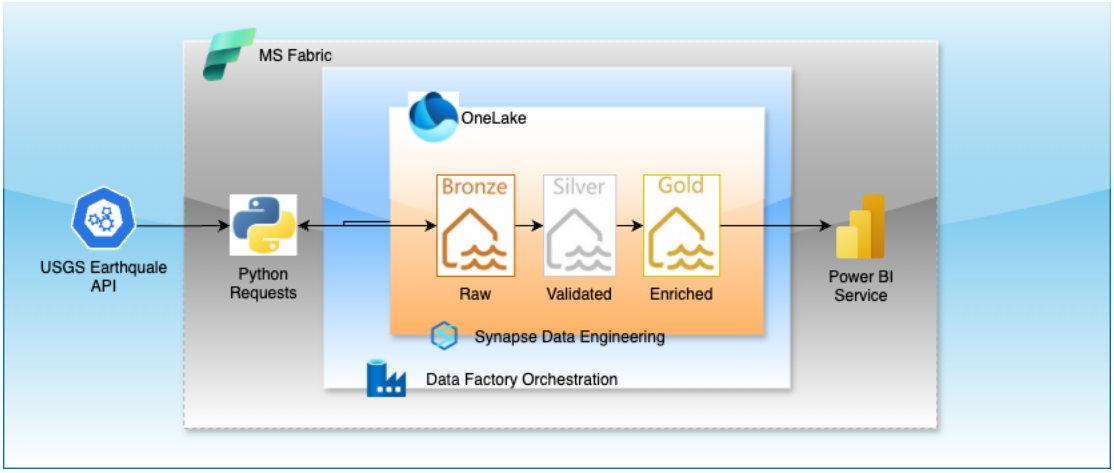


USGS Earthquake Data Pipeline

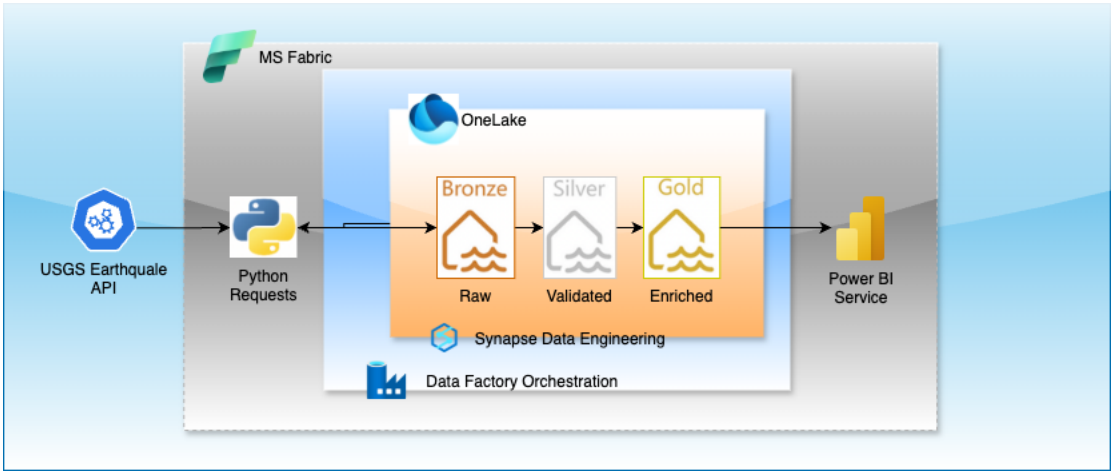
End-to-End Data Engineering Project

by: Randy A. Velasco

I. Objective

To develop and deploy an automated data pipeline to extract, transform, and visualize earthquake data from the USGS Earthquake API, showcasing real-time data processing and analytics.

II. Project Overview



This project demonstrates an end-to-end data engineering workflow that integrates data extraction, transformation, and visualization. By leveraging Microsoft Fabric, PySpark, and Power BI, the pipeline processes daily earthquake data and provides actionable insights through interactive dashboards.

