Handling Data Types

February 4, 2025

1 PySpark Basics: Handling Different Data Types

This notebook is designed for beginners to learn the basics of PySpark, focusing on handling different data types (integer, string, float, and date). We'll also add more date columns to demonstrate how different date formats are handled.

1.1 Step 1: Set Up PySpark

Before we start, we need to install and set up PySpark in the notebook.

[1]:
```
# Import SparkSession
from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder \
    .appName("PySpark Basics") \
    .getOrCreate()
```

```
25/02/02 02:44:59 WARN SparkSession: Using an existing Spark session; only
runtime SQL configurations will take effect.
```

1.1.1 Explanation:

- `SparkSession`: This is the entry point to use PySpark. It allows us to create DataFrames and interact with Spark.
- `builder`: Used to configure the Spark session.
- `appName`: Sets a name for the Spark application.
- `getOrCreate()`: Creates a new Spark session or reuses an existing one.

1.2 Step 2: Create a DataFrame

Let's create a DataFrame from the provided data. A DataFrame is a distributed collection of data organized into named columns.

[2]:
```
# Sample data
data = [
    (1, "John Doe", "Bangalore", "2023-01-15", "152.75", "True"),
    (2, "Jane Smith", "Delhi", "2023-05-20", "89.50", "False"),
    (3, "Robert Brown", "Mumbai", "InvalidDate", "200.00", "True"),
```

```
    (4, "Linda White", "Kolkata", "2023-02-29", None, "yes"),  # Feb 29 invalid␣
 ↪in 2023
    (5, "Mike Green", "Chennai", "2023-08-10", "NaN", "1"),  # NaN needs␣
 ↪handling
    (6, "Sarah Blue", "Hyderabad", "InvalidDate", "300.40", "No")
]

# Define column names
columns = ["id", "name", "city", "date", "amount", "is_active"]

# Create DataFrame
df = spark.createDataFrame(data, schema=columns)

# Show the DataFrame
df.show()
```

```
[Stage 1:====================>                                    (1 + 2) / 3]
```

```
+---+------------+---------+-----------+------+---------+
| id|        name|     city|       date|amount|is_active|
+---+------------+---------+-----------+------+---------+
|  1|    John Doe|Bangalore| 2023-01-15|152.75|     True|
|  2|  Jane Smith|    Delhi| 2023-05-20| 89.50|    False|
|  3|Robert Brown|   Mumbai|InvalidDate|200.00|     True|
|  4| Linda White|  Kolkata| 2023-02-29|  null|      yes|
|  5|  Mike Green|  Chennai| 2023-08-10|   NaN|        1|
|  6|  Sarah Blue|Hyderabad|InvalidDate|300.40|       No|
+---+------------+---------+-----------+------+---------+
```

1.2.1 Explanation:

- `data`: This is the raw data in the form of a list of tuples.
- `columns`: This is a list of column names for the DataFrame.
- `createDataFrame`: This function creates a DataFrame from the data and column names.
- `show()`: Displays the first 20 rows of the DataFrame.

1.3 Step 3: Explore Schema and Data Types

Let's check the schema of the DataFrame to understand the default data types assigned by PySpark.

[3]:
```
# Print the schema
df.printSchema()
```

```
root
 |-- id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- city: string (nullable = true)
```

```
 |-- date: string (nullable = true)
 |-- amount: string (nullable = true)
 |-- is_active: string (nullable = true)
```

1.3.1 Explanation:

- `printSchema()`: Displays the schema of the DataFrame, including column names and data types.
- By default, PySpark infers the data types. For example, `id` is inferred as `integer`, `name` as `string`, and `date` as `string`.

1.4 Step 4: Handle Integer Column (`id`)

Let's perform basic operations on the `id` column (integer type).

[4]:
```
df.id
```

[4]:
```
Column<'id'>
```

[5]:
```
df['id']
```

[5]:
```
Column<'id'>
```

[6]:
```
# Filter rows where id > 3
df.filter(df.id > 3).show()

# Add a new column with id multiplied by 2
df = df.withColumn("id_double", df.id * 2)
df.show()
```

```
+---+-----------+---------+-----------+------+---------+
| id|       name|     city|       date|amount|is_active|
+---+-----------+---------+-----------+------+---------+
|  4|Linda White|  Kolkata| 2023-02-29|  null|      yes|
|  5| Mike Green|  Chennai| 2023-08-10|   NaN|        1|
|  6| Sarah Blue|Hyderabad|InvalidDate|300.40|       No|
+---+-----------+---------+-----------+------+---------+

+---+------------+---------+-----------+------+---------+---------+
| id|        name|     city|       date|amount|is_active|id_double|
+---+------------+---------+-----------+------+---------+---------+
|  1|    John Doe|Bangalore| 2023-01-15|152.75|     True|        2|
|  2|  Jane Smith|    Delhi| 2023-05-20| 89.50|    False|        4|
|  3|Robert Brown|   Mumbai|InvalidDate|200.00|     True|        6|
|  4| Linda White|  Kolkata| 2023-02-29|  null|      yes|        8|
|  5|  Mike Green|  Chennai| 2023-08-10|   NaN|        1|       10|
|  6|  Sarah Blue|Hyderabad|InvalidDate|300.40|       No|       12|
```

```
+---+-----------+--------+----------+------+--------+---------+
```

1.4.1 Explanation:

- `filter()`: Filters rows based on a condition. Here, we filter rows where `id > 3`.
- `withColumn()`: Adds a new column or replaces an existing column. Here, we add a new column `id_double` where each value is `id * 2`.

