

**NAME : AKULA SHARATH CHANDRA**

**DATE : 12-02-2024**

**BATCH : DATA ENGINEERING**

### **PY SPARK CODING ASSESSMENT**

**Questions :**

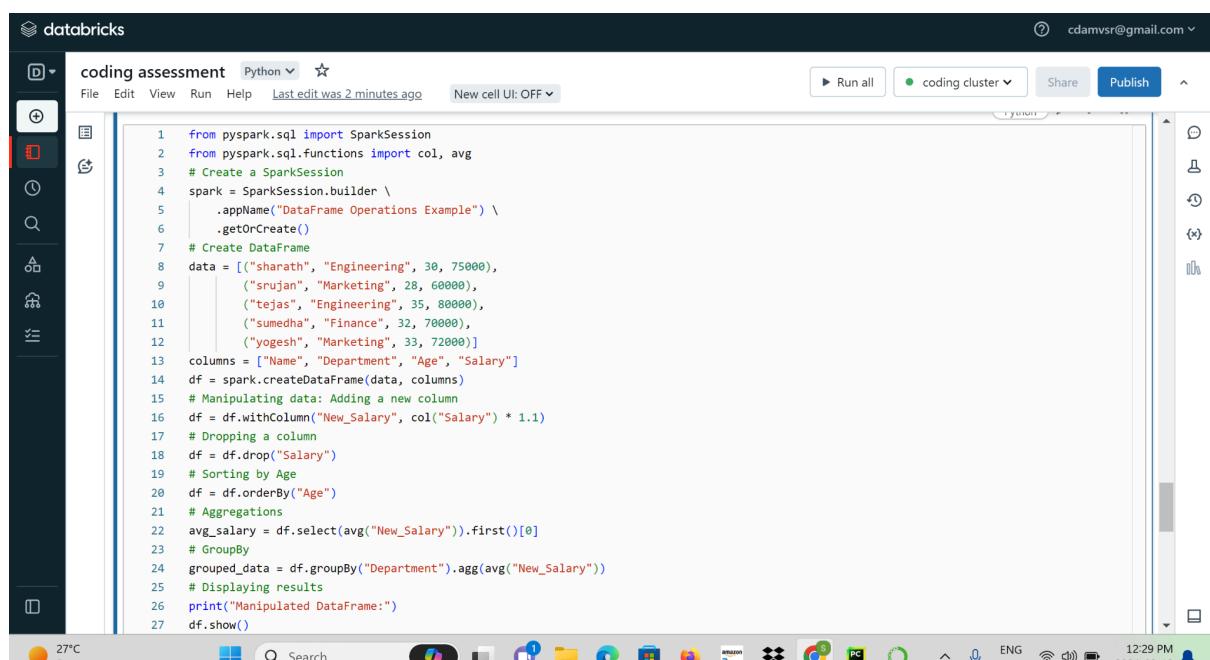
- 1) Execute Manipulating, Dropping, Sorting, Aggregations, Joining, GroupBy DataFrames**
- 2) Execute Pyspark -sparksql joins & Applying Functions in a Pandas DataFrame**

# 1) Execute Manipulating, Dropping, Sorting, Aggregations, Joining, GroupBy DataFrames

## ANSWER:

DataFrame manipulation in PySpark involves various operations such as filtering, dropping unnecessary columns, sorting, aggregating data, joining multiple DataFrames, and grouping data for analysis. Each operation helps in transforming and preparing the data for analysis or visualisation.

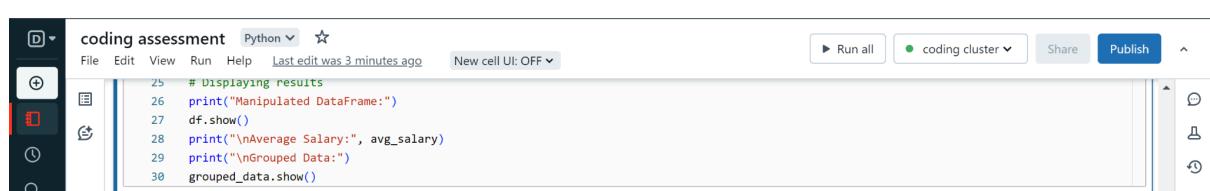
## EXAMPLE :



