

PySpark Coding Challenge

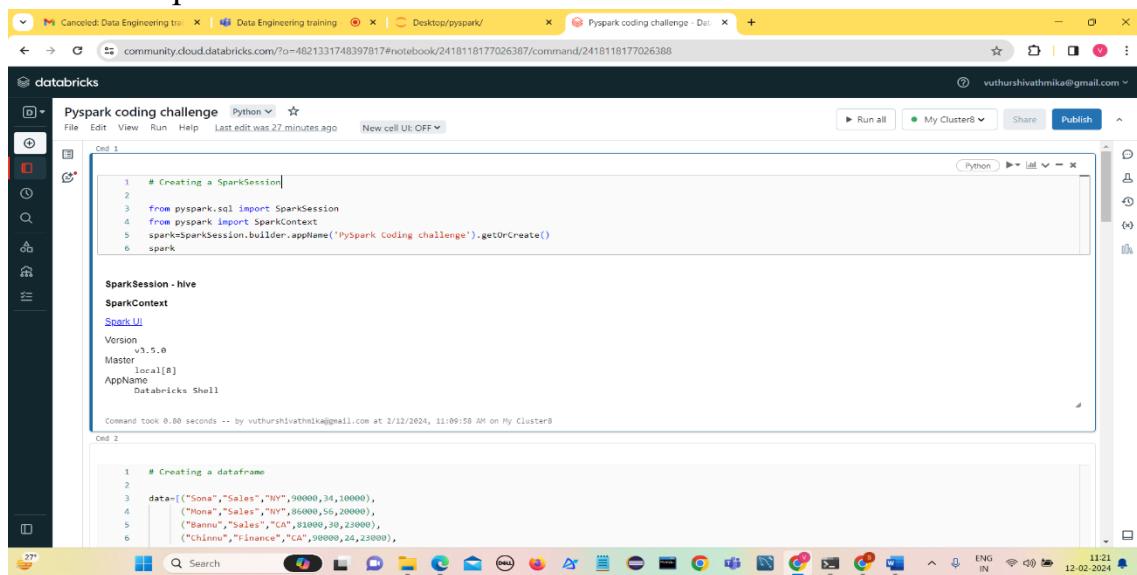
1) Execute Manipulating, Droping, Sorting, Aggregations, Joining, GroupBy DataFrames

a) Manipulating Dataframes

Manipulating dataframes include printing schema, displaying the records, Counting number of rows and columns, names of columns , adding new columns etc.,

Explanation:

Create a Spark Session

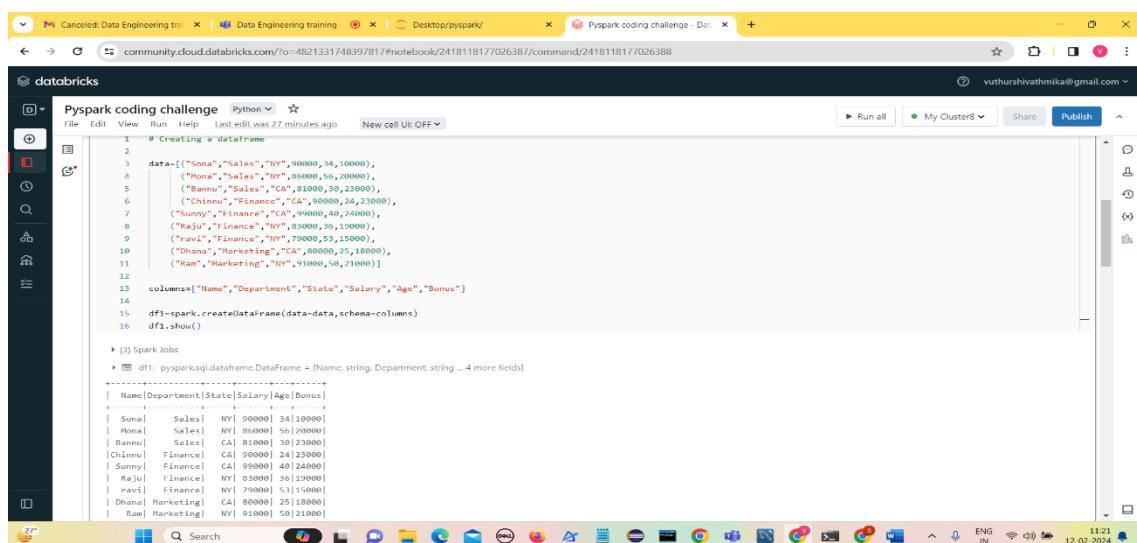


```
1 # Creating a SparkSession
2
3 from pyspark.sql import SparkSession
4 from pyspark import SparkContext
5 spark=SparkSession.builder.appName('PySpark Coding challenge').getOrCreate()
6 spark
```

SparkSession - hive
SparkContext
Spark UI
Version v3.5.0
Master local[8]
AppName databricks Shell

Command took 0.00 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:09:58 AM on My Cluster8

Create a dataframe using Spark Session containing data and columns



```
1 # Creating a DataFrame
2
3 data=[("Sona","Sales","NY",90000,34,10000),
4      ("Monal","Sales","NY",86000,36,10000),
5      ("Bannu","Sales","CA",81000,30,23000),
6      ("Chinmu","Finance","CA",98000,40,23000),
7      ("Sunny","Finance","CA",96000,40,24000),
8      ("Raju","Finance","NY",83000,30,15000),
9      ("Ravi","Finance","NY",79000,53,15000),
10     ("Dhanal","Marketing","CA",80000,25,18000),
11     ("Ram","Marketing","NY",91000,50,21000)]
12
13 columns=['Name','Department','State','Salary','Age','Bonus']
14
15 df1=spark.createDataFrame(data=data,schema=columns)
16 df1.show()
```

(3) Spark Jobs

dt1: pyspark.sql.DataFrame = [Name: string, Department: string ... 4 more fields]

Name	Department	State	Salary	Age	Bonus
Sona	Sales	NY	90000	34	10000
Monal	Sales	NY	86000	36	10000
Bannu	Sales	CA	81000	30	23000
Chinmu	Finance	CA	98000	40	23000
Sunny	Finance	CA	96000	40	24000
Raju	Finance	NY	83000	30	15000
Ravi	Finance	NY	79000	53	15000
Dhanal	Marketing	CA	80000	25	18000
Ram	Marketing	NY	91000	50	21000

a) Manipulating

1) Printing the Schema—prints the schema of dataframe

The screenshot shows a Databricks notebook interface. The code cell contains the following Python code:

```
1 ##### Manipulating
2
3 # 1) schema
4 df1.printSchema()
```

The output shows the schema of the DataFrame:

```
root
 |-- Name: string (nullable = true)
 |-- Department: string (nullable = true)
 |-- State: string (nullable = true)
 |-- Salary: long (nullable = true)
 |-- Age: long (nullable = true)
 |-- Bonus: long (nullable = true)
```

Below the schema, the command took 0.11 seconds to execute.

