Problem Statement

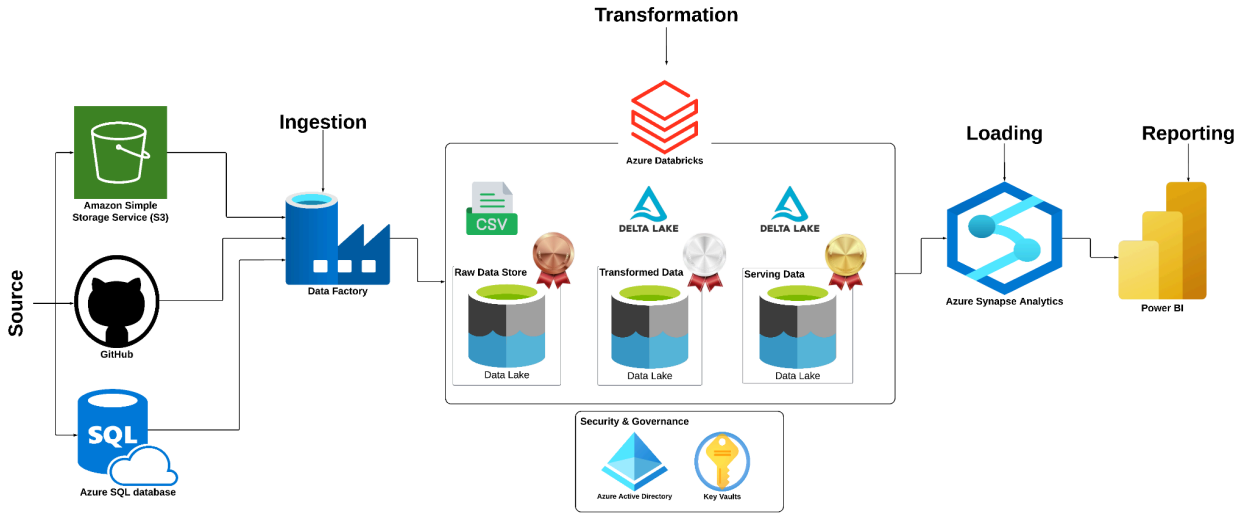
In today's world, the efficient use of renewable energy is crucial to combating climate change and ensuring sustainable development. However, the global progress in renewable energy adoption and efficiency varies significantly across countries, influenced by various economic, social, and political factors. Understanding these variations is key to making informed decisions that can accelerate the transition to renewable energy sources.

This project aims to analyze and visualize key aspects of renewable energy and its relationship with economic factors such as GDP growth. By utilizing various data analysis techniques and visualization tools, we aim to provide insights into the following key areas:

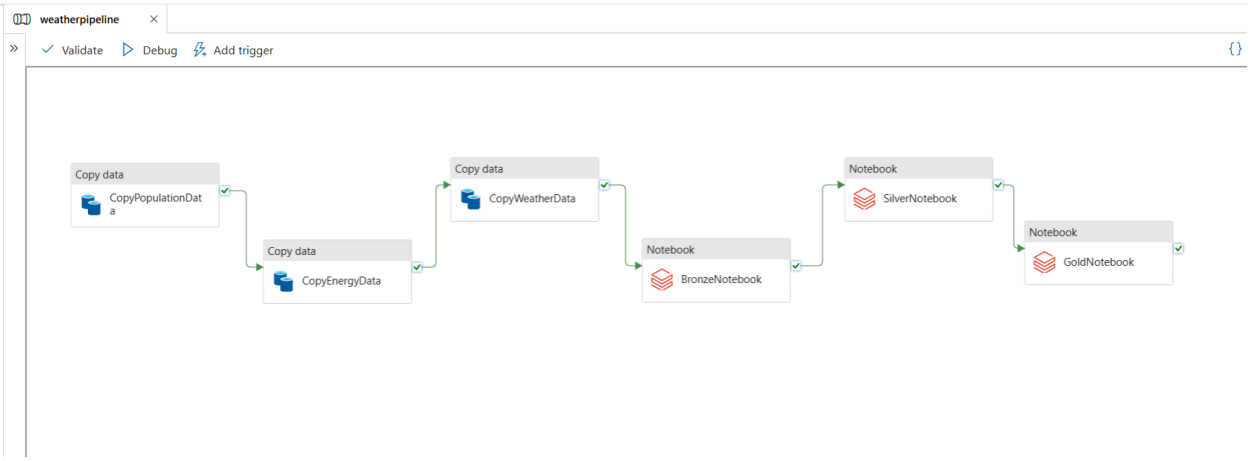
1. **Renewable Energy Share vs. GDP Growth**: Assessing the relationship between renewable energy adoption and economic growth, and understanding how the integration of renewable energy impacts the GDP growth of different countries.
2. **Temperature Analysis by Country and Season**: Exploring the correlation between average temperature and its impact on renewable energy generation across countries and seasons. This will help identify regions that are most affected by climate variations and how they can adapt their renewable energy strategies accordingly.
3. **Renewable Energy Efficiency Across Countries**: Comparing renewable energy efficiency across different countries to understand which regions are excelling in optimizing their renewable energy use and where improvements are needed.
4. **Impact of Renewable Energy on Sustainable Development**: Examining the broader effects of renewable energy adoption on environmental sustainability and economic resilience, and identifying key areas for policy intervention and development.

By the end of this project, the goal is to provide a comprehensive dashboard that enables users to interact with and analyze key data points such as renewable energy share, GDP growth, temperature variations, and energy efficiency across countries. This data-driven approach will help stakeholders, policymakers, and researchers identify trends, make informed decisions, and work towards a more sustainable and efficient global energy future.

Solution Architecture



Pipeline Architecture



Overview of Datasets

For this project, three comprehensive datasets—**Weather**, **Energy**, and **Population**—were utilized to analyze renewable energy efficiency, its relationship with economic factors, and its potential for sustainable development. Below is an overview of each dataset and their relevance to the analysis:

---

1. Weather Dataset

- **Size:** 1,870,160 rows
- **Description:** This dataset provides detailed weather information across various locations globally. It contains daily meteorological data, including temperature, precipitation, and snow depth. The dataset is crucial for understanding how climatic factors, such as temperature and precipitation, impact renewable energy production and efficiency.
- **Key Columns:**
  - `STATION`: Identifier for the weather station.
  - `Country Region`: Country or region where the station is located.
  - `DATE`, `Year`, `Month`, `Day`: Temporal data to analyze trends over time.
  - `PRCP`, `SNWD`, `SNOW`: Precipitation, snow depth, and snowfall data.
  - `TAVG`, `TMAX`, `TMIN`: Average, maximum, and minimum temperature values.
  - `LATITUDE`, `LONGITUDE`, `ELEVATION`: Geographical coordinates and elevation of the station.
  - `DAPR`, `MDPR`, `WESD`: Additional precipitation-related measures.

---

2. Energy Dataset

- **Size:** 69,672 rows
- **Description:** This dataset provides detailed information on energy production, consumption, and sustainability metrics. It plays a central role in analyzing renewable energy trends, economic growth, and environmental impact.
- **Key Columns:**
  - `Entity`: Country or region.
  - `Year`: Year of the data point.
  - `Access to electricity (%)`: Percentage of population with electricity access.
  - `Access to clean fuels for cooking`: Percentage of population with access to clean cooking fuels.
  - `Renewable-electricity-generating-capacity-per-capita`: Capacity per person for renewable electricity generation.
  - `Renewable energy share in the total final energy consumption (%)`: Percentage of energy from renewable sources.
  - `Electricity from renewables (TWh)`: Total renewable electricity generated in terawatt-hours.
  - `Low-carbon electricity (% electricity)`: Share of electricity generated from low-carbon sources.
  - `gdp_growth`, `gdp_per_capita`: Economic indicators to study the relationship between energy use and economic growth.
  - `Density (P/Km²)`, `Land Area (Km²)`: Population density and land area for regional analysis.
  - `Value_co2_emissions_kt_by_country`: Carbon emissions in kilotons, for environmental impact analysis.

---

3. Population Dataset

- **Size:** 2,562 rows
- **Description:** This dataset provides demographic data, such as population size, density, and urbanization. It is used to analyze the relationship between population characteristics and renewable energy adoption.
- **Key Columns:**
  - `Country (or dependency)`: Country or region.
  - `Population (2020)`: Population size in 2020.
  - `Yearly Change`, `Net Change`: Population growth rates and changes.
  - `Density (P/Km²)`: Population density for spatial analysis.
  - `Land Area (Km²)`: Total land area for each country or dependency.
  - `Migrants (net)`: Net migration data.
  - `Fert. Rate`, `Med. Age`: Fertility rate and median age for demographic insights.
  - `Urban Pop %`: Percentage of urban population.
  - `World Share`: Percentage share of the world population.

Relevance of the Datasets

The integration of these datasets enables a multi-faceted analysis of renewable energy efficiency and its determinants. The weather data helps assess the climatic factors influencing renewable energy production. The energy data provides insights into renewable energy adoption, economic growth, and environmental impact. Lastly, the population data aids in understanding the role of demographic factors in renewable energy utilization and sustainability.

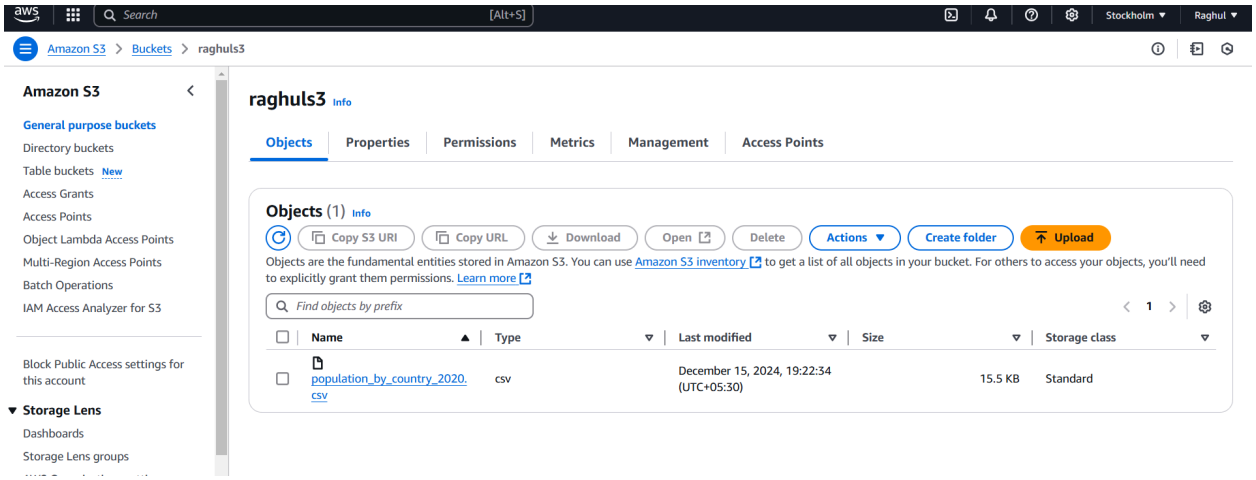
These datasets collectively form the foundation for building visualizations and insights to achieve the project's goals.

