Building an ETL Pipeline using PySpark

By Mauzum Shamil

An ETL (Extract, Transform, and Load) pipeline extracts data from sources, transforms it, and loads it into a storage system. It helps create clean, usable data formats for analysis. PySpark is ideal for building ETL pipelines for large-scale data processing. It offers distributed computing, high performance, and handles structured and unstructured data efficiently. This project will take you through building an ETL pipeline using PySpark

about the dataset

We'll develop an ETL Pipeline using PySpark to process this dataset to handle the following tasks:

1- Extract: Load the dataset from the CSV file.

2- Transform: Clean the data, handle missing values, and pivot year-wise temperature data for analysis.

3- Load: Save the processed data into a new storage format (e.g., Parquet or a database).

Step 1: Setting Up the Environment & Initializing a PySpark Session

Ensure that PySpark is installed and set up. Run the following command to install PySpark if it's not already installed:

```
!pip install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.4)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)
```

```
# Initialize a PySpark session to enable interaction with the Spark framework

from pyspark.sql import SparkSession

# initialize SparkSession
spark  = SparkSession.builder.appName("ETL Pipeline").getOrCreate()
```

Step 2: Extract – Load the Dataset

```
# load the csv file into the spark dataframe
file_path = "/content/temperature.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)

# display the schema and preview the date
df.printSchema()
df.show(5)
```

```
 integer (nullable = true)
string (nullable = true)
ing (nullable = true)
ing (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
```

```
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
uble (nullable = true)
```

In PySpark, we are loading a CSV file into a distributed DataFrame, which is similar to using pandas.read_csv() to load data into a Pandas DataFrame. However, unlike Pandas, which uses memory and runs on a single machine, PySpark handles large datasets distributed across a cluster. The methods df.printSchema() and df.show(5) provide insights into the schema and preview the data, comparable to df.info() and df.head() in Pandas, but designed for scalable data exploration on big data workloads.

Step 3: Transform – Clean and Process the Data

All datasets require different types of cleaning and processing steps. In this data, we will replace missing values in important columns like ISO2 or impute missing temperature values:

```
# fill missing values for country codes
df = df.fillna({"ISO2": "Unknown"})

# DROP ROWS WHERE ALL TEMPERATURE VALUES ARE NULL
temperatre_columns = [col for col in df.columns if col.startswith('F')]
df = df.na.drop(subset=temperatre_columns, how='all')
```

Next, we will transform the dataset to have "Year" as a single column and its temperature value:

```
from pyspark.sql.functions import expr
# reshape temperature data to have 'year' and 'Temperature' columns
df_pivot = df.selectExpr(
    "ObjectId", "Country", "ISO3",
    "stack(62, " +
    ",".join([f"'F{1961 + i}', F{1961 + i}" for i in range(62)]) +
    ") as (Year, Temperature)"
)

# convert 'Year' column to integer
df_pivot = df_pivot.withColumn("Year", expr("int(substring(Year, 2, 4))"))
df_pivot.show(5)
```

```
+--------+--------------------+----+----+-----------+
|ObjectId|             Country|ISO3|Year|Temperature|
+--------+--------------------+----+----+-----------+
|       1|Afghanistan, Isla...| AFG|1961|     -0.113|
|       1|Afghanistan, Isla...| AFG|1962|     -0.164|
|       1|Afghanistan, Isla...| AFG|1963|      0.847|
|       1|Afghanistan, Isla...| AFG|1964|     -0.764|
|       1|Afghanistan, Isla...| AFG|1965|     -0.244|
+--------+--------------------+----+----+-----------+
only showing top 5 rows
```

```
# Collect the DataFrame into a list
data = df_pivot.collect()
```

```
# Get the last 5 rows
last_5_rows = data[-5:]

# Print the last 5 rows
for row in last_5_rows:
    print(row)
```

```
Row(ObjectId=225, Country='Zimbabwe', ISO3='ZWE', Year=2018, Temperature=0.453)
Row(ObjectId=225, Country='Zimbabwe', ISO3='ZWE', Year=2019, Temperature=0.925)
Row(ObjectId=225, Country='Zimbabwe', ISO3='ZWE', Year=2020, Temperature=0.389)
Row(ObjectId=225, Country='Zimbabwe', ISO3='ZWE', Year=2021, Temperature=-0.125)
Row(ObjectId=225, Country='Zimbabwe', ISO3='ZWE', Year=2022, Temperature=-0.49)
```