2) Displaying required number of rows

The code is used to display the required number of rows from dataframe.

The screenshot shows a Databricks notebook interface. The code cell contains the following Python code:

```
1 # 2) show only 2 top records
2 df1.show(2)
```

The output shows the first two rows of the DataFrame:

Name	Department	State	Salary	Age	Bonus
Sona	Sales	NY	90000	34	10000
Mona	Sales	NY	86000	56	20000

Below the output, it says "only showing top 2 rows".

Command took 0.42 seconds to execute.

- 3) **Count number of rows in a dataframe** : It counts number of rows in dataframe
- 4) **Count number of columns**: It returns the number of columns in a dataframe
- 5) **Names of columns**: It returns the names of columns of dataframe

The screenshot shows a Databricks notebook interface with the title 'Pyspark coding challenge'. The notebook contains the following Python code:

```

1 # 3) count number of rows in DF
2 print(df1.count())
3
4 # 4) count of number of columns
5 print(len(df1.columns))
6
7 # 5) names of columns
8 df1.columns
9
10

```

The output of the code is:

```

+-----+
| Name|Department|State|Salary|Age|Bonus|
+-----+
| Sona| Sales    | NY   | 90000| 34|100000|
| Mona| Sales    | NY   | 86000| 56|200000|
+-----+
only showing top 2 rows

Command took 0.42 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:12:30 AM on My Cluster8

```

Below the code cell, there is a section labeled 'Cmd 5' containing the output of the previous command.

- 6) **Adding new columns**—It adds new columns using withColumn and lit() is used to fill the column with constant
- 7) **Select operations**—select() is used to select the required data from dataframe

The screenshot shows a Databricks notebook interface with the title 'Pyspark coding challenge'. The notebook contains the following Python code:

```

1 # 6) Adding new columns
2 from pyspark.sql.functions import lit
3 df1.withColumn("Marks",lit(90)).show()
4
5 # 7) select() operations
6 df1.select(df1.Department,df1.State).show()

```

The output of the code is:

```

+-----+
| Name|Department|State|Salary|Age|Bonus|Marks|
+-----+
| Sona| Sales    | NY   | 90000| 34|100000| 90|
| Mona| Sales    | NY   | 86000| 56|200000| 90|
| Bannu| Sales   | CA   | 81000| 30|23000| 90|
| Chinu| Finance | CA   | 90000| 24|23000| 90|
| Sunny| Finance | CA   | 99000| 40|24000| 90|
| Raju| Finance  | NY   | 83000| 36|19000| 90|
| ravi| Finance | NY   | 79000| 53|15000| 90|
| Dhana| Marketing| CA   | 80000| 25|18000| 90|
| Ram| Marketing| NY   | 91000| 50|21000| 90|
+-----+
|Department|State|
+-----+
| Sales| NY|
| Sales| NY|
| Sales| CA|
| Finance| CA|

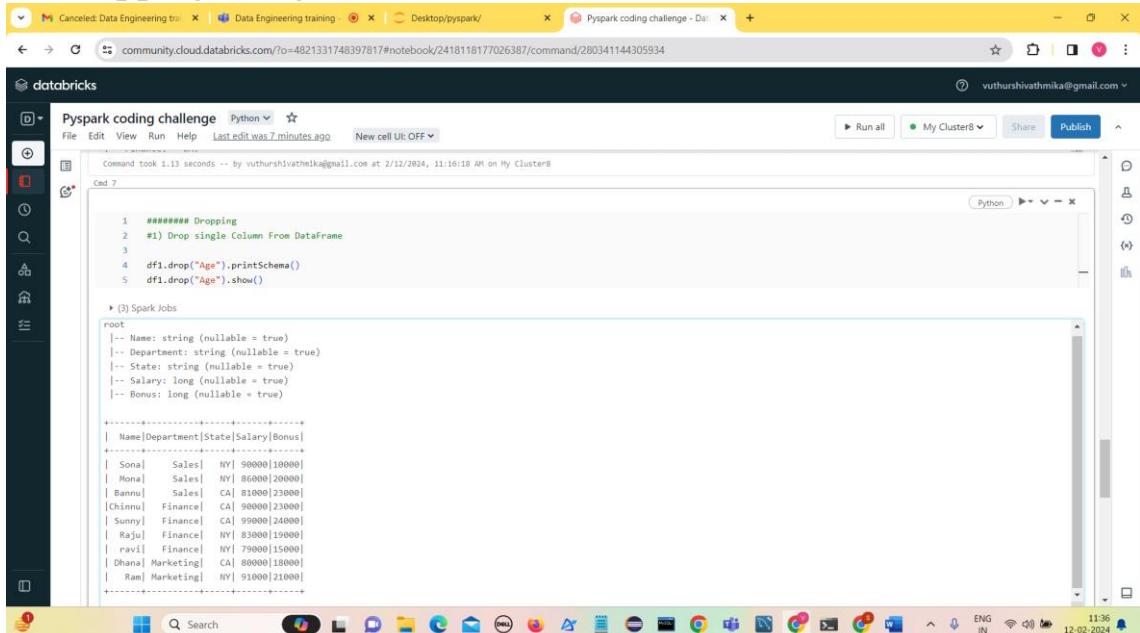
```

Below the code cell, there is a section labeled '(6) Spark Jobs' containing the output of the previous command.

b) Dropping

Dropping in dataframe means dropping the columns from the dataframe. Dropping can be done in two ways like we can drop a single column or at a time we can drop multiple columns.

1) Dropping a single column



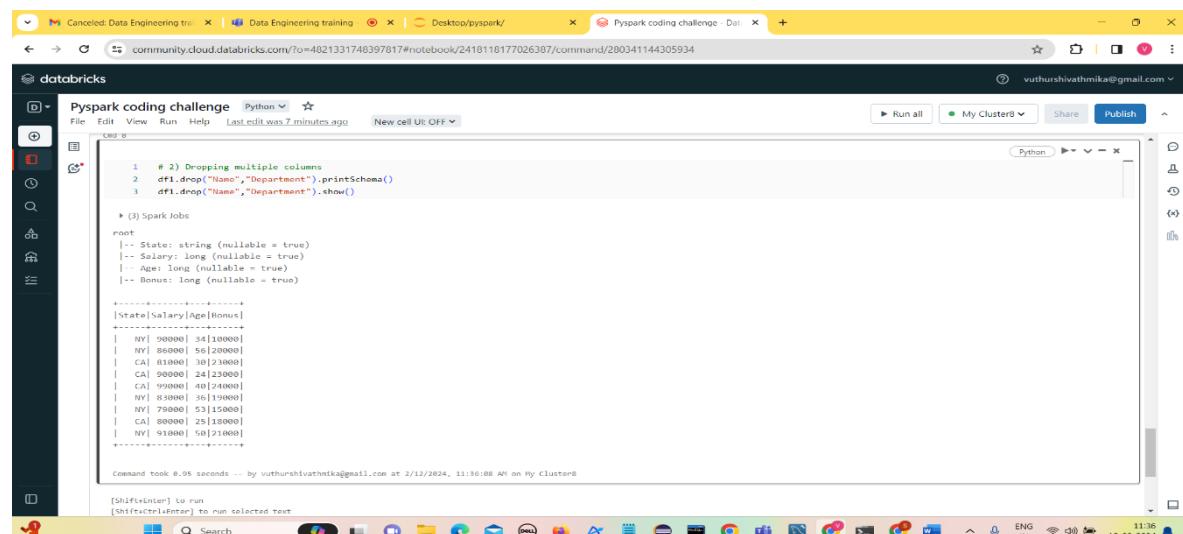
```
1 ##### Dropping
2 #1) Drop single Column From Dataframe
3
4 df1.drop("Age").printSchema()
5 df1.drop("Age").show()
```