III. Tools and Technologies

This project leverages a combination of cutting-edge tools and technologies to ensure efficient data handling, transformation, and visualization:

a. PySpark

PySpark is a robust library for distributed data processing in Python, built on top of Apache Spark. For this project, PySpark is utilized to perform scalable data transformations across multiple layers:

- Bronze Layer: Handles raw data ingestion from the USGS Earthquake API, preserving the original JSON format for audit and traceability.
- Silver Layer: Converts JSON data into structured tabular formats using PySpark's DataFrame APIs, enabling efficient querying and processing.

- Gold Layer: Applies advanced data manipulation techniques, such as column additions and categorization, ensuring enriched data ready for analysis.

PySpark's ability to manage large datasets and parallelize operations ensures the pipeline remains performant even as data volumes grow over time.

b. Microsoft Fabric

Microsoft Fabric is utilized for orchestrating the entire data pipeline, serving as the backbone of this end-to-end solution. Key features include:

- Pipeline Automation: Seamlessly schedules and executes the three PySpark notebooks in succession, ensuring daily updates without manual intervention.
- Integrated Workspace: Provides a unified environment for data engineering, storage, and analytics, streamlining project management and reducing operational complexity.
- Layered Architecture: Implements a Bronze-Silver-Gold data flow model to maintain data governance, enhance clarity, and ensure scalability.

By leveraging Microsoft Fabric, the pipeline benefits from cloud-native capabilities, enabling smooth integration with other Microsoft tools, like Power BI.

c. Power BI

Power BI is the visualization engine used to create an interactive dashboard that communicates insights effectively to stakeholders. Features include:

- Default Semantic Model: Automatically generated from the Gold Layer table, simplifying data relationships and enabling quick dashboard creation.
- Interactive Dashboards: Showcase earthquake data trends through map visualizations, column charts, and other statistical representations.
- Real-Time Insights: Allows users to monitor earthquake occurrences and distributions daily, supporting proactive decision-making.

Power BI’s flexibility and ability to integrate with Microsoft Fabric enhance the overall user experience, making it easier to derive insights from complex datasets.

IV. Workflow Details

Data Source:

The foundation of this project is the **USGS Earthquake API**, a reliable and well-documented source for accessing global earthquake event data. This API provides comprehensive information about earthquake occurrences, including their location, magnitude, depth, and time.

Key Features of the API:

- **Endpoint**: USGS Earthquake API
- **Data Format**: The API returns earthquake data in JSON format, which is ideal for processing and transformation in a data pipeline.
- **Filters and Parameters**: The API allows customization through query parameters, such as time range, magnitude thresholds, and geographic boundaries, to retrieve specific datasets. For this project, it is configured to fetch all earthquake events within the defined time range.

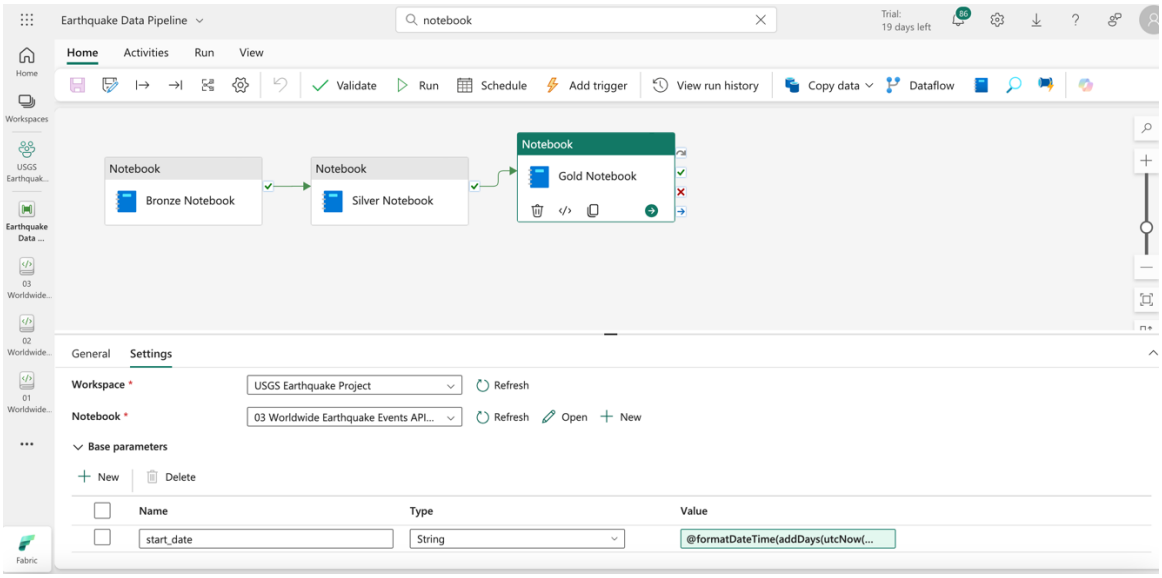
Update Frequency:

- The data is extracted once every day, ensuring the pipeline remains up-to-date with the latest earthquake events.
- The extraction process retrieves all earthquake data recorded between **midnight (00:00)** and **11:59 PM (23:59)** of the previous day.
- The retrieved data is appended to the existing dataset, creating a comprehensive and historical record of earthquake occurrences for analysis.

By using this API, the pipeline ensures near real-time data updates and maintains accuracy, allowing stakeholders to monitor earthquake trends and statistics effectively.

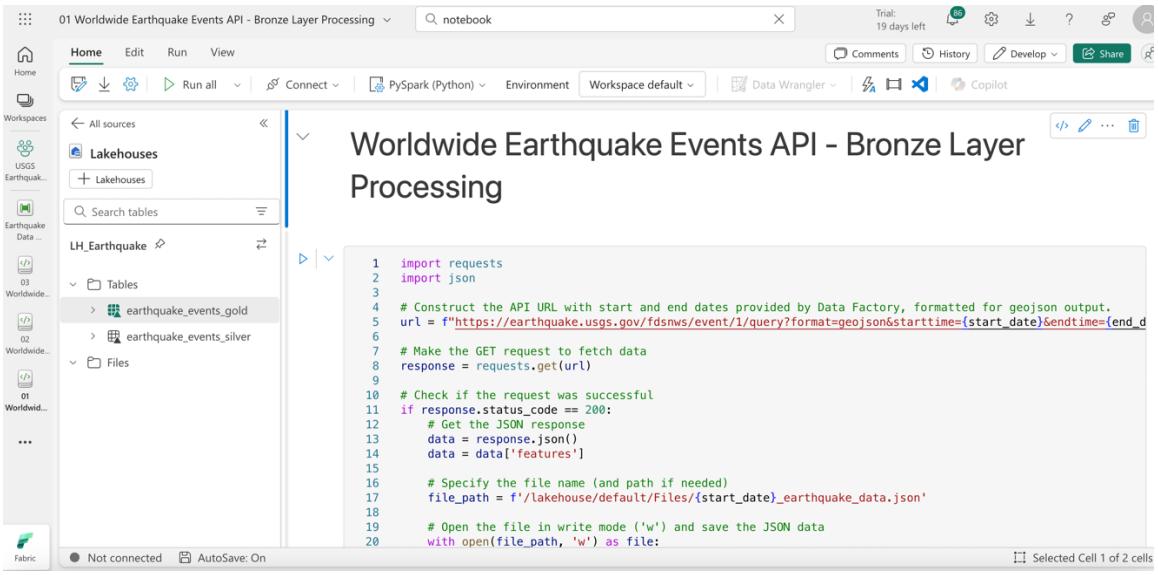
V. Pipeline Overview

The data pipeline is orchestrated using Microsoft Fabric and consists of three PySpark notebooks that process data through Bronze, Silver, and Gold layers.



1. **Bronze Notebook: Data Extraction**

The purpose is to retrieve JSON data from the USGS Earthquake API using Python's `requests` library. Output is saved as raw JSON data to the Bronze Layer in JSON format.



