www.linkedin.com/in/naresh-neelam

Databricks-PySpark DataFrame Scenarios

1. Remove All the rows that are having all columns are null values

Source: Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

Solution: empdf.na.drop(how="all").show()

```
empdf.na.drop(how="all").show()
```

(1) Spark Jobs

```
+----+-------+-------------+------+-------+
|  ID|   NAME|DATE_OF_BIRTH|SALARY|DEPT_ID|
+----+-------+-------------+------+-------+
| 101| Naresh|   11-11-1986| 50000|    D01|
| 102| Prasad|   12-11-1986| 40000|    D02|
| 103| Suresh|   13-11-1986| 20000|    D03|
| 104|  Mohan|   14-11-1986| 10000|    D01|
| 105|  Sadai|   15-11-1986| 70000|    D02|
| 106|  Surya|   16-11-1986| 25000|    D03|
| 107|  Nanda|   17-11-1986| 41000|    D01|
| 108| Govind|   18-11-1986| 25600|    D02|
| 109|Prakash|   19-11-1986| 60000|    D03|
| 110|   Madu|   20-11-1986| 25000|    D01|
|null|  Kumar|         null|  1000|   null|
|null|Surekha|         null|  null|   null|
|null|  Suraj|   20-11-1986| 25000|   null|
|null| Komali|         null|  null|   null|
+----+-------+-------------+------+-------+
```

Command took 1.43 seconds -- by naresh_neelam@outlook.com at 10/15/2021, 9:04:20 PM on NN

2. Remove All the rows that are having any column having null value

Source: Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01

Solution: empdf.na.drop(how="any").show()

```
empdf.na.drop(how="any").show()
```

▸ (1) Spark Jobs

```
+---+-------+-------------+------+-------+
| ID|   NAME|DATE_OF_BIRTH|SALARY|DEPT_ID|
+---+-------+-------------+------+-------+
|101| Naresh|   11-11-1986| 50000|    D01|
|102| Prasad|   12-11-1986| 40000|    D02|
|103| Suresh|   13-11-1986| 20000|    D03|
|104|  Mohan|   14-11-1986| 10000|    D01|
|105|  Sadai|   15-11-1986| 70000|    D02|
|106|  Surya|   16-11-1986| 25000|    D03|
|107|  Nanda|   17-11-1986| 41000|    D01|
|108| Govind|   18-11-1986| 25600|    D02|
|109|Prakash|   19-11-1986| 60000|    D03|
|110|   Madu|   20-11-1986| 25000|    D01|
+---+-------+-------------+------+-------+
```

Command took 1.45 seconds -- by naresh_neelam@outlook.com at 10/15/2021, 9:14:04 PM on NN

3. Identify rows at least two columns having valid value

Source: Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Suraj	20-11-1986	25000	

Solution: empdf.na.drop(how="any",thresh=2).show()

```
empdf.na.drop(how="any",thresh=2).show()
```

(1) Spark Jobs

```
+----+-------+-------------+------+-------+
|  ID|   NAME|DATE_OF_BIRTH|SALARY|DEPT_ID|
+----+-------+-------------+------+-------+
| 101| Naresh|   11-11-1986| 50000|    D01|
| 102| Prasad|   12-11-1986| 40000|    D02|
| 103| Suresh|   13-11-1986| 20000|    D03|
| 104|  Mohan|   14-11-1986| 10000|    D01|
| 105|  Sadai|   15-11-1986| 70000|    D02|
| 106|  Surya|   16-11-1986| 25000|    D03|
| 107|  Nanda|   17-11-1986| 41000|    D01|
| 108| Govind|   18-11-1986| 25600|    D02|
| 109|Prakash|   19-11-1986| 60000|    D03|
| 110|   Madu|   20-11-1986| 25000|    D01|
|null|  Kumar|         null|  1000|   null|
|null|  Suraj|   20-11-1986| 25000|   null|
+----+-------+-------------+------+-------+
```

Command took 1.54 seconds -- by naresh_neelam@outlook.com at 10/15/2021, 9:20:59 PM on NN

4. Identify rows either name or date of birth having valid values

Source: Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Suraj	20-11-1986	25000	

Solution: empdf.na.drop(how="any",subset=['NAME','DATE_OF_BIRTH']).show()

```
1  empdf.na.drop(how="any",subset=['NAME','DATE_OF_BIRTH']).show()
```

▸ (1) Spark Jobs

```
+----+-------+-------------+------+-------+
|  ID|   NAME|DATE_OF_BIRTH|SALARY|DEPT_ID|
+----+-------+-------------+------+-------+
| 101| Naresh|   11-11-1986| 50000|    D01|
| 102| Prasad|   12-11-1986| 40000|    D02|
| 103| Suresh|   13-11-1986| 20000|    D03|
| 104|  Mohan|   14-11-1986| 10000|    D01|
| 105|  Sadai|   15-11-1986| 70000|    D02|
| 106|  Surya|   16-11-1986| 25000|    D03|
| 107|  Nanda|   17-11-1986| 41000|    D01|
| 108| Govind|   18-11-1986| 25600|    D02|
| 109|Prakash|   19-11-1986| 60000|    D03|
| 110|   Madu|   20-11-1986| 25000|    D01|
|null|  Suraj|   20-11-1986| 25000|   null|
+----+-------+-------------+------+-------+
```

Command took 1.23 seconds -- by naresh_neelam@outlook.com at 10/15/2021, 9:28:53 PM on NN

5. Fill the missing values of DEPT_ID as DEPT-ERR

Source: Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	DEPT-ERR
	Surekha			DEPT-ERR
	Suraj	20-11-1986	25000	DEPT-ERR
	Komali			DEPT-ERR
				DEPT-ERR
				DEPT-ERR