The screenshot shows a Databricks notebook titled "coding assessment" in Python. The code performs the following operations:

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, avg
3 # Create a SparkSession
4 spark = SparkSession.builder \
5     .appName("DataFrame Operations Example") \
6     .getOrCreate()
7 # Create DataFrame
8 data = [ ("sharath", "Engineering", 30, 75000),
9         ("srujan", "Marketing", 28, 60000),
10        ("tejas", "Engineering", 35, 80000),
11        ("sumedha", "Finance", 32, 70000),
12        ("yogesh", "Marketing", 33, 72000)]
13 columns = ["Name", "Department", "Age", "Salary"]
14 df = spark.createDataFrame(data, columns)
15 # Manipulating data: Adding a new column
16 df = df.withColumn("New_Salary", col("Salary") * 1.1)
17 # Dropping a column
18 df = df.drop("Salary")
19 # Sorting by Age
20 df = df.orderBy("Age")
21 # Aggregations
22 avg_salary = df.select(avg("New_Salary")).first()[0]
23 # GroupBy
24 grouped_data = df.groupby("Department").agg(avg("New_Salary"))
25 # Displaying results
26 print("Manipulated DataFrame:")
27 df.show()
28 print("\nAverage Salary:", avg_salary)
29 print("\nGrouped Data:")
30 grouped_data.show()
```

The notebook interface includes a sidebar with icons for file operations, a search bar, and a toolbar with various tools and status indicators at the bottom.



The screenshot shows the output of the Python code execution. It displays the manipulated DataFrame, the average salary, and the grouped data.

```
Manipulated DataFrame:
+-----+-----+---+-----+
|    Name|Department|Age| New_Salary|
+-----+-----+---+-----+
|  sharath|Engineering| 30|      82500|
|   srujan|Marketing| 28|      66000|
|     tejas|Engineering| 35|      88000|
| sumedha|Finance| 32|      77000|
|   yogesh|Marketing| 33|      79200|
+-----+-----+---+-----+
Average Salary: 77000.0
Grouped Data:
+-----+-----+
| Department| avg(New_Salary)|
+-----+-----+
|Engineering|      82500.0|
|   Marketing|      66000.0|
|   Finance|      77000.0|
+-----+-----+
```

## Outputs:

### i) manipulated dataframe:

A screenshot of a Jupyter Notebook interface titled "coding assessment" in Python. The code cell contains the following:

```
df: pyspark.sql.dataframe.DataFrame = [Name: string, Department: string ... 2 more fields]
grouped_data: pyspark.sql.dataframe.DataFrame = [Department: string, avg(New_Salary): double]
```

The output shows a manipulated DataFrame with columns Name, Department, Age, and New\_Salary, containing the following data:

Name	Department	Age	New_Salary
srujan	Marketing	28	66000.0
sharath	Engineering	30	82500.0
sumedha	Finance	32	77000.0
yogesh	Marketing	33	79200.0
tejas	Engineering	35	88000.0

### ii) average salary:

A screenshot of a Jupyter Notebook cell showing the output of a print statement: "Average Salary: 78540.0".

### iii) grouped data:

A screenshot of a Jupyter Notebook cell showing the output of a grouped data frame. The code cell contains:

```
grouped_data.show()
```

The output shows the grouped data frame with columns Department and avg(New\_Salary), containing the following data:

Department	avg(New_Salary)
Engineering	85250.0
Finance	77000.0
Marketing	72600.0

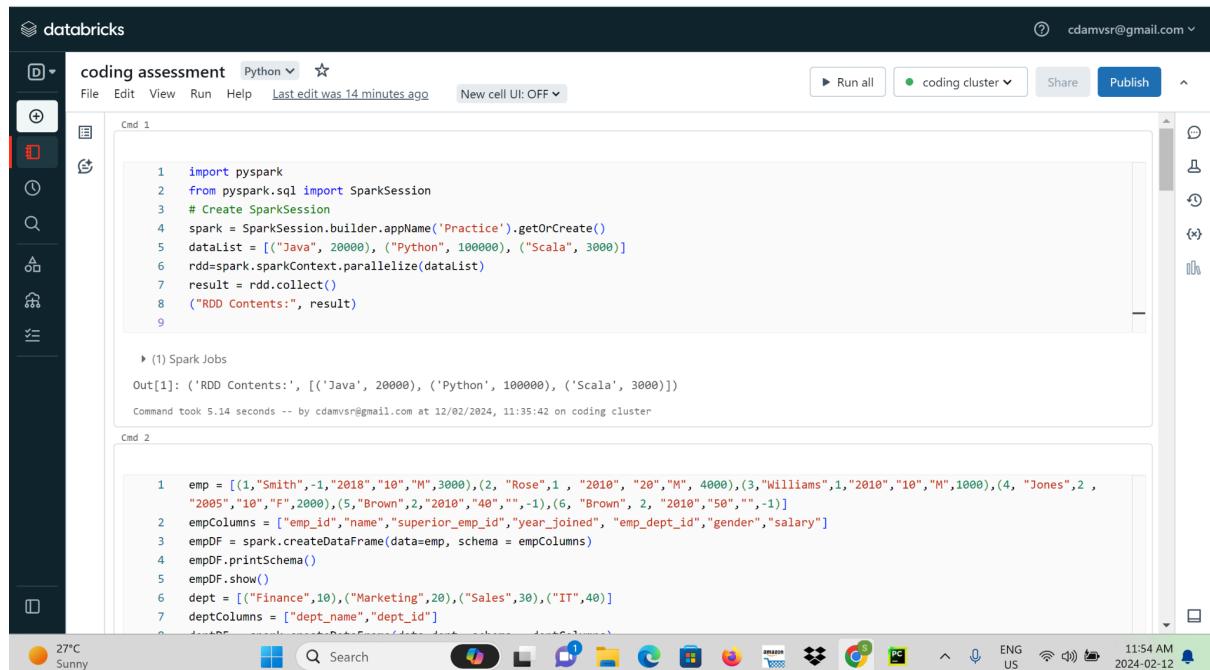
Below the grouped data frame, the average salary is printed: "Average Salary: 78540.0".

## 2) Execute Pyspark -sparksql joins & Applying Functions in a Pandas DataFrame.

### ANSWER:

**JOINS()**: PySpark Join is used to combine two DataFrames and by chaining these you can join multiple DataFrames

### → PERFORMING JOINS: IMPORTING PY SPARK IN DATABRICKS



```
1 import pyspark
2 from pyspark.sql import SparkSession
3 # Create SparkSession
4 spark = SparkSession.builder.appName('Practice').getOrCreate()
5 dataList = [("Java", 2000), ("Python", 10000), ("Scala", 3000)]
6 rdd=spark.sparkContext.parallelize(dataList)
7 result = rdd.collect()
8 ("RDD Contents:", result)
9

(1) Spark Jobs
Out[1]: ('RDD Contents:', [('Java', 2000), ('Python', 10000), ('Scala', 3000)])
Command took 5.14 seconds -- by cdamvsr@gmail.com at 12/02/2024, 11:35:42 on coding cluster

Cmd 2

1 emp = [(1,"Smith",-1,"2018","10","M",3000),(2, "Rose",1 , "2010", "20", "M", 4000),(3,"Williams",1,"2010","10","M",1000),(4, "Jones",2 , "2005","10","F",2000),(5,"Brown",2,"2010","40","","-1),(6, "Brown", 2, "2010","50","","-1)]
2 empColumns = ["emp_id","name","superior_emp_id","year_joined", "emp_dept_id","gender","salary"]
3 empDF = spark.createDataFrame(data=emp, schema = empColumns)
4 empDF.printSchema()
5 empDF.show()
6 dept = [("Finance",10),("Marketing",20),("Sales",30),("IT",40)]
7 deptColumns = ["dept_name","dept_id"]
8 deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
```

# CREATING A DATA FRAME AND DISPLAYING IT :

Databricks Notebook titled "coding assessment" in Python. The code creates two DataFrames, empDF and deptDF, from lists of tuples. The empDF schema includes columns for emp\_id, name, superior\_emp\_id, year\_joined, emp\_dept\_id, gender, and salary. The deptDF schema includes dept\_name and dept\_id. Both DataFrames are printed and shown.