PySpark notebook for Bronze Layer processing.

```python
import requests
import json

# Construct the API URL with start and end dates provided by Data Factory, formatted
for geojson output.
url =
f"https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&starttime={start_da
te}&endtime={end_date}"

# Make the GET request to fetch data
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    # Get the JSON response
    data = response.json()
    data = data['features']

    # Specify the file name (and path if needed)
    file_path = f'/lakehouse/default/Files/{start_date}_earthquake_data.json'

    # Open the file in write mode ('w') and save the JSON data
    with open(file_path, 'w') as file:
        # The `json.dump` method serializes `data` as a JSON formatted stream to
`file`
        # `indent=4` makes the file human-readable by adding whitespace
        json.dump(data, file, indent=4)

    print(f"Data successfully saved to {file_path}")
else:
    print("Failed to fetch data. Status code:", response.status_code)
```

Full PySpark code for the Bronze Layer processing.

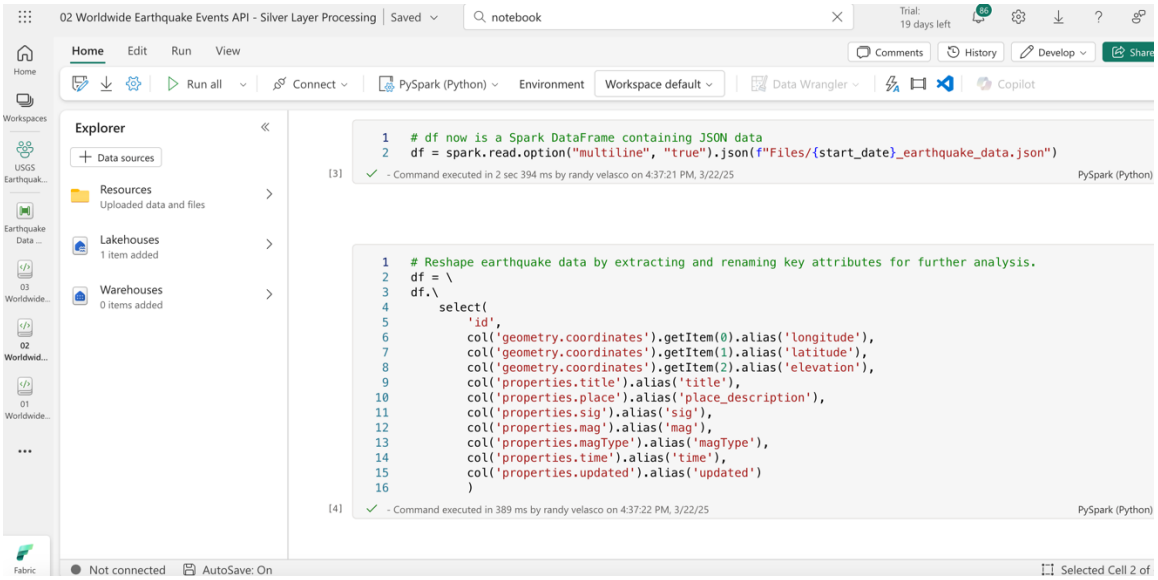
This code automates the process of fetching earthquake data from the **USGS Earthquake API**, processing it, and saving it to a file for further analysis. Here's a brief explanation of its purpose:

a. **API Request**: The code constructs a URL using specified start and end dates (`start_date` and `end_date`) to query the API for earthquake events within that time frame. The output format is set to **GeoJSON** for geospatial data.
b. **Data Retrieval**: Using the `requests.get()` method, it sends an HTTP GET request to the API to fetch data.
c. **Response Validation**: It checks if the request was successful by verifying the status code (200 indicates success). If the request fails, an error message is printed.

d. **Data Extraction**: Upon success, it extracts the `features` field from the JSON response, which contains the earthquake data.
e. **Save Data to File**: The extracted data is saved as a JSON file in a specified location (`file_path`) using the `json.dump()` method. The output is indented for better readability.
f. **Output Confirmation**: A message confirms that the data has been successfully saved, ensuring traceability.

2. **Silver Notebook: Data Transformation**

The purpose of this notebook is to convert JSON data from the Bronze Layer into a tabular format using PySpark. The output is a structured data saved as a table in the Silver Layer.