Solution: empdf.na.fill("DEPT-ERR",["DEPT_ID"]).show()

```
empdf.na.fill("DEPT-ERR",["DEPT_ID"]).show()
```

(1) Spark Jobs

```
+----+-------+-------------+------+--------+
|  ID|   NAME|DATE_OF_BIRTH|SALARY| DEPT_ID|
+----+-------+-------------+------+--------+
| 101| Naresh|   11-11-1986| 50000|     D01|
| 102| Prasad|   12-11-1986| 40000|     D02|
| 103| Suresh|   13-11-1986| 20000|     D03|
| 104|  Mohan|   14-11-1986| 10000|     D01|
| 105|  Sadai|   15-11-1986| 70000|     D02|
| 106|  Surya|   16-11-1986| 25000|     D03|
| 107|  Nanda|   17-11-1986| 41000|     D01|
| 108| Govind|   18-11-1986| 25600|     D02|
| 109|Prakash|   19-11-1986| 60000|     D03|
| 110|   Madu|   20-11-1986| 25000|     D01|
|null|  Kumar|         null|  1000|DEPT-ERR|
|null|Surekha|         null|  null|DEPT-ERR|
|null|  Suraj|   20-11-1986| 25000|DEPT-ERR|
|null| Komali|         null|  null|DEPT-ERR|
|null|   null|         null|  null|DEPT-ERR|
|null|   null|         null|  null|DEPT-ERR|
+----+-------+-------------+------+--------+
```

Command took 1.55 seconds -- by naresh_neelam@outlook.com at 10/15/2021, 9:35:58 PM on NN

7. Find the average salary for each dept_id and add that as a another column, display with order by dept id asc, salary asc and nulls should be last

Source: Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID	AVG_SAL_BY_DEPT
101	Naresh	11-11-1986	50000	D01	31500
104	Mohan	14-11-1986	10000	D01	31500
107	Nanda	17-11-1986	41000	D01	31500
110	Madu	20-11-1986	25000	D01	31500
102	Prasad	12-11-1986	40000	D02	45200
105	Sadai	15-11-1986	70000	D02	45200
108	Govind	18-11-1986	25600	D02	45200
103	Suresh	13-11-1986	20000	D03	35000
106	Surya	16-11-1986	25000	D03	35000
109	Prakash	19-11-1986	60000	D03	35000
	Kumar		1000		13000
	Surekha				13000
	Suraj	20-11-1986	25000		13000
	Komali				13000
					13000
					13000

Solution:

```
from pyspark.sql import Window
from pyspark.sql.functions import avg

win=Window.partitionBy("DEPT_ID")
empdf.withColumn("AVG_SAL_BY_DEPT",avg(empdf.SALARY).over(win)).orderBy((empdf.DEPT_ID).asc_nulls_last(),(empdf.ID).asc_nulls_last()).show()
```

```
from pyspark.sql import Window
from pyspark.sql.functions import avg

win=Window.partitionBy("DEPT_ID")
empdf.withColumn("AVG_SAL_BY_DEPT",avg(empdf.SALARY).over(win)).orderBy((empdf.DEPT_ID).asc_nulls_last(),(empdf.ID).asc_nulls_last()).show()
```

(2) Spark Jobs

```
+----+-------+-------------+------+-------+---------------+
|  ID|   NAME|DATE_OF_BIRTH|SALARY|DEPT_ID|AVG_SAL_BY_DEPT|
+----+-------+-------------+------+-------+---------------+
| 101| Naresh|   11-11-1986| 50000|    D01|        31500.0|
| 104|  Mohan|   14-11-1986| 10000|    D01|        31500.0|
| 107|  Nanda|   17-11-1986| 41000|    D01|        31500.0|
| 110|   Madu|   20-11-1986| 25000|    D01|        31500.0|
| 102| Prasad|   12-11-1986| 40000|    D02|        45200.0|
| 105|  Sadai|   15-11-1986| 70000|    D02|        45200.0|
| 108| Govind|   18-11-1986| 25600|    D02|        45200.0|
| 103| Suresh|   13-11-1986| 20000|    D03|        35000.0|
| 106|  Surya|   16-11-1986| 25000|    D03|        35000.0|
| 109|Prakash|   19-11-1986| 60000|    D03|        35000.0|
|null|  Kumar|         null|  1000|   null|        13000.0|
|null|   null|         null|  null|   null|        13000.0|
|null|Surekha|         null|  null|   null|        13000.0|
|null|  Suraj|   20-11-1986| 25000|   null|        13000.0|
|null| Komali|         null|  null|   null|        13000.0|
|null|   null|         null|  null|   null|        13000.0|
+----+-------+-------------+------+-------+---------------+
```

Command took 1.43 seconds -- by naresh_neelam@outlook.com at 10/15/2021, 9:53:18 PM on NN

9. Find the minimum, maximum, average, sum of SALARY column with proper alias names

Source: Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

min_of_salary	max_of_salary	avg_of_salary	sum_of_salary
1000	70000	32716.66667	392600

Solution:

```
from pyspark.sql import functions as F

empdf.agg( \
        F.min(empdf.SALARY).alias("min_of_salary"), \
        F.max(empdf.SALARY).alias("max_of_salary"), \
        F.avg(empdf.SALARY).alias("avg_of_salary"), \
        F.sum(empdf.SALARY).alias("sum_of_salary")).show()
```

```
1  from pyspark.sql import functions as F
2  empdf.agg( \
3          F.min(empdf.SALARY).alias("min_of_salary"), \
4          F.max(empdf.SALARY).alias("max_of_salary"), \
5          F.avg(empdf.SALARY).alias("avg_of_salary"), \
6          F.sum(empdf.SALARY).alias("sum_of_salary")).show()
```