```
1 emp = [(1,"Smith",1,"2018","10","M",3000),(2, "Rose",1 , "2010", "20","M", 4000),(3,"Williams",1,"2010","10","M",1000),(4, "Jones",2 , "2005","10","F",2000),(5,"Brown",2,"2010","40","","-1),(6, "Brown", 2, "2010", "50","","-1)]
2 empColumns = ["emp_id","name","superior_emp_id","year_joined", "emp_dept_id","gender","salary"]
3 empDF = spark.createDataFrame(data=emp, schema = empColumns)
4 empDF.printSchema()
5 empDF.show()
6 dept = [{"Finance":10}, {"Marketing":20}, {"Sales":30}, {"IT":40}]
7 deptColumns = ["dept_name","dept_id"]
8 deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
9 deptDF.printSchema()
10 deptDF.show()
```

(6) Spark Jobs

```
▶ empDF: pyspark.sql.dataframe.DataFrame = [emp_id: long, name: string ... 5 more fields]
▶ deptDF: pyspark.sql.dataframe.DataFrame = [dept_name: string, dept_id: long]
```

root

```
|-- emp_id: long (nullable = true)
|-- name: string (nullable = true)
|-- superior_emp_id: long (nullable = true)
|-- year_joined: string (nullable = true)
|-- emp_dept_id: string (nullable = true)
|-- gender: string (nullable = true)
|-- salary: long (nullable = true)
```

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary
1  Smith  -1  2018  10  M  3000	2  Rose  1  2010  20  M  4000	3  Williams  1  2010  10  M  1000	4  Jones  2  2005  10  F  2000	5  Brown  2  2010  40    -1	6  Brown  2  2010  50    -1	

Databricks Notebook titled "coding assessment" in Python. The code displays the empDF DataFrame and performs an INNER JOIN between empDF and deptDF based on emp\_dept\_id. The resulting DataFrame is printed and shown.

```
|emp_id|     name|superior_emp_id|year_joined|emp_dept_id|gender|salary|
+----+-----+-----+-----+-----+-----+
|  1|  Smith|        -1|    2018|      10|    M| 3000|
|  2|   Rose|         1|    2010|      20|    M| 4000|
|  3|Williams|        1|    2010|      10|    M| 1000|
|  4|  Jones|        2|    2005|      10|    F| 2000|
|  5|   Brown|         2|    2010|      40|     | -1|
|  6|   Brown|         2|    2010|      50|     | -1|
```

root

```
|-- dept_name: string (nullable = true)
|-- dept_id: long (nullable = true)
```

dept_name	dept_id
Finance	10
Marketing	20
Sales	30

Command took 12.24 seconds -- by cdamvsr@gmail.com at 12/02/2024, 11:36:09 on coding cluster

Cmd 3

```
1 #INNER JOIN
2 empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"inner") .show()
```

(3) Spark Jobs

Databricks

coding assessment Python

File Edit View Run Help Last edit was 17 minutes ago New cell UI: OFF Share Publish

```

| 1| Smith|      -1| 2018|    10| M| 3000|
| 2| Rose|       1| 2010|    20| M| 4000|
| 3| Williams|   1| 2010|    10| M| 1000|
| 4| Jones|      2| 2005|    10| F| 2000|
| 5| Brown|      2| 2010|    20| M| 4000|
| 6| Brown|      2| 2010|    50|   | -1|
+---+-----+-----+-----+-----+-----+-----+
root
|-- dept_name: string (nullable = true)
|-- dept_id: long (nullable = true)

+-----+
|dept_name|dept_id|
+-----+
| Finance| 10|
| Marketing| 20|
| Sales| 30|
| IT| 40|
+-----+

```

Command took 12.24 seconds -- by cdamvsr@gmail.com at 12/02/2024, 11:36:09 on coding cluster

Cmd 3

```

1 #INNER JOIN
2 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"inner") .show()

```

(3) Spark Jobs

27°C Sunny ENG US 11:57 AM 2024-02-12

i) **INNER JOIN:** Join records when key columns are matched, and dropped when they are not matched.

Databricks

coding assessment Python

File Edit View Run Help Last edit was 18 minutes ago New cell UI: OFF Share Publish

