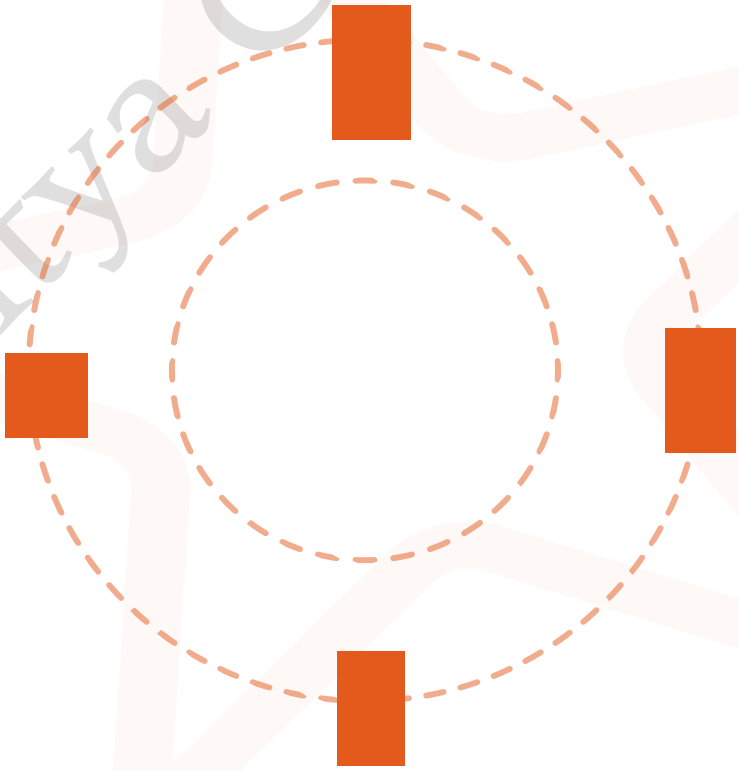
Nitya CloudTech Pvt Ltd.

PySpark Scenario-Based Interview Questions & Answers



Data Transformation & Manipulation

Interviewer: Given a DataFrame with columns `id`, `name`, and `age`, write a PySpark code to find the average age per name. How would you handle missing values in the `age` column?

Candidate: Sure! To find the average age per name, I'd use the `groupBy()` function on the `name` column, followed by `avg()` on the `age` column. To handle missing values in the `age` column, I'd use `fillna()` to replace null values with a specified default, or `dropna()` to remove rows with null values in the `age` column, depending on the requirements.

Here's the code:

```
from pyspark.sql import functions as F

# Filling null values in age column with 0 or a specified
default value
df = df.fillna({'age': 0})

# Grouping by name and calculating average age
avg_age_df =
df.groupBy("name").agg(F.avg("age").alias("avg_age"))
avg_age_df.show()
```

Interviewer: You have a DataFrame with duplicated rows based on multiple columns. How would you remove duplicates while keeping the first occurrence of each?

Candidate: I would use the `dropDuplicates()` function and specify the columns based on which duplicates are identified. This function keeps the first occurrence by default.

Example:

```
# Removing duplicates based on specified columns
df = df.dropDuplicates(["column1", "column2"])
```

Interviewer: How would you implement a window function to calculate the cumulative sum of a column called `sales` within each `region`?

Candidate: I'd use the `Window` function and partition the data by `region`. Then, I'd apply an order by operation, if needed, and use `F.sum("sales").over(window_spec)` to calculate the cumulative sum.

Here's the code:

```
from pyspark.sql import Window
from pyspark.sql import functions as F

# Defining the window specification for cumulative sum
window_spec =
Window.partitionBy("region").orderBy("sales").rowsBetween(Wind
ow.unboundedPreceding, Window.currentRow)

# Adding the cumulative sum column
df = df.withColumn("cumulative_sales",
F.sum("sales").over(window_spec))
df.show()
```

Interviewer: You need to pivot a DataFrame to convert rows into columns. Given columns `country`, `year`, and `population`, demonstrate how to create a new column for each year with its population.

Candidate: I'd use the `pivot()` function on the `year` column and apply `sum()` or `first()` to aggregate the population values.

Here's the code:

```
# Pivoting the DataFrame
pivot_df =
df.groupBy("country").pivot("year").sum("population")
pivot_df.show()
```

Interviewer: How would you use PySpark to split a single `full_name` column into `first_name` and `last_name`?

Candidate: I'd use the `split()` function from PySpark to separate the `full_name` column by a space and extract the parts into `first_name` and `last_name` columns.

Example:

```
# Splitting full_name into first_name and last_name
df = df.withColumn("first_name", F.split(F.col("full_name"), " ")[0])
df = df.withColumn("last_name", F.split(F.col("full_name"), " ")[1])
df.show()
```

Interviewer: Describe how to concatenate multiple columns into a single column, separated by a hyphen, without any null values in the result.