PySpark notebook for Silver Layer processing.

```
from pyspark.sql.functions import col
from pyspark.sql.types import TimestampType

# df now is a Spark DataFrame containing JSON data
df = spark.read.option("multiline",
"true").json(f"Files/{start_date}_earthquake_data.json")
# Reshape earthquake data by extracting and renaming key attributes for further
analysis.
df = \
df.\
```

```
    select(
        'id',
        col('geometry.coordinates').getItem(0).alias('longitude'),
        col('geometry.coordinates').getItem(1).alias('latitude'),
        col('geometry.coordinates').getItem(2).alias('elevation'),
        col('properties.title').alias('title'),
        col('properties.place').alias('place_description'),
        col('properties.sig').alias('sig'),
        col('properties.mag').alias('mag'),
        col('properties.magType').alias('magType'),
        col('properties.time').alias('time'),
        col('properties.updated').alias('updated')
        )

# Convert 'time' and 'updated' columns from milliseconds to timestamp format for
clearer datetime representation.
df = df.\
    withColumn('time', col('time')/1000).\
    withColumn('updated', col('updated')/1000).\
    withColumn('time', col('time').cast(TimestampType())).\
    withColumn('updated', col('updated').cast(TimestampType()))


# appending the data to the gold table
df.write.mode('append').saveAsTable('earthquake_events_silver')
```

Full PySpark code for the Silver Layer processing.

This Python code is designed to process earthquake data stored in a JSON file. Here's how it works step by step:

a. **Import Libraries:** The code begins by importing required functions and types from `pyspark.sql.functions` and `pyspark.sql.types`. These are used to manipulate and cast data in the processing pipeline.
b. **Read the JSON Data:** It reads the JSON file (e.g., `Files/{start_date}_earthquake_data.json`) into a PySpark DataFrame. The `multiline=true` option ensures that the JSON file is parsed correctly if the data spans multiple lines.
c. **Extract and Reshape Key Data:** The JSON structure includes nested data. This step reshapes the dataset by extracting fields of interest, such as `geometry.coordinates` for longitude, latitude, and elevation, as well as properties like `title`, `place_description`, `magnitude (mag)`, and timestamps.

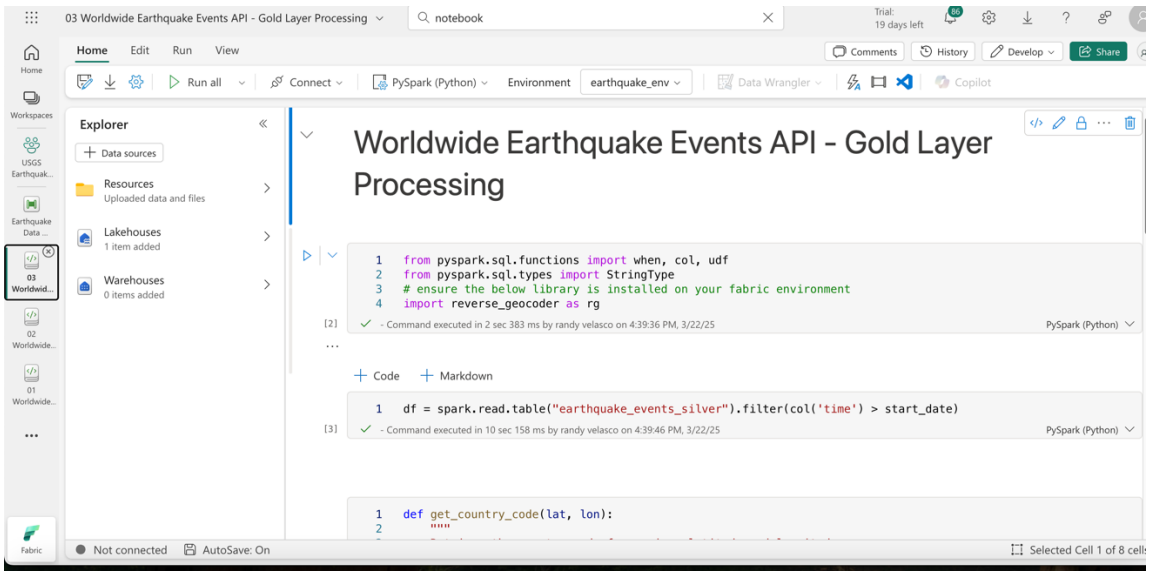
d. **Rename and Assign Columns:** Selected fields are assigned meaningful column names using PySpark functions like `alias`. For instance, longitude and latitude are extracted from JSON arrays and renamed for clarity.

