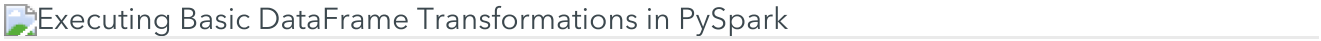
Executing Basic DataFrame Transformations in PySpark

Todd Spark



If you joined us **last time**, you should have some working knowledge of how to get started with PySpark by using a Databricks notebook. Armed with that knowledge, we can now start playing with real data.

For most of the time we spend in PySpark, we'll likely be working with Spark DataFrames: this is our bread and butter for data manipulation in Spark. For this exercise, we'll attempt to execute an elementary string of transformations to get a feel for what the middle portion of an ETL pipeline looks like (also known as the "transform" part 😁).

Loading Up Some Data

As usual, I'll be loading up some sample data from our best friend: Google BigQuery. The example data I'll be using is a public dataset from BigQuery: the results of the MLB 2016 postseason:

Baseball games from BigQuery.

We'll export this data to a CSV. Next, we'll import this data into Databricks the same way as last time. Now we can get started messing with data.

Simple Data Transformations

First things first, we need to load this data into a DataFrame:

```
# File location and type
file_location = "/FileStore/tables/results_20190428_181010-cc606.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df)
```

Nothing new so far! Of course, we should store this data as a table for future use:

```
# Create a view or table

permanent_table_name = "baseball_2016_postseason"
df.write.format("parquet").saveAsTable(permanent_table_name)
```