The screenshot shows a Databricks notebook titled "Pyspark coding challenge". The code cell contains the above Python code. Below the code, the notebook displays the schema and content of the DataFrame `df1` after dropping the "Age" column. The schema shows columns: Name, Department, State, Salary, and Bonus. The data shows rows for Sona, Mona, Banu, Chinu, Sunny, Raju, ravi, Dhana, and Ram, with their respective details across NY and CA.

Explanation:

In the above code Age column is dropped using drop column in dataframe.

2) Dropping two columns or multiple columns at a time



```
1 # 2) Dropping multiple columns
2 df1.drop("Name","Department").printSchema()
3 df1.drop("Name","Department").show()
```

The screenshot shows a Databricks notebook titled "Pyspark coding challenge". The code cell contains the above Python code. Below the code, the notebook displays the schema and content of the DataFrame `df1` after dropping the "Name" and "Department" columns. The schema shows columns: State, Salary, Age, and Bonus. The data shows rows for Sona, Mona, Banu, Chinu, Sunny, Raju, ravi, Dhana, and Ram, with their respective details across NY and CA.

In the above code two columns Name and Department are dropped.

c) Sorting

sort() or orderBy() function of PySpark DataFrame is used to sort DataFrame by ascending or descending order based on single or multiple columns.

1) Sort single column in ascending order.

The screenshot shows a Databricks notebook titled "Pyspark coding challenge". In the code editor, the following Python code is written:

```
1 ##### Sorting
2 # sort() or orderBy() function of PySpark DataFrame to sort DataFrame by ascending or descending order
3 # based on single or multiple columns
4
5 # i) sort()
6
7 # i) sorting one column in ascending order
8 df1.sort("Salary", ascending=True).show()
```

Below the code, the output shows the sorted DataFrame:

Name	Department	State	Salary	Age	Bonus
ravi	Finance	NY	79000	53	15000
Dhana	Marketing	CA	80000	25	18000
Bannu	Sales	CA	81000	30	23000
Rajul	Finance	NY	82000	36	19000
Mona	Sales	NY	86000	56	20000
Sona	Sales	NY	90000	34	10000
Chinnu	Finance	CA	90000	24	23000
Ram	Marketing	NY	91000	50	21000
Sunny	Finance	CA	99000	40	24000

In this code sort method is used to sort Salary column in ascending order

2) Sorting in decending order

The screenshot shows a Databricks notebook titled "Pyspark coding challenge". In the code editor, the following Python code is written:

```
1 # ii) sorting one column in descending order
2
3 df1.sort("Salary", ascending=False).show()
```

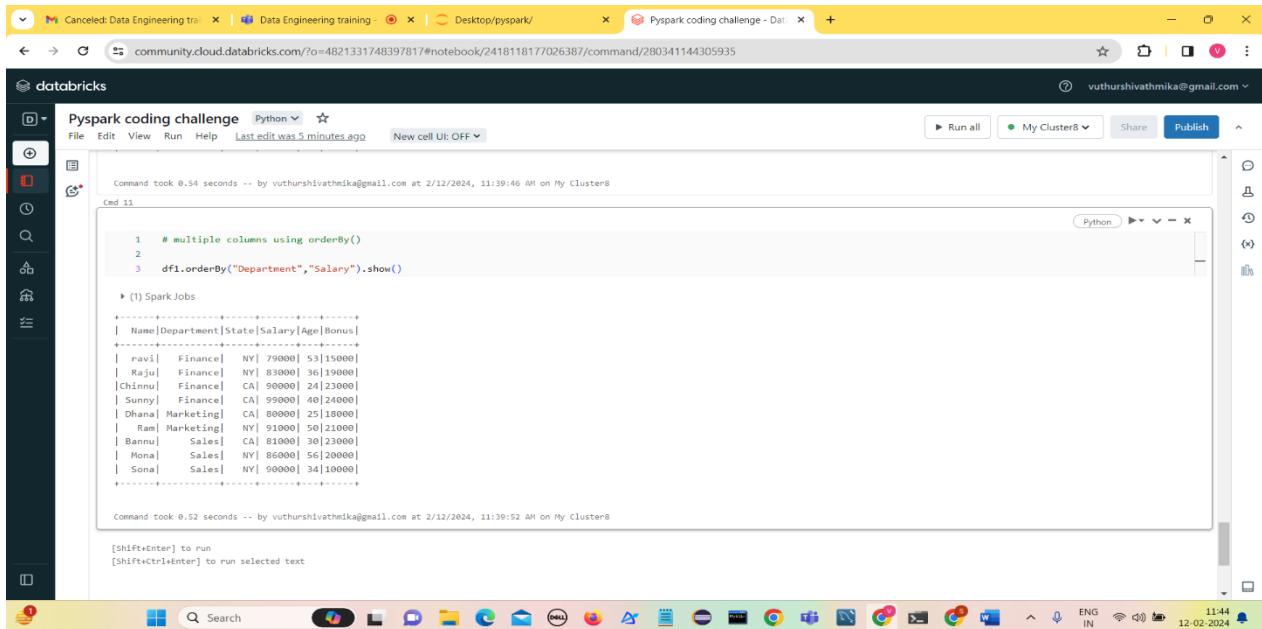
Below the code, the output shows the sorted DataFrame:

Name	Department	State	Salary	Age	Bonus
Mona	Sales	NY	80000	56	20000
Sona	Sales	NY	90000	34	10000
Chinnu	Finance	CA	90000	24	23000
Ram	Marketing	NY	91000	50	21000
Sunny	Finance	CA	99000	40	24000

In this code sort method is used to sort Salary column in descending order

3) orderBy()

orderBy() is also same as sort() method used to sort in ascending or descending order.



```
1 # multiple columns using orderBy()
2
3 df1.orderBy("Department","Salary").show()
```

