# **Spark Setup**

! pip install pyspark

Collecting pyspark  
 Downloading pyspark-3.5.2.tar.gz (317.3 MB)  
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.3/317.3 MB 4.4 MB/s eta 0:00:00  
etadata (setup.py) ... ent already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)  
Building wheels for collected packages: pyspark  
 Building wheel for pyspark (setup.py) ... e=pyspark-3.5.2-py2.py3-none-any.whl size=317812365 sha256=f447e1ce460794723046fbd6afe959f72eb7723244b7d1c4b006d2745b7b8914  
 Stored in directory: /root/.cache/pip/wheels/34/34/bd/03944534c44b677cd5859f248090daa9fb27b3c8f8e5f49574  
Successfully built pyspark  
Installing collected packages: pyspark  
Successfully installed pyspark-3.5.2

# **Spark Example**

# creating session  
from pyspark.sql import SparkSession  
from pyspark.sql.functions import col  
  
spark = SparkSession.builder \  
 .appName("PySpark Notebook Example") \  
 .getOrCreate()

customers = [  
 (1, "Ravi", "Mumbai"),  
 (2, "Priya", "Delhi"),  
 (3, "Vijay", "Bangalore"),  
 (4, "Anita", "Chennai"),  
 (5, "Raj", "Hyderabad"),  
]  
  
transactions = [  
 (1, 1, 10000.50),  
 (2, 2, 20000.75),  
 (3, 1, 15000.25),  
 (4, 3, 30000.00),  
 (5, 2, 40000.50),  
 (6, 4, 25000.00),  
 (7, 5, 18000.75),  
 (8, 1, 5000.00),  
]  
  
customer\_col = ["customer\_id", "Name", "city"]  
transaction\_col = ["transaction\_id", "customer\_id", "amount"]  
  
customer\_df = spark.createDataFrame(customers, schema = customer\_col)  
transaction\_df = spark.createDataFrame(transactions, schema = transaction\_col)

print("Customer DataFrame:")  
customer\_df.show()  
  
print("Transaction DataFrame:")  
transaction\_df.show()

Customer DataFrame:  
+-----------+-----+---------+  
|customer\_id| Name| city|  
+-----------+-----+---------+  
| 1| Ravi| Mumbai|  
| 2|Priya| Delhi|  
| 3|Vijay|Bangalore|  
| 4|Anita| Chennai|  
| 5| Raj|Hyderabad|  
+-----------+-----+---------+  
  
Transaction DataFrame:  
+--------------+-----------+--------+  
|transaction\_id|customer\_id| amount|  
+--------------+-----------+--------+  
| 1| 1| 10000.5|  
| 2| 2|20000.75|  
| 3| 1|15000.25|  
| 4| 3| 30000.0|  
| 5| 2| 40000.5|  
| 6| 4| 25000.0|  
| 7| 5|18000.75|  
| 8| 1| 5000.0|  
+--------------+-----------+--------+

# Join the DataFrames on CustomerID  
customer\_transactions\_df = customer\_df.join(transaction\_df, on="customer\_id")  
print("Customer Transactions DataFrame:")  
customer\_transactions\_df.show()

Customer Transactions DataFrame:  
+-----------+-----+---------+--------------+--------+  
|customer\_id| Name| city|transaction\_id| amount|  
+-----------+-----+---------+--------------+--------+  
| 1| Ravi| Mumbai| 1| 10000.5|  
| 1| Ravi| Mumbai| 3|15000.25|  
| 1| Ravi| Mumbai| 8| 5000.0|  
| 2|Priya| Delhi| 2|20000.75|  
| 2|Priya| Delhi| 5| 40000.5|  
| 3|Vijay|Bangalore| 4| 30000.0|  
| 4|Anita| Chennai| 6| 25000.0|  
| 5| Raj|Hyderabad| 7|18000.75|  
+-----------+-----+---------+--------------+--------+

# Calculate the total amount spent by each customer  
total\_spent\_df = customer\_transactions\_df.groupBy ("Name").sum("Amount").withColumnRenamed ("sum(Amount)", "TotalSpent")  
print("Total Amount Spent by Each Customer: ")  
total\_spent\_df.show()

Total Amount Spent by Each Customer:   
+-----+----------+  
| Name|TotalSpent|  
+-----+----------+  
| Ravi| 30000.75|  
|Priya| 60001.25|  
|Vijay| 30000.0|  
|Anita| 25000.0|  
| Raj| 18000.75|  
+-----+----------+

# Find customers who have spent more than 30,000  
big\_spenders\_df = total\_spent\_df.filter(col ("TotalSpent") > 30000)  
print("Customers Who Spent More Than 30,000:")  
big\_spenders\_df.show()

Customers Who Spent More Than 30,000:  
+-----+----------+  
| Name|TotalSpent|  
+-----+----------+  
| Ravi| 30000.75|  
|Priya| 60001.25|  
+-----+----------+

# Count the number of transactions per customer  
transactions\_count\_df = customer\_transactions\_df.groupBy ("Name").count().withColumnRenamed ("count", "TransactionCount")  
print("Number of Transactions Per Customer: ")  
transactions\_count\_df.show()

Number of Transactions Per Customer:   
+-----+----------------+  
| Name|TransactionCount|  
+-----+----------------+  
| Ravi| 3|  
|Priya| 2|  
|Vijay| 1|  
|Anita| 1|  
| Raj| 1|  
+-----+----------------+

# Sort customers by total amount spent in descending order  
sorted\_spenders\_df = total\_spent\_df.orderBy(col ("TotalSpent").desc())  
print ("Customers Sorted by Total Spent (Descending):")  
sorted\_spenders\_df.show()

Customers Sorted by Total Spent (Descending):  
+-----+----------+  
| Name|TotalSpent|  
+-----+----------+  
|Priya| 60001.25|  
| Ravi| 30000.75|  
|Vijay| 30000.0|  
|Anita| 25000.0|  
| Raj| 18000.75|  
+-----+----------+