```

Marketing| 20|
| Sales| 30|
| IT| 40|
+-----+

```

Command took 12.24 seconds -- by cdamvsr@gmail.com at 12/02/2024, 11:36:09 on coding cluster

Cmd 3

```

1 #INNER JOIN
2 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"inner") .show()

```

(3) Spark Jobs

```

+-----+-----+-----+-----+-----+-----+-----+
|emp_id| name|superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+-----+-----+-----+-----+-----+-----+-----+
| 1| Smith|      -1| 2018|    10| M| 3000| Finance| 10|
| 3| Williams|   1| 2010|    10| M| 1000| Finance| 10|
| 4| Jones|      2| 2005|    10| F| 2000| Finance| 10|
| 2| Rose|       1| 2010|    20| M| 4000| Marketing| 20|
| 5| Brown|      2| 2010|    40|   | -1| IT| 40|
+-----+-----+-----+-----+-----+-----+-----+

```

Command took 5.48 seconds -- by cdamvsr@gmail.com at 12/02/2024, 11:37:59 on coding cluster

Cmd 4

1 HOLLOWED JOIN

27°C Sunny ENG US 11:58 AM 2024-02-12

**ii) OUTER JOIN:** Returns all rows from both datasets, where the Join expression doesn't match it returns null or respective columns.

```

1 #OUTER JOIN
2 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"outer").show()
3 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"full").show()
4 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"fullouter").show()
    
```

The screenshot shows a Databricks notebook titled "coding assessment" in Python. It contains four lines of code demonstrating different types of joins. The first two lines show standard outer and full joins, while the third line shows a full outer join. The fourth line is a comment. Below the code, two tables of data are displayed side-by-side. Both tables have columns: emp\_id, name, superior\_emp\_id, year\_joined, emp\_dept\_id, gender, salary, dept\_name, and dept\_id. The top table (standard outer join) has 9 rows, and the bottom table (full outer join) also has 9 rows, including rows where emp\_dept\_id does not match between the two datasets, resulting in null values in the joined columns.

```

1 #OUTER JOIN
2 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"outer").show()
3 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"full").show()
4 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"fullouter").show()
    
```

This screenshot shows the same Databricks notebook and code as the previous one. The execution results are identical, displaying two side-by-side tables of data with 9 rows each, illustrating the results of standard outer, full, and full outer joins.

**iii) LEFT JOIN/ LEFT OUTER JOIN:** Returns all rows from left dataset regardless of match found on right dataset, when join doesn't match - it assigns null for that record.

```

coding assessment Python
File Edit View Run Help Last edit was 21 minutes ago New cell UI: OFF
Run all coding cluster Share Publish Python
1 #LEFT JOIN
2 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"left").show()
3 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"leftouter").show()

(12) Spark Jobs
+---+-----+-----+-----+-----+
|emp_id| name|superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+---+-----+-----+-----+-----+
| 1| Smith|          -1|    2018|      10|M| 3000| Finance| 10|
| 2| Rose|           1|    2018|      20|M| 4000|Marketing| 20|
| 3|Williams|          1|    2018|      10|M| 1000| Finance| 10|
| 4| Jones|           2|    2005|      10|F| 2000| Finance| 10|
| 5| Brown|           2|    2018|      40| | -1| IT| 40|
| 6| Brown|           2|    2018|      50| | -1| null| null|
+---+-----+-----+-----+-----+
+---+-----+-----+-----+-----+
|emp_id| name|superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+---+-----+-----+-----+-----+
| 1| Smith|          -1|    2018|      10|M| 3000| Finance| 10|
| 2| Rose|           1|    2018|      20|M| 4000|Marketing| 20|
| 3|Williams|          1|    2018|      10|M| 1000| Finance| 10|
| 4| Jones|           2|    2005|      10|F| 2000| Finance| 10|
| 5| Brown|           2|    2018|      40| | -1| IT| 40|
| 6| Brown|           2|    2018|      50| | -1| null| null|
+---+-----+-----+-----+-----+
Command took 5.17 seconds -- by cdamvsr@gmail.com at 12/02/2024, 11:41:08 on coding cluster
27°C Sunny Search ENG US 12:01 PM 2024-02-12

```

**iv) RIGHT JOIN/ RIGHT OUTER JOIN:** Returns all rows from Right dataset regardless of match found on left dataset,when Join doesn't match - it assigns null for that record.