Data Ingestion

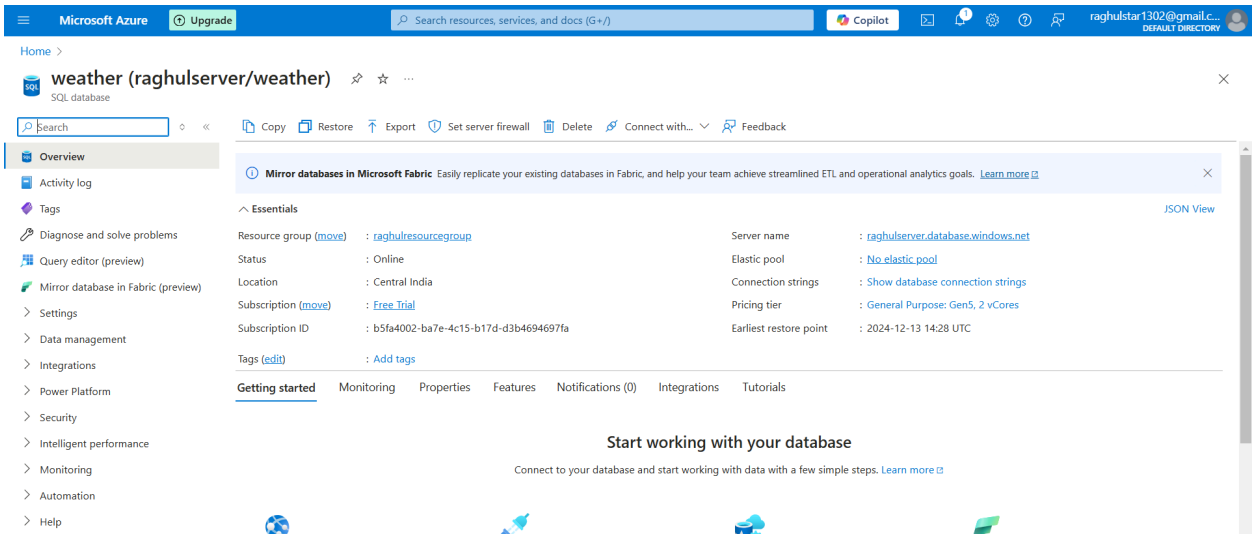
Data Sources

For this project, three distinct datasets—**Population**, **Weather**, and **Energy**—were sourced from diverse platforms, ensuring comprehensive coverage of the data required for analysis. The datasets originate from the following sources:

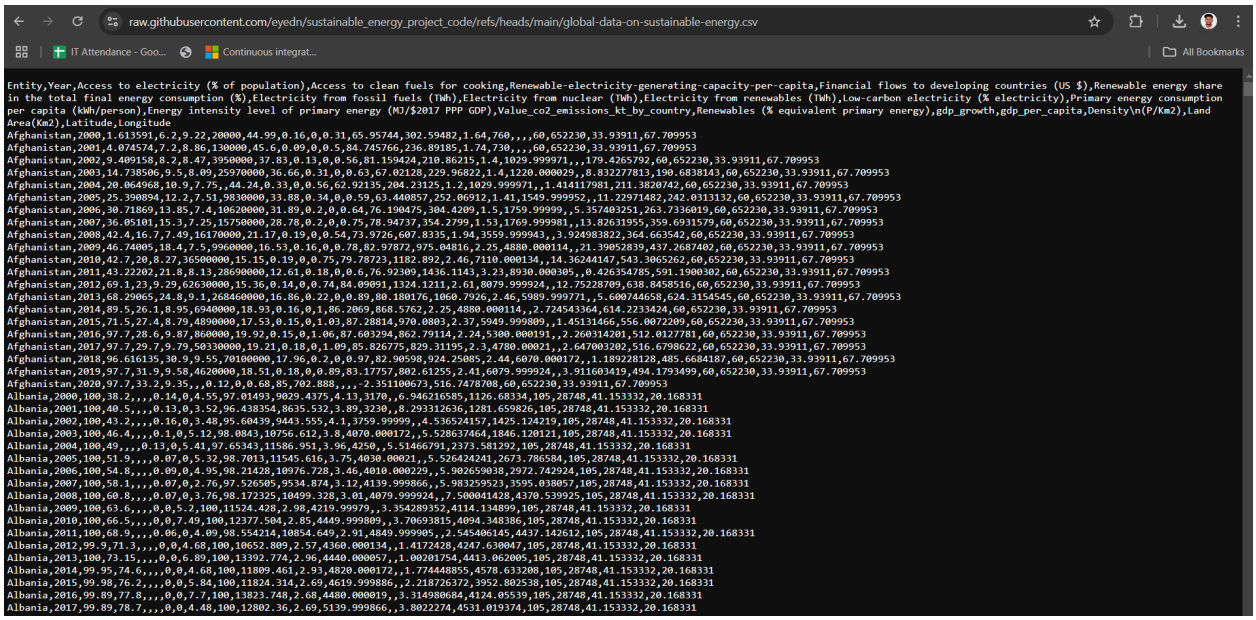
1. **Population Data:** Retrieved from **AWS S3**.



2. **Weather Data:** Stored in **Azure SQL Database**.



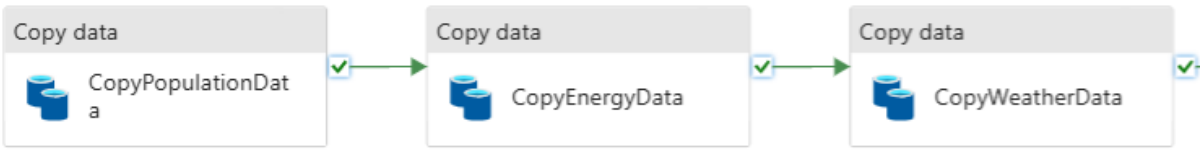
3. **Energy Data:** Accessed from **GitHub**.



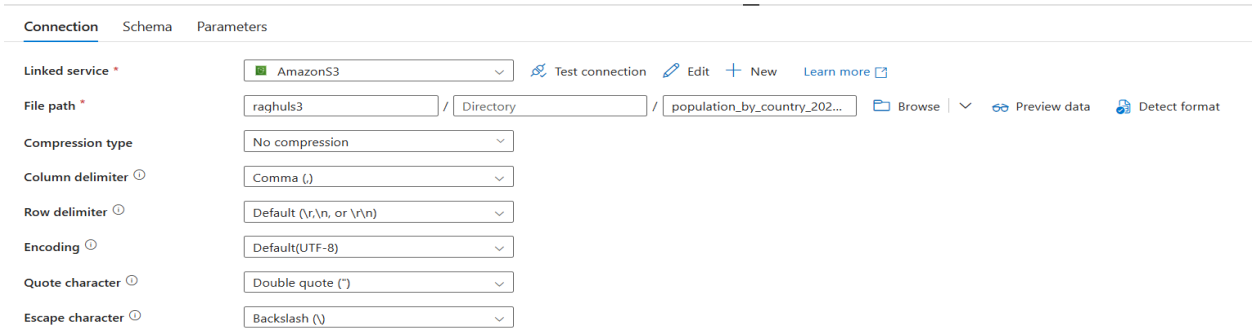
Ingestion Process

To streamline and automate the ingestion of data from these disparate sources, **Azure Data Factory (ADF)** was utilized. ADF served as the orchestration tool to design and implement a robust pipeline for efficient data movement. Below is a detailed outline of the ingestion process:

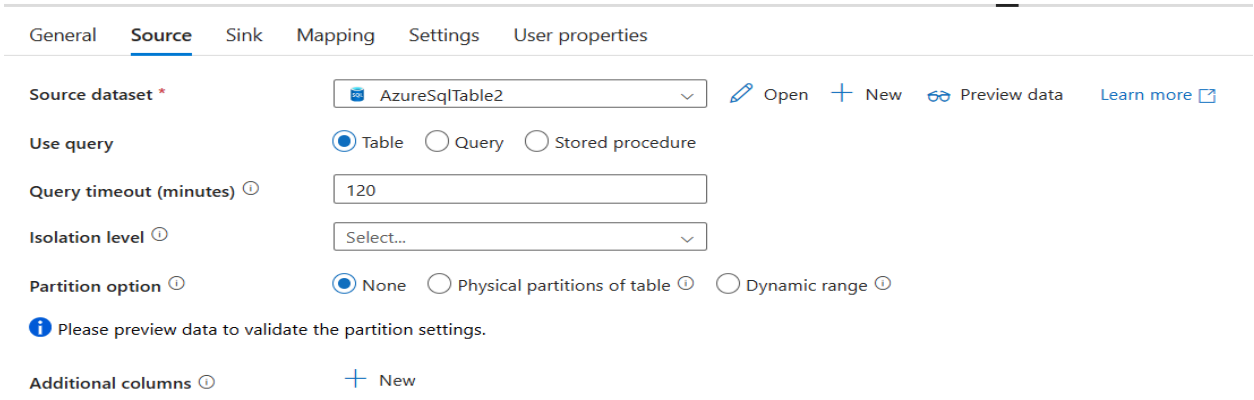
1. **Pipeline Design in ADF:**
    - A dedicated pipeline was created to handle the ingestion of each dataset.
    - The pipeline included **Copy Data activities** to extract data from the source systems and transfer it to the designated sink.



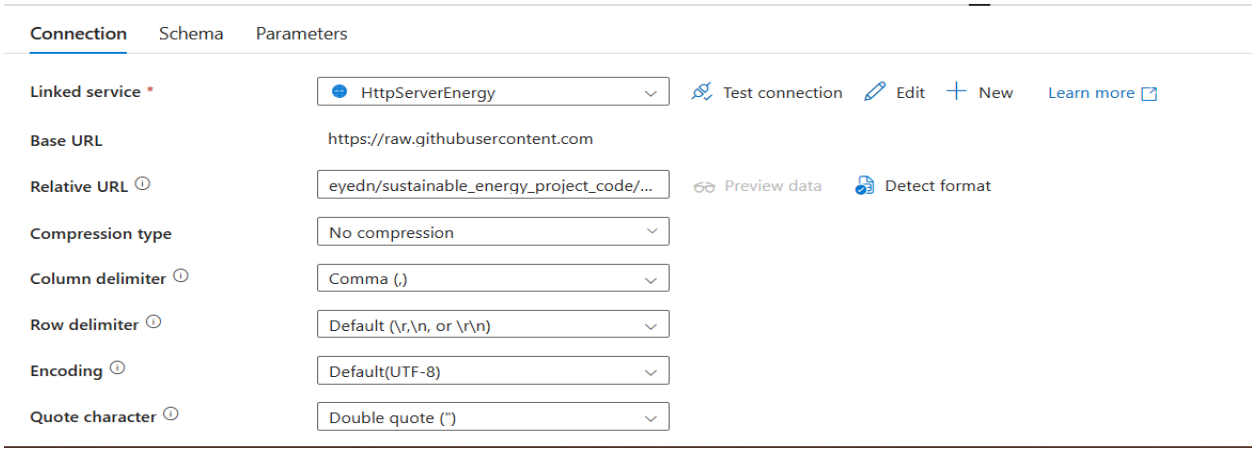
2. **Source Integration:**
    - **Population Data (AWS S3):** Configured a source connection in ADF to fetch the dataset from the AWS S3 bucket.



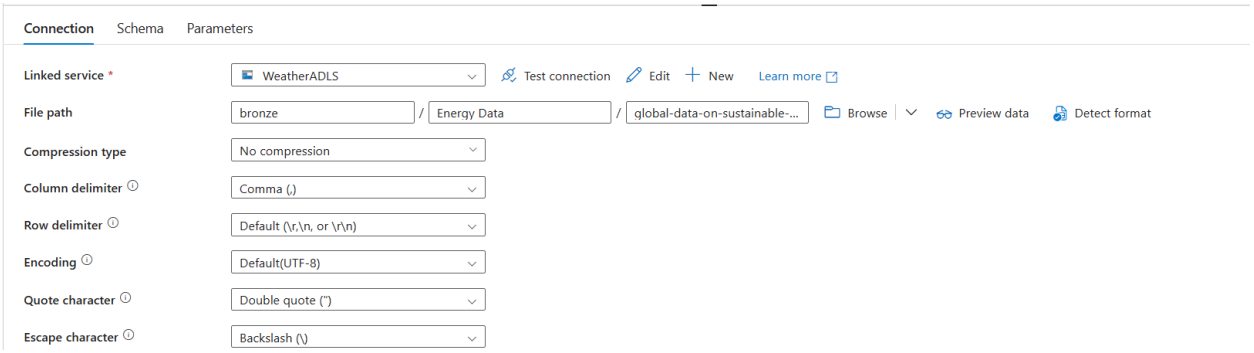
    - **Weather Data (Azure SQL DB):** Established a connection to the Azure SQL Database to retrieve weather-related data.



