SUMMARY-DAY10

Name: Tejaswini Gokanakonda

Roll no:DE142

Date:19-11-2024

PYSPARK OVERVIEW

PySpark is a distributed computing framework built on top of Apache Spark, widely used for large-scale data processing. It provides a Python interface to Spark's core functionalities, allowing efficient data manipulation and transformations across clusters.

1. Loading Data

- PySpark supports various data formats like CSV, JSON, and Parquet.
- The read.load() function loads datasets into a Spark DataFrame, which is Spark's equivalent of a table in SQL or a DataFrame in pandas.
- It also supports options like specifying the separator for CSV files, inferring schema types automatically, and including headers.
- Example Use Case: Loading COVID-19 case data to analyze trends.

2. Viewing Data

- To understand the structure of the data, Spark provides the show() method, which displays the first few rows.

- When working in environments like Jupyter Notebook, converting Spark DataFrames to Pandas using .toPandas() provides a more compact and readable format.
- Example: This feature is particularly useful when you have wide tables, as Spark's default output can truncate or format the data poorly.

3. Renaming and Selecting Columns

- Renaming Columns: Sometimes the names in a dataset might not be intuitive or may not follow naming conventions. PySpark allows you to rename single or multiple columns at once.
- Selecting Columns: To reduce complexity or focus on relevant data, subsets of columns can be selected. This is especially useful when you work with large datasets with many columns.
- Example: Extracting columns like "City," "Province," and "Confirmed Cases" to focus on regional data analysis.

4. Sorting Data

- Sorting allows you to arrange your dataset based on column values, either in ascending or descending order.
- By default, sorting is in ascending order. For descending order, you use a specialized function like desc().
- Use Case: Sorting case counts in descending order helps identify areas with the highest infection rates.

5. Changing Data Types

- Sometimes, data might be loaded in incorrect formats, such as numerical data being interpreted as strings. PySpark allows data type conversion through casting.
- Example: Converting "Confirmed Cases" from string to integer for numerical analysis.

6. Filtering Data

- Filtering is a fundamental operation for extracting relevant subsets of data based on conditions.
- PySpark supports multiple conditions combined with operators like AND, OR, and NOT.
- Example: Filtering rows where cases exceed a certain number in a specific region to study high-risk areas.

7. Grouping and Aggregating Data

- GroupBy: Groups data based on specified columns, enabling aggregation over those groups.
- Aggregations include operations like summing, finding maximum or minimum, and calculating averages.
- Aliases can be used to rename the aggregated columns, improving readability.
- Example: Summing up confirmed cases by city and finding the maximum cases in each province.

8. Joining DataFrames

- Joins are used to combine data from two or more tables based on a common key.
- PySpark supports various types of joins: inner, left, right, and full outer joins.
- A special optimization, broadcast joins, is used when joining a large table with a much smaller one. Broadcasting sends the small table to all nodes, reducing data shuffling.
- Example: Combining a COVID-19 case table with a region table to add demographic details.

9. Using SQL with PySpark

- PySpark provides SQL-like operations by registering a DataFrame as a temporary SQL table.
- Complex queries involving filters, groupings, and aggregations can be written in SQL and executed on Spark DataFrames.
- Example: Extracting all rows with more than 100 confirmed cases using SQL for clarity and familiarity.

10. Creating New Columns

- PySpark allows you to add new columns by applying transformations or calculations on existing ones.
- Spark's built-in functions include operations for strings, dates, and math.
- Example: Adding a column for exponential growth based on confirmed cases to study patterns in infection spread.

Additional Notes on Efficiency:

- PySpark is designed for parallel processing, making it highly efficient for handling big data.
- Techniques like caching intermediate results, using broadcast joins, and selecting only required columns can significantly enhance performance.
- Spark's lazy evaluation means operations are not executed until an action (like showing or saving data) is performed.