e. **Convert Timestamps:** Columns like `time` and `updated` are stored as epoch timestamps (milliseconds since 1970). The code:
   - Divides them by 1000 to convert milliseconds to seconds.
   - Casts the values to `TimestampType()` for better readability and usability in analysis.

f. **Transform and Output Data:** The resulting DataFrame is ready for further transformations, queries, or saving to an enriched layer, such as a table in the Silver Layer of the pipeline.

3. **Gold Notebook: Data Enhancement**

The purpose is to add new columns, such as country code and significance categorization, to the table from the Silver Layer. The output is an enriched data saved to the Gold Layer, ready for analysis.



PySpark notebook for Gold Layer processing.

```
from pyspark.sql.functions import when, col, udf
from pyspark.sql.types import StringType
```

```
# ensure the below library is installed on your fabric environment
import reverse_geocoder as rg
```

```
df = spark.read.table("earthquake_events_silver").filter(col('time') >
start_date)
```

```
def get_country_code(lat, lon):
    """
    Retrieve the country code for a given latitude and longitude.

    Parameters:
    lat (float or str): Latitude of the location.
    lon (float or str): Longitude of the location.

    Returns:
    str: Country code of the location, retrieved using the reverse
geocoding API.

    Example:
    >>> get_country_details(48.8588443, 2.2943506)
    'FR'
    """
    coordinates = (float(lat), float(lon))
    return rg.search(coordinates)[0].get('cc')
```

```
# registering the udfs so they can be used on spark dataframes
get_country_code_udf = udf(get_country_code, StringType())
```

```
# adding country_code and city attributes
df_with_location = \
                df.\
                    withColumn("country_code",
get_country_code_udf(col("latitude"), col("longitude")))
```

```
# adding significance classification
df_with_location_sig_class = \
                            df_with_location.\
                                withColumn('sig_class',
                                           when(col("sig") < 100, "Low").\
                                           when((col("sig") >= 100) &
(col("sig") < 500), "Moderate").\
                                           otherwise("High")
                                           )
# appending the data to the gold table
df_with_location_sig_class.write.mode('append').saveAsTable('earthquake_eve
nts_gold')
```

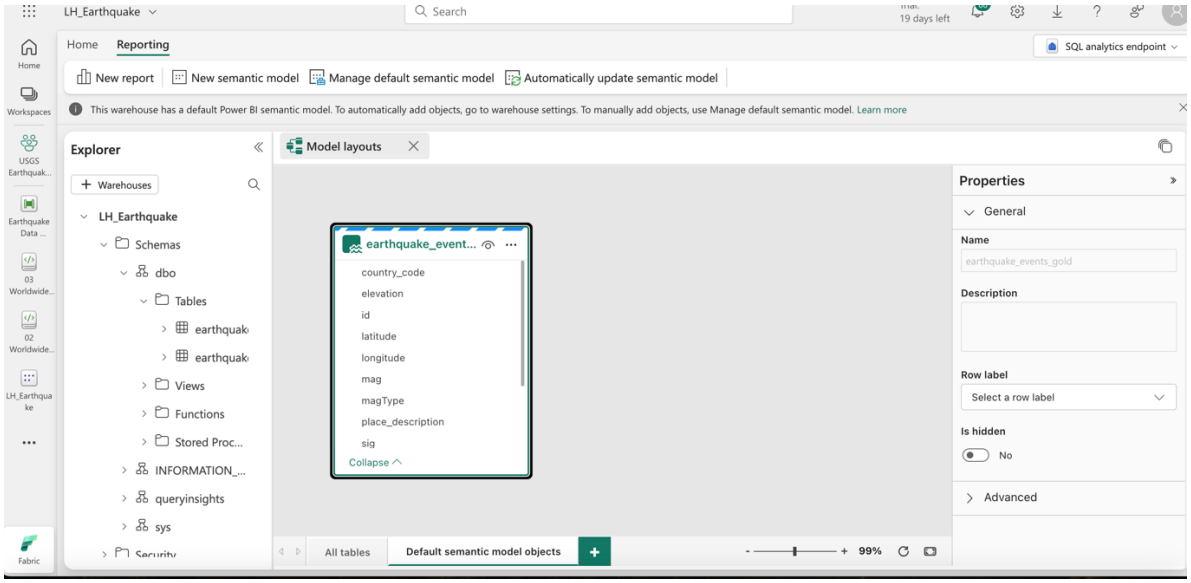
Full PySpark code for the Gold Layer processing.