# **Exercise**

# Exercise: Product Sales Analysis  
  
from pyspark.sql import SparkSession  
from pyspark.sql.functions import col  
  
spark = SparkSession.builder \  
 .appName("Product Sales Analysis") \  
 .getOrCreate()  
  
products = [  
 (1, "Laptop", "Electronics", 50000),  
 (2, "Smartphone", "Electronics", 30000),  
 (3, "Table", "Furniture", 15000),  
 (4, "Chair", "Furniture", 5000),  
 (5, "Headphones", "Electronics", 2000),  
]  
  
sales = [  
 (1, 1, 2),  
 (2, 2, 1),  
 (3, 3, 3),  
 (4, 1, 1),  
 (5, 4, 5),  
 (6, 2, 2),  
 (7, 5, 10),  
 (8, 3, 1),  
]  
  
product\_columns = ["ProductID", "ProductName", "Category", "Price"]  
sales\_columns = ["SaleID", "ProductID", "Quantity"]  
  
product\_df = spark.createDataFrame(products, schema=product\_columns)  
sales\_df = spark.createDataFrame(sales, schema=sales\_columns)  
  
print("Products DataFrame:")  
product\_df.show()  
  
print("Sales DataFrame:")  
sales\_df.show()

Products DataFrame:  
+---------+-----------+-----------+-----+  
|ProductID|ProductName| Category|Price|  
+---------+-----------+-----------+-----+  
| 1| Laptop|Electronics|50000|  
| 2| Smartphone|Electronics|30000|  
| 3| Table| Furniture|15000|  
| 4| Chair| Furniture| 5000|  
| 5| Headphones|Electronics| 2000|  
+---------+-----------+-----------+-----+  
  
Sales DataFrame:  
+------+---------+--------+  
|SaleID|ProductID|Quantity|  
+------+---------+--------+  
| 1| 1| 2|  
| 2| 2| 1|  
| 3| 3| 3|  
| 4| 1| 1|  
| 5| 4| 5|  
| 6| 2| 2|  
| 7| 5| 10|  
| 8| 3| 1|  
+------+---------+--------+

# 1.Join the DataFrames:  
# Join the product\_df and sales\_df DataFrames on ProductID to create a combined DataFrame with product and sales data.  
product\_sales\_df = product\_df.join(sales\_df, on="ProductID")  
print("product\_sales DataFrame:")  
product\_sales\_df.show()

product\_sales DataFrame:  
+---------+-----------+-----------+-----+------+--------+  
|ProductID|ProductName| Category|Price|SaleID|Quantity|  
+---------+-----------+-----------+-----+------+--------+  
| 1| Laptop|Electronics|50000| 1| 2|  
| 1| Laptop|Electronics|50000| 4| 1|  
| 2| Smartphone|Electronics|30000| 2| 1|  
| 2| Smartphone|Electronics|30000| 6| 2|  
| 3| Table| Furniture|15000| 3| 3|  
| 3| Table| Furniture|15000| 8| 1|  
| 4| Chair| Furniture| 5000| 5| 5|  
| 5| Headphones|Electronics| 2000| 7| 10|  
+---------+-----------+-----------+-----+------+--------+

# 2.Calculate Total Sales Value:  
# For each product, calculate the total sales value by multiplying the price by the quantity sold.  
  
total\_sale\_product\_df = product\_sales\_df.withColumn("TotalSalesValue", col("Price") \* col("Quantity"))  
print("Total Sales Value DataFrame:")  
total\_sale\_product\_df.show()

Total Sales Value DataFrame:  
+---------+-----------+-----------+-----+------+--------+---------------+  
|ProductID|ProductName| Category|Price|SaleID|Quantity|TotalSalesValue|  
+---------+-----------+-----------+-----+------+--------+---------------+  
| 1| Laptop|Electronics|50000| 1| 2| 100000|  
| 1| Laptop|Electronics|50000| 4| 1| 50000|  
| 2| Smartphone|Electronics|30000| 2| 1| 30000|  
| 2| Smartphone|Electronics|30000| 6| 2| 60000|  
| 3| Table| Furniture|15000| 3| 3| 45000|  
| 3| Table| Furniture|15000| 8| 1| 15000|  
| 4| Chair| Furniture| 5000| 5| 5| 25000|  
| 5| Headphones|Electronics| 2000| 7| 10| 20000|  
+---------+-----------+-----------+-----+------+--------+---------------+

# 3.Find the Total Sales for Each Product Category:  
# Group the data by the Category column and calculate the total sales value for each product category.  
total\_sale\_by\_category\_df = total\_sale\_product\_df.groupBy("Category").sum("TotalSalesValue").withColumnRenamed("sum(TotalSalesValue)","TotalSales")  
print("Total Sales for Each Product Category:")  
total\_sale\_by\_category\_df.show()

Total Sales for Each Product Category:  
+-----------+----------+  
| Category|TotalSales|  
+-----------+----------+  
|Electronics| 260000|  
| Furniture| 85000|  
+-----------+----------+

# 4.Identify the Top-Selling Product:  
# Find the product that generated the highest total sales value.  
high\_sale\_product = total\_sale\_product\_df.groupBy("ProductName").sum("TotalSalesValue").withColumnRenamed("sum(TotalSalesValue)","TotalSales").orderBy(col("TotalSales").desc()).limit(1)  
print("Top-Selling Product:")  
high\_sale\_product.show()

Top-Selling Product:  
+-----------+----------+  
|ProductName|TotalSales|  
+-----------+----------+  
| Laptop| 150000|  
+-----------+----------+

# 5.Sort the Products by Total Sales Value:  
# Sort the products by total sales value in descending order.  
high\_sale\_product = total\_sale\_product\_df.groupBy("ProductName").sum("TotalSalesValue").withColumnRenamed("sum(TotalSalesValue)","TotalSales").orderBy(col("TotalSales").desc())  
print("product's Total sales value")  
high\_sale\_product.show()

product's Total sales value  
+-----------+----------+  
|ProductName|TotalSales|  
+-----------+----------+  
| Laptop| 150000|  
| Smartphone| 90000|  
| Table| 60000|  
| Chair| 25000|  
| Headphones| 20000|  
+-----------+----------+