▸ (2) Spark Jobs

```
+-------------+-------------+------------------+-------------+
|min_of_salary|max_of_salary|     avg_of_salary|sum_of_salary|
+-------------+-------------+------------------+-------------+
|         1000|        70000|32716.666666666668|       392600|
+-------------+-------------+------------------+-------------+
```

Command took 1.46 seconds -- by naresh_neelam@outlook.com at 10/15/2021, 10:12:24 PM on NN

10. Find the minimum, maximum, average, sum of SALARY column by dept id by ascending with proper alias names

Source: Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

DEPT_ID	min_of_salary	max_of_salary	avg_of_salary	sum_of_salary
D01	10000	50000	31500	126000
D02	25600	70000	45200	135600
D03	20000	60000	35000	105000
	1000	25000	13000	26000

Solution:

```
from pyspark.sql import functions as F

empdf.groupBy(empdf.DEPT_ID)\
    .agg( \
      F.min(empdf.SALARY).alias("min_of_salary"), \
      F.max(empdf.SALARY).alias("max_of_salary"), \
      F.avg(empdf.SALARY).alias("avg_of_salary"), \
      F.sum(empdf.SALARY).alias("sum_of_salary"))\
    .orderBy(empdf.DEPT_ID.asc_nulls_last()).show()
```

```
from pyspark.sql import functions as F
empdf.groupBy(empdf.DEPT_ID)\
      .agg( \
         F.min(empdf.SALARY).alias("min_of_salary"), \
         F.max(empdf.SALARY).alias("max_of_salary"), \
         F.avg(empdf.SALARY).alias("avg_of_salary"), \
         F.sum(empdf.SALARY).alias("sum_of_salary"))\
      .orderBy(empdf.DEPT_ID.asc_nulls_last()).show()
```

▸ (2) Spark Jobs

```
+-------+-------------+-------------+-------------+-------------+
|DEPT_ID|min_of_salary|max_of_salary|avg_of_salary|sum_of_salary|
+-------+-------------+-------------+-------------+-------------+
|    D01|        10000|        50000|      31500.0|       126000|
|    D02|        25600|        70000|      45200.0|       135600|
|    D03|        20000|        60000|      35000.0|       105000|
|   null|         1000|        25000|      13000.0|        26000|
+-------+-------------+-------------+-------------+-------------+
```

Command took 1.53 seconds -- by naresh_neelam@outlook.com at 10/15/2021, 10:19:23 PM on NN

11. Select distinct *, select distinct cols, count(*) group by

empdf.groupBy(empdf.DEPT_ID,empdf.SALARY).count().show()
== SELECT DEPT_ID,SALARY,COUNT(*) FROM EMPLOYESS GROUP BY DEPT_ID,SALARY

empdf.distinct().show() == SELECT DISTINCT * FROM EMPLOYESS

empdf.select(empdf.DEPT_ID).distinct().orderBy((empdf.DEPT_ID).asc_nulls_last()).show()
== SELECT DISTINCT DEPT_ID FROM EMPLOYESS ORDER BY DEPT_ID ASC, NULLS LAST

```
1  empdf.dropDuplicates(['DEPT_ID']).show()
```

(2) Spark Jobs

```
+----+------+-------------+------+-------+
|  ID|  NAME|DATE_OF_BIRTH|SALARY|DEPT_ID|
+----+------+-------------+------+-------+
|null| Kumar|         null|  1000|   null|
| 101|Naresh|   11-11-1986| 50000|    D01|
| 102|Prasad|   12-11-1986| 40000|    D02|
| 103|Suresh|   13-11-1986| 20000|    D03|
+----+------+-------------+------+-------+
```

Command took 1.43 seconds -- by naresh_neelam@outlook.com at 10/15/2021, 10:58:48 PM on NN

12. Having two tables employees, department and join by dept id and get the sum of salary by Department Name

Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

Departmens.csv

DEPT_ID	DEPT_NAME
D01	Administrator
D02	Manager
D03	Developer

DEPT_ID	DEPT_NAME	SUM_OF_SALARY
D01	Administrator	126000
D02	Manager	135600
D03	Developer	105000

Solution:

```
from pyspark.sql import functions as F

empdf.join(deptdf, \
                    empdf.DEPT_ID == deptdf.DEPT_ID, 'inner' )\
        .groupBy(empdf.DEPT_ID,deptdf.DEPT_NAME) \
        .agg (\ F.sum(empdf.SALARY).alias("sum_of_salary")) \
        .orderBy(empdf.DEPT_ID.asc_nulls_last()).show()
```

```
1  from pyspark.sql import functions as F
2
3  empdf.join(deptdf, \
4                     empdf.DEPT_ID == deptdf.DEPT_ID, 'inner'  )\
5       .groupBy(empdf.DEPT_ID,deptdf.DEPT_NAME) \
6       .agg (\
7            F.sum(empdf.SALARY).alias("sum_of_salary")) \
8       .orderBy(empdf.DEPT_ID.asc_nulls_last()).show()
```

▸ (2) Spark Jobs

```
+-------+-------------+-------------+
|DEPT_ID|    DEPT_NAME|sum_of_salary|
+-------+-------------+-------------+
|    D01|Administrator|       126000|
|    D02|      Manager|       135600|
|    D03|    Developer|       105000|
+-------+-------------+-------------+
```

Command took 1.83 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 8:16:45 AM on NN