    - **Energy Data (GitHub):** Utilized ADF's HTTP connector to extract the energy dataset from the GitHub repository.



3. **Data Sink:**
    - The ingested data from all three sources was consolidated into **Azure Data Lake Storage (ADLS) Gen 2**. ADLS Gen 2 served as the centralized repository for storing and managing the datasets in a structured format, enabling downstream processes like transformation and analysis.



Benefits of the Ingestion Process

- **Scalability:** ADF's integration capabilities allow seamless handling of large datasets from multiple sources.
- **Automation:** The pipeline eliminates manual effort by automating the extraction and loading processes.
- **Centralized Storage:** Storing the data in ADLS Gen 2 facilitates easy access for analysis and visualization.
- **Reliability:** The use of ADF ensures reliable and consistent data movement with built-in monitoring and error-handling mechanisms.

This ingestion process lays the foundation for data transformation, modeling, and analysis, enabling meaningful insights into renewable energy efficiency, population trends, and climatic influences.

Data Transformation

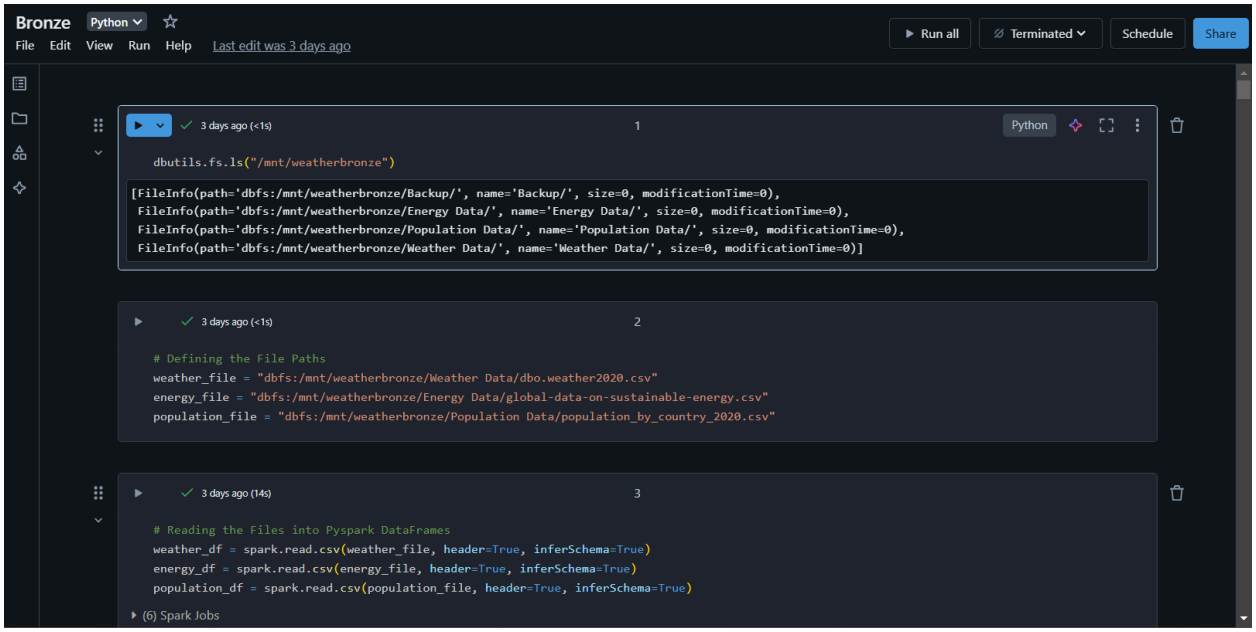
Overview

For the data transformation layer, I utilized **Azure Databricks** to implement the **Medallion Architecture**. This approach structures data into three layers: Bronze, Silver, and Gold. The transformation process was managed through four Databricks notebooks:

1. **Mounting Notebook**: Configured Azure Data Lake Storage (ADLS) mount points to access raw data.
2. **Bronze Layer**: Ingested raw data into Databricks, retaining the original structure.
3. **Silver Layer**: Performed data cleansing and transformation to prepare data for analysis.
4. **Gold Layer**: Created optimized, analytics-ready datasets.

Bronze Layer

- **Objective**: Load raw data into Databricks while preserving its original structure.
- **Data Sources**:
    - Weather data (`dbo.weather2020.csv`)
    - Energy data (`global-data-on-sustainable-energy.csv`)
    - Population data (`population_by_country_2020.csv`)
- **Steps**:
    - Configured file paths for each dataset.
    - Loaded data into PySpark DataFrames using `spark.read.csv`.
    - Performed initial exploration by displaying schemas, summaries, and sample data.



3 days ago (1s) 4

```
weather_df.display()
```

(1) Spark Jobs

Table

	STATION	Country Region	DATE	Year	Month	Day	PRCP	SNWD	
1	VEM00080428	Saint Lucia	2020-03-01T00:00:00.000+00:...	2020	3	1	0	0	
2	VEM00080428	Saint Lucia	2020-04-01T00:00:00.000+00:...	2020	4	1	0	0	
3	VEM00080428	Saint Lucia	2020-05-01T00:00:00.000+00:...	2020	5	1	0	0	
4	VEM00080428	Saint Lucia	2020-06-01T00:00:00.000+00:...	2020	6	1	0	0	
5	VEM00080428	Saint Lucia	2020-07-01T00:00:00.000+00:...	2020	7	1	0	0	
6	VEM00080457	Saint Lucia	2020-02-01T00:00:00.000+00:...	2020	2	1	0	0	
7	VEM00080457	Saint Lucia	2020-03-01T00:00:00.000+00:...	2020	3	1	0	0	
8	VEM00080457	Saint Lucia	2020-04-01T00:00:00.000+00:...	2020	4	1	0	0	
9	VEM00080457	Saint Lucia	2020-05-01T00:00:00.000+00:...	2020	5	1	0	0	
10	VEM00080457	Saint Lucia	2020-06-01T00:00:00.000+00:...	2020	6	1	0	0	
11	VEM00080457	Saint Lucia	2020-07-01T00:00:00.000+00:...	2020	7	1	0	0	
12	VEM00080410	Saint Lucia	2020-02-01T00:00:00.000+00:...	2020	2	1	0	0	
13	VEM00080410	Saint Lucia	2020-03-01T00:00:00.000+00:...	2020	3	1	0	0	
14	VEM00080410	Saint Lucia	2020-04-01T00:00:00.000+00:...	2020	4	1	0	0	

3 days ago (1s) 5

```
energy_df.display()
```

(1) Spark Jobs

Table +

	Entity	Year	Access to electricity (% of population)	Access to clean fuels for cooking	Renewable-electricity-generating-cap
1	Afghanistan	2000	1.613591	6.2	
2	Afghanistan	2001	4.074574	7.2	
3	Afghanistan	2002	9.409158	8.2	
4	Afghanistan	2003	14.738506	9.5	
5	Afghanistan	2004	20.064968	10.9	
6	Afghanistan	2005	25.390894	12.2	
7	Afghanistan	2006	30.71869	13.85	
8	Afghanistan	2007	36.05101	15.3	
9	Afghanistan	2008	42.4	16.7	
10	Afghanistan	2009	46.74005	18.4	
11	Afghanistan	2010	42.7	20	
12	Afghanistan	2011	43.22202	21.8	
13	Afghanistan	2012	69.1	23	
14	Afghanistan	2013	68.29065	24.8	

3,649 rows | 0.81 seconds runtime Refreshed 3 days ago

3 days ago (<1s) 6

```
population_df.display()
```

(1) Spark Jobs

Table +

	Country (or dependency)	Population (2020)	Yearly Change	Net Change	Density (P/Km²)	Land Area (Km²)
1	China	1440297825	0.39 %	5540090	153	9388211
2	India	1382345085	0.99 %	13586631	464	2973190
3	United States	331341050	0.59 %	1937734	36	9147420
4	Indonesia	274021604	1.07 %	2898047	151	1811570
5	Pakistan	221612785	2.00 %	4327022	287	770880
6	Brazil	212821986	0.72 %	1509890	25	8358140
7	Nigeria	206984347	2.58 %	5175990	226	910770
8	Bangladesh	164972348	1.01 %	1643222	1265	130170
9	Russia	145945524	0.04 %	62206	9	16376870
10	Mexico	129166028	1.06 %	1357224	66	1943950
11	Japan	126407422	-0.30 %	-383840	347	364555
12	Ethiopia	115434444	2.57 %	2884858	115	1000000
13	Philippines	109830324	1.35 %	1464463	368	298170
14	Egypt	102659126	1.94 %	1946331	103	995450

235 rows | 0.48 seconds runtime Refreshed 3 days ago

▶ ✓ 3 days ago (1s) 12

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

# Define individual transformations as reusable functions
def replace_nulls(df: DataFrame, column: str, replacement: str) -> DataFrame:
    """
    Replace null values in a specific column with a given replacement.

    Args:
        df (DataFrame): Input DataFrame.
        column (str): Column name to replace nulls.
        replacement (str): Value to replace nulls with.

    Returns:
        DataFrame: Updated DataFrame.
    """
    return df.fillna({column: replacement})
```

```python
def validate_lat_long(df: DataFrame) -> DataFrame:
    """
    Ensure latitude and longitude values are within valid ranges.

    Args:
        df (DataFrame): Input DataFrame.

    Returns:
        DataFrame: Updated DataFrame with valid latitudes and longitudes.
    """
    return (
        df.withColumn(
            "LATITUDE",
            when((col("LATITUDE") >= -90) & (col("LATITUDE") <= 90), col("LATITUDE")).otherwise(None)
        )
        .withColumn(
            "LONGITUDE",
            when((col("LONGITUDE") >= -180) & (col("LONGITUDE") <= 180), col("LONGITUDE")).otherwise(None)
        )
    )
```

```python
def ensure_temperature_consistency(df: DataFrame) -> DataFrame:
    """
    Ensure that TMIN ≤ TAVG ≤ TMAX and handle nulls in TAVG appropriately.

    Args:
        df (DataFrame): Input DataFrame.

    Returns:
        DataFrame: DataFrame with consistent temperature values.
    """
    return df.withColumn(
        "TAVG",
        when(
            (col("TAVG") >= col("TMIN")) & (col("TAVG") <= col("TMAX")), col("TAVG")
        )
        .when(
            col("TAVG").isNull() & col("TMIN").isNotNull() & col("TMAX").isNotNull(),
            (col("TMIN") + col("TMAX")) / 2
        )
        .otherwise(col("TAVG"))  # Retain the original TAVG for other cases
    )
```

```python
def drop_duplicates(df: DataFrame, subset: list) -> DataFrame:
    """
    Drop duplicate rows based on a subset of columns.

    Args:
        df (DataFrame): Input DataFrame.
        subset (list): List of columns to consider for duplicates.