# 6.Count the Number of Sales for Each Product:  
# Count the number of sales transactions for each product.  
product\_sales\_count\_df = product\_sales\_df.groupBy("ProductID").count().withColumnRenamed("count","TransactionCount")  
print("Number of Sales for Each Product:")  
product\_sales\_count\_df.show()

Number of Sales for Each Product:  
+---------+----------------+  
|ProductID|TransactionCount|  
+---------+----------------+  
| 1| 2|  
| 2| 2|  
| 3| 2|  
| 4| 1|  
| 5| 1|  
+---------+----------------+

# 7.Filter the Products with Total Sales Value Greater Than ₹50,000:  
# Filter out the products that have a total sales value greater than ₹50,000.  
filtered\_high\_sale\_product = high\_sale\_product.filter(col("TotalSales") > 50000)  
print("Products with Total Sales Value Greater Than ₹50,000:")  
filtered\_high\_sale\_product.show()

Products with Total Sales Value Greater Than ₹50,000:  
+-----------+----------+  
|ProductName|TotalSales|  
+-----------+----------+  
| Laptop| 150000|  
| Smartphone| 90000|  
| Table| 60000|  
+-----------+----------+

#**RDD Transformation** (Resilient Distributed Dataset)

spark = SparkSession.builder \  
 .appName("RDD Transformation Example") \  
 .getOrCreate()  
  
sc = spark.sparkContext  
print("Spark Session Created")

Spark Session Created

data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
rdd = sc.parallelize(data)  
  
print("original RDD:", rdd.collect())

original RDD: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

rdd2 = rdd.map(lambda x: x \* 2)  
  
print("RDD after transformation (x\*\*2):", rdd2.collect())

RDD after transformation (x\*\*2): [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

rdd3 = rdd2.filter(lambda x: x % 2 == 0)  
  
print("RDD after filtering (even):", rdd3.collect())

RDD after filtering (even): [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

sentence = ["Hello World","py spark is geat", "RDD transformations"]  
rdd4 = sc.parallelize(sentence)  
ScentenceToWords\_rdd = rdd4.flatMap(lambda x: x.split(" "))  
  
print("RDD after flatMap transformation:", ScentenceToWords\_rdd.collect())

RDD after flatMap transformation: ['Hello', 'World', 'py', 'spark', 'is', 'geat', 'RDD', 'transformations']

# Actions  
# 1. collect()  
result = rdd.collect()  
print("Result of collect action:", result)  
# 2. count()  
result = rdd.count()  
print("Result of count action:", result)  
# 3. reduce()  
result = rdd.reduce(lambda x, y: x + y)  
print("Result of reduce action:", result)

Result of collect action: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
Result of count action: 10  
Result of reduce action: 55

# **RDD Exercise**

# https://codeshare.io/ez3VNJ  
# Initialize SparkSession  
spark = SparkSession.builder.appName("SalesDataAnalysis").getOrCreate()

sales\_data = [  
 ("ProductA", 100),  
 ("ProductB", 150),  
 ("ProductA", 200),  
 ("ProductC", 300),  
 ("ProductB", 250),  
 ("ProductC", 100)  
]  
  
# step 1 -> spark context  
sc = spark.sparkContext  
  
# step 2  
# task 1 -> creating RDD of sales\_data and Printing the first few elements of the RDD  
sales\_rdd = sc.parallelize(sales\_data)  
print(sales\_rdd.take(3))

[('ProductA', 100), ('ProductB', 150), ('ProductA', 200)]

# step 3 -> Grouping and Aggregating Data  
  
# taks 2 -> Group data by product name  
grouped\_sales\_rdd = sales\_rdd.groupByKey()  
print("Grouped data:")  
for k,v in grouped\_sales\_rdd.collect():  
 print(k,list(v))  
  
# taks 3 -> Calculate total sales by product  
total\_sales\_by\_product = sales\_rdd.reduceByKey(lambda x, y: x + y)  
print("Total sales by product:")  
print(total\_sales\_by\_product.collect())  
  
# taks 4 -> Sort products by total sales  
sorted\_products = total\_sales\_by\_product.sortBy(lambda x: x[1], ascending=False)  
print("Sorted products by total sales:")  
print(sorted\_products.collect())

Grouped data:  
ProductA [100, 200]  
ProductB [150, 250]  
ProductC [300, 100]  
Total sales by product:  
[('ProductA', 300), ('ProductB', 400), ('ProductC', 400)]  
Sorted products by total sales:  
[('ProductB', 400), ('ProductC', 400), ('ProductA', 300)]

#step 4 -> Additional transformations  
  
# taks 5 -> Filter products with high sales  
high\_sales\_products = total\_sales\_by\_product.filter(lambda x: x[1] > 300)  
print("Products with high sales:")  
print(high\_sales\_products.collect())  
  
# task 6 -> Combine Regional Sales Data  
  
# regional sales data RDD  
regional\_sales\_data = [  
 ("ProductA", 50),  
 ("ProductC", 150)  
]  
regional\_sales\_rdd = sc.parallelize(regional\_sales\_data)  
  
# Combining the two RDDs  
combined\_sales\_rdd = sales\_rdd.union(regional\_sales\_rdd)  
  
# Calculating new total sales  
new\_total\_sales\_by\_product = combined\_sales\_rdd.reduceByKey(lambda x, y: x + y)  
print("Combined sales data:")  
print(new\_total\_sales\_by\_product.collect())

Products with high sales:  
[('ProductB', 400), ('ProductC', 400)]  
Combined sales data:  
[('ProductA', 350), ('ProductC', 550), ('ProductB', 400)]

# step 5 -> Perform Actions on the RDD  
  
# task 7 -> Count the number of distinct products  
distinct\_products\_count = sales\_rdd.map(lambda x: x[0]).distinct().count()  
print("Count of distinct products:", distinct\_products\_count)  
  
# task 8 -> Identify the product with maximum sales  
max\_sales = total\_sales\_by\_product.max()[1]  
max\_sales\_products = total\_sales\_by\_product.filter(lambda x: x[1] == max\_sales)  
print("Products with maximum sales:", max\_sales\_products.map(lambda x: x[0]).collect())

Count of distinct products: 3  
Products with maximum sales: ['ProductB', 'ProductC']