13. Having two tables employees, department and join by dept id order by dept_id, id and get last row of each department

Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

Departmens.csv

DEPT_ID	DEPT_NAME
D01	Administrator
D02	Manager
D03	Developer

Highlighted Green records needs to be displayed

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
104	Mohan	14-11-1986	10000	D01
107	Nanda	17-11-1986	41000	D01
110	Madu	20-11-1986	25000	D01
102	Prasad	12-11-1986	40000	D02
105	Sadai	15-11-1986	70000	D02
108	Govind	18-11-1986	25600	D02
103	Suresh	13-11-1986	20000	D03
106	Surya	16-11-1986	25000	D03
109	Prakash	19-11-1986	60000	D03

Solution:

```
from pyspark.sql.window import Window
from pyspark.sql.functions import count

empdeptrn=empdf.join(deptdf, \
                     empdf.DEPT_ID == deptdf.DEPT_ID, 'inner' )\
            .withColumn("rn",row_number().over(Window.partitionBy(empdf.DEPT_ID).orderBy(empdf.ID.asc_nulls_last())))\
            .withColumn("cnt",count(empdf.ID).over(Window.partitionBy(empdf.DEPT_ID)))

empdeptrn.filter(empdeptrn.rn-empdeptrn.cnt == 0).show()
```

```
from pyspark.sql.window import Window
from pyspark.sql.functions import count


empdeptrn=empdf.join(deptdf, \
                     empdf.DEPT_ID == deptdf.DEPT_ID, 'inner'  )\
      .withColumn("rn",row_number().over(Window.partitionBy(empdf.DEPT_ID).orderBy(empdf.ID.asc_nulls_last())))\
      .withColumn("cnt",count(empdf.ID).over(Window.partitionBy(empdf.DEPT_ID)))

empdeptrn.filter(empdeptrn.rn-empdeptrn.cnt == 0).show()
```

▸ (2) Spark Jobs

▸ empdeptrn: pyspark.sql.dataframe.DataFrame = [ID: integer, NAME: string ... 7 more fields]

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID	DEPT_ID	DEPT_NAME	rn	cnt
110	Madu	20-11-1986	25000	D01	D01	Administrator	4	4
108	Govind	18-11-1986	25600	D02	D02	Manager	3	3
109	Prakash	19-11-1986	60000	D03	D03	Developer	3	3

Command took 2.14 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 8:52:18 AM on NN

14. display cumulative sum of salary by department order by id

Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

Departmens.csv

DEPT_ID	DEPT_NAME
D01	Administrator
D02	Manager
D03	Developer

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID	cum_sum_sal
101	Naresh	11-11-1986	50000	D01	50000
104	Mohan	14-11-1986	10000	D01	60000
107	Nanda	17-11-1986	41000	D01	101000
110	Madu	20-11-1986	25000	D01	126000
102	Prasad	12-11-1986	40000	D02	40000
105	Sadai	15-11-1986	70000	D02	110000
108	Govind	18-11-1986	25600	D02	135600
103	Suresh	13-11-1986	20000	D03	20000
106	Surya	16-11-1986	25000	D03	45000
109	Prakash	19-11-1986	60000	D03	105000

Solution:

```
from pyspark.sql.window import Window
from pyspark.sql.functions import sum

win=(Window.partitionBy(empdf.DEPT_ID).orderBy(empdf.ID.asc_nulls_last()) .rangeBetween(Window.unboundedPreceding,0))

empdeptrn=empdf.join(deptdf, \
                     empdf.DEPT_ID == deptdf.DEPT_ID, 'inner' )\
                .withColumn("cum_sum_sal",sum(empdf.SALARY).over(win)).show()
```

```
1  from pyspark.sql.window import Window
2  from pyspark.sql.functions import sum
3
4  win=(Window.partitionBy(empdf.DEPT_ID).orderBy(empdf.ID.asc_nulls_last())
5            .rangeBetween(Window.unboundedPreceding,0))
6
7  empdeptrn=empdf.join(deptdf, \
8                    empdf.DEPT_ID == deptdf.DEPT_ID, 'inner'  )\
9      .withColumn("cum_sum_sal",sum(empdf.SALARY).over(win)).show()
```

▸ (3) Spark Jobs

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID	DEPT_ID	DEPT_NAME	cum_sum_sal
101	Naresh	11-11-1986	50000	D01	D01	Administrator	50000
104	Mohan	14-11-1986	10000	D01	D01	Administrator	60000
107	Nanda	17-11-1986	41000	D01	D01	Administrator	101000
110	Madu	20-11-1986	25000	D01	D01	Administrator	126000
102	Prasad	12-11-1986	40000	D02	D02	Manager	40000
105	Sadai	15-11-1986	70000	D02	D02	Manager	110000
108	Govind	18-11-1986	25600	D02	D02	Manager	135600
103	Suresh	13-11-1986	20000	D03	D03	Developer	20000
106	Surya	16-11-1986	25000	D03	D03	Developer	45000
109	Prakash	19-11-1986	60000	D03	D03	Developer	105000

Command took 2.48 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 1:04:37 PM on NN

15. Concatenate the list of employees by comma(,) by dept id order by id

Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

Departmens.csv

DEPT_ID	DEPT_NAME
D01	Administrator
D02	Manager
D03	Developer

DEPT_NAME	der_emp_list
Administrator	Naresh,Mohan,Nanda,Madu
Manager	Prasad,Sadai,Govind
Developer	Suresh,Surya,Prakash