    Returns:
        DataFrame: DataFrame without duplicates.
    """
    return df.dropDuplicates(subset)


# Main transformation pipeline
weather_silver_df = weather_df.transform(replace_nulls, column="Country Region", replacement="Unknown") \
    .transform(validate_lat_long) \
    .transform(ensure_temperature_consistency) \
    .transform(drop_duplicates, subset=["STATION", "DATE"])

weather_silver_df = weather_silver_df.withColumnRenamed("Country Region", "Country_Region")
```

energy_df

3 days ago (<1s) 16

```
energy_df.printSchema()
```

```
root
 |-- Entity: string (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Access to electricity (% of population): double (nullable = true)
 |-- Access to clean fuels for cooking: double (nullable = true)
 |-- Renewable-electricity-generating-capacity-per-capita: double (nullable = true)
 |-- Financial flows to developing countries (US $): long (nullable = true)
 |-- Renewable energy share in the total final energy consumption (%): double (nullable = true)
 |-- Electricity from fossil fuels (TWh): double (nullable = true)
 |-- Electricity from nuclear (TWh): double (nullable = true)
 |-- Electricity from renewables (TWh): double (nullable = true)
 |-- Low-carbon electricity (% electricity): double (nullable = true)
 |-- Primary energy consumption per capita (kWh/person): double (nullable = true)
 |-- Energy intensity level of primary energy (MJ/$2017 PPP GDP): double (nullable = true)
 |-- Value_co2_emissions_kt_by_country: double (nullable = true)
 |-- Renewables (% equivalent primary energy): double (nullable = true)
 |-- gdp_growth: double (nullable = true)
 |-- gdp_per_capita: double (nullable = true)
 |-- Density\n(P/Km2): string (nullable = true)
 |-- Land Area(Km2): integer (nullable = true)
 |-- Latitude: double (nullable = true)
```

3 days ago (2s) 20 Python

```
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType

# Rename columns
energy_silver_df = energy_df \
    .withColumnRenamed("Access to electricity (% of population)", "access_to_electricity_percent") \
    .withColumnRenamed("Access to clean fuels for cooking", "access_to_clean_fuels") \
    .withColumnRenamed("Renewable-electricity-generating-capacity-per-capita", "renewable_electricity_capacity_per_capita") \
    .withColumnRenamed("Density\\n(P/Km2)", "density_p_km2") \
    .withColumnRenamed("Land Area(Km2)", "land_area_km2") \
    .withColumnRenamed("Renewables (% equivalent primary energy)", "Renewables_percent_equivalent_primary_energy") \
    .withColumnRenamed("Electricity from fossil fuels (TWh)", "electricity_from_fossil_fuels_TWh") \
    .withColumnRenamed("Renewable energy share in the total final energy consumption (%)",
    "renewable_energy_share_in_total_final_energy_consumption") \
    .withColumnRenamed("Value_co2_emissions_kt_by_country", "co2_emissions_kt") \
    .withColumnRenamed("Financial flows to developing countries (US $)", "financial_flows_to_developing_countries_USD") \
    .withColumnRenamed("Energy intensity level of primary energy (MJ/$2017 PPP GDP)", "energy_intensity_mj_per_ppp_gdp") \
    .withColumnRenamed("Electricity from nuclear (TWh)", "electricity_from_nuclear_TWh") \
    .withColumnRenamed("Electricity from renewables (TWh)", "electricity_from_renewables_TWh") \
    .withColumnRenamed("Low-carbon electricity (% electricity)", "low_carbon_electricity_percent") \
    .withColumnRenamed("Primary energy consumption per capita (kWh/person)", "primary_energy_consumption_per_capita_kWh_per_person")

# Replace invalid values (e.g., "NA", "null") with None
energy_silver_df = energy_silver_df.replace(["NA", "null"], None)
```

3 days ago (2s) 20

```
# Convert density to numeric
energy_silver_df = energy_silver_df.withColumn("density_p_km2", F.col("density_p_km2").cast(DoubleType()))

# Fill missing values for specific columns with their mean
mean_values = energy_silver_df.agg(*[
    F.mean(F.col(c)).alias(c) for c in [
        "access_to_electricity_percent",
        "access_to_clean_fuels",
        "renewable_electricity_capacity_per_capita",
        "financial_flows_to_developing_countries_USD",
        "renewable_energy_share_in_total_final_energy_consumption",
        "electricity_from_fossil_fuels_TWh",
        "electricity_from_nuclear_TWh",
        "electricity_from_renewables_TWh",
        "low_carbon_electricity_percent",
        "primary_energy_consumption_per_capita_kWh_per_person",
        "energy_intensity_mj_per_ppp_gdp",
        "co2_emissions_kt",
        "Renewables_percent_equivalent_primary_energy",
        "gdp_growth",
        "gdp_per_capita",
        "density_p_km2",
        "land_area_km2",
        "Latitude",
        "Longitude"
    ]
]).first()
```

```
energy_silver_df = energy_silver_df.fillna(mean_values.asDict())

# Handle missing values in derived column inputs before creating it
energy_silver_df = energy_silver_df.fillna({
    "electricity_from_nuclear_TWh": mean_values["electricity_from_nuclear_TWh"],
    "electricity_from_renewables_TWh": mean_values["electricity_from_renewables_TWh"]
})

# Filter rows with valid year range
energy_silver_df = energy_silver_df.filter((F.col("Year") >= 2000) & (F.col("Year") <= 2020))

# Remove duplicates and sort by Entity and Year
energy_silver_df = energy_silver_df.dropDuplicates().orderBy("Entity", "Year")
```

▸ (2) Spark Jobs

▸ energy_silver_df: pyspark.sql.dataframe.DataFrame = [Entity: string, Year: integer ... 19 more fields]

population_df

✓ 3 days ago (<1s)

```
population_df.printSchema()
```

```
root
 |-- Country (or dependency): string (nullable = true)
 |-- Population (2020): integer (nullable = true)
 |-- Yearly Change: string (nullable = true)
 |-- Net Change: integer (nullable = true)
 |-- Density (P/Km²): integer (nullable = true)
 |-- Land Area (Km²): integer (nullable = true)
 |-- Migrants (net): double (nullable = true)
 |-- Fert. Rate: string (nullable = true)
 |-- Med. Age: string (nullable = true)
 |-- Urban Pop %: string (nullable = true)
 |-- World Share: string (nullable = true)
```

▶ ✓ 3 days ago (1s) 26

```
from pyspark.sql import functions as F
from pyspark.sql.types import FloatType, IntegerType

# Step 1: Create a copy of the DataFrame
population_silver_df = population_df

# Step 2: Standardize Column Names
population_silver_df = population_silver_df.toDF(
    "country_or_dependency",
    "population_2020",
    "yearly_change",
    "net_change",
    "density_p_km2",
    "land_area_km2",
    "migrants_net",
    "fertility_rate",
    "median_age",
    "urban_population_percent",
    "world_share"
)
```

```
# Step 3: Handle Missing Values
# Replace "N.A." and other invalid values with None
population_silver_df = population_silver_df.replace(["N.A.", "NA", ""], None)

# Step 4: Convert Columns to Correct Data Types
# Remove '%' and convert percentage columns to float
population_silver_df = population_silver_df.withColumn(
    "yearly_change",
    F.regexp_replace("yearly_change", "%", "").cast(FloatType()) / 100
).withColumn(
    "urban_population_percent",
    F.regexp_replace("urban_population_percent", "%", "").cast(FloatType())
).withColumn(
    "world_share",
    F.regexp_replace("world_share", "%", "").cast(FloatType()) / 100
)
```

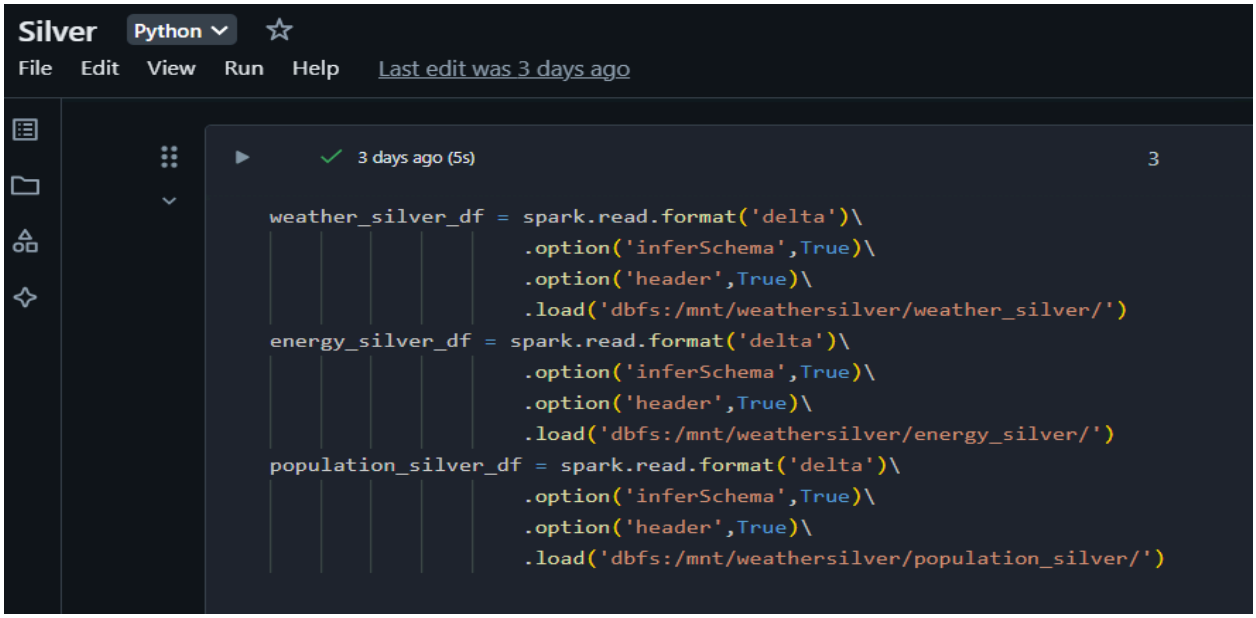
```
# Convert other columns to numeric, handling invalid values
population_silver_df = population_silver_df.withColumn(
    "fertility_rate", F.when(F.col("fertility_rate").rlike("^\d+(\.\d+)?$"), F.col("fertility_rate")).otherwise(None).cast(FloatType())
).withColumn(
    "median_age", F.when(F.col("median_age").rlike("^\d+$"), F.col("median_age")).otherwise(None).cast(IntegerType())
).withColumn(
    "migrants_net", F.when(F.col("migrants_net").rlike("^\d+(\.\d+)?$"), F.col("migrants_net")).otherwise(None).cast(FloatType())  # Changed
    to FloatType
)

# Step 5: Handle Outliers for Migrants Net (Optional)
population_silver_df = population_silver_df.withColumn(
    "migrants_net",
    F.when((F.col("migrants_net") > 1e6) | (F.col("migrants_net") < -1e6), None).otherwise(F.col("migrants_net"))
)
```

```python
# Step 6: Fill missing values with default values (for numeric columns)
population_silver_df = population_silver_df.fillna({
    "yearly_change": 0.0,
    "urban_population_percent": 0.0,
    "world_share": 0.0,
    "fertility_rate": 0.0,
    "median_age": 0,
    "migrants_net": 0.0,
    "population_2020": 0,
    "net_change": 0,
    "density_p_km2": 0,
    "land_area_km2": 0
})

# Step 7: Display the cleaned DataFrame
population_silver_df.display()
```

Writing All DF to Silver Container

Silver Layer

- **Objective**: Apply data cleansing, transformation, and validation to enhance data quality.
- **Key Operations**:
  - **Weather Data Transformations**:
    - Replaced null values in critical columns (e.g., "Country Region") with defaults like "Unknown".
    - Validated latitude and longitude ranges to ensure correctness.
    - Ensured consistency in temperature data (`TMIN ≤ TAVG ≤ TMAX`) and calculated missing `TAVG` values when possible.
    - Dropped duplicate rows based on key columns (`STATION`, `DATE`).
  - **Energy Data Transformations**:
    - Renamed columns for better readability (e.g., `Access to electricity (% of population)` → `access_to_electricity_percent`).
    - Replaced invalid values (e.g., "NA") with `None`.
    - Filled missing values with column means for numerical fields.
    - Filtered data to retain only records from 2000–2020.
    - Removed duplicates and sorted by `Entity` and `Year`.
  - **Population Data Transformations**:
    - Standardized column names (e.g., `Country_or_dependency`, `Population_2020`).
    - Replaced invalid values (e.g., "N.A.") with `None`.
    - Converted percentage-based columns to float after removing `%`.
    - Ensured correct data types for numerical fields.
- **Validation**:
  - Checked for null values in transformed DataFrames.
  - Verified schemas and displayed sample records to ensure data integrity.




