Master PySpark: From Zero to Big Data Hero!!

Null Handling in Dataframe

Here's an example of how you can use PySpark functions for null handling with sales data. The code includes null detection, dropping rows with nulls, filling null values, and using coalesce() to handle nulls in aggregations. I will provide the notes alongside the code.

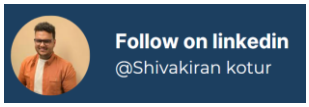
Sample Sales Data with Null Values

```
# Sample data: sales data with nulls
data = [
    ("John", "North", 100, None),
    ("Doe", "East", None, 50),
    (None, "West", 150, 30),
    ("Alice", None, 200, 40),
    ("Bob", "South", None, None),
    (None, None, None, None)
]
columns = ["Name", "Region", "UnitsSold", "Revenue"]
# Create DataFrame
df = spark.createDataFrame(data, columns)
df.show()
```

```
+-----+------+---------+-------+
| Name|Region|UnitsSold|Revenue|
+-----+------+---------+-------+
| John| North|      100|   null|
|  Doe|  East|     null|     50|
| null|  West|      150|     30|
|Alice|  null|      200|     40|
|  Bob| South|     null|   null|
| null|  null|     null|   null|
+-----+------+---------+-------+
```

Notes:

1. **Detecting Null Values**:
   The isNull() function identifies rows where a specified column has null values. The output shows a boolean flag for each row to indicate whether the value in the column is null.



```
from pyspark.sql.functions import *
# Detecting Null Values in the "Region" Column
df.select("Name", "Region", isnull("Region").alias("is_Region_Null")).show()
```

▸ (3) Spark Jobs

```
+-----+------+--------------+
| Name|Region|is_Region_Null|
+-----+------+--------------+
| John| North|         false|
|  Doe|  East|         false|
| null|  West|         false|
|Alice|  null|          true|
|  Bob| South|         false|
| null|  null|          true|
+-----+------+--------------+
```

2. Dropping Rows with Null Values:

- **dropna()** removes rows that contain null values in any column when the default mode is used.
- Specifying "all" ensures rows are only removed if *all* columns contain null values.
- You can also apply null handling only on specific columns by providing a list of column names to the subset parameter.

```
# Dropping Rows with Null Values (if any value in the row is null)
df2 = df.dropna()
df2.show()
```

▸ (3) Spark Jobs

▸ df2: pyspark.sql.dataframe.DataFrame = [Name: string, Region: string ... 2 more fields]

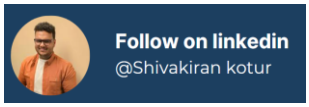
```
+----+------+---------+-------+
|Name|Region|UnitsSold|Revenue|
+----+------+---------+-------+
+----+------+---------+-------+
```

```
# Dropping Rows where all values are Null
df3 = df.na.drop("all")
df3.show()
```

▸ (3) Spark Jobs

▸ df3: pyspark.sql.dataframe.DataFrame = [Name: stri

```
+-----+------+---------+-------+
| Name|Region|UnitsSold|Revenue|
+-----+------+---------+-------+
| John| North|      100|   null|
|  Doe|  East|     null|     50|
| null|  West|      150|     30|
|Alice|  null|      200|     40|
|  Bob| South|     null|   null|
+-----+------+---------+-------+
```



Follow me on LinkedIn – Shivakiran kotur

```
# Dropping Rows if null values exist in "Name" or "Region" columns
df4 =df.na.drop("all", subset=["Name", "Region"])
df4.show()
```

(3) Spark Jobs

df4: pyspark.sql.dataframe.DataFrame = [Name: string, Region: string ... 2 more fields]

```
+-----+------+---------+-------+
| Name|Region|UnitsSold|Revenue|
+-----+------+---------+-------+
| John| North|      100|   null|
|  Doe|  East|     null|     50|
| null|  West|      150|     30|
|Alice|  null|      200|     40|
|  Bob| South|     null|   null|
+-----+------+---------+-------+
```

3. **Filling Null Values**:
   - **fillna()** allows replacing null values with specified replacements, either for all columns or selectively.
   - In the example, nulls in Region are replaced with "Unknown", while UnitsSold and Revenue nulls are filled with 0.

