Querying Big Data with PySpark

Data Boot Camp

Lesson 22.2

WELCOME

Class Objectives

By the end of this lesson, you will be able to:

- Apply grouping and aggregation functions to a dataset by using Spark.
- Parse and format date timestamps by using Spark.
- Use temporary tables to prepare data for SQL.
- Combine PySpark and SQL to run queries.

Grouping and Aggregating Data

We use the `groupBy()` function to group rows together, based on specific columns.

Aggregation Functions

PySpark makes several aggregation functions available. Here are a few:

- `avg()`
- `countDistinct()`
- `count()`
- `first()`
- `last()`
- `max()`
- `min()`

In PySpark, we need to combine the `groupBy()` function with an aggregation function to return the rows that we want.

Instructor Demonstration

PCard Transactions

Questions?

Activity: Retail Transactions

In this activity, you'll practice using the `groupBy()` and `agg()` functions with the Retail Transaction Data dataset.

Understanding how these functions work is important, because that's another step toward being able to transform data into various forms.

Suggested Time:

15 minutes

Time's Up! Let's Review.

Questions?

Reading and Plotting Time Series

Instructor Demonstration

Spark DataFrame Dates

Handling Date Formats (1 of 5)

To avoid errors when reading the data, we tell Spark to infer the schema:

`inferSchema=True`

We commonly encounter various date and timestamp formats.

Luckily, Spark provides a functions library that includes date and timestamp conversion functions.

Handling Date Formats (2 of 5)

We import the year() function, which we can use to select the year from a timestamp column:

```
# Import date time functions
from pyspark.sql.functions import year

# Show the year for the date column
df.select(year(df["date"])).show()
```

We can now create a new column that stores only the year:

```
# Save the year as a new column
df = df.withColumn("year", year(df['date']))
df.show()
```

Handling Date Formats (3 of 5)

With the new column, we can now group by year and find the average precipitation:

```
# Find the average precipitation per year
averages = df.groupBy("year").avg()
averages.orderBy("year").select("year", "avg(prcp)").show()
```

Handling Date Formats (4 of 5)

We can also use the month() function to select the month from a timestamp column:

```
# Import the month function to get the month from the "Date" column.
from pyspark.sql.functions import month
df.select(month(df['Date'])).show()
```

And, create a new column that stores only the month:

```
# Add a column, "month" to the DataFrame.
df = df.withColumn("month", month(df['date']))
df.show()
```

Handling Date Formats (5 of 5)

We can create a new DataFrame for the maximum precipitation:

```
# Get the maximum precipitation for each month.
max_ppt = df.groupBy("month").max()
max_ppt.orderBy("month").select("month", "max(prcp)").show()
```

And, convert the Spark DataFrame to a Pandas DataFrame:

```
# Import the summarized data to a Pandas DataFrame for plotting.
pandas_df = max_ppt.orderBy("month").select("month",
"max(prcp)").toPandas()
pandas_df.head()
```

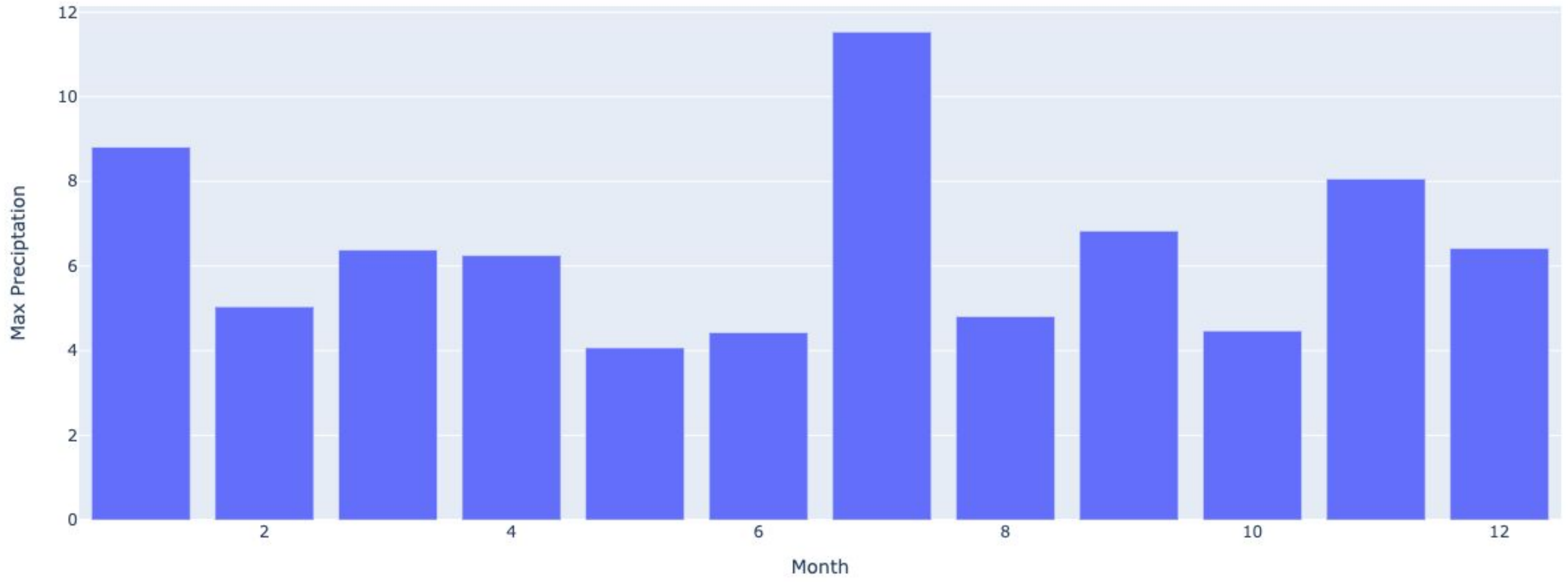
Plotting Time Series Data (1 of 2)