Solution:

```
from pyspark.sql import functions as F

empdeptrn=empdf.join(deptdf, \
                    empdf.DEPT_ID == deptdf.DEPT_ID, 'inner' )\
          .orderBy(empdf.DEPT_ID.asc_nulls_last(),empdf.ID.asc_nulls_last())\
          .groupBy(empdf.DEPT_ID).agg(F.concat_ws(",",F.collect_list(empdf.NAME)).alias("dev_emp_list"))\
          .orderBy(empdf.DEPT_ID.asc_nulls_last())\
          .show(truncate=False)
```

```
from pyspark.sql import functions as F

empdeptrn=empdf.join(deptdf, \
                    empdf.DEPT_ID == deptdf.DEPT_ID, 'inner'  )\
     .orderBy(empdf.DEPT_ID.asc_nulls_last(),empdf.ID.asc_nulls_last())\
     .groupBy(empdf.DEPT_ID).agg(F.concat_ws(",",F.collect_list(empdf.NAME)).alias("dev_emp_list")) \
     .orderBy(empdf.DEPT_ID.asc_nulls_last())\
     .show(truncate=False)
```

▸ (4) Spark Jobs

```
+-------+----------------------+
|DEPT_ID|dev_emp_list          |
+-------+----------------------+
|D01    |Naresh,Mohan,Nanda,Madu|
|D02    |Prasad,Sadai,Govind   |
|D03    |Suresh,Surya,Prakash  |
+-------+----------------------+
```

Command took 2.45 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 3:03:35 PM on NN

16. Rows into columns

Products.csv

PRODUCT_ID1	PRODUCT_ID2	PRODUCT_ID3
P000001	P000002	P000003
P000004	P000005	P000006

PRODUCT_ID
P000001
P000002
P000003
P000004
P000005
P000006

Solution:

```
undf=prddf.select(prddf.PRODUCT_ID1.alias("PRODUCT_ID"))\
.union(prddf.select(prddf.PRODUCT_ID2.alias("PRODUCT_ID")))\
.union(prddf.select(prddf.PRODUCT_ID3.alias("PRODUCT_ID")))

undf.orderBy(undf.PRODUCT_ID.asc_nulls_last()).show()
```

```
1
2  undf=prddf.select(prddf.PRODUCT_ID1.alias("PRODUCT_ID"))\
3  .union(prddf.select(prddf.PRODUCT_ID2.alias("PRODUCT_ID")))\
4  .union(prddf.select(prddf.PRODUCT_ID3.alias("PRODUCT_ID")))
5
6  undf.orderBy(undf.PRODUCT_ID.asc_nulls_last()).show()
7
```

▸ (1) Spark Jobs

▸ undf: pyspark.sql.dataframe.DataFrame = [PRODUCT_ID: string]

```
+----------+
|PRODUCT_ID|
+----------+
|   P000001|
|   P000002|
|   P000003|
|   P000004|
|   P000005|
|   P000006|
+----------+
```

Command took 1.44 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 3:08:06 PM on NN

17. Columns into Rows

Products_Sku.csv

PRODUCT_ID	SKU_ID
P000001	AA
P000001	BB
P000001	CC
P000002	AA
P000002	BB
P000002	CC

PRODUCT_ID	PRIMARY_SKU_ID	SECONDARY_SKU_ID	OTHER_SKU_ID
P000001	AA	BB	CC
P000001	AA	BB	CC

Solution:

```python
from pyspark.sql.window import Window
from pyspark.sql import functions as F

psdf=prdskudf.withColumn("rn",F.row_number().over(Window.partitionBy(prdskudf.PRODUCT_ID).orderBy(prdskudf.PRODUCT_ID)))

psdfmax=psdf.select(psdf.PRODUCT_ID,F.when(psdf.rn == 1, psdf.SKU_ID).otherwise("").alias("primary_sku_id") \
                    ,F.when(psdf.rn == 2, psdf.SKU_ID).otherwise("").alias("secondary_sku_id") \
                    ,F.when(psdf.rn == 3, psdf.SKU_ID).otherwise("").alias("other_sku_id"))

psdfmax.groupBy(psdfmax.PRODUCT_ID)\
        .agg( \
              F.max(psdfmax.primary_sku_id).alias("primary_sku_id") \
             ,F.max(psdfmax.secondary_sku_id).alias("secondary_sku_id") \
             ,F.max(psdfmax.other_sku_id).alias("other_sku_id") \
            ) \
        .show()
```


```python
from pyspark.sql.window import Window
from pyspark.sql import functions as F

psdf=prdskudf.withColumn("rn",F.row_number().over(Window.partitionBy(prdskudf.PRODUCT_ID).orderBy(prdskudf.PRODUCT_ID)))

psdfmax=psdf.select(psdf.PRODUCT_ID,F.when(psdf.rn == 1, psdf.SKU_ID).otherwise("").alias("primary_sku_id") \
                    ,F.when(psdf.rn == 2, psdf.SKU_ID).otherwise("").alias("secondary_sku_id") \
                    ,F.when(psdf.rn == 3, psdf.SKU_ID).otherwise("").alias("other_sku_id"))

psdfmax.groupBy(psdfmax.PRODUCT_ID)\
        .agg( \
              F.max(psdfmax.primary_sku_id).alias("primary_sku_id") \
             ,F.max(psdfmax.secondary_sku_id).alias("secondary_sku_id") \
             ,F.max(psdfmax.other_sku_id).alias("other_sku_id") \
            ) \
        .show()
```


▸ (2) Spark Jobs

▸ psdf: pyspark.sql.dataframe.DataFrame = [PRODUCT_ID: string, SKU_ID: string ... 1 more field]

▸ psdfmax: pyspark.sql.dataframe.DataFrame = [PRODUCT_ID: string, primary_sku_id: string ... 2 more fields]