```

coding assessment Python
File Edit View Run Help Last edit was 21 minutes ago New cell UI: OFF
Run all coding cluster Share Publish Python
1 #RIGHT JOIN
2 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"right").show()
3 empDF.join(deptDF,empDF.emp_dept_id == deptDF.dept_id,"rightouter").show()
4

(12) Spark Jobs
+---+-----+-----+-----+-----+
|emp_id| name|superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+---+-----+-----+-----+-----+
| 4| Jones|           2|    2005|      10|F| 2000| Finance| 10|
| 3|Williams|          1|    2010|      10|M| 1000| Finance| 10|
| 1| Smith|          -1|    2018|      10|M| 3000| Finance| 10|
| 2| Rose|           1|    2018|      20|M| 4000|Marketing| 20|
| null| null|        null|    null|      null| null| null| Sales| 30|
| 5| Brown|           2|    2018|      40| | -1| IT| 40|
+---+-----+-----+-----+-----+
+---+-----+-----+-----+-----+
|emp_id| name|superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+---+-----+-----+-----+-----+
| 4| Jones|           2|    2005|      10|F| 2000| Finance| 10|
| 3|Williams|          1|    2010|      10|M| 1000| Finance| 10|
| 1| Smith|          -1|    2018|      10|M| 3000| Finance| 10|
| 2| Rose|           1|    2018|      20|M| 4000|Marketing| 20|
| null| null|        null|    null|      null| null| null| Sales| 30|
| 5| Brown|           2|    2018|      40| | -1| IT| 40|
+---+-----+-----+-----+-----+
27°C Sunny Search ENG US 12:01 PM 2024-02-12

```

v) **LEFT SEMI JOIN**: Returns columns from the only left dataset for the matched records in the right dataset on join expression.

The screenshot shows a Jupyter Notebook interface with a Python code cell. The code performs a LEFT SEMI JOIN between two DataFrames, `empDF` and `deptDF`, based on the condition `empDF.emp\_dept\_id == deptDF.dept\_id`. The resulting DataFrame is named "leftsemi". The output displays the joined data as a table:

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary
1	Smith	-1	2018	10	M	3000
3	Williams	1	2018	10	M	1000
4	Jones	2	2005	10	F	2000
2	Rose	1	2018	20	M	4000
5	Brown	2	2018	40		-1

Below the code cell, a message indicates the command took 1.70 seconds to run. The system status bar at the bottom shows the date as 2024-02-12 and the time as 12:02 PM.

vi) **LEFT ANTI JOIN**: Returns only columns from the left dataset for non-matched records.

The screenshot shows a Jupyter Notebook interface with a Python code cell. The code performs a LEFT ANTI JOIN between two DataFrames, `empDF` and `deptDF`, based on the condition `empDF.emp\_dept\_id == deptDF.dept\_id`. The resulting DataFrame is named "leftanti". The output displays the joined data as a table:

emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary
6	Brown	2	2010	50		-1

Below the code cell, a message indicates the command took 1.98 seconds to run. The system status bar at the bottom shows the date as 2024-02-12 and the time as 12:03 PM.

# →APPLYING FUNCTIONS IN A PANDAS DATAFRAME:

## i)selecting renaming filtering data frames :

The screenshot shows a Jupyter Notebook interface with a Python kernel. The code cell contains the following Python code:

```
1 #SELECTING RENAMING FILTERING DATA'S IN PANDAS
2 from pyspark.sql import SparkSession
3 spark = SparkSession.builder.appName('pyspark - example join').getOrCreate()
4 data = [ ('sharath'), '2002-04-01', 'M', 3000),
5         ('srujan'), '2001-05-19', 'M', 4000),
6         ('tejas'), '2002-09-05', 'M', 4000),
7         ('sumedha'), '2001-12-01', 'M', 4000),
8         ('yogesh'), '2000-02-17', 'M', 1200)
9 columns = ["Name", "DOB", "Gender", "salary"]
10 # Create the spark dataframe
11 df = spark.createDataFrame(data=data, schema=columns)
12 df.show()
```

The output of the code is displayed below the code cell, showing the data frame structure:

Name	DOB	Gender	salary
sharath	2002-04-01	M	3000
srujan	2001-05-19	M	4000
tejas	2002-09-05	M	4000
sumedha	2001-12-01	M	4000
yogesh	2000-02-17	M	1200

## ii)ColumnRename():

The screenshot shows a Jupyter Notebook interface with a Python kernel. The code cell contains the following Python code:

```
1 df.withColumnRenamed("DOB","DateOfBirth").show()
```

The output of the code is displayed below the code cell, showing the data frame structure after renaming:

Name	DateOfBirth	Gender	salary
sharath	2002-04-01	M	3000
srujan	2001-05-19	M	4000
tejas	2002-09-05	M	4000
sumedha	2001-12-01	M	4000
yogesh	2000-02-17	M	1200

Below the first code cell, there is a command history entry:

```
Command took 0.55 seconds -- by cdamvsr@gmail.com at 12/02/2024, 12:40:16 on coding cluster
```

The second code cell contains the following Python code:

```
1 #RENAME MULTIPLE COLUMNS
2 df.withColumnRenamed("Gender","Sex").withColumnRenamed("salary","Amount").show()
```

### iii) renaming multiple column:

The screenshot shows a Jupyter Notebook interface with two code cells. The first cell contains:

```
1 #RENAME MULTIPLE COLUMNS
2 df.withColumnRenamed("Gender","Sex").withColumnRenamed("salary","Amount").show()
```

The second cell contains:

```
1 #using expr
2 data = df.selectExpr("Name as name","DOB","Gender","salary")
3 data.show()
```

Both cells output the same data frame, which is a table with columns: Name, DOB, Sex, and Amount. The data is:

	Name	DOB	Sex	Amount
1	sharath	2002-04-01	M	3000
2	srujan	2001-05-19	M	4000
3	tejas	2002-09-05	M	4000
4	sumedha	2001-12-01	M	4000
5	yogesh	2000-02-17	M	1200

### iv) using select expr():

The screenshot shows a Databricks notebook interface with one code cell. The cell contains:

```
1 #using expr
2 data = df.selectExpr("Name as name","DOB","Gender","salary")
3 data.show()
```

The output shows the same data frame as the previous screenshot, with columns: name, DOB, Gender, and salary. The data is:

	name	DOB	Gender	salary
1	sharath	2002-04-01	M	3000
2	srujan	2001-05-19	M	4000
3	tejas	2002-09-05	M	4000
4	sumedha	2001-12-01	M	4000
5	yogesh	2000-02-17	M	1200

## v)using select():

A screenshot of a Jupyter Notebook interface. The title bar says "pyspark coding assessment Python". The code in the cell is:

```
1 #USING SELECT METHOD
2 from pyspark.sql.functions import col
3
4 # Select the 'salary' as 'Amount' using aliasing
5 # Select remaining with their original name
6 data = df.select(col("Name"), col("DOB"),
7                   col("Gender"),
8                   col("salary").alias("Amount"))
9
10 data.show()
```

The output shows the command took 0.58 seconds and displays the DataFrame:

Name	DOB	Gender	Amount
sharath	2002-04-01	M	3000
srujan	2001-05-19	M	4000
tejas	2002-09-05	M	4000
sumedha	2001-12-01	M	4000
yogesh	2000-02-17	M	1200

Command took 0.58 seconds -- by cdamvsr@gmail.com at 12/02/2024, 12:41:29 on coding cluster

## vi)using data frame:

A screenshot of a Jupyter Notebook interface. The title bar says "pyspark coding assessment Python". The code in the cell is:

```
1 #USING DF
2 Data_list = ["Emp Name","Date of Birth",
3              "Gender-m/f","Paid salary"]
4
5 new_df = df.toDF(*Data_list)
6 new_df.show()
7
```

The output shows the command took 0.58 seconds and displays the DataFrame:

Emp Name	Date of Birth	Gender-m/f	Paid salary
sharath	2002-04-01	M	3000
srujan	2001-05-19	M	4000
tejas	2002-09-05	M	4000
sumedha	2001-12-01	M	4000
yogesh	2000-02-17	M	1200

Command took 0.86 seconds -- by cdamvsr@gmail.com at 12/02/2024, 12:41:38 on coding cluster