This code enriches earthquake data by:

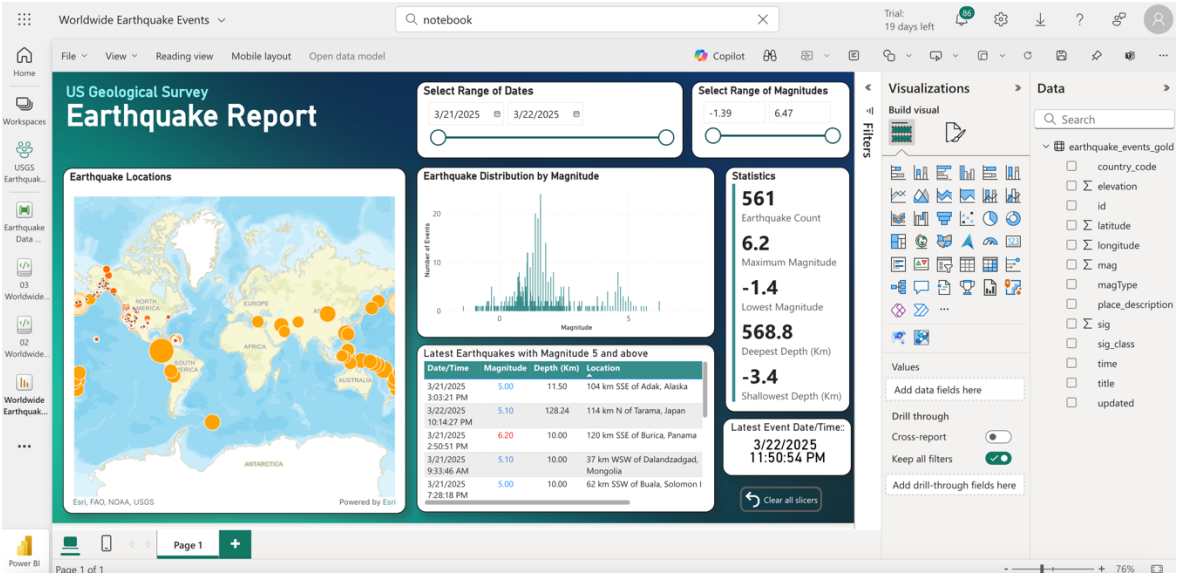
- **Adding Location Data**: Captures geographic details (country code) for each event based on latitude and longitude.
- **Significance Classification**: Provides meaningful insights into event severity by grouping earthquakes into classes.
- **Pipeline Enhancement**: Saves enriched data to the Gold Layer, preparing it for analysis and visualization in dashboards.