```
# Step 3: Join the result with population data
final_df = weather_energy_df.join(
    population_silver_df,
    weather_energy_df["Country"] == population_silver_df["Country"],
    how="inner"
).drop(population_silver_df["Country"])  # Drop duplicate column after join

# Display the result
display(final_df)
```

3 days ago (<1s) 15

```
final_df.printSchema()
```

```
root
 |-- STATION: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- DATE: timestamp (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Month: integer (nullable = true)
 |-- Day: integer (nullable = true)
 |-- PRCP: double (nullable = true)
 |-- SNWD: double (nullable = true)
 |-- TAVG: double (nullable = true)
 |-- TMAX: double (nullable = true)
 |-- TMIN: double (nullable = true)
 |-- SNOW: integer (nullable = true)
 |-- Weather_Latitude: double (nullable = true)
 |-- Weather_Longitude: double (nullable = true)
 |-- ELEVATION: double (nullable = true)
 |-- DAPR: integer (nullable = true)
 |-- MDPR: double (nullable = true)
 |-- WESD: double (nullable = true)
 |-- access_to_electricity_percent: double (nullable = true)
 |-- access to clean fuels: double (nullable = true)
```

Enriching the Dataset

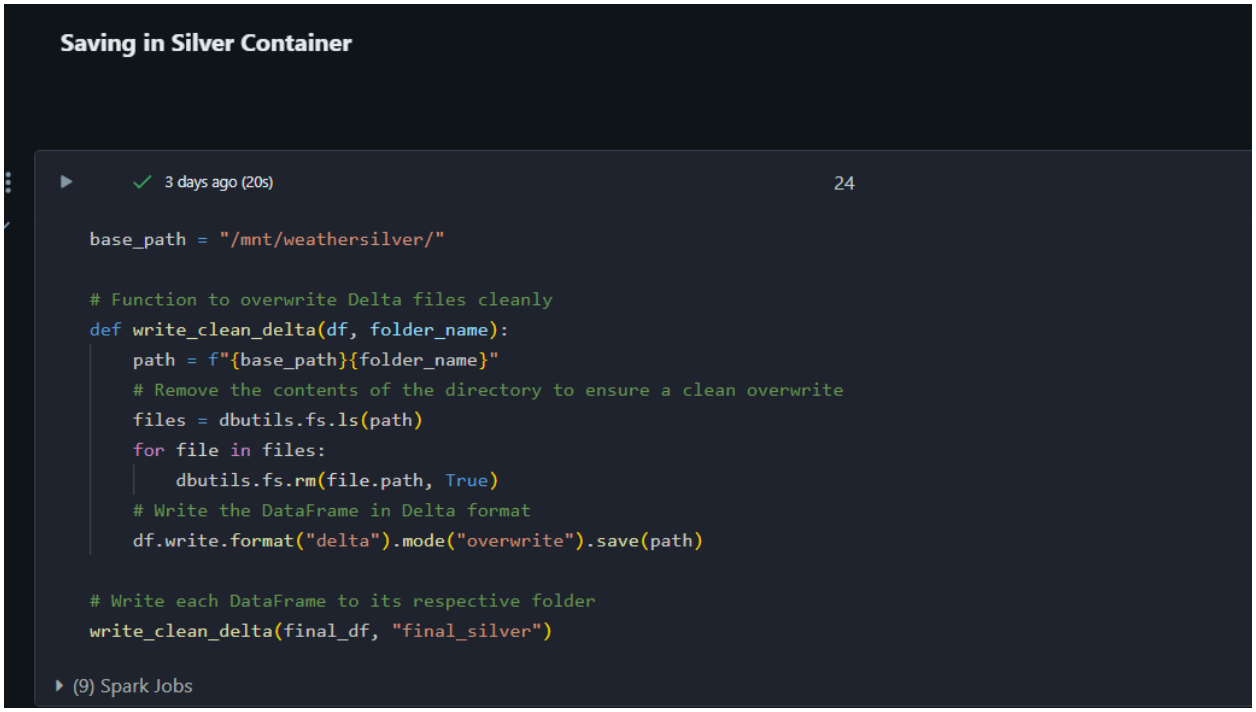
3 days ago (13s) 22

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# Step 1: Data Cleansing and Preparation
# 1.1 Remove duplicate records
final_df = final_df.dropDuplicates()

# 1.2 Handle remaining null values
# Replace nulls with default values or drop rows, depending on the column
final_df = final_df.fillna({
    'TAVG': 0,  # Replace nulls in TAVG (average temperature) column with 0
    'renewable_electricity_capacity_per_capita': 0,  # Replace nulls in renewable capacity with 0
    'electricity_from_fossil_fuels_TWh': 0,  # Replace nulls in TotalCapacity with 0
    'co2_emissions_kt': 0  # Replace nulls in CO2Emissions with 0
}).dropna(subset=['Country', 'Year', 'Month'])  # Drop rows where critical fields are null
```

```python
# 1.3 Create calculated columns
# Add 'Season' column based on Month (considering hemispheric differences)
def get_season(month):
    if month in [12, 1, 2]:
        return "Winter"
    elif month in [3, 4, 5]:
        return "Spring"
    elif month in [6, 7, 8]:
        return "Summer"
    elif month in [9, 10, 11]:
        return "Fall"

get_season_udf = F.udf(get_season)
final_df = final_df.withColumn("Season", get_season_udf(F.col("Month")))

# Add 'Temperature Classification' column
final_df = final_df.withColumn(
    "TemperatureClassification",
    F.when(F.col("TAVG") < 10, "Cold")
     .when((F.col("TAVG") >= 10) & (F.col("TAVG") < 25), "Moderate")
     .otherwise("Hot")
)
```

```python
# Add 'Renewable Energy Performance Index'
# Formula: (RenewableCapacity / TotalCapacity) * 100
final_df = final_df.withColumn(
    "RenewableEnergyPerformanceIndex",
    F.when(F.col("electricity_from_fossil_fuels_TWh") > 0,
           (F.col("renewable_electricity_capacity_per_capita") / F.col("electricity_from_fossil_fuels_TWh")) * 100)
     .otherwise(0)
)

final_df.display()
```

Saving in Silver Container



```
✓ 3 days ago (20s)                                                    24

base_path = "/mnt/weathersilver/"

# Function to overwrite Delta files cleanly
def write_clean_delta(df, folder_name):
    path = f"{base_path}{folder_name}"
    # Remove the contents of the directory to ensure a clean overwrite
    files = dbutils.fs.ls(path)
    for file in files:
        dbutils.fs.rm(file.path, True)
    # Write the DataFrame in Delta format
    df.write.format("delta").mode("overwrite").save(path)

# Write each DataFrame to its respective folder
write_clean_delta(final_df, "final_silver")

▸ (9) Spark Jobs
```