(1) Spark Jobs

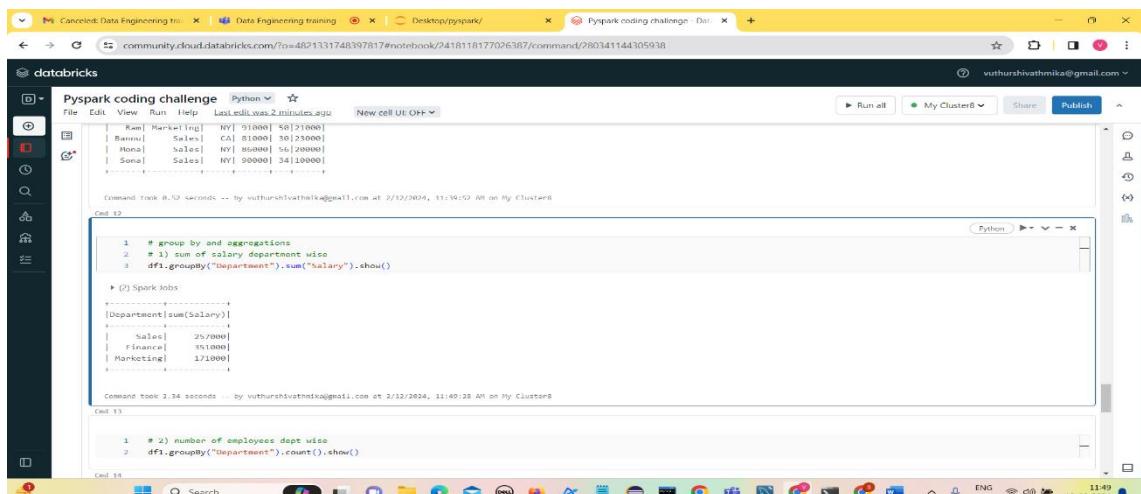
Name	Department	State	Salary	Age	Bonus
Ram	Marketing	NY	81000	58	12000
Bannu	Sales	CA	81000	30	23000
Hona	Sales	NY	86000	56	20000
Sona	Sales	NY	90000	34	10000
Kaju	Finance	NY	83000	36	19000
Chinu	Finance	CA	90000	24	23000
Sunny	Finance	CA	99000	40	24000
Dhana	Marketing	CA	80000	25	18000
Rami	Marketing	NY	91000	50	21000
Mona	Sales	CA	81000	30	23000
Banu	Sales	NY	86000	56	20000

d) Group by

e) Aggregations

PySpark groupBy() function is used to collect the identical data into groups on DataFrame and perform count, sum, avg, min, and max functions on the grouped data.

1) sum of salary department wise



```
1 # group by and aggregations
2 # i) sum of salary department wise
3 df1.groupby("Department").sum("salary").show()
```

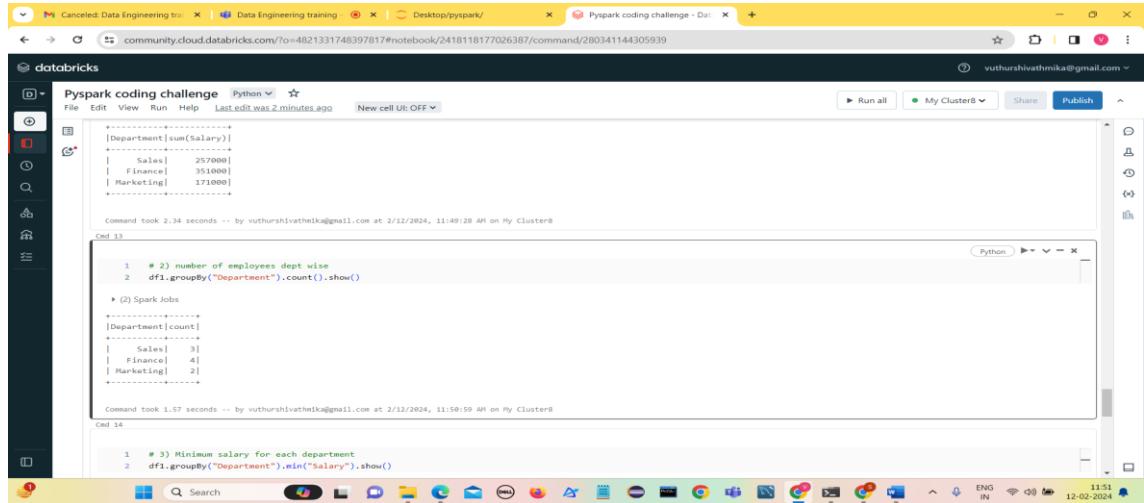
(2) Spark Jobs

Department	sum(Salary)
Sales	257000
Finance	351000
Marketing	171000

The above code gives the sum of salary department wise

2) Count()

This code gives the number of employees in each department



The screenshot shows a Databricks notebook titled "Pyspark coding challenge" in Python. The notebook contains three cells:

- Cell 13:** Displays the result of a group-by operation on the "Department" column and summing the "Salary" column. The output is:

Department sum(Salary)
Sales 257000
Finance 351000
Marketing 171000

Command took 2.34 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:49:28 AM on My Cluster8
- Cell 14:** Displays the result of a group-by operation on the "Department" column and counting the number of employees. The output is:

Department count
Sales 3
Finance 4
Marketing 2

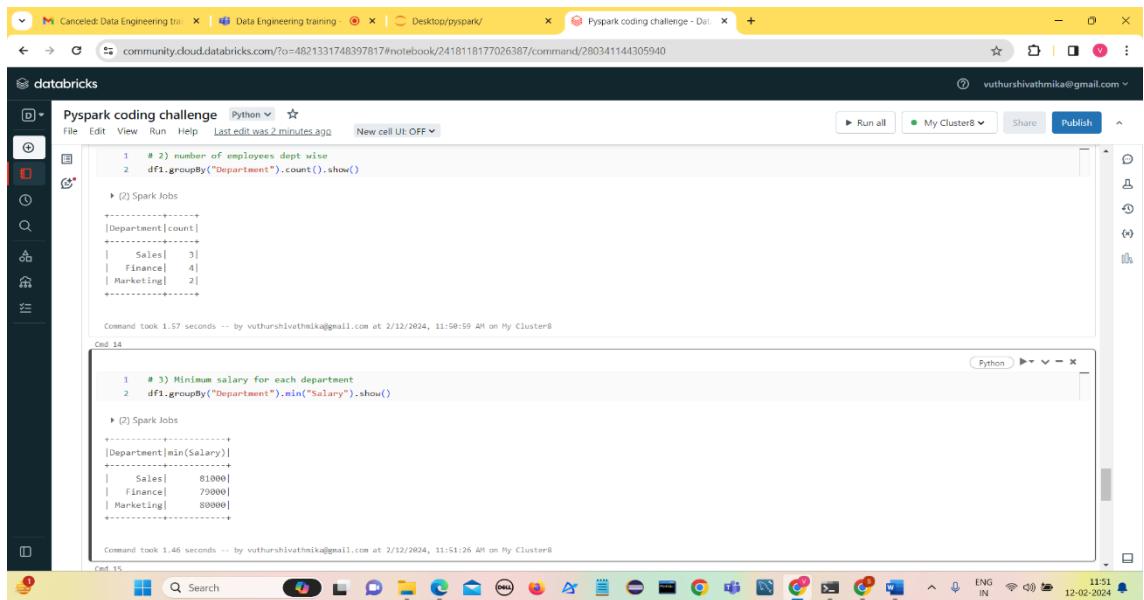
Command took 1.57 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:50:59 AM on My Cluster8
- Cell 15:** Displays the result of a group-by operation on the "Department" column and finding the minimum salary. The output is:

Department min(Salary)
Sales 81000
Finance 79000
Marketing 80000

Command took 1.46 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:51:26 AM on My Cluster8

3) Min()

This code finds the minimum salary in each department



The screenshot shows a Databricks notebook titled "Pyspark coding challenge" in Python. The notebook contains three cells:

- Cell 13:** Displays the result of a group-by operation on the "Department" column and summing the "Salary" column. The output is:

Department sum(Salary)
Sales 257000
Finance 351000
Marketing 171000

Command took 1.57 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:50:59 AM on My Cluster8
- Cell 14:** Displays the result of a group-by operation on the "Department" column and counting the number of employees. The output is:

Department count
Sales 3
Finance 4
Marketing 2

Command took 1.46 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:51:26 AM on My Cluster8
- Cell 15:** Displays the result of a group-by operation on the "Department" column and finding the minimum salary. The output is:

Department min(Salary)
Sales 81000
Finance 79000
Marketing 80000

Command took 1.46 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:51:26 AM on My Cluster8

4) Max() This code gives maximum salary in each department

Databricks Notebook Screenshot showing Python code to find the maximum salary in each department:

```
1 # 4) Maximum salary of each dept
2 df1.groupby("Department").max("Salary").show()
```

Output:

Department	max(Salary)
Sales	91000
Finance	79000
Marketing	68000

5) Avg() This code gives average salary in each department

Databricks Notebook Screenshot showing Python code to find the average salary in each department:

```
1 # 5) Average salary of each dept
2 df1.groupby("Department").avg("Salary").show()
```

Output:

Department	avg(Salary)
Sales	90000
Finance	99000
Marketing	91000

6) Mean() This code gives mean salary in each department

Databricks Notebook Screenshot showing Python code to find the mean salary in each department:

```
1 # 6) Mean of salary dept wise
2 df1.groupby("Department").mean("Salary").show()
```

Output:

Department	avg(Salary)
Sales	85666.66666666667
Finance	87750.0
Marketing	85500.0

7) Group by multiple columns and sum of two columns

Pyspark coding challenge

```
[Department] avg(Salary)
+-----+
| Sales| 85666.66666666667|
| Finance| 87750.0|
| Marketing| 85500.0|
+-----+
```

Command took 0.84 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:52:42 AM on My Cluster8

Cmd 18

```
1 # 7) group by multiple columns
2 # group by department,state and find sum of salary and bonus
3 df1.groupBy("Department","State").sum("Salary","Bonus").show()
4
5
```

▶ (2) Spark jobs

```
+-----+-----+-----+
|Department|State|sum(Salary)|sum(Bonus)|
+-----+-----+-----+
| Sales| NY| 176000| 30000|
| Sales| CA| 81000| 23000|
| Finance| CA| 189000| 47000|
| Finance| NY| 162000| 34000|
| Marketing| NY| 91000| 21000|
| Marketing| CA| 88000| 18000|
+-----+-----+-----+
```

Command took 1.56 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:53:38 AM on My Cluster8

8) Running more than one aggregates at a time using agg() function

Pyspark coding challenge

```
| Sales| CA| 81000| 23000|
| Finance| CA| 189000| 47000|
| Finance| NY| 162000| 34000|
| Marketing| NY| 91000| 21000|
| Marketing| CA| 88000| 18000|
+-----+
```

Command took 1.56 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:53:38 AM on My Cluster8

Cmd 19

```
1 #8) Running more aggregates at a time
2 from pyspark.sql.functions import sum,avg,min,max,mean,count
3 df1.groupBy("Department").agg(sum("Salary").alias("sum_salary"),
4                               avg("Salary").alias("avg_salary"),
5                               min("Salary").alias("Min_salary"),
6                               max("Salary").alias("Max_Salary"),
7                               mean("Salary").alias("Mean_Salary"),
8                               count("*").alias("Total count")).show()
```

▶ (2) Spark jobs

```
+-----+-----+-----+-----+-----+
|Department|sum_salary| avg_salary|Min_salary|Max_Salary| Mean Salary|Total count|
+-----+-----+-----+-----+-----+
| Sales| 257000|85666.66666666667| 81000| 90000|85666.66666666667| 3|
| Finance| 351000| 87750.0| 79000| 95000| 87750.0| 4|
| Marketing| 171000| 85500.0| 80000| 91000| 85500.0| 2|
+-----+-----+-----+-----+-----+
```

Command took 1.59 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 11:54:19 AM on My Cluster8

f) Joins

PySpark Join is used to combine two DataFrames based on the common column.

1) Inner Join:

Join records when key values are matched, and dropped when they are not matched.

Create two dataframes employee personal details and employee salary details for performing joins.

The screenshot shows a Databricks notebook interface. The notebook title is "Pyspark coding challenge". The code cell contains the following Python code:

```
1 # joins
2
3 # Creating a dataframe for Employee Personal Details
4 data=[("1","Sona","TCS"),
5       ("2","Mona","Infosys"),
6       ("3","Bannu","Deloitte"),
7       ("4","Sunny","TCS"),
8       ("5","Chinnu","Accenture")]
9 cols=["ID","Name","Company"]
10 df1=spark.createDataFrame(data,cols)
11 df1.show()
```

The output of the code shows a DataFrame named df1 with the following data:

ID	Name	Company
1	Sona	TCS
2	Mona	Infosys
3	Bannu	Deloitte
4	Sunny	TCS
5	Chinnu	Accenture

The screenshot shows two Databricks notebooks side-by-side. Both notebooks have the title "Pyspark coding challenge" and are set to Python. The top notebook (Cmd 21) contains the following code:

```

1 # Creating a dataframe of employee salary details
2 data=[("1",45000,"HR"),
3       ("2",90000,"Manager"),
4       ("6",78000,"IT"),
5       ("5",50000,"Sales")]
6
7 cols=["ID","Salary","Department"]
8 df2=spark.createDataFrame(data=data,schema=cols)
9 df2.show()

```

The output shows a DataFrame with four rows:

ID	Salary	Department
1	45000	HR
2	90000	Manager
6	78000	IT
5	50000	Sales

The bottom notebook (Cmd 22) contains the following code:

```

1 # 1) Inner join
2 df1.join(df2,df1.ID==df2.ID,"inner").show()

```

The output shows the result of an inner join between the two DataFrames:

ID	Name	Company	ID	Salary	Department
1	Sona	TCS	1	45000	HR
2	Mona	Infosys	2	90000	Manager
5	Chinu	Accenture	5	50000	Sales

Both notebooks were run on a cluster named "My Cluster8". The timestamp on the bottom notebook indicates it was run at 12:01 PM on 12-02-2024.

