daily hospital admissions. We utilize a line chart, utilizing the result from the query above. The date is placed on the x-axis and the total daily admissions on the y-axis. Tracking trends over time helps in identifying significant fluctuations.

Variations Between Countries:

- Comparing variations between countries offers critical insights into the disparities in daily hospital admissions. We opt for a Bar Chart visualization, which enables side-by-side comparisons of different countries.



Comparaison in Mars

also here we will use The “Play Axis” feature.

Seasonal Patterns:

- Identifying seasonal patterns can provide essential context. To achieve this, we employ a query that calculates a seven-day moving average. This moving average is pivotal in recognizing recurring trends in hospital admissions. It can be instrumental in resource allocation and preparedness.

These visualizations are instrumental in uncovering patterns, trends, and disparities in the data.

This screen is just a part of the visualization files for a specific moment, similar to the comparison — it’s only on Mars. It’s more like an animation.