codecademy

Spark DataFrames with PySpark SQL

Starting a SparkSession

A SparkSession is the entry point to Spark SQL. The session is a wrapper around a SparkContext and contains all the metadata required to start working with distributed data.

```
# start a SparkSession
spark = SparkSession.builder.getOrCreate()
```

PySpark DataFrames

PySpark DataFrames are distributed collections of data in tabular format that are built on top of RDDs. They function almost identically to pandas DataFrames, and allow users to manipulate data in Spark easily, especially when compared to RDDs.

RDDs to DataFrames

PySpark DataFrames can be created from RDDs using `rdd.toDF()`. They can also be converted back to RDDs with `DataFrame.rdd`.

```
# Create an RDD from a list
hrly_views_rdd  =
spark.sparkContext.parallelize([
    ["Betty_White" , 288886],
    ["Main_Page", 139564],
    ["New_Year's_Day", 7892],
    ["ABBA", 8154]
])

# Convert RDD to DataFrame
hrly_views_df = hrly_views_rdd\
    .toDF(["article_title", "view_count"])

# Convert DataFrame back to RDD
hrly_views_rdd = hrly_views_df.rdd
```

Inspecting DataFrame Schemas

All DataFrames have a schema that defines their structure, columns, datatypes, and value restrictions. We can use `DataFrame.printSchema()` to show a DataFrame's schema.

```
# view schema DataFrame df
df.printSchema()

# output:
root
|-- language_code: string (nullable =
true)
|-- article_title: string (nullable =
true)
|-- hourly_count: integer (nullable =
true)
|-- monthly_count: integer (nullable =
true)
```

Summarizing PySpark DataFrames

Similarly to pandas, we can display a high-level summary of PySpark DataFrames by using the `.describe()` function to quickly inspect the stored data.

```
df_desc = df.describe()
df_desc.show()

# output:
+-------+---------+---------+
|summary|  columnA|  columnB|
+-------+---------+---------+
|  count|465409100|465409100|
|   mean|  4.52417| 213.0394|
| stddev|182.92502| 27.54093|
|    min|        1|        0|
|    max|   288886|      628|
+-------+---------+-----+
```

DataFrame Columns

Similar to pandas DataFrames, PySpark columns can be dropped and renamed.

```
# Dropping a column
df = df.drop('column_name')

# Renaming a column
df = df.withColumnRenamed('old_name',
'new_name')
```

Querying DataFrames with SQL

PySpark allows users to query DataFrames using standard SQL queries.

```
# create a view
df.createTempView("tiny_df")

# query from a PySpark DataFrame
query = """SELECT * FROM tiny_df """
spark.sql(query).show()
```

Creating a Temp View

If there is a query that is often executed, we can save some time by saving that query as a temporary view. This saves the results as a table that can be stored in memory and used for future analysis.

```
# create a view from an existing dataframe and then query from it
tiny_df.createOrReplaceTempView('tiny_view')
spark.sql("SELECT * FROM tiny_view").show()
```

Using Parquet Files

Parquet is a file format used with Spark to save DataFrames. Parquet format offers many benefits over traditional file formats like CSV:

- Parquet files are efficiently compressed, so they are smaller than CSV files.
- Parquet files preserve information about a DataFrame's schema.
- Performing analysis on parquet files is often faster than CSV files.

```
# Write DataFrame to Parquet
df.write.parquet('./cleaned/parquet/views/', mode="overwrite")

# Read Parquet as DataFrame
df_restored = spark.read.parquet('./cleaned/parquet/views/')
```

Reading and Writing using PySpark

PySpark allows users to work with external data by reading from or writing to those files. Developers can use `spark.read.<file-format>(filename)` and `spark.write.<fileformat>(filename)` to read and write data between external files and Spark DataFrames.

```
# read from an external parquet file
df = spark.read.parquet('parquet_file.parquet')

# write to an external parquet file
spark.write.parquet('parquet_file.parquet', mode="overwrite")
```