Candidate: I'd first filter out null values in the columns to be concatenated using `coalesce()` or `fillna()`, then concatenate them with `concat_ws()` to add the hyphen.

Here's the code:

```
# Concatenating columns with hyphen as separator and handling nulls
df = df.withColumn("combined_column", F.concat_ws("-", F.coalesce("column1", F.lit("")), F.coalesce("column2", F.lit(""))))
df.show()
```

Interviewer: Explain how you would handle a scenario where you need to join two large DataFrames on a specific column, but the column has many null values.

Candidate: For large DataFrames, I'd first filter out rows with null values in the join column on both sides, using `dropna()`. If keeping nulls is essential, I'd consider filling them with unique placeholder values to avoid erroneous joins.

Example:

```
# Dropping rows with null join keys
df1 = df1.dropna(subset=["join_column"])
df2 = df2.dropna(subset=["join_column"])

# Performing the join
joined_df = df1.join(df2, "join_column", "inner")
joined_df.show()
```

Interviewer: How would you use PySpark to unnest JSON data stored in a single column into multiple columns?

Candidate: If the column has JSON objects, I'd use `from_json()` to parse the JSON into a struct and then `select()` the fields individually to unnest them.

Example:

```
from pyspark.sql.types import StructType, StructField,
StringType, IntegerType
from pyspark.sql import functions as F

# Defining schema for JSON data
schema = StructType([StructField("field1", StringType(),
True), StructField("field2", IntegerType(), True)])

# Parsing and selecting JSON fields
df = df.withColumn("json_data",
F.from_json(F.col("json_column"), schema))
df = df.select("json_data.*")
df.show()
```

Interviewer: You have a column with comma-separated values in a DataFrame. How would you split these values into separate rows?

Candidate: I'd use `split()` to convert the comma-separated values into an array and then `explode()` to create separate rows for each value.

Example:

```
#
Splitting and
exploding the column
df = df.withColumn("split_column",
F.split(F.col("comma_separated_column"), ","))
df = df.select(F.col("id"),
F.explode(F.col("split_column")).alias("separated_value"))
df.show()
```

Interviewer: How would you filter out rows with null values in more than two specific columns in a PySpark DataFrame?

Candidate: I'd use `isNotNull()` on each column and then combine these conditions to filter rows that satisfy at least two non-null conditions.

Example:

```
# Filtering rows with at least two non-null values
df = df.filter((F.col("col1").isNotNull() &
F.col("col2").isNotNull()) | (F.col("col1").isNotNull() &
F.col("col3").isNotNull()) | (F.col("col2").isNotNull() &
F.col("col3").isNotNull()))
df.show()
```

Data Aggregation & Analysis

Interviewer: Explain how you would compute a rolling average for a specific metric over a 7-day window.

Candidate: To compute a rolling average over a 7-day window, I'd use the `Window` function, specifying the range as 6 days before the current day and including the current row. Then, I'd use `avg()` to compute the rolling average within that window.

Example:

```
from pyspark.sql import Window
from pyspark.sql import functions as F

# Define the window specification for a 7-day rolling average
window_spec = Window.orderBy("date").rowsBetween(-6, 0)
```

```
#
Calculate the 7-day rolling average for a specific metric
df = df.withColumn("rolling_avg",
F.avg("metric_column").over(window_spec))
df.show()
```

Interviewer: How would you group data by two columns, such as `city` and `product`, and then calculate the average sales for each combination?

Candidate: I'd use `groupBy()` on both `city` and `product` columns and then apply `avg()` to calculate the average sales for each combination.

Example:

```
# Group by city and product, and calculate average sales
df = df.groupBy("city",
"product").agg(F.avg("sales").alias("average_sales"))
df.show()
```

Interviewer: How would you calculate the percentile rank of a column, such as `salary`, within a DataFrame partitioned by department?

Candidate: I'd use the `percent_rank()` window function, partitioned by `department`, and ordered by `salary`. This will assign a percentile rank to each row within each department.

Example:

```
from pyspark.sql import Window
from pyspark.sql import functions as F

# Define the window specification
window_spec =
Window.partitionBy("department").orderBy("salary")

# Calculate percentile rank for salary
df = df.withColumn("salary_percentile",
F.percent_rank().over(window_spec))
df.show()
```

Interviewer:
Given a sales dataset, explain how to identify the top 5 products with the highest revenue per region.

Candidate: I'd partition the data by `region`, order by `revenue` in descending order, and use `row_number()` to assign a rank to each product within each region. Then, I'd filter for rows where the rank is 5 or below.

Example:

```
from pyspark.sql import Window
from pyspark.sql import functions as F

# Define window to rank products by revenue within each region
window_spec =
Window.partitionBy("region").orderBy(F.desc("revenue"))

# Rank products and filter top 5 per region
df = df.withColumn("rank",
F.row_number().over(window_spec)).filter(F.col("rank") <= 5)
df.show()
```

Interviewer: You have a dataset with `purchase_date` and `customer_id`. How would you calculate the average time between purchases per customer?