Before going any further, we need to decide what we actually want to do with this data (I'd hope that under normal circumstances, this is the *first* thing we do)! Let's get a quick look at what we're working with, by using `print(df.info())`:

```
root
 |-- gameId: string (nullable = true)
 |-- seasonId: string (nullable = true)
 |-- seasonType: string (nullable = true)
 |-- year: integer (nullable = true)
 |-- startTime: string (nullable = true)
 |-- gameStatus: string (nullable = true)
 |-- attendance: integer (nullable = true)
 |-- dayNight: string (nullable = true)
 |-- duration: string (nullable = true)
 |-- durationMinutes: integer (nullable = true)
 |-- awayTeamId: string (nullable = true)
 |-- awayTeamName: string (nullable = true)
 |-- homeTeamId: string (nullable = true)
 |-- homeTeamName: string (nullable = true)
```

```
 |-- venueId: string (nullable = true)
 |-- venueName: string (nullable = true)
 |-- venueSurface: string (nullable = true)
 |-- venueCapacity: integer (nullable = true)
 |-- venueCity: string (nullable = true)
 |-- venueState: string (nullable = true)
 |-- venueZip: integer (nullable = true)
 |-- venueMarket: string (nullable = true)
 |-- venueOutfieldDistances: string (nullable = true)
 |-- homeFinalRuns: integer (nullable = true)
 |-- homeFinalHits: integer (nullable = true)
 |-- homeFinalErrors: integer (nullable = true)
 |-- awayFinalRuns: integer (nullable = true)
 |-- awayFinalHits: integer (nullable = true)
 |-- awayFinalErrors: integer (nullable = true)
 |-- homeFinalRunsForInning: integer (nullable = true)
 |-- awayFinalRunsForInning: integer (nullable = true)
 |-- inningNumber: integer (nullable = true)
 |-- inningHalf: string (nullable = true)
 |-- inningEventType: string (nullable = true)
 |-- inningHalfEventSequenceNumber: integer (nullable = true)
 |-- description: string (nullable = true)
 |-- atBatEventType: string (nullable = true)
 |-- atBatEventSequenceNumber: integer (nullable = true)
 |-- createdAt: string (nullable = true)
 |-- updatedAt: string (nullable = true)
 |-- status: string (nullable = true)
 |-- outcomeId: string (nullable = true)
 |-- outcomeDescription: string (nullable = true)
 |-- hitterId: string (nullable = true)
 |-- hitterLastName: string (nullable = true)
 |-- hitterFirstName: string (nullable = true)
 |-- hitterWeight: integer (nullable = true)
 |-- hitterHeight: integer (nullable = true)
 |-- hitterBatHand: string (nullable = true)
 |-- pitcherId: string (nullable = true)
 |-- pitcherFirstName: string (nullable = true)
 |-- pitcherLastName: string (nullable = true)
 |-- pitcherThrowHand: string (nullable = true)
 |-- pitchType: string (nullable = true)
 |-- pitchTypeDescription: string (nullable = true)
 |-- pitchSpeed: integer (nullable = true)
 |-- pitchZone: integer (nullable = true)
 |-- pitcherPitchCount: integer (nullable = true)
 |-- hitterPitchCount: integer (nullable = true)
 |-- hitLocation: integer (nullable = true)
 |-- hitType: string (nullable = true)
 |-- startingBalls: integer (nullable = true)
 |-- startingStrikes: integer (nullable = true)
 |-- startingOuts: integer (nullable = true)
 |-- balls: integer (nullable = true)
 |-- strikes: integer (nullable = true)
 |-- outs: integer (nullable = true)
 |-- rob0_start: string (nullable = true)
 |-- rob0_end: integer (nullable = true)
```

```
|-- rob0_isOut: string (nullable = true)
|-- rob0_outcomeId: string (nullable = true)
|-- rob0_outcomeDescription: string (nullable = true)
|-- rob1_start: string (nullable = true)
|-- rob1_end: integer (nullable = true)
|-- rob1_isOut: string (nullable = true)
|-- rob1_outcomeId: string (nullable = true)
|-- rob1_outcomeDescription: string (nullable = true)
|-- rob2_start: string (nullable = true)
|-- rob2_end: integer (nullable = true)
|-- rob2_isOut: string (nullable = true)
|-- rob2_outcomeId: string (nullable = true)
|-- rob2_outcomeDescription: string (nullable = true)
|-- rob3_start: string (nullable = true)
|-- rob3_end: integer (nullable = true)
|-- rob3_isOut: string (nullable = true)
|-- rob3_outcomeId: string (nullable = true)
|-- rob3_outcomeDescription: string (nullable = true)
|-- is_ab: integer (nullable = true)
|-- is_ab_over: integer (nullable = true)
|-- is_hit: integer (nullable = true)
|-- is_on_base: integer (nullable = true)
|-- is_bunt: integer (nullable = true)
|-- is_bunt_shown: integer (nullable = true)
|-- is_double_play: integer (nullable = true)
|-- is_triple_play: integer (nullable = true)
|-- is_wild_pitch: integer (nullable = true)
|-- is_passed_ball: integer (nullable = true)
|-- homeCurrentTotalRuns: integer (nullable = true)
|-- awayCurrentTotalRuns: integer (nullable = true)
|-- awayFielder1: string (nullable = true)
|-- awayFielder2: string (nullable = true)
|-- awayFielder3: string (nullable = true)
|-- awayFielder4: string (nullable = true)
|-- awayFielder5: string (nullable = true)
|-- awayFielder6: string (nullable = true)
|-- awayFielder7: string (nullable = true)
|-- awayFielder8: string (nullable = true)
|-- awayFielder9: string (nullable = true)
|-- awayFielder10: string (nullable = true)
|-- awayFielder11: string (nullable = true)
|-- awayFielder12: string (nullable = true)
|-- awayBatter1: string (nullable = true)
|-- awayBatter2: string (nullable = true)
|-- awayBatter3: string (nullable = true)
|-- awayBatter4: string (nullable = true)
|-- awayBatter5: string (nullable = true)
|-- awayBatter6: string (nullable = true)
|-- awayBatter7: string (nullable = true)
|-- awayBatter8: string (nullable = true)
|-- awayBatter9: string (nullable = true)
|-- homeFielder1: string (nullable = true)
|-- homeFielder2: string (nullable = true)
|-- homeFielder3: string (nullable = true)
|-- homeFielder4: string (nullable = true)
```

```
 |-- homeFielder5: string (nullable = true)
 |-- homeFielder6: string (nullable = true)
 |-- homeFielder7: string (nullable = true)
 |-- homeFielder8: string (nullable = true)
 |-- homeFielder9: string (nullable = true)
 |-- homeFielder10: string (nullable = true)
 |-- homeFielder11: string (nullable = true)
 |-- homeFielder12: string (nullable = true)
 |-- homeBatter1: string (nullable = true)
 |-- homeBatter2: string (nullable = true)
 |-- homeBatter3: string (nullable = true)
 |-- homeBatter4: string (nullable = true)
 |-- homeBatter5: string (nullable = true)
 |-- homeBatter6: string (nullable = true)
 |-- homeBatter7: string (nullable = true)
 |-- homeBatter8: string (nullable = true)
 |-- homeBatter9: string (nullable = true)
 |-- lineupTeamId: string (nullable = true)
 |-- lineupPlayerId: string (nullable = true)
 |-- lineupPosition: integer (nullable = true)
 |-- lineupOrder: integer (nullable = true)
```