PRODUCT_ID	primary_sku_id	secondary_sku_id	other_sku_id
P000001	AA	BB	CC
P000002	AA	BB	CC

Command took 1.64 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 4:09:17 PM on NN

18. Combine all below three tables based on column names

Sequence_Poll_1.csv

SEQ1	SEQ2	SEQ3	SEQ4
1	2	3	4

Sequence_Poll_2.csv

SEQ4	SEQ3	SEQ2	SEQ1
8	7	6	5

Sequence_Poll_3.csv

SEQ1	SEQ4
9	10

SEQ1	SEQ2	SEQ3	SEQ4
1	2	3	4
5	6	7	8
9	null	null	10

Solution:

```
seq1df.unionByName(seq2df,allowMissingColumns=True)\
        .unionByName(seq3df,allowMissingColumns=True)\
.show()
```

```
seq1df.unionByName(seq2df,allowMissingColumns=True)\
        .unionByName(seq3df,allowMissingColumns=True)\
.show()
```

(2) Spark Jobs

```
+----+----+----+----+
|SEQ1|SEQ2|SEQ3|SEQ4|
+----+----+----+----+
|   1|   2|   3|   4|
|   5|   6|   7|   8|
|   9|null|null|  10|
+----+----+----+----+
```

Command took 2.54 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 4:35:59 PM on NN

19. Find the age of the employees

Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

Departmens.csv

DEPT_ID	DEPT_NAME
D01	Administrator
D02	Manager
D03	Developer

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID	AGE
101	Naresh	11-11-1986	50000	D01	XX.XX
102	Prasad	12-11-1986	40000	D02	XX.XX
103	Suresh	13-11-1986	20000	D03	XX.XX
104	Mohan	14-11-1986	10000	D01	XX.XX
105	Sadai	15-11-1986	70000	D02	XX.XX
106	Surya	16-11-1986	25000	D03	XX.XX
107	Nanda	17-11-1986	41000	D01	XX.XX
108	Govind	18-11-1986	25600	D02	XX.XX
109	Prakash	19-11-1986	60000	D03	XX.XX
110	Madu	20-11-1986	25000	D01	XX.XX

Solution: from pyspark.sql.functions import *

```
empdf.join(deptdf,\
                    empdf.DEPT_ID == deptdf.DEPT_ID,'inner')\
        .withColumn("AGE",round(months_between(current_date(),to_date(empdf.DATE_OF_BIRTH,'dd-MM-yyyy'))/12,2)).show()
```

```
from pyspark.sql.functions import *

empdf.join(deptdf,\
                    empdf.DEPT_ID == deptdf.DEPT_ID,'inner')\
        .withColumn("AGE",round(months_between(current_date(),to_date(empdf.DATE_OF_BIRTH,'dd-MM-yyyy'))/12,2)).show()
```

(1) Spark Jobs

```
+---+-------+-------------+------+-------+-------+-------------+-----+
| ID|   NAME|DATE_OF_BIRTH|SALARY|DEPT_ID|DEPT_ID|    DEPT_NAME|  AGE|
+---+-------+-------------+------+-------+-------+-------------+-----+
|101| Naresh|   11-11-1986| 50000|    D01|    D01|Administrator|34.93|
|102| Prasad|   12-11-1986| 40000|    D02|    D02|      Manager|34.93|
|103| Suresh|   13-11-1986| 20000|    D03|    D03|    Developer|34.92|
|104|  Mohan|   14-11-1986| 10000|    D01|    D01|Administrator|34.92|
|105|  Sadai|   15-11-1986| 70000|    D02|    D02|      Manager|34.92|
|106|  Surya|   16-11-1986| 25000|    D03|    D03|    Developer|34.92|
|107|  Nanda|   17-11-1986| 41000|    D01|    D01|Administrator|34.91|
|108| Govind|   18-11-1986| 25600|    D02|    D02|      Manager|34.91|
|109|Prakash|   19-11-1986| 60000|    D03|    D03|    Developer|34.91|
|110|   Madu|   20-11-1986| 25000|    D01|    D01|Administrator|34.91|
+---+-------+-------------+------+-------+-------+-------------+-----+
```

Command took 1.93 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 7:53:28 PM on NN

20. Data and Timestamp scenarios

Truncate –Year, Quarter, Month ,Week, Day, Hour, Minute, Second

```
from pyspark.sql.functions import *

dualdf=empdf.select(empdf.ID).filter(empdf.ID == 101)

dualdf.select(dualdf.ID, \
              current_timestamp().alias("Current_Timestamp"),\
              date_trunc("Year",current_timestamp()).alias("Start_of_Year")  , \
              date_trunc("Quarter",current_timestamp()).alias("Start_of_Quarter"),\
              date_trunc("Month",current_timestamp()).alias("Start_of_Month"),\
              date_trunc("Week",current_timestamp()).alias("Start_of_Week"),\
              date_trunc("Day",current_timestamp()).alias("Start_of_Day"), \
              date_trunc("Hour",current_timestamp()).alias("Start_of_Hour"),\
              date_trunc("Minute",current_timestamp()).alias("Start_of_Minute"),\
              date_trunc("Second",current_timestamp()).alias("Start_of_Second")\
             ).show(truncate=False)
```

(1) Spark Jobs

dualdf: pyspark.sql.dataframe.DataFrame = [ID: integer]

```
+---+-----------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+
|ID |Current_Timestamp      |Start_of_Year      |Start_of_Quarter   |Start_of_Month     |Start_of_Week      |Start_of_Day       |Start_of_Hour      |Start_of_Minute    |Start_of_Second    |
+---+-----------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+
|101|2021-10-16 14:46:40.864|2021-01-01 00:00:00|2021-10-01 00:00:00|2021-10-01 00:00:00|2021-10-11 00:00:00|2021-10-16 00:00:00|2021-10-16 14:00:00|2021-10-16 14:46:00|2021-10-16 14:46:40|
+---+-----------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+
```