1.5 Step 5: Handle String Column (`name` and `city`)

Let's perform basic operations on string columns.

[8]:
```
from pyspark.sql.functions import *
```

[9]:
```
# Convert name to uppercase
df = df.withColumn("name_upper", upper(df.name))
df.show()

# Filter rows where city starts with 'B'
df.filter(df.city.startswith("B")).show()
```

```
+---+------------+---------+-----------+------+---------+---------+------------+
| id|        name|     city|       date|amount|is_active|id_double|  name_upper|
+---+------------+---------+-----------+------+---------+---------+------------+
|  1|    John Doe|Bangalore| 2023-01-15|152.75|     True|        2|    JOHN DOE|
|  2|  Jane Smith|    Delhi| 2023-05-20| 89.50|    False|        4|  JANE SMITH|
|  3|Robert Brown|   Mumbai|InvalidDate|200.00|     True|        6|ROBERT BROWN|
|  4| Linda White|  Kolkata| 2023-02-29|  null|      yes|        8| LINDA WHITE|
|  5|  Mike Green|  Chennai| 2023-08-10|   NaN|        1|       10|  MIKE GREEN|
|  6|  Sarah Blue|Hyderabad|InvalidDate|300.40|       No|       12|  SARAH BLUE|
+---+------------+---------+-----------+------+---------+---------+------------+

+---+--------+---------+----------+------+---------+---------+----------+
| id|    name|     city|      date|amount|is_active|id_double|name_upper|
+---+--------+---------+----------+------+---------+---------+----------+
|  1|John Doe|Bangalore|2023-01-15|152.75|     True|        2|  JOHN DOE|
+---+--------+---------+----------+------+---------+---------+----------+
```

1.5.1 Explanation:

- `upper()`: Converts a string column to uppercase.
- `startswith()`: Filters rows where the string column starts with a specific value.

1.6 Step 6: Handle Float Column (`amount`)

Let's handle the `amount` column (float type).

[11]:
```
# Replace 'NaN' with null and cast to float
from pyspark.sql.functions import col
df = df.withColumn("amount", col("amount").cast("float"))
df.show()
```

```
+---+------------+---------+-----------+------+---------+---------+------------+
| id|        name|     city|       date|amount|is_active|id_double|  name_upper|
+---+------------+---------+-----------+------+---------+---------+------------+
|  1|    John Doe|Bangalore| 2023-01-15|152.75|     True|        2|    JOHN DOE|
|  2|  Jane Smith|    Delhi| 2023-05-20|  89.5|    False|        4|  JANE SMITH|
|  3|Robert Brown|   Mumbai|InvalidDate| 200.0|     True|        6|ROBERT BROWN|
|  4| Linda White|  Kolkata| 2023-02-29|  null|      yes|        8| LINDA WHITE|
|  5|  Mike Green|  Chennai| 2023-08-10|   NaN|        1|       10|  MIKE GREEN|
|  6|  Sarah Blue|Hyderabad|InvalidDate| 300.4|       No|       12|  SARAH BLUE|
+---+------------+---------+-----------+------+---------+---------+------------+
```

1.6.1 Explanation:

- `cast()`: Converts a column to a specific data type. Here, we convert `amount` to `float`.
- `selectExpr()`: Allows us to run SQL expressions. Here, we calculate the average of the `amount` column.

1.7 Step 7: Handle Date Column (`date`)

Let's handle the `date` column, including invalid dates. We'll also add more date columns to demonstrate different date formats.