# challenge -> Calculate the Average Sales per Product  
for k,v in grouped\_sales\_rdd.collect():  
 print(k,sum(list(v))/len(list(v)))

ProductA 150.0  
ProductB 200.0  
ProductC 200.0

# **PySpark Exercise Sep\_4**

# https://codeshare.io/w90yOJ  
  
spark = SparkSession.builder \  
 .appName("Employee Data Analysis") \  
 .getOrCreate()

# Sample employee data  
data = [  
 (1, 'Arjun', 'IT', 75000),  
 (2, 'Vijay', 'Finance', 85000),  
 (3, 'Shalini', 'IT', 90000),  
 (4, 'Sneha', 'HR', 50000),  
 (5, 'Rahul', 'Finance', 60000),  
 (6, 'Amit', 'IT', 55000)  
]  
  
# Define schema (columns)  
columns = ['EmployeeID', 'EmployeeName', 'Department', 'Salary']  
  
# Create DataFrame  
employee\_df = spark.createDataFrame(data, columns)  
  
# Show the DataFrame  
employee\_df.show()

+----------+------------+----------+------+  
|EmployeeID|EmployeeName|Department|Salary|  
+----------+------------+----------+------+  
| 1| Arjun| IT| 75000|  
| 2| Vijay| Finance| 85000|  
| 3| Shalini| IT| 90000|  
| 4| Sneha| HR| 50000|  
| 5| Rahul| Finance| 60000|  
| 6| Amit| IT| 55000|  
+----------+------------+----------+------+

# Task 1: Filter Employees by Salary  
  
high\_salary\_employees = employee\_df.filter(col("Salary") > 60000)  
print("Employees with salary greater than 60000:")  
high\_salary\_employees.show()

Employees with salary greater than 60000:  
+----------+------------+----------+------+  
|EmployeeID|EmployeeName|Department|Salary|  
+----------+------------+----------+------+  
| 1| Arjun| IT| 75000|  
| 2| Vijay| Finance| 85000|  
| 3| Shalini| IT| 90000|  
+----------+------------+----------+------+

# Task 2: Calculate the Average Salary by Department  
  
avg\_salary\_by\_dept = employee\_df.groupBy("Department").avg("Salary").withColumnRenamed("avg(Salary)", "AvgerageSalary")  
print("Average salary by department:")  
avg\_salary\_by\_dept.show()

Average salary by department:  
+----------+-----------------+  
|Department| AvgerageSalary|  
+----------+-----------------+  
| Finance| 72500.0|  
| IT|73333.33333333333|  
| HR| 50000.0|  
+----------+-----------------+

# Task 3: Sort Employees by Salary (Descending)  
  
sorted\_by\_salary\_desc = employee\_df.orderBy(col("Salary").desc())  
print("Employees sorted by salary descending:")  
sorted\_by\_salary\_desc.show()

Employees sorted by salary descending:  
+----------+------------+----------+------+  
|EmployeeID|EmployeeName|Department|Salary|  
+----------+------------+----------+------+  
| 3| Shalini| IT| 90000|  
| 2| Vijay| Finance| 85000|  
| 1| Arjun| IT| 75000|  
| 5| Rahul| Finance| 60000|  
| 6| Amit| IT| 55000|  
| 4| Sneha| HR| 50000|  
+----------+------------+----------+------+

# Task 4: Add a Bonus Column  
  
employee\_df\_with\_bonus = employee\_df.withColumn("Bonus", col("Salary") \* 0.1)  
print("Employees with bonus column:")  
employee\_df\_with\_bonus.show()

Employees with bonus column:  
+----------+------------+----------+------+------+  
|EmployeeID|EmployeeName|Department|Salary| Bonus|  
+----------+------------+----------+------+------+  
| 1| Arjun| IT| 75000|7500.0|  
| 2| Vijay| Finance| 85000|8500.0|  
| 3| Shalini| IT| 90000|9000.0|  
| 4| Sneha| HR| 50000|5000.0|  
| 5| Rahul| Finance| 60000|6000.0|  
| 6| Amit| IT| 55000|5500.0|  
+----------+------------+----------+------+------+

# **Data Handling - NULL Values**

spark = SparkSession.builder \  
 .appName("Employee Data Handling") \  
 .getOrCreate()  
  
# Sample employee data with null values  
data = [  
 (1, 'Arjun', 'IT', 75000),  
 (2, 'Vijay', 'Finance', 85000),  
 (3, None, 'IT', 90000),  
 (4, 'Sneha', 'HR', None),  
 (5, 'Rahul', None, 60000),  
 (6, 'Amit', 'IT', 55000)  
]  
columns = ['EmployeeID', 'EmployeeName', 'Department', 'Salary']  
  
# Create DataFrame  
employee\_df = spark.createDataFrame(data, columns)  
  
# Show the DataFrame  
employee\_df.show()

+----------+------------+----------+------+  
|EmployeeID|EmployeeName|Department|Salary|  
+----------+------------+----------+------+  
| 1| Arjun| IT| 75000|  
| 2| Vijay| Finance| 85000|  
| 3| NULL| IT| 90000|  
| 4| Sneha| HR| NULL|  
| 5| Rahul| NULL| 60000|  
| 6| Amit| IT| 55000|  
+----------+------------+----------+------+

# fillna  
filled\_df = employee\_df.fillna({'EmployeeName': 'Unknown', 'Department': 'Unknown'})  
filled\_df.show()

+----------+------------+----------+------+  
|EmployeeID|EmployeeName|Department|Salary|  
+----------+------------+----------+------+  
| 1| Arjun| IT| 75000|  
| 2| Vijay| Finance| 85000|  
| 3| Unknown| IT| 90000|  
| 4| Sneha| HR| NULL|  
| 5| Rahul| Unknown| 60000|  
| 6| Amit| IT| 55000|  
+----------+------------+----------+------+

# drop where salary is NULL  
dropped\_null\_salary = employee\_df.na.drop(subset=["Salary"])  
dropped\_null\_salary.show()

+----------+------------+----------+------+  
|EmployeeID|EmployeeName|Department|Salary|  
+----------+------------+----------+------+  
| 1| Arjun| IT| 75000|  
| 2| Vijay| Finance| 85000|  
| 3| NULL| IT| 90000|  
| 5| Rahul| NULL| 60000|  
| 6| Amit| IT| 55000|  
+----------+------------+----------+------+