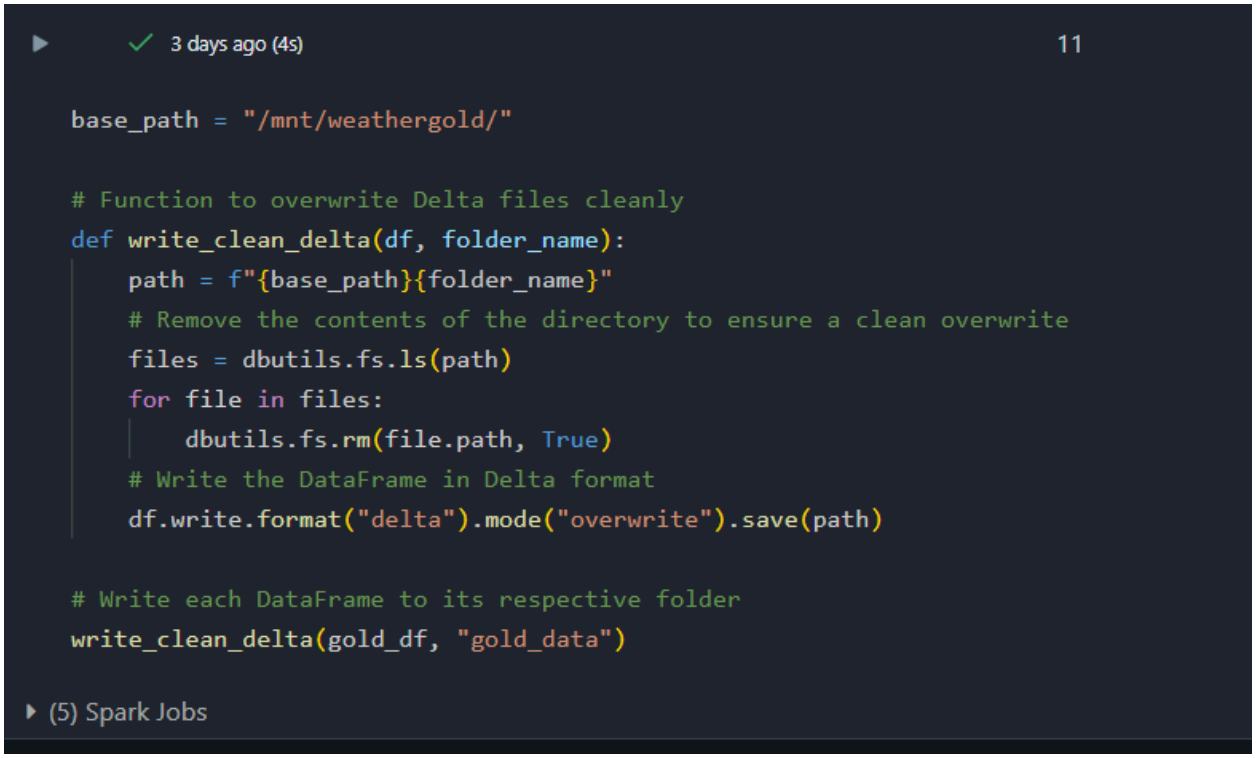
Gold Layer

- **Objective**: Generate optimized datasets tailored for specific analytical requirements.
- **Highlights**:
  - Joined cleaned datasets (Weather, Energy, and Population) based on shared keys (e.g., country, year).
  - Derived new metrics and calculated aggregations as needed.
  - Ensured the Gold layer was optimized for performance and ready for use in downstream analytics.

```
Gold  Python ☆
File  Edit  View  Run  Help  Last edit was 3 days ago        ▸ Run all  Terminated  Schedu

✓ 3 days ago (1s)                                                     7

import pyspark.sql.functions as F

# 1. **Data Aggregation and Enrichment**
# Group by dimensions like Country, Year, and Season and aggregate relevant statistics.

aggregated_df = final_silver_df.groupBy("Country", "Year", "Season").agg(
    F.avg("TAVG").alias("avg_temp"),
    F.avg("PRCP").alias("avg_precipitation"),
    F.avg("SNWD").alias("avg_snow_depth"),
    F.sum("co2_emissions_kt").alias("total_co2_emissions"),
    F.avg("renewable_energy_share_in_total_final_energy_consumption").alias("avg_renewable_share"),
    F.avg("gdp_growth").alias("avg_gdp_growth"),
    F.sum("financial_flows_to_developing_countries_USD").alias("total_financial_flows"),
    F.avg("gdp_per_capita").alias("avg_gdp_per_capita"),  # Include GDP per capita
    F.avg("primary_energy_consumption_per_capita_kWh_per_person").alias("avg_energy_consumption_per_capita")  # Include energy consumption
)
```

```python
# 2. **Feature Engineering**
# Calculate GDP per capita growth rate
gold_df = aggregated_df.withColumn(
    "gdp_per_capita_growth_rate",
    (F.col("avg_gdp_growth") * F.col("avg_gdp_per_capita")) / 100
)

# Calculate renewable energy efficiency: renewable energy share vs total energy consumption
gold_df = gold_df.withColumn(
    "renewable_energy_efficiency",
    F.col("avg_renewable_share") / F.col("avg_energy_consumption_per_capita")
)

# Example of creating a weather severity index combining TAVG, PRCP, and SNWD
gold_df = gold_df.withColumn(
    "weather_severity_index",
    F.when(F.col("avg_temp") < 0, 1).otherwise(0) +  # Cold weather
    F.when(F.col("avg_precipitation") > 50, 1).otherwise(0) +  # High precipitation
    F.when(F.col("avg_snow_depth") > 100, 1).otherwise(0)  # Snow depth
)
```

```python
# 3. **Final Cleaning and Transformation**
# Handle null values, type casting, and drop unnecessary columns
gold_df = gold_df.fillna({
    "avg_temp": 0,
    "avg_precipitation": 0,
    "avg_snow_depth": 0,
    "total_co2_emissions": 0,
    "avg_renewable_share": 0,
    "avg_gdp_growth": 0,
    "total_financial_flows": 0,
    "gdp_per_capita_growth_rate": 0,
    "renewable_energy_efficiency": 0,
    "weather_severity_index": 0
})

# Display the transformed DataFrame
display(gold_df)
```

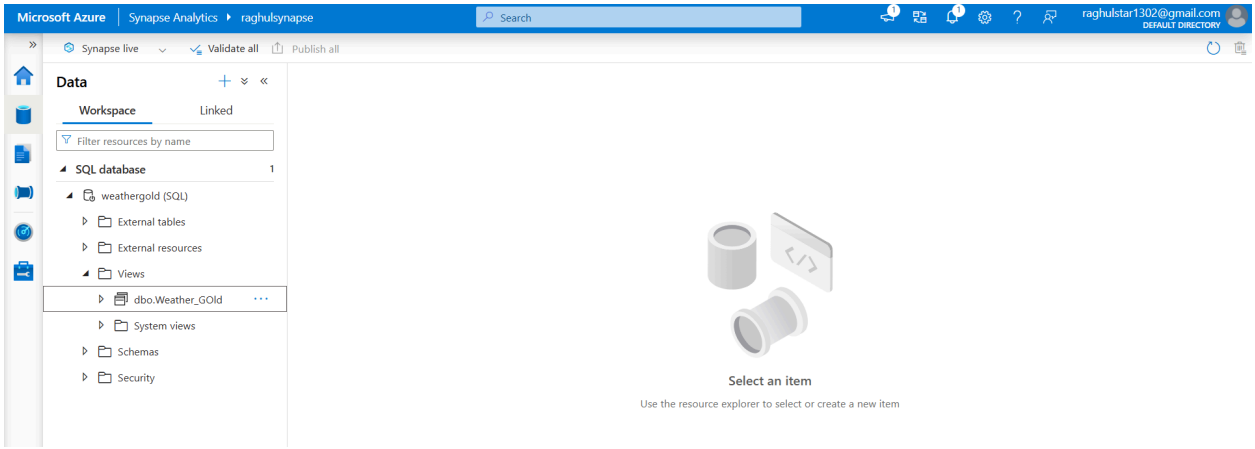


This transformation pipeline ensures that raw data from disparate sources is cleansed, validated, and structured for analytical use, adhering to best practices in modern data engineering.

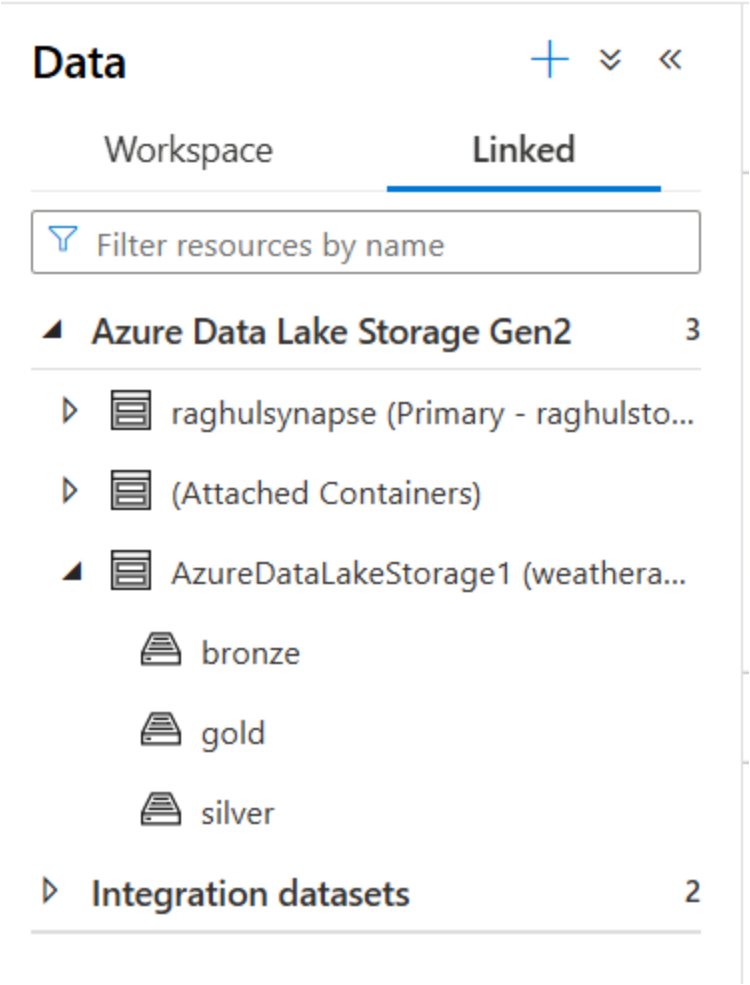
Data Loading and Integration

For efficient data loading and integration, **Azure Synapse Analytics** was utilized. The process involved the following steps:

1. **Database Creation**:
   A database was created within Azure Synapse Analytics to serve as the central repository for the project's data.



2. **Integration with ADLS Gen2**:
   The data pipeline leveraged **Azure Data Lake Storage Gen2 (ADLS Gen2)** as the primary data source. The gold-level file, containing the cleaned and transformed data, was stored in ADLS Gen2.

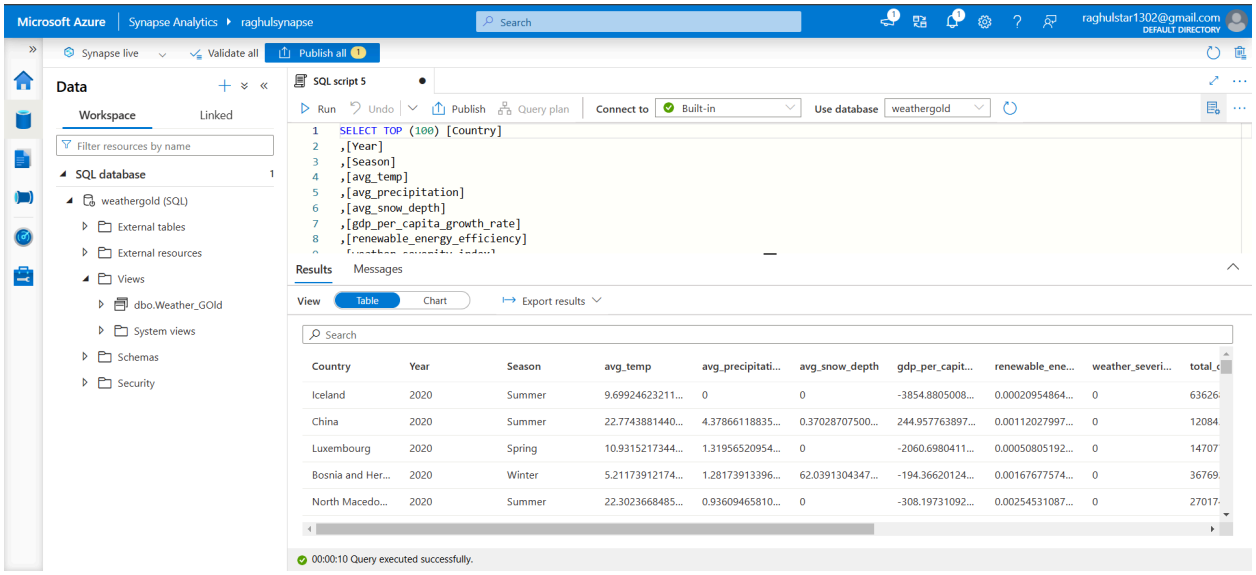


3. **View Creation for Gold-Level Data**:
    - A **view** was created within the Synapse Analytics database to represent the final gold-level data.
    - This view serves as an abstraction layer, enabling streamlined querying and analysis without directly exposing the underlying raw data.
    - The view aggregates, filters, and organizes the data from the gold file stored in ADLS Gen2, ensuring that it is ready for downstream analytics and reporting.

SQL script 5 ● | SQL script 2 | SQL script 3 | SQL script 4 ×

Run | Undo | Publish | Query plan | Connect to: Built-in | Use databas

```sql
-- This is auto-generated code
CREATE VIEW Weather_GOld AS
SELECT
    *
FROM
    OPENROWSET(
        BULK 'https://weatheradls.dfs.core.windows.net/gold/gold_data/',
        FORMAT = 'DELTA'
    ) AS [result]

```



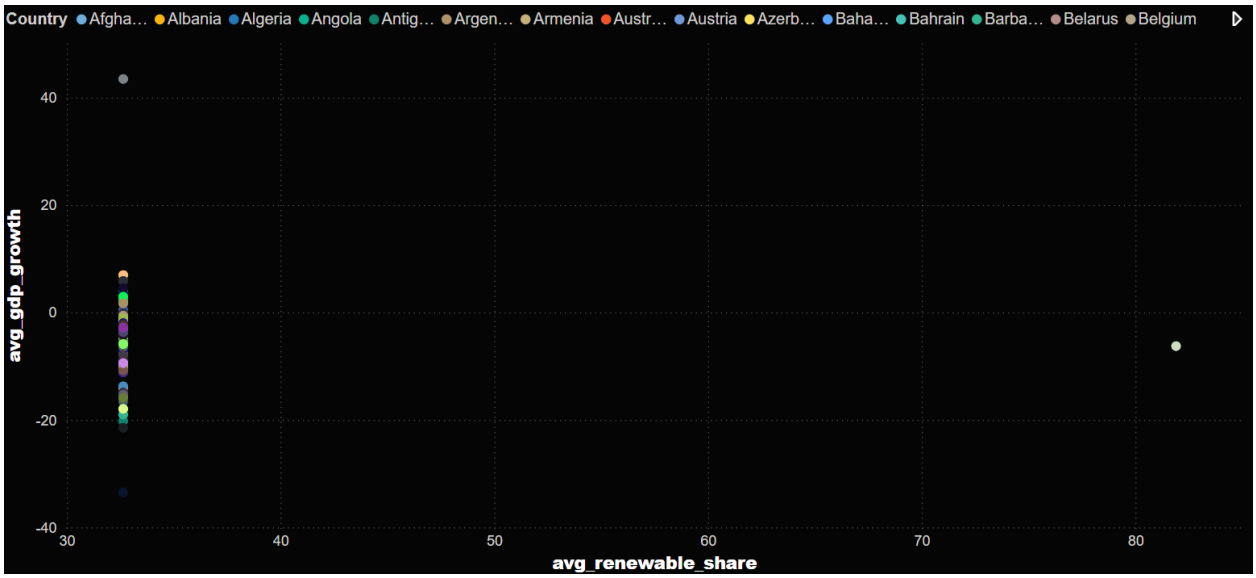
4. **Advantages of Using Azure Synapse Analytics**:
    - Unified analytics platform for large-scale data processing.
    - Seamless integration with ADLS Gen2, ensuring smooth data flow and management.
    - Optimized query performance for gold-level data, enhancing the efficiency of data consumption by downstream systems.