2) Outer Join

Returns all rows from both datasets, if join expression doesn't match it returns null values. (right join + left join)

```

1 # 2) outer join
2 df1.join(df2,df1.ID==df2.ID,"outer").show()
3

▶ (3) Spark Jobs
+-----+
| ID| Name| Company| ID|Salary|Department|
+-----+
| 1| Sona| TCS| 1| 45000| HR|
| 2| Mona| Infosys| 2| 90000| Manager|
| 3| Bannu| Deloitte|NULL| NULL| NULL|
| 4| Sunny| TCS|NULL| NULL| NULL|
| 5|Chinu|Accenture| 5| 50000| Sales|
|NULL| NULL| NULL| 6| 78000| IT|
+-----+

Command took 1.30 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:02:48 PM on My Cluster8

```

3) Left Join

Returns all rows from the left dataset irrespective of match found on the right dataset ,when join doesn't match it assigns null for that record

```

1 # 3) left join
2 df1.join(df2,df1.ID==df2.ID,"leftouter").show()
3

▶ (5) Spark Jobs
+-----+
| ID| Name| Company| ID|Salary|Department|
+-----+
| 1| Sona| TCS| 1| 45000| HR|
| 2| Mona| Infosys| 2| 90000| Manager|
| 3| Bannu| Deloitte|NULL| NULL| NULL|
| 4| Sunny| TCS|NULL| NULL| NULL|
| 5|Chinu|Accenture| 5| 50000| Sales|
|NULL| NULL| NULL| 6| 78000| IT|
+-----+

Command took 1.38 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:02:48 PM on My Cluster8

```

4) Right join

Returns all rows from the right dataset irrespective of match found on the left dataset ,when join doesn't match it assigns null for that record

```

| ID| Name | Company| ID|Salary|Department|
| 1| Sona| TCS| 1| 45000| HR|
| 2| Mona| Infosys| 2| 90000| Manager|
| 3| Banu| Deloitte|NULL| NULL| NULL|
| 4| Sunny| TCS|NULL| NULL| NULL|
| 5| Chinu| Accenture| 5| 50000| Sales|

```

Command took 2.04 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:04:44 PM on My Cluster8

```

1 # 4) Right join
2 df1.join(df2,df1.ID==df2.ID,"right").show()

# (3) Spark Jobs

```

```

| ID| Name | Company| ID|Salary|Department|
| 1| Sona| TCS| 1| 45000| HR|
| 2| Mona| Infosys| 2| 90000| Manager|
| NULL| NULL| NULL| 6| 78000| IT|
| 5| Chinu| Accenture| 5| 50000| Sales|

```

Command took 1.00 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:05:13 PM on My Cluster8

5) Left semi join

returns columns from the only left dataset for the records match in the right dataset on join expression, records not matched on join expression are ignored from both left and right datasets.

```

1 # 4) Right join
2 df1.join(df2,df1.ID==df2.ID,"right").show()

# (3) Spark Jobs

```

```

| ID| Name | Company| ID|Salary|Department|
| 1| Sona| TCS| 1| 45000| HR|
| 2| Mona| Infosys| 2| 90000| Manager|
| NULL| NULL| NULL| 6| 78000| IT|
| 5| Chinu| Accenture| 5| 50000| Sales|

```

Command took 1.00 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:05:13 PM on My Cluster8

```

1 # 5) left semi
2 df1.join(df2,df1.ID==df2.ID,"leftsemi").show()

# (3) Spark Jobs

```

```

| ID| Name | Company|
| 1| Sona| TCS|
| 2| Mona| Infosys|
| 5| Chinu| Accenture|

```

Command took 1.91 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:06:21 PM on My Cluster8

6) Left Anti Join

leftanti join returns only columns from the left dataset for non-matched record

The screenshot shows a Databricks notebook interface with three tabs at the top: "Cancelled: Data Engineering trial", "Data Engineering training", and "Pyspark coding challenge - Dat...". The notebook title is "Pyspark coding challenge". The code cell contains the following Python code:

```
1 # 6) left anti
2 df1.join(df2,df1.ID==df2.ID,"leftanti").show()
```

The output of the code shows the following data frame:

ID	Name	Company
1	Sona	TCS
2	Mona	Infosys
5	Chinu	Accenture

Below the code cell, there is a note: "[Shift+Enter] to run [Shift+Ctrl+Enter] to run selected text". The status bar at the bottom right shows the date and time: "12-02-2024 12:07".

Question-2

Execute Pyspark -sparksql joins & Applying Functions in a Pandas DataFrame

Pyspark SQL JOINS

Create a SparkSession and using sparksql create a database

The screenshot shows a Databricks notebook titled "Pyspark coding challenge" in Python. The notebook contains three code cells:

```
1 # Question-2
2 from pyspark.sql import SparkSession
3 spark = SparkSession.builder.appName("SparkCreateTableExample").getOrCreate()
```

```
1 # creating a database
2 spark.sql("CREATE DATABASE IF NOT EXISTS db;")
```

```
1 # creating a table Employee3
2 spark.sql("CREATE TABLE IF NOT EXISTS Employee3(emp_id Int, emp_name String,Salary INT,age INT,city String)")
```

After running these commands, a database named "db" is created, and a table named "Employee3" is created with the specified schema.

After creating a database create a table Employee3 and insert values into it

The screenshot shows a Databricks notebook titled "Pyspark coding challenge" in Python. The notebook contains four code cells:

```
1 # creating a database
2 spark.sql("CREATE DATABASE IF NOT EXISTS db;")
```

```
1 # creating a table Employee3
2 spark.sql("CREATE TABLE IF NOT EXISTS Employee3(emp_id Int, emp_name String,Salary INT,age INT,city String)")
```

```
1 spark.sql("INSERT INTO Employee3 VALUES (1,'Sona',80000,22,'Hyderabad')")
3 spark.sql("insert into employee3 values(2,'Sunny',70000,23,'Pune'),(3,'Mona',90000,19,'Hyderabad'),(7,'Chinu',45000,25,'Mumbai'),(10,'Bannu',12,27,'Pune'),(6,'Raju',70000,45,'Mumbai'),(18,'Dhana',89999,34,'Kolkata')")
```

```
DataFrame[1]: num_affected_rows: bigint, num_inserted_rows: bigint]
```

After running these commands, data is inserted into the "Employee3" table. The table has columns: emp_id, emp_name, Salary, age, and city. The data includes rows for Sona, Sunny, Mona, Chinu, Bannu, Raju, and Dhana.