[14]:
```
from pyspark.sql.functions import to_date

# Add more date columns with different formats
df = df.withColumn("date_alt_format", to_date(df.date, "dd-MM-yyyy"))  #␣
 ↪Alternate format
df = df.withColumn("date_with_time", to_date(df.date, "yyyy-MM-dd HH:mm:ss")) ␣
 ↪# Date with time

# Convert valid dates and handle invalid dates
df = df.withColumn("parsed_date", to_date(df.date, "yyyy-MM-dd"))

df = df.withColumn("date_alt_format", to_date(df.parsed_date, "dd-MM-yyyy"))  #␣
 ↪Alternate format
df = df.withColumn("date_with_time", to_date(df.parsed_date, "yyyy-MM-dd HH:mm:
 ↪ss"))  # Date with time


# Show the DataFrame
df.show()
```

```
# Filter rows with valid dates
df.filter(df.parsed_date.isNotNull()).show()
```

```
+---+------------+---------+-----------+------+---------+---------+------------
+--------------+-------------+-----------+
| id|        name|     city|       date|amount|is_active|id_double|
name_upper|date_alt_format|date_with_time|parsed_date|
+---+------------+---------+-----------+------+---------+---------+------------
+--------------+-------------+-----------+
|  1|    John Doe|Bangalore| 2023-01-15|152.75|     True|        2|    JOHN DOE|
2023-01-15|    2023-01-15| 2023-01-15|
|  2|  Jane Smith|    Delhi| 2023-05-20|  89.5|    False|        4|  JANE SMITH|
2023-05-20|    2023-05-20| 2023-05-20|
|  3|Robert Brown|   Mumbai|InvalidDate| 200.0|     True|        6|ROBERT BROWN|
null|          null|       null|
|  4| Linda White|  Kolkata| 2023-02-29|  null|      yes|        8| LINDA WHITE|
null|          null|       null|
|  5|  Mike Green|  Chennai| 2023-08-10|   NaN|        1|       10|  MIKE GREEN|
2023-08-10|    2023-08-10| 2023-08-10|
|  6|  Sarah Blue|Hyderabad|InvalidDate| 300.4|       No|       12|  SARAH BLUE|
null|          null|       null|
+---+------------+---------+-----------+------+---------+---------+------------
+--------------+-------------+-----------+

+---+----------+---------+----------+------+---------+---------+----------+-----
----------+-------------+-----------+
| id|      name|     city|      date|amount|is_active|id_double|name_upper|date_
alt_format|date_with_time|parsed_date|
+---+----------+---------+----------+------+---------+---------+----------+-----
----------+-------------+-----------+
|  1|  John Doe|Bangalore|2023-01-15|152.75|     True|        2|  JOHN DOE|
2023-01-15|    2023-01-15| 2023-01-15|
|  2|Jane Smith|    Delhi|2023-05-20|  89.5|    False|        4|JANE SMITH|
2023-05-20|    2023-05-20| 2023-05-20|
|  5|Mike Green|  Chennai|2023-08-10|   NaN|        1|       10|MIKE GREEN|
2023-08-10|    2023-08-10| 2023-08-10|
+---+----------+---------+----------+------+---------+---------+----------+-----
----------+-------------+-----------+
```

1.7.1 Explanation:

- `to_date()`: Converts a string column to a date column using the specified format.
- `isNotNull()`: Filters rows where the column is not null.

1.8 Step 8: Add 3 More Columns

Let's add 3 more columns to the DataFrame to showcase additional operations.

[15]:
```
from pyspark.sql.functions import lit, when

# Add a boolean column based on `is_active`
df = df.withColumn("is_active_bool", when(df.is_active.isin("True", "yes",␣
 ↪"1"), True).otherwise(False))

# Add a constant column
df = df.withColumn("constant_col", lit("PySpark"))

# Add a calculated column (amount + 10)
df = df.withColumn("amount_plus_10", df.amount + 10)

# Show the final DataFrame
df.show()
```

```
+---+------------+---------+-----------+------+---------+---------+------------
+--------------+--------------+-----------+--------------+------------+--------
------+
| id|        name|     city|       date|amount|is_active|id_double|  name_upper|
date_alt_format|date_with_time|parsed_date|is_active_bool|constant_col|amount_pl
us_10|
+---+------------+---------+-----------+------+---------+---------+------------
+--------------+--------------+-----------+--------------+------------+--------
------+
|  1|    John Doe|Bangalore| 2023-01-15|152.75|     True|        2|    JOHN DOE|
2023-01-15|    2023-01-15| 2023-01-15|          true|     PySpark|
162.75|
|  2|  Jane Smith|    Delhi| 2023-05-20|  89.5|    False|        4|  JANE SMITH|
2023-05-20|    2023-05-20| 2023-05-20|         false|     PySpark|
99.5|
|  3|Robert Brown|   Mumbai|InvalidDate| 200.0|     True|        6|ROBERT BROWN|
null|          null|       null|          true|     PySpark|         210.0|
|  4| Linda White|  Kolkata| 2023-02-29|  null|      yes|        8| LINDA WHITE|
null|          null|       null|          true|     PySpark|          null|
|  5|  Mike Green|  Chennai| 2023-08-10|   NaN|        1|       10|  MIKE GREEN|
2023-08-10|    2023-08-10| 2023-08-10|          true|     PySpark|
NaN|
|  6|  Sarah Blue|Hyderabad|InvalidDate| 300.4|       No|       12|  SARAH BLUE|
null|          null|       null|         false|     PySpark|         310.4|
+---+------------+---------+-----------+------+---------+---------+------------
+--------------+--------------+-----------+--------------+------------+--------
------+
```

1.8.1 Explanation:

- `when()`: Used for conditional logic. Here, we convert `is_active` to a boolean column.
- `lit()`: Adds a constant value to a column.

- `amount + 10`: Performs arithmetic operations on a column.

[16]:
```
spark.stop()
```