# fill NULL salary with 50000  
filled\_null\_salary = employee\_df.fillna({'Salary': 50000})  
filled\_null\_salary.show()

+----------+------------+----------+------+  
|EmployeeID|EmployeeName|Department|Salary|  
+----------+------------+----------+------+  
| 1| Arjun| IT| 75000|  
| 2| Vijay| Finance| 85000|  
| 3| NULL| IT| 90000|  
| 4| Sneha| HR| 50000|  
| 5| Rahul| NULL| 60000|  
| 6| Amit| IT| 55000|  
+----------+------------+----------+------+

# check for NULL values in entire table  
null\_check = employee\_df.select([col(column).isNull().alias(column) for column in employee\_df.columns])  
null\_check.show()

+----------+------------+----------+------+  
|EmployeeID|EmployeeName|Department|Salary|  
+----------+------------+----------+------+  
| false| false| false| false|  
| false| false| false| false|  
| false| true| false| false|  
| false| false| false| true|  
| false| false| true| false|  
| false| false| false| false|  
+----------+------------+----------+------+

# replace all NULL values with 'N/A'  
replaced\_df = employee\_df.na.fill('N/A')  
replaced\_df.show()

+----------+------------+----------+------+  
|EmployeeID|EmployeeName|Department|Salary|  
+----------+------------+----------+------+  
| 1| Arjun| IT| 75000|  
| 2| Vijay| Finance| 85000|  
| 3| N/A| IT| 90000|  
| 4| Sneha| HR| NULL|  
| 5| Rahul| N/A| 60000|  
| 6| Amit| IT| 55000|  
+----------+------------+----------+------+

# **Window and Dates** - Advanced DataFrame operations

from pyspark.sql import SparkSession  
from pyspark.sql import functions as F  
from pyspark.sql.window import Window  
  
spark = SparkSession.builder \  
 .appName("Advanced DataFrame operations") \  
 .getOrCreate()  
  
data1 = [  
 (1, 'Arjun', 'IT', 75000, '2022-01-15'),  
 (2, 'Vijay', 'Finance', 85000, '2022-03-12'),  
 (3, 'Shalini', 'IT', 90000, '2021-06-10'),  
 (4, 'Sneha', 'IT', 90000, '2022-02-28'),  
]  
  
data2 = [  
 (5, 'Vikram', 'HR', 50000, '2022-05-01'),  
 (6, 'Amit', 'Finance', 60000, '2022-04-05'),  
 (7, 'Priya', 'IT', 55000, '2022-03-15'),  
 (8, 'Rahul', 'Finance', 70000, '2022-02-20'),  
 (9, 'Anjali', 'HR', 65000, '2022-01-25'),  
]  
  
columns = ['EmployeeID', 'EmployeeName', 'Department', 'Salary', 'JoiningDate']  
  
employee\_df1 = spark.createDataFrame(data1, columns)  
employee\_df2 = spark.createDataFrame(data2, columns)  
  
employee\_df1.show()  
employee\_df2.show()

+----------+------------+----------+------+-----------+  
|EmployeeID|EmployeeName|Department|Salary|JoiningDate|  
+----------+------------+----------+------+-----------+  
| 1| Arjun| IT| 75000| 2022-01-15|  
| 2| Vijay| Finance| 85000| 2022-03-12|  
| 3| Shalini| IT| 90000| 2021-06-10|  
| 4| Sneha| IT| 90000| 2022-02-28|  
+----------+------------+----------+------+-----------+  
  
+----------+------------+----------+------+-----------+  
|EmployeeID|EmployeeName|Department|Salary|JoiningDate|  
+----------+------------+----------+------+-----------+  
| 5| Vikram| HR| 50000| 2022-05-01|  
| 6| Amit| Finance| 60000| 2022-04-05|  
| 7| Priya| IT| 55000| 2022-03-15|  
| 8| Rahul| Finance| 70000| 2022-02-20|  
| 9| Anjali| HR| 65000| 2022-01-25|  
+----------+------------+----------+------+-----------+

# union 2 dataframes  
  
# remove duplicates  
union\_df = employee\_df1.union(employee\_df2).dropDuplicates()  
  
# include duplicates  
union\_df = employee\_df1.union(employee\_df2)  
union\_df.show()

+----------+------------+----------+------+-----------+  
|EmployeeID|EmployeeName|Department|Salary|JoiningDate|  
+----------+------------+----------+------+-----------+  
| 1| Arjun| IT| 75000| 2022-01-15|  
| 2| Vijay| Finance| 85000| 2022-03-12|  
| 3| Shalini| IT| 90000| 2021-06-10|  
| 4| Sneha| IT| 90000| 2022-02-28|  
| 5| Vikram| HR| 50000| 2022-05-01|  
| 6| Amit| Finance| 60000| 2022-04-05|  
| 7| Priya| IT| 55000| 2022-03-15|  
| 8| Rahul| Finance| 70000| 2022-02-20|  
| 9| Anjali| HR| 65000| 2022-01-25|  
+----------+------------+----------+------+-----------+

from pyspark.sql.window import Window  
from pyspark.sql.functions import rank  
  
# window specification to rank employees by salary witin each department  
window\_spec = Window.partitionBy("Department").orderBy(col("Salary").desc())  
  
# add a new column 'Rank' to the DataFrame  
ranked\_df = union\_df.withColumn("Rank", rank().over(window\_spec))  
ranked\_df.show()

+----------+------------+----------+------+-----------+----+  
|EmployeeID|EmployeeName|Department|Salary|JoiningDate|Rank|  
+----------+------------+----------+------+-----------+----+  
| 2| Vijay| Finance| 85000| 2022-03-12| 1|  
| 8| Rahul| Finance| 70000| 2022-02-20| 2|  
| 6| Amit| Finance| 60000| 2022-04-05| 3|  
| 9| Anjali| HR| 65000| 2022-01-25| 1|  
| 5| Vikram| HR| 50000| 2022-05-01| 2|  
| 3| Shalini| IT| 90000| 2021-06-10| 1|  
| 4| Sneha| IT| 90000| 2022-02-28| 1|  
| 1| Arjun| IT| 75000| 2022-01-15| 3|  
| 7| Priya| IT| 55000| 2022-03-15| 4|  
+----------+------------+----------+------+-----------+----+

from pyspark.sql.functions import sum  
  
# window specification for cumulative sum of salaries within each departmrnt  
window\_spec\_sum = Window.partitionBy("Department").orderBy("JoiningDate").rowsBetween(Window.unboundedPreceding, Window.currentRow)  
  