With the Pandas DataFrame, we can use Plotly to create a bar graph of the data.

```
import plotly.express as px
# Create a Plotly graph.
fig = px.bar(x=pandas_df['month'], y=pandas_df['max(prcp)'])
fig.update_layout(xaxis_title='Month', yaxis_title='Max Precipitation')
fig.show()
```

Plotting Time Series Data (2 of 2)

The bar graph of max precipitation per month:



Questions?

Group Programming Activity:

Spark Operations

In this activity, you'll first use Spark and Pandas to clean a bigfoot time series dataset, and you'll then create a plot.

Suggested Time:

20 minutes

Spark SQL

Today, we'll transition to using Spark SQL with Spark.

One of the primary goals of the Spark development team is to make Spark easy to use for people in various roles. And, almost every data professional is familiar with SQL

The Spark engine has a query optimization engine (named Catalyst optimizer) that creates a plan for how to run the SQL code.

We'll both rewrite some of our earlier activities and write new ones with fresh data.

Questions?

Break

Creating Spark Temporary Tables

One one of the many advantages of Spark is the ability to write code against it by using various APIs—all of which perform in a similar way.

What’s a Temporary Table?

A temporary table, or **temporary view**, provides a way to lazily load a Spark table into memory.

- Lazy loading doesn’t mean slow.
- Lazy loading means not completely loading anything until it’s needed.
- So, applications typically run faster.

Temporary Tables

Once we load a temporary table, we can query the table with SQL statements to produce rapid results.

- It’s important to differentiate a global temp view from a local temp view.
- To do so, we’ll discuss the Spark Application and the Spark Session.

When you define Spark on your system, you first create a Spark Application.

A **Spark Application** can hold one to many Spark jobs, views, and Sessions.

Global Temp Views

Until the application is stopped or the table is explicitly removed, a global temp view remains accessible to all parts of the application.

All the Sessions in that Spark Application can access a **global temp view**.

We create a global temp view with the following command:

```
df.createOrReplaceGlobalTempView(<viewName>)
```

Local Temp Views

Other jobs or sessions can't access a **local temp view**. This can be helpful when we want to limit the scope of a session to a specific task..

We create a local temp view by using the following command:

```
df.createOrReplaceTempView(<viewName>)
```

Instructor Demonstration

Creating a Temporary Table

Questions?

Activity: Amazon Vine Temporary View

In this activity, you'll import an Amazon Vine dataset, create a temporary table, and then use Spark SQL to run queries on the temporary table.

Suggested Time:

15 Minutes

Time’s Up! Let’s Review.

Questions?

Combining PySpark with Spark SQL

Combining PySpark with Spark SQL (1 of 3)

Now that you’re familiar with both the PySpark and Spark SQL frameworks, it’s time to bring them together to provide some flexibility when querying data.

PySpark + Apache Spark SQL

Combining PySpark with Spark SQL (2 of 3)

One of the best features of the Spark framework is that we can easily switch among querying styles and languages.

The Spark infrastructure is built to accommodate dedicated Spark users, SQL enthusiasts, and Python programmers alike.

Combining PySpark with Spark SQL (3 of 3)

Regardless of which language or part of the Spark infrastructure we prefer, the performance of all the parts of the Spark infrastructure is the same.

- The Spark engine optimizes XXX based on what you’re doing and not on which language you’re using.
- While coding in Spark, people commonly switch between languages.

Instructor Demonstration

Combining PySpark with Spark SQL

Questions?

Activity: Earn a Medal in Spark SQL

In this activity, you'll evaluate a dataset that contains Olympic athlete data by using Spark SQL.

Suggested Time:

20 Minutes

Time’s Up! Let’s Review.

Questions?