VISION BOARD

# PYSPARK INTERVIEW Q&A

DEVIKRISHNA R

LinkedIn: @Devikrishna R Email: visionboard044@gmail.com

1. Create a DataFrame for an Indian Employee Database
Sample Data:
data = [
(1, "Amit", "IT", 60000),
(2, "Priya", "HR", 55000),
(3, "Rahul", "Finance", 75000),
(4, "Sneha", "IT", 80000),
(5, "Karan", "HR", 65000)]
columns = ["EmpID", "Name", "Department", "Salary"]
df = spark.createDataFrame(data, columns)

Task: Display the schema and first 3 rows.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, avg, count, sum, max
# Initialize Spark session
spark = SparkSession.builder.appName("EmployeeDataAnalysis"
    ).getOrCreate()
# Sample Data
data = [
    (1, "Amit", "IT", 60000),
    (2, "Priya", "HR", 55000),
    (3, "Rahul", "Finance", 75000),
    (4, "Sneha", "IT", 80000),
    (5, "Karan", "HR", 65000)
]
columns = ["EmpID", "Name", "Department", "Salary"]
# Create DataFrame
df = spark.createDataFrame(data, columns)
# Display schema
df.printSchema()
# Display first 3 rows
df.show(3)
```

**OUTPUT**

```
root
 |-- EmpID: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Department: string (nullable = true)
 |-- Salary: integer (nullable = true)

+-----+-----+----------+------+
|EmpID|Name |Department|Salary|
+-----+-----+----------+------+
|1    |Amit |IT        |60000 |
|2    |Priya|HR        |55000 |
|3    |Rahul|Finance   |75000 |
+-----+-----+----------+------+
```

2. Filter Employees Earning More than 70,000

Task: Write a query to filter employees earning more than ₹70,000.

```
df.filter(col("Salary") > 70000).show()
```

**OUTPUT**

```
+-----+------+----------+------+
|EmpID|Name  |Department|Salary|
+-----+------+----------+------+
|3    |Rahul |Finance   |75000 |
|4    |Sneha |IT        |80000 |
+-----+------+----------+------+
```

3. Calculate Average Salary per Department

Task: Use `groupBy` to get the average salary for each department.

```
df.groupBy("Department").agg(avg("Salary").alias("Avg_Salary")).show()
```

**OUTPUT**

```
+----------+----------+
|Department|Avg_Salary|
+----------+----------+
|IT        |70000.0   |
|HR        |60000.0   |
|Finance   |75000.0   |
+----------+----------+
```

4. Find Employees whose Name Starts with 'A'

Task: Filter employees whose names start with the letter 'A'.

```
df.filter(col("Name").startswith("A")).show()
```

**OUTPUT**

```
+-----+----+----------+------+
|EmpID|Name|Department|Salary|
+-----+----+----------+------+
|1    |Amit|IT        |60000 |
+-----+----+----------+------+
```

5. Count the Number of Employees per Department

Task: Use `groupBy` and `count()` to find the number of employees in each department.

```
df.groupBy("Department").agg(count("EmpID").alias("Employee_Count")).show()
```

**OUTPUT**

```
+----------+--------------+
|Department|Employee_Count|
+----------+--------------+
|IT        |2             |
|HR        |2             |
|Finance   |1             |
+----------+--------------+
```

6. Add a New Column for Tax Deduction (10% of Salary)

Task: Add a new column `Tax` that deducts 10% from `Salary`.

```
df = df.withColumn("Tax", col("Salary") * 0.10)
df.show()
```

**OUTPUT**

```
+-----+------+----------+------+------+
|EmpID|Name  |Department|Salary|Tax   |
+-----+------+----------+------+------+
|1    |Amit  |IT        |60000 |6000.0|
|2    |Priya |HR        |55000 |5500.0|
|3    |Rahul |Finance   |75000 |7500.0|
|4    |Sneha |IT        |80000 |8000.0|
|5    |Karan |HR        |65000 |6500.0|
+-----+------+----------+------+------+
```

7. Sort Employees by Salary in Descending Order

Task: Display employees sorted in descending order of salary.

```
df.orderBy(col("Salary").desc()).show()
```

**OUTPUT**

```
+-----+------+----------+------+------+
|EmpID|Name  |Department|Salary|Tax   |
+-----+------+----------+------+------+
|4    |Sneha |IT        |80000 |8000.0|
|3    |Rahul |Finance   |75000 |7500.0|
|5    |Karan |HR        |65000 |6500.0|
|1    |Amit  |IT        |60000 |6000.0|
|2    |Priya |HR        |55000 |5500.0|
+-----+------+----------+------+------+
```

8. Get the Second Highest Salary

Task: Find the second highest salary without using `LIMIT` and `OFFSET`.

```
second_highest_salary = df.orderBy(col("Salary").desc()).limit(2).collect()[-1]["Salary"]
df.filter(col("Salary") == second_highest_salary).show()
```

**OUTPUT**

```
+-----+-----+----------+------+------+
|EmpID|Name |Department|Salary|Tax   |
+-----+-----+----------+------+------+
|3    |Rahul|Finance   |75000 |7500.0|
+-----+-----+----------+------+------+
```

9. Get Employees Who are in the HR or IT Department

Task: Filter records where the department is either "HR" or "IT".