This approach ensures that the data loading process is both robust and scalable, meeting the project's analytical and reporting needs effectively.

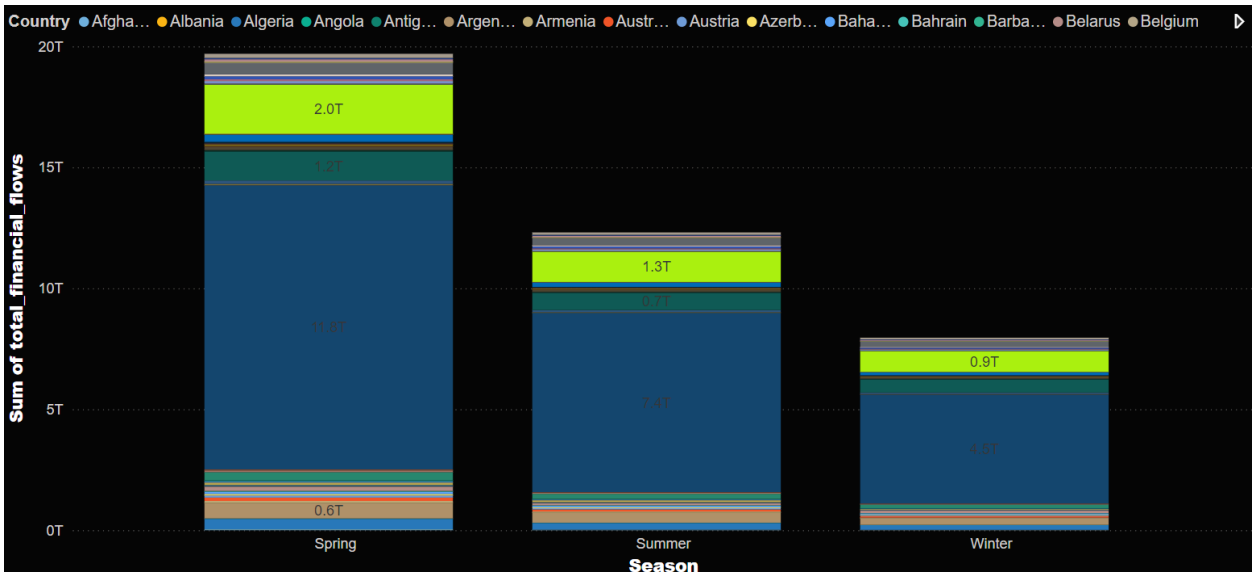
Reporting

To analyze and present insights from the processed data, **Power BI** was utilized for reporting and visualization. The process and outcomes are described below:

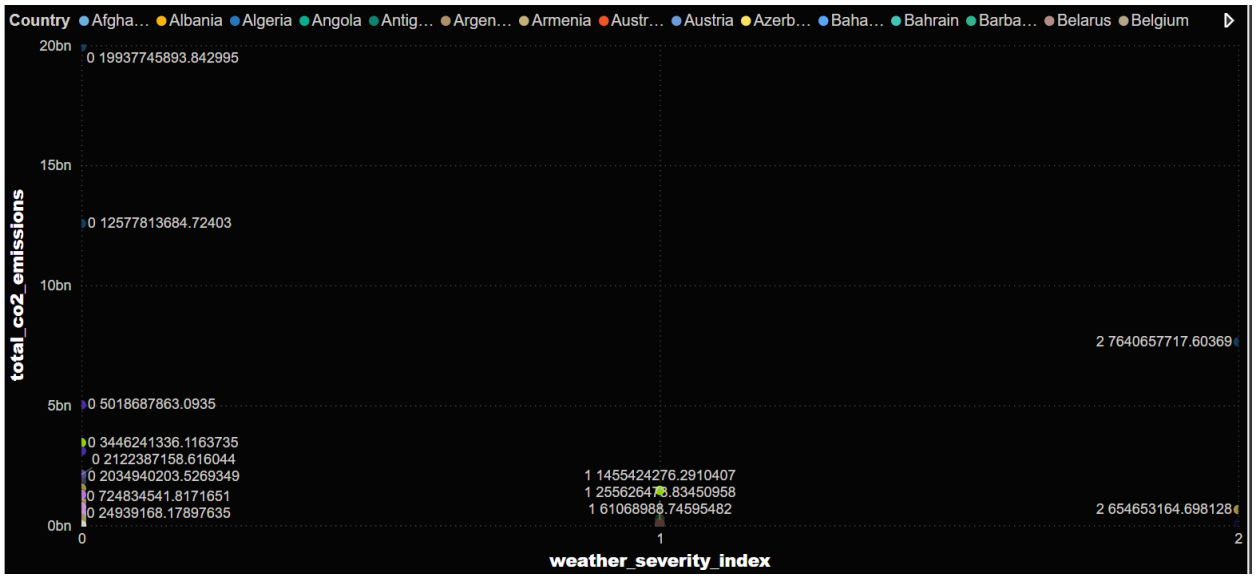
1. **Integration with Azure Synapse Analytics**:
    - Power BI was connected to **Azure Synapse Analytics** using Azure credentials.
    - Data from the Synapse database, including the gold-level data view, was seamlessly loaded into Power BI for visualization purposes.
2. **Visualization Development**:
   Various visualizations and charts were created in Power BI to uncover trends, patterns, and relationships within the data. These visualizations include:
    - **Renewable Energy Share vs. GDP Growth**: Explores the relationship between a country's renewable energy adoption and its economic growth.



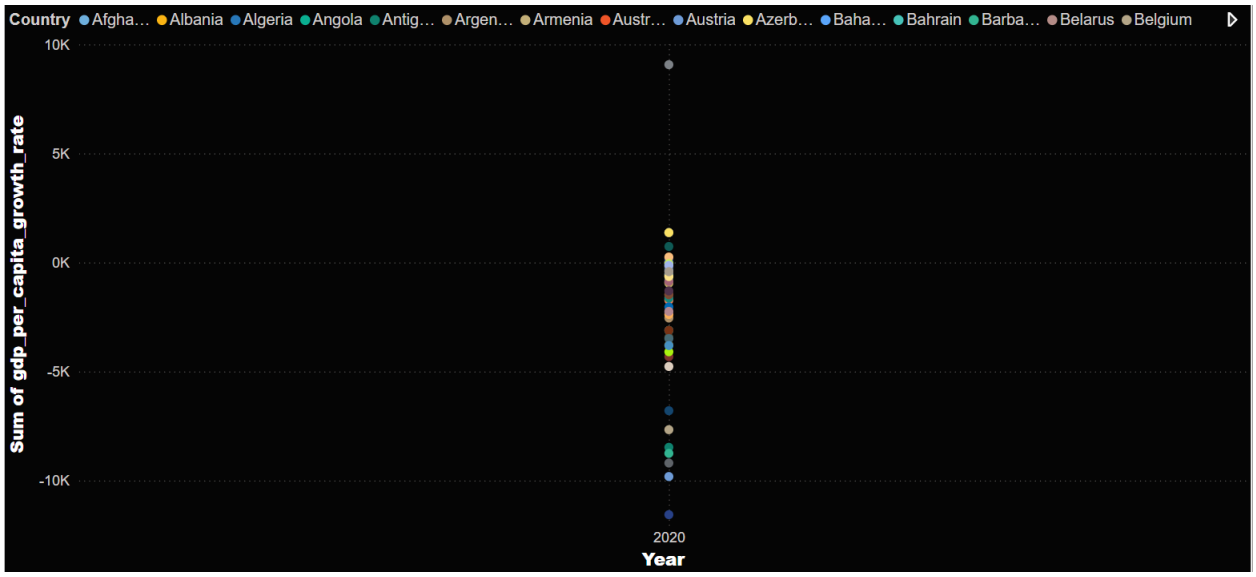
- **Seasonal Financial Flows**: Analyzes financial activity variations across different seasons.



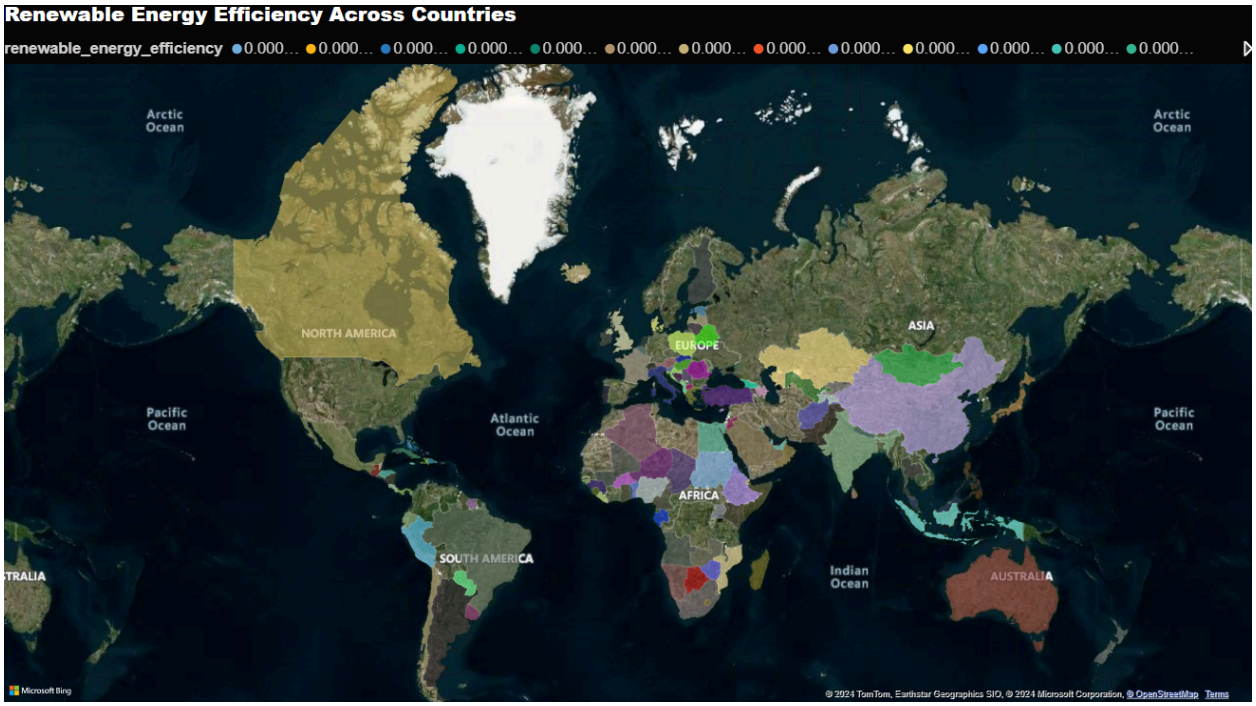
- **Weather Severity and CO_2 Emissions**: Examines how severe weather conditions correlate with carbon dioxide emissions.