Step 4: Load – Save the Processed Data

After completing all the processing steps, you save the transformed data to a Parquet file for efficient storage and querying:

```
output_path = "/processed_temperature.parquet"
df_pivot.write.mode("overwrite").parquet(output_path)
```

This operation saves the transformed DataFrame as a Parquet file, which optimizes it for storage and querying in a distributed environment.

```
#We can load the saved Parquet file to ensure the data was correctly saved:
# load the saved parquet file
processed_df = spark.read.parquet(output_path)
processed_df.show(5)
```

```
+--------+--------------------+----+----+-----------+
|ObjectId|             Country|ISO3|Year|Temperature|
+--------+--------------------+----+----+-----------+
|       1|Afghanistan, Isla...| AFG|1961|     -0.113|
|       1|Afghanistan, Isla...| AFG|1962|     -0.164|
|       1|Afghanistan, Isla...| AFG|1963|      0.847|
|       1|Afghanistan, Isla...| AFG|1964|     -0.764|
|       1|Afghanistan, Isla...| AFG|1965|     -0.244|
+--------+--------------------+----+----+-----------+
only showing top 5 rows
```

Summary

In this project, we built an ETL (Extract, Transform, Load) pipeline using PySpark, a powerful framework for distributed data processing. The goal of the pipeline was to process a dataset containing temperature data for various countries over several years. The dataset was initially in a CSV format, with each year's temperature data stored in separate columns. The pipeline was designed to extract the data, clean and transform it into a more usable format, and finally load it into a storage system optimized for efficient querying and analysis.

Key Steps in the Pipeline:

1-Extract:

The dataset was loaded from a CSV file into a PySpark DataFrame. This step involved reading the file and inferring the schema to ensure the data types were correctly identified.

The dataset contained columns for country information (e.g., Country, ISO2, ISO3) and yearly temperature data from 1961 to 2022.

2-Transform:

Data Cleaning: Missing values in the ISO2 column were replaced with "Unknown". Additionally, rows where all temperature values were missing were dropped to ensure data quality.

Data Reshaping: The dataset was transformed from a wide format (with separate columns for each year) to a long format, where each row represented a single year's temperature for a country. This was achieved using PySpark's stack function.

Data Type Conversion: The Year column was converted from a string (e.g., "F1961") to an integer (e.g., 1961) for easier analysis.

3-Load:

The transformed data was saved in Parquet format, a columnar storage format optimized for distributed systems. This format is highly efficient for storage and querying, especially in big data environments.

The saved Parquet file was reloaded to verify that the data was correctly processed and stored.

Key Insights:

Scalability: PySpark's distributed computing capabilities make it ideal for processing large datasets that cannot fit into the memory of a single machine. This project demonstrated how PySpark can handle data transformation tasks efficiently, even with a dataset spanning multiple decades and countries.

Data Quality: The pipeline included steps to handle missing data, ensuring that the final dataset was clean and ready for analysis.

Optimized Storage: By saving the data in Parquet format, the pipeline ensured that the processed data could be efficiently queried and analyzed in the future.

Conclusion

This project showcased the power of PySpark in building robust and scalable ETL pipelines. By leveraging PySpark's distributed computing capabilities, we were able to process a large dataset efficiently, clean and transform it into a more usable format, and store it in an optimized format for future analysis.

ETL pipelines are a critical component of data engineering workflows, and PySpark provides the tools necessary to handle large-scale data processing tasks with ease. Whether you're working with structured or unstructured data, PySpark's flexibility and performance make it an excellent choice for building data pipelines.

Profiles for Reference If you'd like to explore more of my work or connect with me, feel free to check out my profiles:

- GitHub: https://github.com/mauzumshamil Here, you can find more projects and code samples related to data engineering, machine learning, and software development.
- LinkedIn: http://linkedin.com/in/mauzum-shamil Connect with me on LinkedIn to stay updated on my latest projects and professional endeavors.
- Portfolio Link: https://linktr.ee/mauzum_shamil This is a centralized hub for all my work, including projects, blogs, and social media profiles.

Thank you for reading! I hope this project provided valuable insights into building ETL pipelines using PySpark. If you have any questions or feedback, feel free to reach out!

```
Start coding or generate with AI.
```