Holy hell, that's a lot of columns! Let's see what the deal is with these columns by inspecting our data via `display(df)`:

gameId	seasonId	seasonType	year	startTime	gameS
01d76e9f-6095-40dd…	565de4be-dc80-4849…	PST	2016	2016-10-19 00:08:00 …	closed
01d76e9f-6095-40dd…	565de4be-dc80-4849…	PST	2016	2016-10-19 00:08:00 …	closed
01d76e9f-6095-40dd…	565de4be-dc80-4849…	PST	2016	2016-10-19 00:08:00 …	closed
01d76e9f-6095-40dd…	565de4be-dc80-4849…	PST	2016	2016-10-19 00:08:00 …	closed
01d76e9f-6095-40dd…	565de4be-dc80-4849…	PST	2016	2016-10-19 00:08:00 …	closed
01d76e9f-6095-40dd…	565de4be-dc80-4849…	PST	2016	2016-10-19 00:08:00 …	closed
01d76e9f-6095-40dd…	565de4be-dc80-4849…	PST	2016	2016-10-19 00:08:00 …	closed
01d76e9f-6095-40dd…	565de4be-dc80-4849…	PST	2016	2016-10-19 00:08:00 …	closed
01d76e9f-6095-40dd…	565de4be-dc80-4849…	PST	2016	2016-10-19 00:08:00 …	closed
01d76e9f-6095-40dd…	565de4be-dc80-4849…	PST	2016	2016-10-19 00:08:00 …	closed

You'll notice that there are multiple duplicates for `gameId` in our results. It turns out our dataset isn't just giving us the results of MLB games - it's giving us the result of *every score in every game.* That's a *lot* of data!

Let's say we only care about the outcome of entire games, as opposed to every score. That seems reasonable. First things first, let's rid ourselves of all these extra columns:

```
specific_columns_df = df.select("gameId",
                                "awayTeamName",
                                "homeTeamName",
                                "durationMinutes",
                                "homeFinalRuns",
                                "awayFinalRuns")

display(specific_columns_df)
```

This will give us columns which are *only* relevant to entire games, as opposed to every score. But wait, we still have separate rows for every game! Let's make sure our dataset only has a single record per game:

```
dropped_df = specific_columns_df.dropDuplicates(subset = ['gameId'])

display(dropped_df)
```

A quick comparison:

```
print('Original column count = ', specific_columns_df.count())
print('Dropped column count = ', dropped_df.count())
```

```
Original column count = 8676
Dropped column count = 28
```

We've gone from 8676 records to 28... that sounds more reasonable.

User-Defined Functions

What if we want to transform our data in a less predictable way, like by creating our own customizations? Is that so much much to ask? It isn't, but it's significantly more difficult/convoluted than, say, Pandas. Optimizing code for speed at runtime sure is a bitch.

Custom transformations in PySpark can happen via User-Defined Functions (also known as **udf**s). As you may imagine, a user-defined function is just a function we create ourselves and apply to our DataFrame (think of Pandas' `.apply()`).

To user **udf**s, we need to import `udf` from `pyspark.sql.functions`, as well as any other imports we'll be using within that UDF. Furthermore, we'll need to import the *type* of data we're expecting to be returned from our function:

```
from pyspark.sql.functions import udf, array
from pyspark.sql.types import StringType
```

We're importing `array` because we're going to compare two values in an array we pass, with value 1 being the value in our DataFrame's `homeFinalRuns` column, and value 2 being `awayFinalRuns`. We're also importing `StringType`, because we'll be returning the *name of the team* which wins:

```
from pyspark.sql.functions import udf, array
from pyspark.sql.types import StringType

determine_winner_udf = udf(lambda arr: arr[2] if arr[0] > arr[1] else arr[3], StringType

df_with_winner = dropped_df.withColumn("winner", determine_winner_udf(array('homeFinalRu
display(df_with_winner)
```

It isn't beautiful, but it gets the job done. For each row in our DataFrame, we pass 4 values:

- The home team score.
- The away team score.
- The home team name.
- The away team name.

