**PySpark DataFrame Case Study: Employee Performance Review Analysis**

Scenario: You have a dataset containing employee performance reviews over time and want to analyze performance trends, average ratings, and identify high performers. The dataset contains the following columns:

• emp\_id: Employee ID

• review\_date: Date of the performance review

• department: Department name

• rating: Performance rating (1-5 scale)

• reviewer: Name of the person conducting the review

**Sample Data**

+--------+-------------+-------------+-------+------------+  
| emp\_id | review\_date | department | rating| reviewer |  
+--------+-------------+-------------+-------+------------+  
| 1 | 2024-01-10 | Engineering | 5 | John |  
| 2 | 2024-01-11 | HR | 4 | Jane |  
| 3 | 2024-01-12 | Sales | 3 | Sam |  
| 4 | 2024-02-01 | Engineering | 5 | John |  
| 1 | 2024-03-10 | Engineering | 4 | Jane |  
| 2 | 2024-03-11 | HR | NULL | Sam |  
+--------+-------------+-------------+-------+------------+

**Code to Create DataFrame**

from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, avg, coalesce  
from pyspark.sql.window import Window  
  
# Initialize Spark session  
spark = SparkSession.builder.appName("Employee Performance Review Analysis").getOrCreate()  
  
# Sample data  
data = [  
 (1, '2024-01-10', 'Engineering', 5, 'John'),  
 (2, '2024-01-11', 'HR', 4, 'Jane'),  
 (3, '2024-01-12', 'Sales', 3, 'Sam'),  
 (4, '2024-02-01', 'Engineering', 5, 'John'),  
 (1, '2024-03-10', 'Engineering', 4, 'Jane'),  
 (2, '2024-03-11', 'HR', None, 'Sam')  
]  
  
# Create DataFrame  
columns = ["emp\_id", "review\_date", "department", "rating", "reviewer"]  
df = spark.createDataFrame(data, schema=columns)  
df.show()

**1. Filter**

Find all reviews where the rating was 4 or higher.

df\_filtered = df.filter(df.rating >= 4)  
df\_filtered.show()

**2. Handle Null Values**

Fill null values in the 'rating' column with the department average rating.

window\_spec = Window.partitionBy('department')  
df\_filled = df.withColumn('rating', coalesce(df.rating, avg('rating').over(window\_spec)))  
df\_filled.show()

**3. Drop Duplicates**

Remove duplicate reviews by 'emp\_id' and 'review\_date'.

df\_no\_duplicates = df.dropDuplicates(['emp\_id', 'review\_date'])  
df\_no\_duplicates.show()

**4. Select Specific Columns**

Select 'emp\_id', 'department', and 'rating' columns.

df\_selected = df.select('emp\_id', 'department', 'rating')  
df\_selected.show()

**5. Grouping and Aggregating**

Calculate the average rating per department.

df\_grouped = df.groupBy('department').agg({'rating': 'avg'})  
df\_grouped.show()

**6. Joining DataFrames**

Join with another DataFrame 'df\_employees' containing 'emp\_id' and 'employee\_name'.

# Assuming df\_employees is another DataFrame that contains employee details  
df\_joined = df.join(df\_employees, on='emp\_id', how='inner')  
df\_joined.show()

**7. Union of DataFrames**

Union with another 'df\_new\_reviews' DataFrame containing additional reviews.

df\_union = df.union(df\_new\_reviews)  
df\_union.show()

**8. Temporary View and SQL**

Create a temp view and find the average rating for each employee.

df.createOrReplaceTempView('performance\_reviews')  
sql\_result = spark.sql('SELECT emp\_id, AVG(rating) as avg\_rating FROM performance\_reviews GROUP BY emp\_id')  
sql\_result.show()

**9. Window Functions**

Calculate the cumulative average rating for each employee over time.

window\_spec = Window.partitionBy('emp\_id').orderBy('review\_date')  
df\_with\_cumulative\_avg = df.withColumn('cumulative\_avg', avg('rating').over(window\_spec))  
df\_with\_cumulative\_avg.show()