Displaying the records of Employee3 table

Databricks Notebook interface showing the execution of PySpark code to insert data into Employee3 and then select all records from it.

```
Cmd 31
1 spark.sql("INSERT INTO Employee3 VALUES (1,'Sona',80000,22,'Hyderabad')")
2 spark.sql("insert into employee3 values(2,'Sunny',70000,23,'Pune'),(3,'Mona',90000,19,'Hyderabad'),(7,'Chinu',45000,25,'Mumbai'),(10,'Bannu',12,27,'Pune'),(6,'Raju',70000,45,'Mumbai'),(18,'Dhana',89999,34,'Kolkata')")

▶ (12) Spark Jobs
DataFrame[num_affected_rows: bigint, num_inserted_rows: bigint]
Command took 13.15 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:15:33 PM on My Cluster8

Cmd 32
1 spark.sql(" select * from employee3").show()

▶ (3) Spark Jobs
+-----+-----+-----+
|emp_id|emp_name|Salary|age|city|
+-----+-----+-----+
|  2|  Sunny| 70000| 23|  Pune|
|  3|  Mona| 90000| 19|Hyderabad|
|  7| Chinu| 45000| 25|  Mumbai|
| 10| Bannu|   12| 27|  Pune|
|  6|  Raju| 70000| 45|  Mumbai|
| 18| Dhana| 89999| 34|Kolkata|
|  1|  Sona| 80000| 22|Hyderabad|
+-----+-----+-----+
```

Command took 3.75 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:18:18 PM on My Cluster8

Create another table for Department1 using spark.sql command

Databricks Notebook interface showing the creation of Department1 table and its contents.

```
Cmd 33
1 # Joins
2 # create another table department
3 spark.sql("create table if not exists Department1(dept_id int,emp_id int,dept_name string)")
4
5 # insert records into department table
6 spark.sql("insert into Department1 values(100,1,'IT'),(102,6,'HR'),(103,13,'Manager'),(109,7,'Developer'),(110,10,'Tester')")
7

▶ (9) Spark Jobs
DataFrame[num_affected_rows: bigint, num_inserted_rows: bigint]
Command took 9.32 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:19:17 PM on My Cluster8

Cmd 34
1 spark.sql(" select * from department1").show()

▶ (2) Spark Jobs
+-----+-----+
|dept_id|emp_id|dept_name|
+-----+-----+
| 100|     1|      IT|
| 102|     6|      HR|
| 103|    13| Manager|
| 109|     7|Developer|
| 110|    10| Tester|
+-----+-----+
```

Command took 1.54 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:19:32 PM on My Cluster8

1) Inner Join

Join records when key values are matched, and dropped when they are not matched.

```

1 # Inner join
2 spark.sql("select * from employee3 inner join department1 on employee3.emp_id=department1.emp_id").show()

```

(3) Spark Jobs

```

| [emp_id][emp_name][Salary][age] | [city][dept_id][emp_id][dept_name] | | | | | | |
| 7| Chinnu| 45000| 25| Mumbai| 109| 7|Developer|
| 10| Banu| 12| 27| Pune| 118| 10| Tester|
| 6| Raju| 70000| 45| Mumbai| 102| 6| HR|
| 1| Sona| 80000| 22|Hyderabad| 100| 1| IT|

```

2) Outer join

Returns all rows from both tables, if join expression doesn't match it returns null values. (right join + left join)

```

1 # Outer join
2 # Full outer join
3 spark.sql("select * from employee3 full outer join department1 on employee3.emp_id=department1.emp_id").show()
4
5 spark.sql("select * from employee3 full join department1 on employee3.emp_id=department1.emp_id").show()

```

(6) Spark Jobs

```

| [emp_id][emp_name][Salary][age] | [city][dept_id][emp_id][dept_name] | | | | | | |
| 1| Sona| 80000| 22|Hyderabad| 100| 1| IT|
| 2| Sunny| 70000| 23| Pune| NULL| NULL| NULL|
| 3| Mona| 90000| 19|Hyderabad| NULL| NULL| NULL|
| 6| Raju| 70000| 45| Mumbai| 102| 6| HR|
| 7| Chinnu| 45000| 25| Mumbai| 109| 7|Developer|
| 10| Banu| 12| 27| Pune| 118| 10| Tester|
| NULL| NULL| NULL| NULL| 103| 13| Manager|
| 18| Dhana| 89999| 34| Kolkata| NULL| NULL| NULL|

```



```

| [emp_id][emp_name][Salary][age] | [city][dept_id][emp_id][dept_name] | | | | | | |
| 1| Sona| 80000| 22|Hyderabad| 100| 1| IT|
| 2| Sunny| 70000| 23| Pune| NULL| NULL| NULL|
| 3| Mona| 90000| 19|Hyderabad| NULL| NULL| NULL|
| 6| Raju| 70000| 45| Mumbai| 102| 6| HR|
| 7| Chinnu| 45000| 25| Mumbai| 109| 7|Developer|

```

3) Left Join

Returns all rows from the left table irrespective of match found on the right table ,when join doesn't match it assigns null for that record

```

1 # Left join
2 # Left outer join
3 spark.sql("select * from employee3 left join department1 on employee3.emp_id=department1.emp_id").show()
4
5 spark.sql("select * from employee3 left outer join department1 on employee3.emp_id=department1.emp_id").show()

```

(6) Spark Jobs

	[emp_id]	[emp_name]	[Salary]	[age]	[city]	[dept_id]	[emp_id]	[dept_name]
1	Sunny	78000	23	Pune	NULL	NULL	NULL	
2	Mona	98000	19	Hyderabad	NULL	NULL	NULL	
7	Chinu	45000	25	Mumbai	109	7	Developer	
10	Bannu	12	27	Pune	110	10	Tester	
6	Raju	78000	45	Mumbai	102	6	HR	
18	Dhana	89999	34	Kolkata	NULL	NULL	NULL	
1	Sona	88000	22	Hyderabad	100	1	IT	

	[emp_id]	[emp_name]	[Salary]	[age]	[city]	[dept_id]	[emp_id]	[dept_name]
2	Sunny	78000	23	Pune	NULL	NULL	NULL	
3	Mona	98000	19	Hyderabad	NULL	NULL	NULL	
7	Chinu	45000	25	Mumbai	109	7	Developer	
10	Bannu	12	27	Pune	110	10	Tester	
6	Raju	78000	45	Mumbai	102	6	HR	
18	Dhana	89999	34	Kolkata	NULL	NULL	NULL	