VI. Visualization

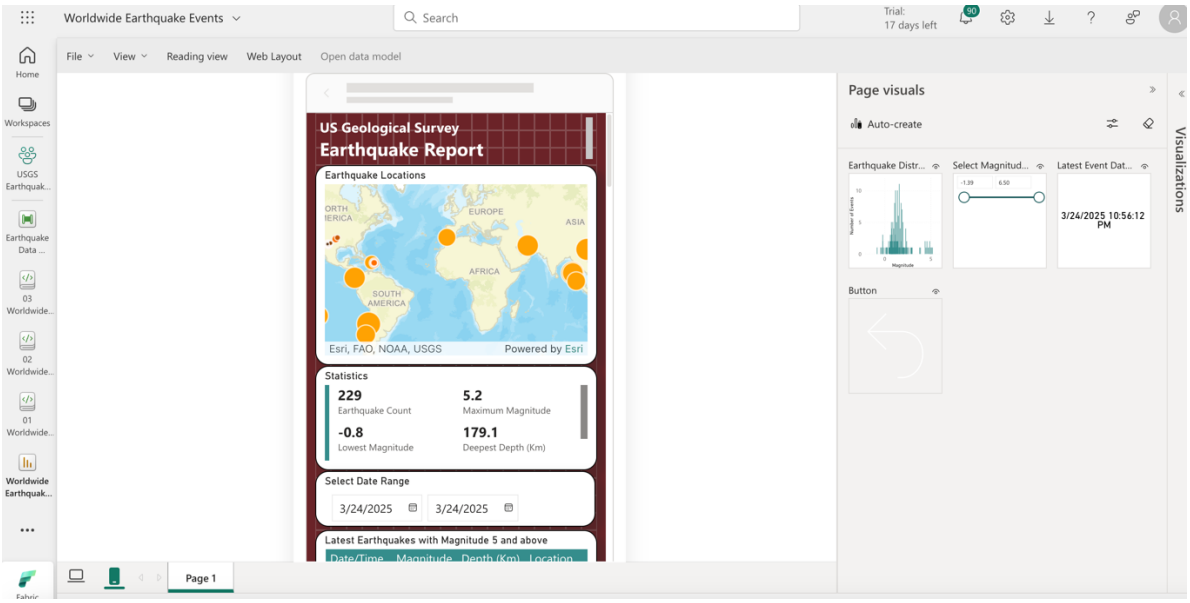
The Gold Layer table is used to create a default semantic model in Power BI. This model powers an interactive dashboard hosted in the Power BI Service.



The default semantic model is created and consists only of a single table from the Gold Layer.



Power BI Report's desktop version utilizes data from the semantic model, derived as an output of the pipeline.



Power BI Report's mobile version utilizes data from the semantic model, derived as an output of the pipeline.

Key dashboard features:

- **Map**: Shows earthquake locations with details like magnitude, depth, and coordinates.
- **Charts**: Highlights magnitude distribution and other stats.
- **Insights**: Provides daily updates for tracking trends in real time.

VII. Technical Highlights

1. **Automation**:
   The pipeline ensures data is automatically extracted daily and seamlessly appended to the existing dataset, eliminating the need for manual intervention.

2. **Cloud Integration**:
   The entire workflow is deployed within Microsoft Fabric, leveraging its cloud-native capabilities for smooth orchestration and scalability.

3. **Data Governance:**
   Implements a structured layered approach (Bronze, Silver, Gold) to enhance clarity, maintain data quality, and ensure reliability throughout the process.

VIII. Key Outcomes

- Scalable Data Pipeline: Handles growing data volumes while maintaining performance.
- Actionable Insights: Enables stakeholders to monitor earthquake trends and make informed decisions.
- Skill Demonstration: Showcases expertise in cloud-based ETL workflows, data visualization, and semantic modeling.

This project demonstrates my ability to design, build, and deploy robust data engineering pipelines, utilizing cloud technologies and industry-best practices.

About me:

I am **Randy A. Velasco**, a seasoned Professional Electronics Engineer with a strong foundation in data governance, data engineering, data analysis, and database management. I am a certified Data Engineer and Data Analyst. I am also certified in Python, SQL, Data Literacy, Microsoft Azure AI Fundamentals (AI-900) and Microsoft Azure Fundamentals (AZ-900).

By working on end-to-end projects like this, I mimic real-world workflows, such as designing pipelines, ensuring data quality, and presenting insights. It solidifies my understanding of tools, technologies and techniques, such as ETL processes, data pipelines, and visualization tools like Power BI. I gain experience handling issues like missing data, data cleaning, and integrating multiple sources into cohesive datasets. Tackling real-world scenarios helps you bridge the gap between theory and practice, which is invaluable in mastering concepts like SQL queries, cloud technologies, and Python data manipulation.