Command took 1.43 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 8:16:40 PM on NN

20. Data and Timestamp scenarios

Extracts –Year, Quarter, Month ,Week, Day, Hour, Minute, Second

```python
from pyspark.sql.functions import *

dualdf=empdf.select(empdf.ID).filter(empdf.ID == 101)

dualdf.select(dualdf.ID, \
              current_timestamp().alias("Current_Timestamp"),\
              year(current_timestamp()).alias("Year")  , \
              dayofyear(current_timestamp()).alias("Day_Of_Year")  , \
              quarter(current_timestamp()).alias("Quarter"),\
              month(current_timestamp()).alias("Month"),\
              weekofyear(current_timestamp()).alias("Week_Of_Year"),\
              dayofweek(current_timestamp()).alias("Day_Of_Week"),\
              dayofmonth(current_timestamp()).alias("Day"), \
              hour(current_timestamp()).alias("Hour"),\
              minute(current_timestamp()).alias("Minute"),\
              second(current_timestamp()).alias("Second"),\
              last_day(current_timestamp()).alias("Last_of_Month")\
             ).show(truncate=False)
```

(1) Spark Jobs

dualdf: pyspark.sql.dataframe.DataFrame = [ID: integer]

ID	Current_Timestamp	Year	Day_Of_Year	Quarter	Month	Week_Of_Year	Day_Of_Week	Day	Hour	Minute	Second	Last_of_Month
101	2021-10-16 14:57:43.626	2021	289	4	10	41	7	16	14	57	43	2021-10-31

Command took 1.43 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 8:27:43 PM on NN

20. Data and Timestamp scenarios

Addition –Year, Quarter, Month ,Week, Day, Hour, Minute, Second

```python
from pyspark.sql.functions import *

dualdf=empdf.select(empdf.ID).filter(empdf.ID == 101)

dualdf.select(dualdf.ID, \
              current_timestamp().alias("Current_Timestamp"),\
              date_add(current_timestamp(),2).alias("Date_After_Two_Days")  , \
              date_add(current_timestamp(),-2).alias("Date_Before_Two_Days")  , \
              add_months(current_timestamp(),2).alias("Day_After_Two_Months")  , \
              add_months(current_timestamp(),-2).alias("Day_Before_Two_Months")  , \
              add_months(current_timestamp(),12).alias("Day_After_One_Year"),\
              add_months(current_timestamp(),-12).alias("Day_Before_One_Year")
             ).show(truncate=False)
```

(1) Spark Jobs

dualdf: pyspark.sql.dataframe.DataFrame = [ID: integer]

```
+---+-----------------------+-------------------+--------------------+--------------------+---------------------+-------------------+--------------------+
|ID |Current_Timestamp      |Date_After_Two_Days|Date_Before_Two_Days|Day_After_Two_Months|Day_Before_Two_Months|Day_After_One_Year|Day_Before_One_Year|
+---+-----------------------+-------------------+--------------------+--------------------+---------------------+-------------------+--------------------+
|101|2021-10-16 16:17:07.624|2021-10-18         |2021-10-14          |2021-12-16          |2021-08-16           |2022-10-16         |2020-10-16          |
+---+-----------------------+-------------------+--------------------+--------------------+---------------------+-------------------+--------------------+
```

Command took 1.54 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 9:47:07 PM on NN

20. Data and Timestamp scenarios

Addition –Year, Quarter, Month ,Week, Day, Hour, Minute, Second

```python
from pyspark.sql.functions import *

dualdf=empdf.select(empdf.ID).filter(empdf.ID == 101)

dualdf.select(dualdf.ID, \
                current_timestamp().alias("Current_Timestamp"),\
               (current_timestamp()+expr('INTERVAL 2 HOURS')).alias("Adding_Two_Hours"),\
               (current_timestamp()+expr('INTERVAL 1 MINUTES')).alias("Adding_One_Minute"),\
               (current_timestamp()+expr('INTERVAL 5 SECONDS')).alias("Adding_Five_Seconds"),\
               (current_timestamp()+expr('INTERVAL 2 HOURS 1 MINUTES 5 SECONDS')).alias("Adding_Time")
              ).show(truncate=False)
```

(1) Spark Jobs

dualdf: pyspark.sql.dataframe.DataFrame = [ID: integer]

```
+---+-----------------------+-----------------------+-----------------------+-----------------------+-----------------------+
|ID |Current_Timestamp      |Adding_Two_Hours       |Adding_One_Minute      |Adding_Five_Seconds    |Adding_Time            |
+---+-----------------------+-----------------------+-----------------------+-----------------------+-----------------------+
|101|2021-10-16 16:21:00.802|2021-10-16 18:21:00.802|2021-10-16 16:22:00.802|2021-10-16 16:21:05.802|2021-10-16 18:22:05.802|
+---+-----------------------+-----------------------+-----------------------+-----------------------+-----------------------+
```

Command took 1.44 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 9:51:00 PM on NN

Use (current_timestamp() – exprt → to substract

20. Data and Timestamp scenarios

Difference –date, month

```
from pyspark.sql.functions import *

dualdf=empdf.select(empdf.ID).filter(empdf.ID == 101)

dualdf.select(dualdf.ID, \
                current_date().alias("Current_Date"),\
                to_date(lit("2020-01-01"),'yyyy-MM-dd').alias("Old_Date"),\
                (datediff(current_date(),to_date(lit("2020-01-01"),'yyyy-MM-dd'))).alias("Date_Diff_In_Days"),\
                (months_between(current_date(),to_date(lit("2020-01-01"),'yyyy-MM-dd'))).alias("Date_Diff_In_Months")
              ).show(truncate=False)
```