# add a new column 'CumulativeSalary' to the DataFrame  
cumulative\_salary\_df = union\_df.withColumn("CumulativeSalary", sum("Salary").over(window\_spec\_sum))  
cumulative\_salary\_df.show()

+----------+------------+----------+------+-----------+----------------+  
|EmployeeID|EmployeeName|Department|Salary|JoiningDate|CumulativeSalary|  
+----------+------------+----------+------+-----------+----------------+  
| 8| Rahul| Finance| 70000| 2022-02-20| 70000|  
| 2| Vijay| Finance| 85000| 2022-03-12| 155000|  
| 6| Amit| Finance| 60000| 2022-04-05| 215000|  
| 9| Anjali| HR| 65000| 2022-01-25| 65000|  
| 5| Vikram| HR| 50000| 2022-05-01| 115000|  
| 3| Shalini| IT| 90000| 2021-06-10| 90000|  
| 1| Arjun| IT| 75000| 2022-01-15| 165000|  
| 4| Sneha| IT| 90000| 2022-02-28| 255000|  
| 7| Priya| IT| 55000| 2022-03-15| 310000|  
+----------+------------+----------+------+-----------+----------------+

# joining date from String -> Date type  
  
date\_converted\_df = union\_df.withColumn("JoiningDate", F.to\_date(col("JoiningDate"), "yyyy-MM-dd"))  
date\_converted\_df.show()

+----------+------------+----------+------+-----------+  
|EmployeeID|EmployeeName|Department|Salary|JoiningDate|  
+----------+------------+----------+------+-----------+  
| 1| Arjun| IT| 75000| 2022-01-15|  
| 2| Vijay| Finance| 85000| 2022-03-12|  
| 3| Shalini| IT| 90000| 2021-06-10|  
| 4| Sneha| IT| 90000| 2022-02-28|  
| 5| Vikram| HR| 50000| 2022-05-01|  
| 6| Amit| Finance| 60000| 2022-04-05|  
| 7| Priya| IT| 55000| 2022-03-15|  
| 8| Rahul| Finance| 70000| 2022-02-20|  
| 9| Anjali| HR| 65000| 2022-01-25|  
+----------+------------+----------+------+-----------+

# number of years since joining  
experienced\_df = date\_converted\_df.withColumn("YearsOfExperience", F.round(F.datediff(F.current\_date(), col("JoiningDate")) / 365,2))  
experienced\_df.show()

+----------+------------+----------+------+-----------+-----------------+  
|EmployeeID|EmployeeName|Department|Salary|JoiningDate|YearsOfExperience|  
+----------+------------+----------+------+-----------+-----------------+  
| 1| Arjun| IT| 75000| 2022-01-15| 2.64|  
| 2| Vijay| Finance| 85000| 2022-03-12| 2.48|  
| 3| Shalini| IT| 90000| 2021-06-10| 3.24|  
| 4| Sneha| IT| 90000| 2022-02-28| 2.52|  
| 5| Vikram| HR| 50000| 2022-05-01| 2.35|  
| 6| Amit| Finance| 60000| 2022-04-05| 2.42|  
| 7| Priya| IT| 55000| 2022-03-15| 2.48|  
| 8| Rahul| Finance| 70000| 2022-02-20| 2.54|  
| 9| Anjali| HR| 65000| 2022-01-25| 2.61|  
+----------+------------+----------+------+-----------+-----------------+

# adding column for next evaluation date  
eval\_date\_df = date\_converted\_df.withColumn("NextEvaluationDate", F.date\_add(col("JoiningDate"), 365))  
eval\_date\_df.show()

+----------+------------+----------+------+-----------+------------------+  
|EmployeeID|EmployeeName|Department|Salary|JoiningDate|NextEvaluationDate|  
+----------+------------+----------+------+-----------+------------------+  
| 1| Arjun| IT| 75000| 2022-01-15| 2023-01-15|  
| 2| Vijay| Finance| 85000| 2022-03-12| 2023-03-12|  
| 3| Shalini| IT| 90000| 2021-06-10| 2022-06-10|  
| 4| Sneha| IT| 90000| 2022-02-28| 2023-02-28|  
| 5| Vikram| HR| 50000| 2022-05-01| 2023-05-01|  
| 6| Amit| Finance| 60000| 2022-04-05| 2023-04-05|  
| 7| Priya| IT| 55000| 2022-03-15| 2023-03-15|  
| 8| Rahul| Finance| 70000| 2022-02-20| 2023-02-20|  
| 9| Anjali| HR| 65000| 2022-01-25| 2023-01-25|  
+----------+------------+----------+------+-----------+------------------+

# average salary per department  
avg\_salary\_df = date\_converted\_df.groupBy("Department").agg(F.avg("Salary").alias("AverageSalary"))  
avg\_salary\_df.show()

+----------+-----------------+  
|Department| AverageSalary|  
+----------+-----------------+  
| Finance|71666.66666666667|  
| IT| 77500.0|  
| HR| 57500.0|  
+----------+-----------------+

# total number of employees  
total\_employees\_df = date\_converted\_df.agg(F.count("EmployeeID").alias("TotalEmployees"))  
total\_employees\_df.show()

+--------------+  
|TotalEmployees|  
+--------------+  
| 9|  
+--------------+

# Employees name to upper  
upper\_name\_df = date\_converted\_df.withColumn("EmployeeNameUpper", F.upper(col("EmployeeName")))  
upper\_name\_df.show()