Candidate: I'd first create a lagged version of `purchase_date` for each `customer_id` using `lag()`. Then, I'd calculate the difference in days between each purchase and the previous one, and finally, take the average for each customer.

Example:

```
from pyspark.sql import Window
from pyspark.sql import functions as F

# Define window for each customer
window_spec =
Window.partitionBy("customer_id").orderBy("purchase_date")
```

```
#
Calculate time
difference between purchases
df = df.withColumn("previous_purchase_date",
F.lag("purchase_date").over(window_spec))
df = df.withColumn("days_between_purchases",
F.datediff("purchase_date", "previous_purchase_date"))

# Calculate average time between purchases per customer
avg_time_df =
df.groupBy("customer_id").agg(F.avg("days_between_purchases").
alias("avg_days_between_purchases"))
avg_time_df.show()
```

Interviewer: Describe how you would create a summary report showing the maximum, minimum, and average values for each numeric column in a DataFrame.

Candidate: I'd use the `agg()` function with `max()`, `min()`, and `avg()` for each numeric column to create a summary report.

Example:

```
# Creating a summary report
summary_df = df.agg(
    F.max("column1").alias("max_column1"),
    F.min("column1").alias("min_column1"),
    F.avg("column1").alias("avg_column1"),
    F.max("column2").alias("max_column2"),
    F.min("column2").alias("min_column2"),
    F.avg("column2").alias("avg_column2")
)
summary_df.show()
```

Interviewer: Explain how to identify the number of unique customers who made purchases over the last month, grouped by product category.

Candidate: I'd filter the data to include only the last month's records based on `purchase_date`, then group by `product_category`, and use `countDistinct()` on `customer_id` to get the unique count of customers.

Example:

```
from pyspark.sql import functions as F

# Filter for last month's data
last_month_df = df.filter(F.col("purchase_date") >= "2024-10-
01")  # Assuming today is Nov 1, 2024

# Group by product category and count distinct customers
unique_customers_df =
last_month_df.groupBy("product_category").agg(F.countDistinct(
"customer_id").alias("unique_customers"))
unique_customers_df.show()
```

Interviewer: In a retail dataset, how would you calculate the monthly revenue trend for each product?

Candidate: I'd extract the month and year from the `purchase_date` column, group by `product` and the extracted month, and then use `sum()` on the `revenue` column.

Example:

```
# Extracting month and year, grouping by product and month
monthly_revenue_df = df.withColumn("month",
F.date_format("purchase_date", "yyyy-MM")).groupBy("product",
"month").agg(F.sum("revenue").alias("monthly_revenue"))
monthly_revenue_df.show()
```

Interviewer: You have sales data from different regions. Explain how you would identify regions with an increasing sales trend over the last three quarters.

Candidate: I'd calculate the total sales per region for each of the last three quarters. Then, I'd use a window function to create a column showing whether each quarter's sales are greater than the previous one. Finally, I'd filter regions where sales consistently increase.

Example:

```
from pyspark.sql import Window
```

```
from pyspark.sql import functions as F

# Extract quarter and year
df = df.withColumn("quarter", F.quarter("sales_date")).withColumn("year", F.year("sales_date"))

# Calculate quarterly sales per region
quarterly_sales_df = df.groupBy("region", "year", "quarter").agg(F.sum("sales").alias("total_sales"))

# Define a window to compare current and previous quarter sales
window_spec = Window.partitionBy("region").orderBy("year", "quarter")

# Calculate the difference between current and previous quarter's sales
quarterly_sales_df = quarterly_sales_df.withColumn("sales_increase", F.when(F.col("total_sales") > F.lag("total_sales").over(window_spec), 1).otherwise(0))

# Filter regions with consistent sales increase over last three quarters
increasing_sales_regions = quarterly_sales_df.groupBy("region").agg(F.sum("sales_increase").alias("increase_count")).filter(F.col("increase_count") == 3)
increasing_sales_regions.show()
```

Interviewer: Given a column of scores, how would you assign grades such as A, B, C, and D based on ranges using PySpark?

Candidate: I'd use `when()` and `otherwise()` to define the ranges for each grade and create a new column for the assigned grade.

Example:

```
from pyspark.sql import functions as F

# Assigning grades based on score ranges
df = df.withColumn(
    "grade",
    F.when(F.col("score") >= 90, "A")
```

```
.when((F.col("score") >= 80) & (F.col("score") < 90), "B")
    .when((F.col("score") >= 70) & (F.col("score") < 80), "C")
    .otherwise("D")
)
df.show()
```

If you like the content, please consider supporting us by sharing it with others who may benefit. Your support helps us continue creating valuable resources!

Scan any QR using PhonePe App

Aditya chandak