Our udf, `determine_winner_udf`, determines a winner from the first two array values. Depending on who wins, our lambda returns the **home team name** (arr[2]) or the **away team name** (arr[3]).

Let's see what we've got:

gameId	awayTeamName	homeTeamName	durationMinutes	homeFinalRuns	a
15557732-6fbb-481f-…	Red Sox	Indians	199	6	(
0e8fb2a4-93f4-4642-…	Nationals	Dodgers	252	3	8

gameId	awayTeamName	homeTeamName	durationMinutes	homeFinalRuns	a
c7c45139-0266-48de…	Orioles	Blue Jays	205	5	2
bac7845d-32ae-4202…	Dodgers	Nationals	272	3	4
877def36-ec67-41ee-…	Cubs	Dodgers	238	2	1
7910731d-d014-44d9…	Indians	Red Sox	221	3	4
c6949116-bd88-4b54…	Red Sox	Indians	213	5	4
681dd595-cd0f-440f-…	Nationals	Dodgers	224	6	5
d0992c0e-f771-4da5-…	Blue Jays	Rangers	210	3	5
6d62f75d-2021-46d8…	Rangers	Blue Jays	201	7	6
892f4258-cfcf-45ef-b…	Giants	Cubs	150	1	0
9dc592e4-a5c8-4222-…	Dodgers	Cubs	165	0	1
01d76e9f-6095-40dd…	Cubs	Dodgers	198	6	0
1a39d635-16f2-4b4d-	Dodgers	Cubs	156	5	0

Visualizing the Results

It seems like we've got what we wanted! We now have a dataset which can tell us the winningest team in the 2016 post-season (kind of: we're using a limited dataset, but whatever). How can we visualize this data? Why, with the Databricks built-in plot options, of course! Each time we use `display()` to show our DataFrame, we can modify the plot options to show us a chart representing our data, as opposed to a table:

Databricks plot options.

From the chart-looking dropdown, select **bar chart**. Next, check out the **plot options** button which results in this modal:

Modifying our plot options to show us winning teams.

I've messed with the settings to show us a distribution of wins above. We aggregate by `COUNT`, thus counting the number of instances where the **winner** column contains each of the team names involved.

A Moment of Reflection

You've done great, young Padawan. We took a real-life instance of some data we wanted to change, and we changed it: all in PySpark.

As you've probably noticed, working with PySpark isn't at all like working in pure-Python alternatives for modifying data. Sure, we're writing code in Python, but as we implement explicit type-setting and navigate user-defined functions, it becomes painfully evident that we're using an API that hooks into a Java application. On the bright side, we've built a respectable string of transformations which can occur across multiple nodes without writing a single line of Scala (or worse yet, *Java).*

When we consider the scalability, speed, and power of what we've just built, the little quirks don't seem so bad anymore.

Spark | Apache | Data Engineering | Big Data | ETL

Todd Birchard

110 Posts
New York City
Website
Twitter

Engineer with an ongoing identity crisis. Breaks everything before learning best practices. Completely normal and emotionally stable.

ADD A COMMENT

Performing Macro Operations on PySpark DataFrames

Spark, Apache

Working with PySpark RDDs

Spark, Apache

Manage Data Pipelines with Apache Airflow

Apache, Python

Are you into data to the point where it's almost embarrasing? Toss us your email and we'll promise to only give you the good stuff.

Your email address Send

Hackers and Slackers

Links

About
Series
Join
RSS
Donate
Sitemap

Tags

Python
Software Development
Data Engineering
Data Science
Machine Learning
DevOps
Architecture
Pandas

Authors

Matthew Alhonte
Todd Birchard
Max Mileaf
Ryan Rosado
David Aquino
Graham Beckley
David Moore