Master PySpark: From Zero to Big Data Hero!!

Key Notes on cast() and printSchema()

In PySpark, the cast() function is used to change the data type of a column within a DataFrame. This is helpful when you need to standardize column data types for data processing, schema consistency, or compatibility with other operations.

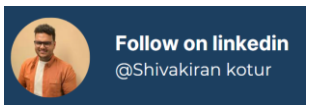
- **Purpose**: The cast() function allows you to change the data type of a column, useful in situations like standardizing formats (e.g., converting strings to dates or integers).
- **Syntax**: The cast() function is applied on individual columns and requires specifying the target data type in quotes.
- **Multiple Columns**: You can cast multiple columns at once by using a list of cast expressions and passing them to select().
- **Supported Data Types**: PySpark supports various data types for casting, including:
    - StringType
    - IntegerType (or "int")
    - DoubleType (or "double")
    - DateType
    - TimestampType
    - BooleanType
    - Others, based on the data types available in PySpark.

Basic Syntax for cast()

```
from pyspark.sql.functions import col

# Single column cast
df = df.withColumn("column_name",
col("column_name").cast("target_data_type"))

# Multiple columns cast with select
cast_expr = [
    col("column1_name").cast("target_data_type1"),
    col("column2_name").cast("target_data_type2"),
    # More columns and data types as needed
]
df = df.select(*cast_expr)
```

Follow me on LinkedIn – Shivakiran kotur

Example

Let's create a dataset and apply cast() to change the data types of multiple columns

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.types import StructType, StructField, StringType,
IntegerType, FloatType

# Initialize Spark session
spark = SparkSession.builder.appName("CastExample").getOrCreate()

# Define the schema for the dataset
schema = StructType([
    StructField("name", StringType(), True),
    StructField("age", StringType(), True),    # Stored as
StringType initially
    StructField("height", StringType(), True)   # Stored as
StringType initially
])

# Create a sample dataset
data = [
    ("Alice", "25", "5.5"),
    ("Bob", "35", "6.1"),
    ("Charlie", "40", "5.8"),
]

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

# Assuming you have already created a DataFrame 'df'
df.printSchema()

df.show()
```

```
root
 |-- name: string (nullable = true)
 |-- age: string (nullable = true)
 |-- height: string (nullable = true)

+-------+---+------+
|   name|age|height|
+-------+---+------+
|  Alice| 25|   5.5|
|    Bob| 35|   6.1|
|Charlie| 40|   5.8|
+-------+---+------+
```

```
# Define cast expressions for multiple columns
cast_expr = [
    col("name").cast("string"),
    col("age").cast("int"),        # Casting age to IntegerType
    col("height").cast("double")   # Casting height to DoubleType
]

# Apply the cast expressions to the DataFrame
df = df.select(*cast_expr)

# Show the result
df.printSchema()
df.show()
```

```
root
 |-- name: string (nullable = true)
 |-- age: integer (nullable = true)
 |-- height: double (nullable = true)

+-------+---+------+
|   name|age|height|
+-------+---+------+
|  Alice| 25|   5.5|
|    Bob| 35|   6.1|
|Charlie| 40|   5.8|
+-------+---+------+
```

Explanation

- **"age" column**: Initially stored as StringType, it's cast to IntegerType (or "int").
- **"height" column**: Initially stored as StringType, it's cast to DoubleType (or "double").

Advantages of Using cast()

- **Schema Alignment**: Ensures data types in different tables or DataFrames are compatible for joining or union operations.
- **Data Consistency**: Ensures all columns conform to expected data types for downstream data processing.
- **Error Reduction**: Minimizes issues arising from mismatched data types in computations or transformations.

This approach using cast() provides a flexible and powerful way to manage data types in PySpark.

printSchema() Method in PySpark

- **Purpose**:
  - To display the schema of a DataFrame, which includes the column names, data types, and nullability of each column.
- **Output Structure**:
  - The schema is presented in a tree-like structure showing:
    - **Column Name**: The name of the column.
    - **Data Type**: The data type of the column (e.g., string, integer, double, boolean, etc.).
    - **Nullability**: Indicates whether the column can contain null values (e.g., nullable = true).

Usage:

- Call df.printSchema() on a DataFrame df to see its structure.
- Useful for verifying the structure of the DataFrame after operations like select(), withColumn(), or cast().