```
df.filter((col("Department") == "HR") | (col("Department") == "IT")).show()
```

**OUTPUT**

```
+-----+------+----------+------+
|EmpID|Name  |Department|Salary|
+-----+------+----------+------+
|1    |Amit  |IT        |60000 |
|2    |Priya |HR        |55000 |
|4    |Sneha |IT        |80000 |
|5    |Karan |HR        |65000 |
+-----+------+----------+------+
```

10. Find the Total Salary Paid by the Company

Task: Calculate the sum of all salaries.

```
df.agg(sum("Salary").alias("Total_Salary")).show()
```

**OUTPUT**

```
+------------+
|Total_Salary|
+------------+
|335000      |
+------------+
```

11. Read a CSV File of Cricket Players
Sample CSV (`players.csv`):
`csv
Player,Country,Runs,Wickets
Virat Kohli,India,12000,4
Rohit Sharma,India,11000,8
Jasprit Bumrah,India,1200,200
Steve Smith,Australia,9500,20

Task: Read this CSV file into a DataFrame and display its contents.

```
players_df = spark.read.csv("players.csv", header=True, inferSchema=True)
players_df.show()
```

**OUTPUT**

```
+-------------+---------+-----+-------+
|Player       |Country  |Runs |Wickets|
+-------------+---------+-----+-------+
|Virat Kohli  |India    |12000|4      |
|Rohit Sharma |India    |11000|8      |
|Jasprit Bumrah|India   |1200 |200    |
|Steve Smith  |Australia|9500 |20     |
+-------------+---------+-----+-------+
```

12. Find the Player with Maximum Runs

Task: Find the player who has scored the maximum runs.

```
players_df.orderBy(col("Runs").desc()).limit(1).show()
```

**OUTPUT**

```
+-------------+---------+-----+-------+
|Player       |Country  |Runs |Wickets|
+-------------+---------+-----+-------+
|Virat Kohli  |India    |12000|4      |
+-------------+---------+-----+-------+
```

13. Find the Average Runs Scored by Indian Players

Task: Filter players from "India" and calculate the average runs scored.

```
players_df.filter(col("Country") == "India").agg(avg("Runs").alias("Avg_Runs")).show()
```

**OUTPUT**

```
+---------+
|Avg_Runs|
+---------+
|8066.67  |
+---------+
```

14. Get Players Who Have Taken More than 50 Wickets

Task: Filter players who have taken more than 50 wickets.

```
players_df.filter(col("Wickets") > 50).show()
```

**OUTPUT**

```
+--------------+---------+----+-------+
|Player        |Country  |Runs|Wickets|
+--------------+---------+----+-------+
|Jasprit Bumrah|India    |1200|200    |
+--------------+---------+----+-------+
```

15. Read a JSON File Containing Indian Cities Population
Sample JSON (`cities.json`):
json
[
{"City": "Mumbai", "State": "Maharashtra", "Population": 20000000},
{"City": "Delhi", "State": "Delhi", "Population": 18000000},
{"City": "Bangalore", "State": "Karnataka", "Population": 12000000},
{"City": "Hyderabad", "State": "Telangana", "Population": 10000000}
]

Task: Read this JSON file into a DataFrame and display its contents.

```
cities_df = spark.read.json("cities.json")
cities_df.show()
```

**OUTPUT**

```
+---------+----------+----------+
|City     |Population|State     |
+---------+----------+----------+
|Mumbai   |20000000  |Maharashtra|
|Delhi    |18000000  |Delhi     |
|Bangalore|12000000  |Karnataka |
|Hyderabad|10000000  |Telangana |
+---------+----------+----------+
```

16. Find Cities with a Population Greater than 15 Million

Task: Filter cities with a population greater than 15 million.

```
cities_df.filter(col("Population") > 15000000).show()
```

**OUTPUT**

```
+-------+----------+-------------+
|City   |Population|State        |
+-------+----------+-------------+
|Mumbai |20000000  |Maharashtra  |
|Delhi  |18000000  |Delhi        |
+-------+----------+-------------+
```

17. Calculate Total Population per State

Task : Group by `State` and sum the `Population`.

```
cities_df.groupBy("State").agg(sum("Population").alias("Total_Population")).show()
```

**OUTPUT**

```
+-------------+----------------+
|State        |Total_Population|
+-------------+----------------+
|Maharashtra  |20000000        |
|Delhi        |18000000        |
|Karnataka    |12000000        |
|Telangana    |10000000        |
+-------------+----------------+
```

18. Find the State with the Highest Total Population

Task: Identify which state has the highest total population.

```
cities_df.groupBy("State").agg(sum("Population").alias
    ("Total_Population")).orderBy(col("Total_Population").desc()).limit
    (1).show()
```

**OUTPUT**

```
+-------------+----------------+
|State        |Total_Population|
+-------------+----------------+
|Maharashtra  |20000000        |
+-------------+----------------+
```

19. Convert a DataFrame to Pandas

Task: Convert the `df` DataFrame into a Pandas DataFrame.

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

**OUTPUT**

```
   EmpID   Name Department  Salary     Tax
0      1   Amit         IT   60000  6000.0
1      2  Priya         HR   55000  5500.0
2      3  Rahul    Finance   75000  7500.0
3      4  Sneha         IT   80000  8000.0
4      5  Karan         HR   65000  6500.0
```