+----------+------------+----------+------+-----------+-----------------+  
|EmployeeID|EmployeeName|Department|Salary|JoiningDate|EmployeeNameUpper|  
+----------+------------+----------+------+-----------+-----------------+  
| 1| Arjun| IT| 75000| 2022-01-15| ARJUN|  
| 2| Vijay| Finance| 85000| 2022-03-12| VIJAY|  
| 3| Shalini| IT| 90000| 2021-06-10| SHALINI|  
| 4| Sneha| IT| 90000| 2022-02-28| SNEHA|  
| 5| Vikram| HR| 50000| 2022-05-01| VIKRAM|  
| 6| Amit| Finance| 60000| 2022-04-05| AMIT|  
| 7| Priya| IT| 55000| 2022-03-15| PRIYA|  
| 8| Rahul| Finance| 70000| 2022-02-20| RAHUL|  
| 9| Anjali| HR| 65000| 2022-01-25| ANJALI|  
+----------+------------+----------+------+-----------+-----------------+

# **Advance DataFrame Exercise 4th Sep**

# https://codeshare.io/BdPVKx  
  
# Data Setup:  
  
from pyspark.sql import SparkSession  
from pyspark.sql import functions as F  
from pyspark.sql.window import Window  
  
# Initialize a Spark session  
spark = SparkSession.builder \  
 .appName("Advanced DataFrame Operations - Different Dataset") \  
 .getOrCreate()  
  
# Create two sample DataFrames for Product Sales  
data1 = [  
 (1, 'Product A', 'Electronics', 1200, '2022-05-10'),  
 (2, 'Product B', 'Clothing', 500, '2022-07-15'),  
 (3, 'Product C', 'Electronics', 1800, '2021-11-05')  
]  
  
data2 = [  
 (4, 'Product D', 'Furniture', 3000, '2022-03-25'),  
 (5, 'Product E', 'Clothing', 800, '2022-09-12'),  
 (6, 'Product F', 'Electronics', 1500, '2021-10-19')  
]  
  
# Define schema (columns)  
columns = ['ProductID', 'ProductName', 'Category', 'Price', 'SaleDate']  
  
# Create DataFrames  
sales\_df1 = spark.createDataFrame(data1, columns)  
sales\_df2 = spark.createDataFrame(data2, columns)  
  
# Show the DataFrames  
sales\_df1.show()  
sales\_df2.show()

+---------+-----------+-----------+-----+----------+  
|ProductID|ProductName| Category|Price| SaleDate|  
+---------+-----------+-----------+-----+----------+  
| 1| Product A|Electronics| 1200|2022-05-10|  
| 2| Product B| Clothing| 500|2022-07-15|  
| 3| Product C|Electronics| 1800|2021-11-05|  
+---------+-----------+-----------+-----+----------+  
  
+---------+-----------+-----------+-----+----------+  
|ProductID|ProductName| Category|Price| SaleDate|  
+---------+-----------+-----------+-----+----------+  
| 4| Product D| Furniture| 3000|2022-03-25|  
| 5| Product E| Clothing| 800|2022-09-12|  
| 6| Product F|Electronics| 1500|2021-10-19|  
+---------+-----------+-----------+-----+----------+

# 1.Union of DataFrames (removing duplicates):  
# Combine the two DataFrames (`sales\_df1` and `sales\_df2`) using `union` and remove any duplicate rows.  
  
combined\_df\_noDuplicates = sales\_df1.union(sales\_df2).dropDuplicates()  
print("Combined DataFrame (removing duplicates):")  
combined\_df\_noDuplicates.show()

Combined DataFrame (removing duplicates):  
+---------+-----------+-----------+-----+----------+  
|ProductID|ProductName| Category|Price| SaleDate|  
+---------+-----------+-----------+-----+----------+  
| 1| Product A|Electronics| 1200|2022-05-10|  
| 2| Product B| Clothing| 500|2022-07-15|  
| 3| Product C|Electronics| 1800|2021-11-05|  
| 4| Product D| Furniture| 3000|2022-03-25|  
| 6| Product F|Electronics| 1500|2021-10-19|  
| 5| Product E| Clothing| 800|2022-09-12|  
+---------+-----------+-----------+-----+----------+

# 2.Union of DataFrames (including duplicates):  
# Combine both DataFrames using `unionAll` (replaced by `union`) and include duplicate rows.  
  
combined\_df\_Duplicates = sales\_df1.unionAll(sales\_df2)  
print("Combined DataFrame (including duplicates):")  
combined\_df\_Duplicates.show()

Combined DataFrame (including duplicates):  
+---------+-----------+-----------+-----+----------+  
|ProductID|ProductName| Category|Price| SaleDate|  
+---------+-----------+-----------+-----+----------+  
| 1| Product A|Electronics| 1200|2022-05-10|  
| 2| Product B| Clothing| 500|2022-07-15|  
| 3| Product C|Electronics| 1800|2021-11-05|  
| 4| Product D| Furniture| 3000|2022-03-25|  
| 5| Product E| Clothing| 800|2022-09-12|  
| 6| Product F|Electronics| 1500|2021-10-19|  
+---------+-----------+-----------+-----+----------+

# 3.Rank products by price within their category:  
# Use window functions to rank the products in each category by price in descending order.  
  
window1 = Window.partitionBy("Category").orderBy(col("Price").desc())  
ranked\_df = combined\_df\_noDuplicates.withColumn("Rank", F.rank().over(window1))  
print("Ranked products by price within their category:")  
ranked\_df.show()

Ranked products by price within their category:  
+---------+-----------+-----------+-----+----------+----+  
|ProductID|ProductName| Category|Price| SaleDate|Rank|  
+---------+-----------+-----------+-----+----------+----+  
| 5| Product E| Clothing| 800|2022-09-12| 1|  
| 2| Product B| Clothing| 500|2022-07-15| 2|  
| 3| Product C|Electronics| 1800|2021-11-05| 1|  
| 6| Product F|Electronics| 1500|2021-10-19| 2|  
| 1| Product A|Electronics| 1200|2022-05-10| 3|  
| 4| Product D| Furniture| 3000|2022-03-25| 1|  
+---------+-----------+-----------+-----+----------+----+

# 4.Calculate cumulative price per category:  
# Use window functions to calculate the cumulative price of products within each category.  
  