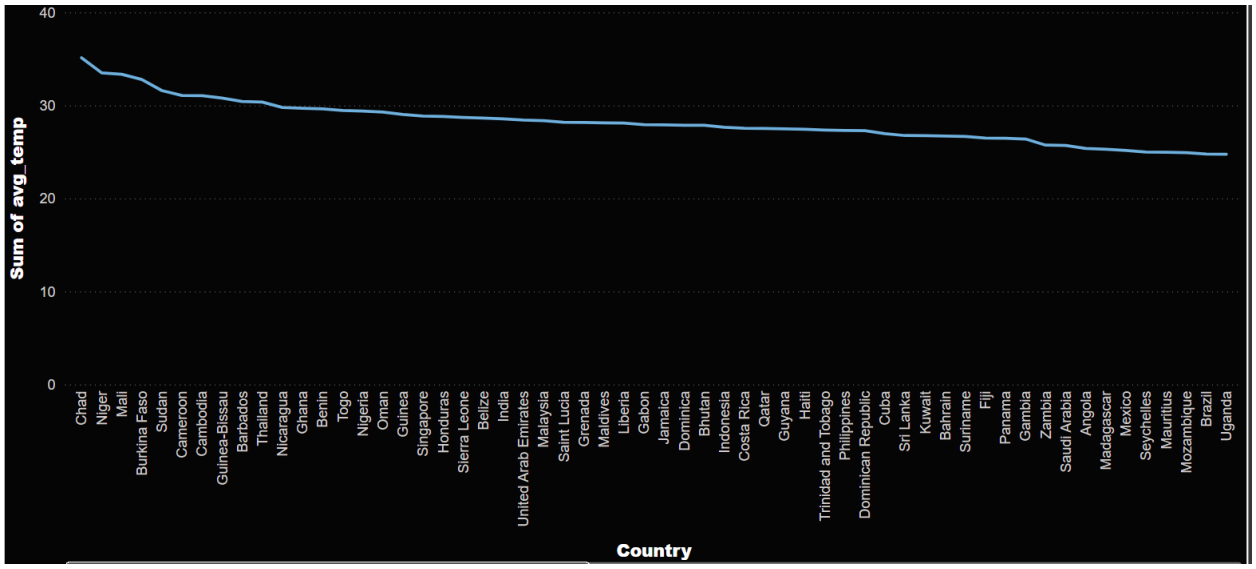
- **GDP Per Capita Growth Trends**: Tracks the growth trends in GDP per capita over time for different regions.



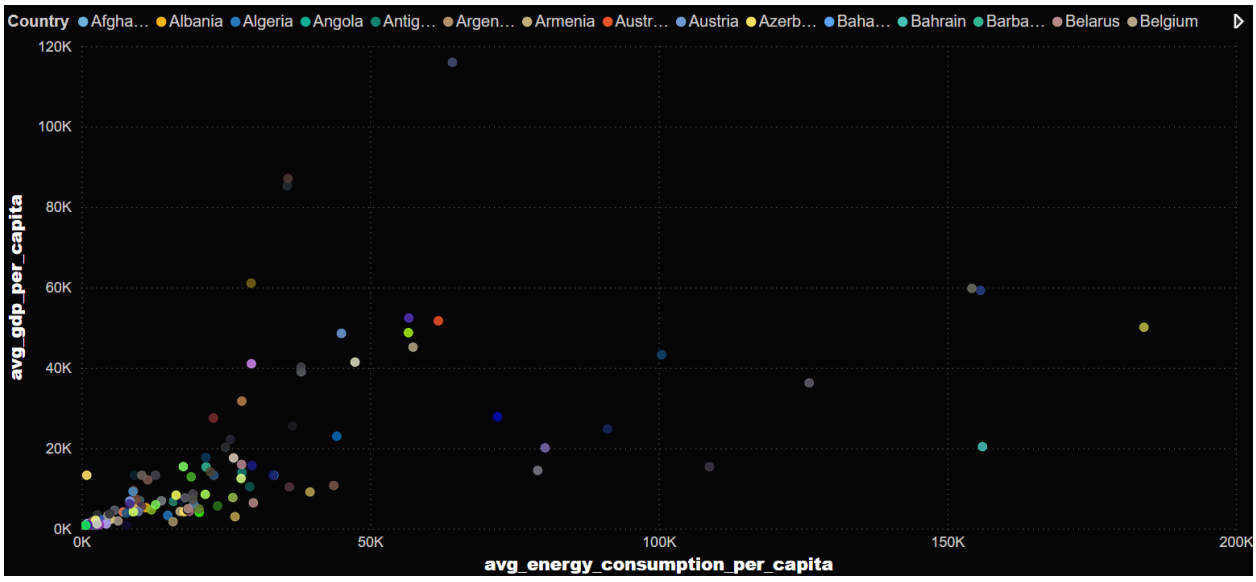
- **Renewable Energy Efficiency Across Countries**: Compares the efficiency of renewable energy utilization across nations.



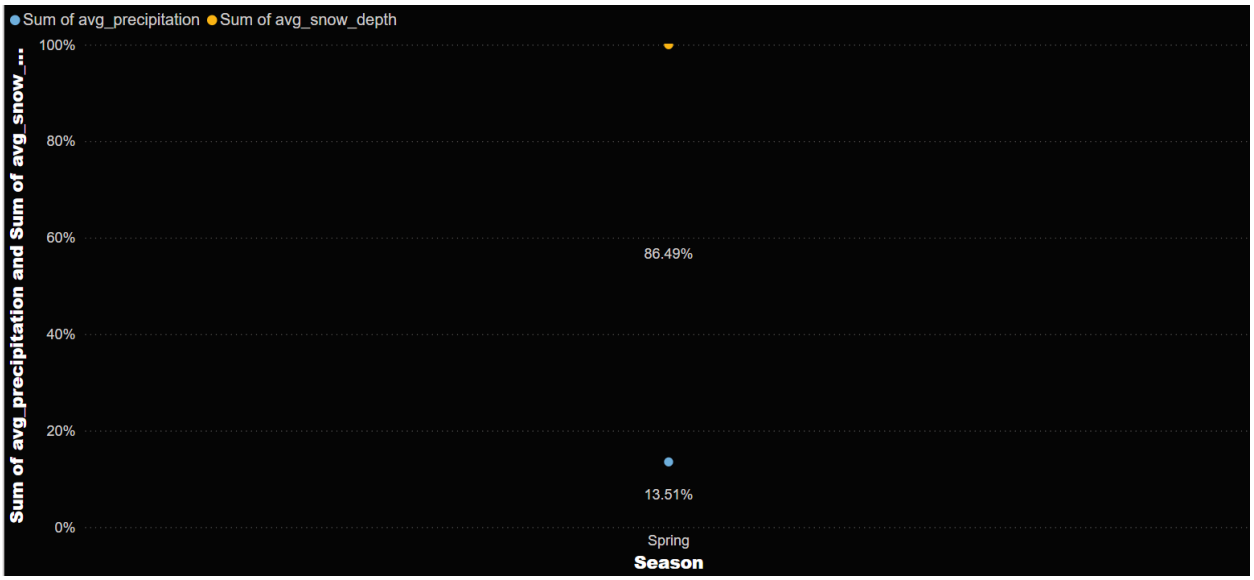
- **Temperature Analysis by Country and Season**: Provides insights into temperature variations across countries and seasons.



- **Correlation Between Energy Consumption and GDP Per Capita**: Investigates the relationship between a country's energy consumption and its GDP per capita.



- **Average Precipitation and Snow Depth by Season**: Analyzes seasonal precipitation and snow depth averages across regions.

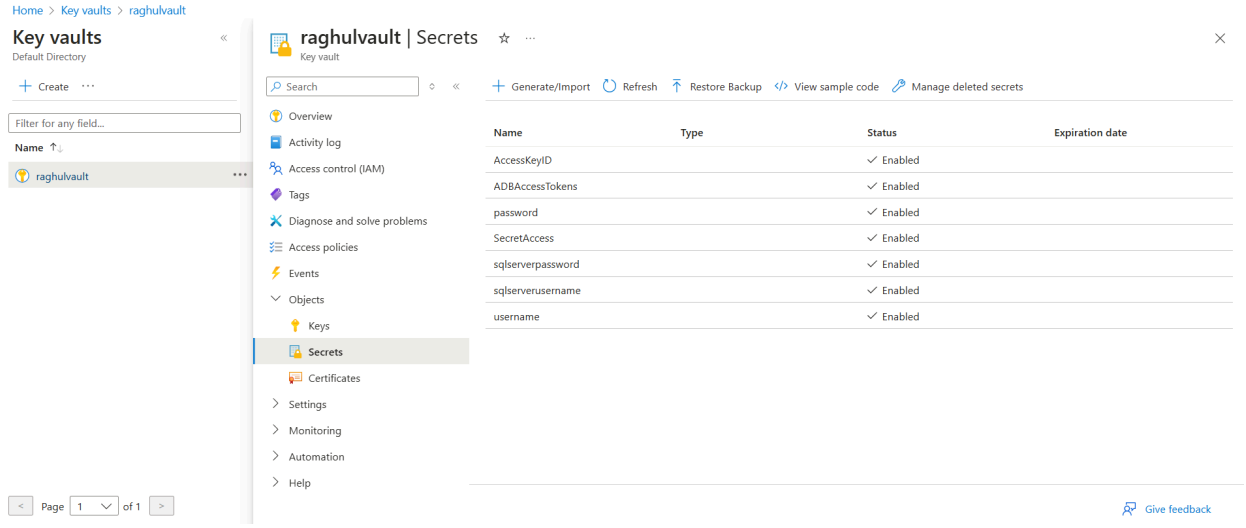


3. **Advantages of Using Power BI**:
    - Intuitive interface for creating rich, interactive visualizations.
    - Direct integration with Azure Synapse Analytics ensures real-time access to processed data.
    - Enables stakeholders to derive actionable insights and make data-driven decisions.

These visualizations played a key role in transforming raw data into meaningful insights, helping stakeholders better understand critical relationships and trends.

Security & Governance

For **Security and Governance** in my project, I utilized **Azure Key Vault** to securely store and manage sensitive information, such as API keys, database credentials, and other secrets. This ensures that sensitive data is protected and access is restricted to authorized users and services.



To enable collaboration and manage access control within the project, I leveraged **Azure Active Directory (AAD)**. I created an **Azure Active Directory group** and added my teammate to this group, granting her the necessary permissions. I provided her with access to the entire resource group, ensuring that both of us could collaborate effectively while maintaining proper access control and security.

This approach not only secured sensitive information but also streamlined the collaboration process by allowing both team members to work seamlessly on the project with appropriate access to resources and the ability to manage the infrastructure securely.

Conclusion

This project successfully demonstrates the end-to-end data engineering and analytics workflow, leveraging modern cloud technologies and visualization tools to derive actionable insights. By integrating **Azure Synapse Analytics** for data storage and processing, **ADLS Gen2** for scalable data management, and **Power BI** for interactive reporting, the project achieves its objectives of transforming raw data into valuable insights.

Key achievements include:

1. Efficient data ingestion and transformation pipelines to process raw datasets into gold-level data.
2. Seamless integration of Azure Synapse Analytics with Power BI for real-time data visualization and reporting.
3. Development of insightful visualizations such as **Renewable Energy Share vs. GDP Growth**, **Weather Severity and CO_2 Emissions**, and **Seasonal Financial Flows**, which provide stakeholders with a deeper understanding of trends and patterns.
4. Utilization of best practices in data engineering and cloud technology to ensure scalability, reliability, and performance.

This project not only highlights the importance of cloud-based data solutions but also showcases how data-driven decision-making can be enabled through effective reporting and analytics. It provides a strong foundation for future enhancements, including incorporating advanced analytics, predictive modeling, and further automation of the data lifecycle.

The successful completion of this project reflects a robust understanding of data engineering principles, cloud computing, and visualization techniques, underscoring the potential of modern data platforms in solving real-world business challenges.