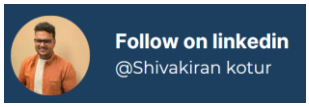
09:28 PM (1s) 6

```
# Filling Null Values with Specific Values
df5 = df.fillna({"Region": "Unknown", "UnitsSold": 0, "Revenue": 0})
df5.show()
```

(3) Spark Jobs

df5: pyspark.sql.dataframe.DataFrame = [Name: string, Region: string ... 2 more fields]

```
+-----+-------+---------+-------+
| Name| Region|UnitsSold|Revenue|
+-----+-------+---------+-------+
| John|  North|      100|      0|
|  Doe|   East|        0|     50|
| null|   West|      150|     30|
|Alice|Unknown|      200|     40|
|  Bob|  South|        0|      0|
| null|Unknown|        0|      0|
+-----+-------+---------+-------+
```



```
# Filling all Null values in "Region" and "Name" columns
df6 = df.na.fill("N/A", subset=["Name", "Region"])
df6.show()
```

▸ (3) Spark Jobs

▸ df6: pyspark.sql.dataframe.DataFrame = [Name: string, Region: strin

```
+-----+------+---------+-------+
| Name|Region|UnitsSold|Revenue|
+-----+------+---------+-------+
| John| North|      100|   null|
|  Doe|  East|     null|     50|
|  N/A|  West|      150|     30|
|Alice|   N/A|      200|     40|
|  Bob| South|     null|   null|
|  N/A|   N/A|     null|   null|
+-----+------+---------+-------+
```

4. **Coalesce Function**:

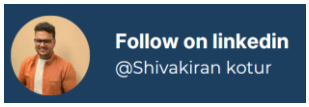
   The **coalesce()** function returns the first non-null value in a list of columns. It's useful when you need to handle missing data by providing alternative values from other columns.

```
# Using coalesce() to handle nulls by taking the first non-null value
df7 = df.withColumn("Adjusted_UnitsSold", coalesce("UnitsSold", "Revenue"))
df7.show()
```

▸ (3) Spark Jobs

▸ df7: pyspark.sql.dataframe.DataFrame = [Name: string, Region: string ... 3 more fields]

```
+-----+------+---------+-------+------------------+
| Name|Region|UnitsSold|Revenue|Adjusted_UnitsSold|
+-----+------+---------+-------+------------------+
| John| North|      100|   null|               100|
|  Doe|  East|     null|     50|                50|
| null|  West|      150|     30|               150|
|Alice|  null|      200|     40|               200|
|  Bob| South|     null|   null|              null|
| null|  null|     null|   null|              null|
+-----+------+---------+-------+------------------+
```

Handling Nulls in Aggregations:

Null values can distort aggregate functions like mean(). Using coalesce() in an aggregation ensures that any null values are replaced with a default (e.g., 0.0) to avoid skewing the results.

```
# Aggregating while handling null values using coalesce
df8 = df.groupBy("Region").agg(coalesce(mean("UnitsSold"), lit(0)).alias("Avg_UnitsSold"))
df8.show()
```

▸ (2) Spark Jobs

▸ df8: pyspark.sql.dataframe.DataFrame = [Region: string, Avg_UnitsSold: double]

```
+------+-------------+
|Region|Avg_UnitsSold|
+------+-------------+
| North|        100.0|
|  East|          0.0|
|  West|        150.0|
|  null|        200.0|
| South|          0.0|
+------+-------------+
```

Null Handling in DataFrames - Summary

1. **Detecting Nulls**: Use isNull() to identify null values in specific columns.
2. **Dropping Nulls**: dropna() removes rows with null values, either in any or all columns. You can target specific columns using the subset parameter.
3. **Filling Nulls**: fillna() replaces nulls with specified default values, either for all or selected columns.
4. **Coalesce Function**: coalesce() returns the first non-null value from multiple columns, providing a fallback when some columns contain nulls.
5. **Aggregations**: Use coalesce() during aggregations like mean() to handle nulls by substituting them with defaults (e.g., 0), ensuring accurate results.