window2 = Window.partitionBy("Category").orderBy("SaleDate").rowsBetween(Window.unboundedPreceding, Window.currentRow)  
cumulative\_price\_df = combined\_df\_noDuplicates.withColumn("CumulativePrice", F.sum("Price").over(window2))  
print("Cumulative price per category:")  
cumulative\_price\_df.show()

Cumulative price per category:  
+---------+-----------+-----------+-----+----------+---------------+  
|ProductID|ProductName| Category|Price| SaleDate|CumulativePrice|  
+---------+-----------+-----------+-----+----------+---------------+  
| 2| Product B| Clothing| 500|2022-07-15| 500|  
| 5| Product E| Clothing| 800|2022-09-12| 1300|  
| 6| Product F|Electronics| 1500|2021-10-19| 1500|  
| 3| Product C|Electronics| 1800|2021-11-05| 3300|  
| 1| Product A|Electronics| 1200|2022-05-10| 4500|  
| 4| Product D| Furniture| 3000|2022-03-25| 3000|  
+---------+-----------+-----------+-----+----------+---------------+

# 5.Convert `SaleDate` from string to date type:  
# Convert the `SaleDate` column from string format to a PySpark date type.  
  
date\_converted\_df = combined\_df\_noDuplicates.withColumn("SaleDate", F.to\_date(col("SaleDate"), "yyyy-MM-dd"))  
print("DataFrame with SaleDate in date type:")  
date\_converted\_df.show()

DataFrame with SaleDate in date type:  
+---------+-----------+-----------+-----+----------+  
|ProductID|ProductName| Category|Price| SaleDate|  
+---------+-----------+-----------+-----+----------+  
| 1| Product A|Electronics| 1200|2022-05-10|  
| 2| Product B| Clothing| 500|2022-07-15|  
| 3| Product C|Electronics| 1800|2021-11-05|  
| 4| Product D| Furniture| 3000|2022-03-25|  
| 6| Product F|Electronics| 1500|2021-10-19|  
| 5| Product E| Clothing| 800|2022-09-12|  
+---------+-----------+-----------+-----+----------+

# 6.Calculate the number of days since each sales:  
# Calculate the number of days since each product was sold using the current date.  
  
days\_since\_sale\_df = date\_converted\_df.withColumn("DaysSinceSale", F.datediff(F.current\_date(), col("SaleDate")))  
print("DataFrame with DaysSinceSale column:")  
days\_since\_sale\_df.show()

DataFrame with DaysSinceSale column:  
+---------+-----------+-----------+-----+----------+-------------+  
|ProductID|ProductName| Category|Price| SaleDate|DaysSinceSale|  
+---------+-----------+-----------+-----+----------+-------------+  
| 1| Product A|Electronics| 1200|2022-05-10| 848|  
| 2| Product B| Clothing| 500|2022-07-15| 782|  
| 3| Product C|Electronics| 1800|2021-11-05| 1034|  
| 4| Product D| Furniture| 3000|2022-03-25| 894|  
| 6| Product F|Electronics| 1500|2021-10-19| 1051|  
| 5| Product E| Clothing| 800|2022-09-12| 723|  
+---------+-----------+-----------+-----+----------+-------------+

# 7.Add a column for the next sale deadline:  
# Add a new column `NextSaleDeadline`, which should be 30 days after the `SaleDate`.  
  
next\_sale\_deadline\_df = date\_converted\_df.withColumn("NextSaleDeadline", F.date\_add(col("SaleDate"), 30))  
print("DataFrame with NextSaleDeadline column:")  
next\_sale\_deadline\_df.show()

DataFrame with NextSaleDeadline column:  
+---------+-----------+-----------+-----+----------+----------------+  
|ProductID|ProductName| Category|Price| SaleDate|NextSaleDeadline|  
+---------+-----------+-----------+-----+----------+----------------+  
| 1| Product A|Electronics| 1200|2022-05-10| 2022-06-09|  
| 2| Product B| Clothing| 500|2022-07-15| 2022-08-14|  
| 3| Product C|Electronics| 1800|2021-11-05| 2021-12-05|  
| 4| Product D| Furniture| 3000|2022-03-25| 2022-04-24|  
| 6| Product F|Electronics| 1500|2021-10-19| 2021-11-18|  
| 5| Product E| Clothing| 800|2022-09-12| 2022-10-12|  
+---------+-----------+-----------+-----+----------+----------------+

# 8.Calculate total revenue and average price per category:  
# Find the total revenue (sum of prices) and the average price per category.  
  
total\_revenue\_df = date\_converted\_df.groupBy("Category").agg(F.sum("Price").alias("TotalRevenue"), F.avg("Price").alias("AveragePrice"))  
print("Total revenue and average price per category:")  
total\_revenue\_df.show()

Total revenue and average price per category:  
+-----------+------------+------------+  
| Category|TotalRevenue|AveragePrice|  
+-----------+------------+------------+  
|Electronics| 4500| 1500.0|  
| Clothing| 1300| 650.0|  
| Furniture| 3000| 3000.0|  
+-----------+------------+------------+

# 9.Convert all product names to lowercase:  
# Create a new column with all product names in lowercase.  
  
lowercase\_names\_df = combined\_df\_noDuplicates.withColumn("ProductNameLower", F.lower(col("ProductName")))  
print("DataFrame with ProductName in lowercase:")  
lowercase\_names\_df.show()

DataFrame with ProductName in lowercase:  
+---------+-----------+-----------+-----+----------+----------------+  
|ProductID|ProductName| Category|Price| SaleDate|ProductNameLower|  
+---------+-----------+-----------+-----+----------+----------------+  
| 1| Product A|Electronics| 1200|2022-05-10| product a|  
| 2| Product B| Clothing| 500|2022-07-15| product b|  
| 3| Product C|Electronics| 1800|2021-11-05| product c|  
| 4| Product D| Furniture| 3000|2022-03-25| product d|  
| 6| Product F|Electronics| 1500|2021-10-19| product f|  
| 5| Product E| Clothing| 800|2022-09-12| product e|  
+---------+-----------+-----------+-----+----------+----------------+

# **Topic**

# **Topic**

# **Topic**

# **Topic**