Must Know Pyspark Coding Before Your Next Databricks Interview

Document by – Siddhartha Subudhi

Visit my LinkedIn profile

1. Find the second highest salary in a DataFrame using PySpark.

Scenario: You have a DataFrame of employee salaries and want to find the second highest salary.

```
from pyspark.sql import Window
from pyspark.sql.functions import col, dense_rank

windowSpec = Window.orderBy(col("salary").desc())
df_with_rank = df.withColumn("rank", dense_rank().over(windowSpec))
second_highest_salary = df_with_rank.filter(col("rank") == 2).select("salary")
second_highest_salary.show()
```

---

2. Count the number of null values in each column of a PySpark DataFrame.

Scenario: Given a DataFrame, identify how many null values each column contains.

```
from pyspark.sql.functions import col, isnan, when, count

df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

---

3. Calculate the moving average over a window of 3 rows.

Scenario: For a stock price dataset, calculate a moving average over the last 3 days.

```
from pyspark.sql import Window
from pyspark.sql.functions import avg

windowSpec = Window.orderBy("date").rowsBetween(-2, 0)
df_with_moving_avg = df.withColumn("moving_avg", avg("price").over(windowSpec))
df_with_moving_avg.show()
```

---

4. Remove duplicate rows based on a subset of columns in a PySpark DataFrame.

Scenario: You need to remove duplicates from a DataFrame based on certain columns.

```
df = df.dropDuplicates(["column1", "column2"])
df.show()
```

5. Split a single column with comma-separated values into multiple columns.

Scenario: Your DataFrame contains a column with comma-separated values. You want to split this into multiple columns.

```
from pyspark.sql.functions import split


df_split = df.withColumn("new_column1", split(df["column"], ",").getItem(0)) \
        .withColumn("new_column2", split(df["column"], ",").getItem(1))
df_split.show()
```

---

6. Group data by a specific column and calculate the sum of another column.

Scenario: Group sales data by "product" and calculate the total sales.

```
df.groupBy("product").sum("sales").show()
```

---

7. Join two DataFrames on a specific condition.

Scenario: You have two DataFrames: one for customer data and one for orders. Join these DataFrames on the customer ID.

```
df_joined = df_customers.join(df_orders, df_customers.customer_id == df_orders.customer_id, "inner")
df_joined.show()
```

---

8. Create a new column based on conditions from existing columns.

Scenario: Add a new column "category" that assigns "high", "medium", or "low" based on the value of the "sales" column.

```
from pyspark.sql.functions import when
df = df.withColumn("category", when(df.sales > 500, "high")
                   .when((df.sales <= 500) & (df.sales > 200), "medium")
                   .otherwise("low"))
df.show()
```

9. Calculate the percentage contribution of each value in a column to the total.

Scenario: For a sales dataset, calculate the percentage contribution of each product's sales to the total sales.

```
from pyspark.sql.functions import sum, col
total_sales = df.agg(sum("sales").alias("total_sales")).collect()[0]["total_sales"]
df = df.withColumn("percentage", (col("sales") / total_sales) * 100)
df.show()
```

---

10. Find the top N records from a DataFrame based on a column.

Scenario: You need to find the top 5 highest-selling products.

```
df.orderBy(col("sales").desc()).limit(5).show()
```

---

11. Write PySpark code to pivot a DataFrame.

Scenario: You have sales data by "year" and "product", and you want to pivot the table to show "product" sales by year.

```
df_pivot = df.groupBy("product").pivot("year").sum("sales")
df_pivot.show()
```

---

12. Add row numbers to a PySpark DataFrame based on a specific ordering.

Scenario: Add row numbers to a DataFrame ordered by "sales" in descending order.

```
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
windowSpec = Window.orderBy(col("sales").desc())
df_with_row_number = df.withColumn("row_number", row_number().over(windowSpec))
df_with_row_number.show()
```

13. Filter rows based on a condition.

Scenario: You want to filter only those customers who made purchases over ₹1000.

```
df_filtered = df.filter(df.purchase_amount > 1000)
df_filtered.show()
```

---

14. Flatten a JSON column in PySpark.

Scenario: Your DataFrame contains a JSON column, and you want to extract specific fields from it.

```
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StructField, StringType
schema = StructType([
   StructField("name", StringType(), True),
   StructField("age", StringType(), True)
])
df = df.withColumn("json_data", from_json(col("json_column"), schema))
df.select("json_data.name", "json_data.age").show()
```

---

15. Convert a PySpark DataFrame column to a list.

Scenario: Convert a column from your DataFrame into a list for further processing.

```
column_list = df.select("column_name").rdd.flatMap(lambda x: x).collect()
```

---

16. Handle NULL values by replacing them with a default value.

Scenario: Replace all NULL values in the "sales" column with 0.

```
df = df.na.fill({"sales": 0})
df.show()
```

---

17. Perform a self-join on a PySpark DataFrame.

Scenario: You have a hierarchy of employees and want to find each employee's manager.

```
df_self_join = df.alias("e1").join(df.alias("e2"), col("e1.manager_id") == col("e2.employee_id"), "inner") \
        .select(col("e1.employee_name"), col("e2.employee_name").alias("manager_name"))
df_self_join.show()
```


18. Write PySpark code to unpivot a DataFrame.

Scenario: You have a DataFrame with "year" columns and want to convert them to rows.

```
from pyspark.sql.functions import expr
df_unpivot = df.selectExpr("id", "stack(2, '2021', sales_2021, '2022', sales_2022) as (year, sales)")
df_unpivot.show()
```

---

19. Write a PySpark code to group data based on multiple columns and calculate aggregate functions.

Scenario: Group data by "product" and "region" and calculate the average sales for each group.

```
df.groupBy("product", "region").agg({"sales": "avg"}).show()
```

---

20. Write PySpark code to remove rows with duplicate values in any column.

Scenario: You want to remove rows where any column has duplicate values.

```
df_cleaned = df.dropDuplicates()
df_cleaned.show()
```

---

21. Write PySpark code to read a CSV file and infer its schema.

Scenario: You need to load a CSV file into a DataFrame, ensuring the schema is inferred.

```
df = spark.read.option("header", "true").option("inferSchema", "true").csv("path_to_csv")
df.show()
```

---

22. Write PySpark code to merge multiple small files into a single file.

Scenario: You have multiple small files in HDFS, and you want to consolidate them into one large file.

```
df.coalesce(1).write.mode("overwrite").csv("output_path")
```

23. Write PySpark code to calculate the cumulative sum of a column.

Scenario: You want to calculate a cumulative sum of sales in your DataFrame.

```
from pyspark.sql.window import Window
from pyspark.sql.functions import sum

windowSpec = Window.orderBy("date").rowsBetween(Window.unboundedPreceding, 0)
df_with_cumsum = df.withColumn("cumulative_sum", sum("sales").over(windowSpec))
df_with_cumsum.show()
```

---

24. Write PySpark code to find outliers in a dataset.

Scenario: Detect outliers in the "sales" column based on the 1.5 * IQR rule.

```
from pyspark.sql.functions import expr
q1 = df.approxQuantile("sales", [0.25], 0.01)[0]
q3 = df.approxQuantile("sales", [0.75], 0.01)[0]
iqr = q3 - q1
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr
df_outliers = df.filter((col("sales") < lower_bound) | (col("sales") > upper_bound))
df_outliers.show()
```

25. Write PySpark code to convert a DataFrame to a Pandas DataFrame.

Scenario: Convert your PySpark DataFrame into a Pandas DataFrame for local processing.

```
pandas_df = df.toPandas()
```