Command took 3.29 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:25:58 PM on My Cluster8

4) Right Join

Returns all rows from the right table irrespective of match found on the left table ,when join doesn't match it assigns null for that record

```

1 # Right join
2 # Right outer join
3 spark.sql("select * from employee3 right outer join department1 on employee3.emp_id=department1.emp_id").show()
4
5 spark.sql("select * from employee3 right join department1 on employee3.emp_id=department1.emp_id").show()

```

(4) Spark Jobs

	[emp_id]	[emp_name]	[Salary]	[age]	[city]	[dept_id]	[emp_id]	[dept_name]
1	Sona	80000	22	Hyderabad	100	1	IT	
6	Raju	70000	45	Mumbai	102	6	HR	
NULL	NULL	NULL	NULL	NULL	103	13	Manager	
7	Chinu	45000	25	Mumbai	109	7	Developer	
10	Bannu	12	27	Pune	110	10	Tester	

	[emp_id]	[emp_name]	[Salary]	[age]	[city]	[dept_id]	[emp_id]	[dept_name]
1	Sona	80000	22	Hyderabad	100	1	IT	
6	Raju	70000	45	Mumbai	102	6	HR	
NULL	NULL	NULL	NULL	NULL	103	13	Manager	
7	Chinu	45000	25	Mumbai	109	7	Developer	
10	Bannu	12	27	Pune	110	10	Tester	

Command took 3.29 seconds -- by VUTHURSHIVATHMIIKA@gmail.com at 2/12/2024, 12:25:58 PM on My Cluster8

5) Left semi join

returns columns from the only left table for the records match in the right table on join expression, records not matched on join expression are ignored from both left and right tables.

```

1 | 1| Sona| 88000| 22|Hyderabad| 100| 1| IT|
| 6| Rajul| 70000| 45| Mumbai| 102| 6| HR|
| NULL| NULL| NULL| NULL| NULL| 103| 13| Manager|
| 7| Chinnu| 45000| 25| Mumbai| 109| 7| Developer|
| 10| Bannu| 12| 27| Pune| 118| 10| Tester|
+---+---+---+---+---+---+---+---+

```

Command took 3.07 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:26:24 PM on My Cluster8

```

1 # Left semi join
2 spark.sql("select * from employee3 left semi join department1 on employee3.emp_id=department1.emp_id").show()

▶ (3) Spark Jobs
+---+---+---+---+---+
|emp_id|emp_name|Salary|age|city|
+---+---+---+---+---+
| 7| Chinnu| 45000| 25| Mumbai|
| 10| Bannu| 12| 27| Pune|
| 6| Rajul| 70000| 45| Mumbai|
| 1| Sona| 88000| 22|Hyderabad|
+---+---+---+---+---+

```

Command took 1.68 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:28:15 PM on My Cluster8

```

1 # Left anti join
2 spark.sql("select * from employee3 left anti join department1 on employee3.emp_id=department1.emp_id").show()

```

6) Left anti join

leftanti join returns only columns from the left table for non-matched record

```

+---+---+---+---+---+
|emp_id|emp_name|Salary|age|city|
+---+---+---+---+---+
| 7| Chinnu| 45000| 25| Mumbai|
| 10| Bannu| 12| 27| Pune|
| 6| Rajul| 70000| 45| Mumbai|
| 1| Sona| 88000| 22|Hyderabad|
+---+---+---+---+---+

```

Command took 1.68 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:28:15 PM on My Cluster8

```

1 # Left anti join
2 spark.sql("select * from employee3 left anti join department1 on employee3.emp_id=department1.emp_id").show()

▶ (3) Spark Jobs
+---+---+---+---+---+
|emp_id|emp_name|Salary|age|city|
+---+---+---+---+---+
| 2| Sunny| 70000| 23| Pune|
| 3| Mona| 90000| 19|Hyderabad|
| 18| Dhana| 89999| 34| Kolkata|
+---+---+---+---+---+

```

Command took 1.42 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:28:19 PM on My Cluster8

b) Applying functions in pandas dataframe

1) Transform():

- Each function takes a pandas Series, and the pandas API on Spark computes the functions in a distributed manner
- It should return output as same length of input.

The screenshot shows a Databricks notebook interface. The notebook title is "Pyspark coding challenge". The code cell contains the following Python code:

```
1 # b) Applying Functions in pandas DF
2 # 1) transform()
3
4 import pandas as pd
5 psdf = pd.DataFrame({'a': [1,2,3], 'b':[4,5,6]})
6 print(psdf)
7
8 # Adding 10 to each element
9
10 def pandas_plus(count):
11     | return count + 10 # should always return the same length as input.
12
13 psdf.transform(pandas_plus)
```

The output of the code shows two DataFrames:

	a	b
0	1	4
1	2	5
2	3	6

	a	b
0	11	14
1	12	15
2	13	16

Below the code cell, a message indicates the command took 0.06 seconds. The bottom of the screen shows the Windows taskbar with various application icons.

2) apply()

- Each function takes a pandas Series, and the pandas API on Spark computes the functions in a distributed manner
- It is not mandatory to return same length as output

Databricks Notebook titled "Pyspark coding challenge" in Python. The notebook shows the following code and its output:

```

1   0  1  4
2   1  2  5
3   2  3  6

a b
0 11 14
1 12 15
2 13 16

Command took 0.06 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:34:59 PM on My Cluster8

```

```

Cmd 42:
1  # 2) apply
2
3  psdf = pd.DataFrame({'a': [1,2,3], 'b':[5,6,7]})
4  def pandas_plus(x):
5  |    return x % 2 == 1 # allows an arbitrary length
6
7  psdf.apply(pandas_plus)

a b
0 1 5
2 3 7

Command took 0.13 seconds -- by vuthurshivathmika@gmail.com at 2/12/2024, 12:40:03 PM on My Cluster8

```

Example:

Databricks Notebook titled "Pyspark coding challenge" in Python. The notebook shows the following code and its output:

```

1  import pyspark.pandas as pd
2  import numpy as np
3
4  technologies = ({
5      'Fee' :[20000,25000,30000,22000,np.NaN],
6      'Discount':[1000,2500,1500,1200,3000]
7      })
8
9  psdf = pd.DataFrame(technologies)
10 print(type(psdf))
11 print(psdf)
12
13
14  def add(data):
15  |    return data[0]+data[1]
16
17  addDF = psdf.apply(add)
18  print(addDF)

(3) Spark Jobs
<class 'pyspark.pandas.frame.DataFrame'>
Fee  Discount
0  20000.0     1000
1  25000.0     2500
2  30000.0     1500
3  22000.0     1200
4    NaN        3000
Fee       45000.0
Discount  35000.0
dtype: float64

```

Vuthur Sriganga
Data Engineering Batch-1
vuthursriganga@gmail.com