▸ (1) Spark Jobs

▸ dualdf: pyspark.sql.dataframe.DataFrame = [ID: integer]

```
+---+------------+----------+-----------------+-------------------+
|ID |Current_Date|Old_Date  |Date_Diff_In_Days|Date_Diff_In_Months|
+---+------------+----------+-----------------+-------------------+
|101|2021-10-16  |2020-01-01|654              |21.48387097        |
+---+------------+----------+-----------------+-------------------+
```

Command took 1.54 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 10:13:52 PM on NN

21. Add processing file name to the data frame

```python
from pyspark.sql.functions import *

empdf=spark.read.csv(f"{filepath}/Employees.csv",header=True,inferSchema=True)\
        .withColumn("input_File_Name",split(input_file_name(),"/").getItem(5)).show()
```

▸ (3) Spark Jobs

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID	input_File_Name
101	Naresh	11-11-1986	50000	D01	Employees.csv
102	Prasad	12-11-1986	40000	D02	Employees.csv
103	Suresh	13-11-1986	20000	D03	Employees.csv
104	Mohan	14-11-1986	10000	D01	Employees.csv
105	Sadai	15-11-1986	70000	D02	Employees.csv
106	Surya	16-11-1986	25000	D03	Employees.csv
107	Nanda	17-11-1986	41000	D01	Employees.csv
108	Govind	18-11-1986	25600	D02	Employees.csv
109	Prakash	19-11-1986	60000	D03	Employees.csv
110	Madu	20-11-1986	25000	D01	Employees.csv
null	Kumar	null	1000	null	Employees.csv
null	Surekha	null	null	null	Employees.csv
null	Suraj	20-11-1986	25000	null	Employees.csv
null	Komali	null	null	null	Employees.csv
null	null	null	null	null	Employees.csv
null	null	null	null	null	Employees.csv

Command took 3.46 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 10:34:32 PM on NN

22. Bad records handling while reading files

PERMISSIVE – process the bad records and storing the bad records in the one of the column in schema definition by mentioned with columnNameOfCorruptRecord ="*bad_records*"

DROPMALFORMED – to ignore corrupt record

FAILFAST – throws an exception if bad records found

ID	NAME
123	Naresh
124	Venki
XKH	Mohan

```
empRawSchema=StructType([
                    StructField("ID",IntegerType(),True),
                    StructField("Name",StringType(),False),
                    StructField("bad_records",StringType(),False)
                ])
```

```
empdf=spark.read.csv(f"{filepath}/Bad_File.csv",\
     header=True,schema=empRawSchema,mode="PERMISSIVE",\
     columnNameOfCorruptRecord="bad_records").show()
```

(1) Spark Jobs

```
+----+------+-----------+
|  ID|  Name|bad_records|
+----+------+-----------+
| 123|Naresh|       null|
| 124| Venki|       null|
|null| Mohan|  XKH,Mohan|
+----+------+-----------+

Command took 3.35 seconds -- by nares
```

```
empdf=spark.read.csv(f"{filepath}/Bad_File.csv",\
     header=True,schema=empRawSchema,mode="DROPMALFORMED",\
     columnNameOfCorruptRecord="bad_records").show()
```

(1) Spark Jobs

```
+---+------+-----------+
| ID|  Name|bad_records|
+---+------+-----------+
|123|Naresh|       null|
|124| Venki|       null|
+---+------+-----------+

Command took 3.56 seconds -- by naresh_n
```

```
empdf=spark.read.csv(f"{filepath}/Bad_File.csv",\
     header=True,schema=empRawSchema,mode="FAILFAST",\
     columnNameOfCorruptRecord="bad_records").show()
```

(1) Spark Jobs

```
org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stag
on: Error while reading file dbfs:/mnt/azblob/rawsrc/customer_data/Bad_File.csv.
Command took 5.32 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 11:37:44 PM on NN
```

Cmd 47

22. lead and lag

```python
from pyspark.sql.functions import *
from pyspark.sql.window import Window

win=Window.partitionBy(empdf.DEPT_ID).orderBy(empdf.ID)

empdf.join(deptdf,\
                empdf.DEPT_ID == deptdf.DEPT_ID,'inner')\
     .select (empdf.ID,empdf.NAME,empdf.DEPT_ID,empdf.SALARY)\
     .withColumn("lag_1_row",lag(empdf.SALARY,1).over(win))\
     .withColumn("lag_2_row",lag(empdf.SALARY,2).over(win))\
     .withColumn("lead_1_row",lead(empdf.SALARY,1).over(win))\
     .withColumn("lead_2_row",lead(empdf.SALARY,2).over(win)).show()
```

▸ (3) Spark Jobs

ID	NAME	DEPT_ID	SALARY	lag_1_row	lag_2_row	lead_1_row	lead_2_row
101	Naresh	D01	50000	null	null	10000	41000
104	Mohan	D01	10000	50000	null	41000	25000
107	Nanda	D01	41000	10000	50000	25000	null
110	Madu	D01	25000	41000	10000	null	null
102	Prasad	D02	40000	null	null	70000	25600
105	Sadai	D02	70000	40000	null	25600	null
108	Govind	D02	25600	70000	40000	null	null
103	Suresh	D03	20000	null	null	25000	60000
106	Surya	D03	25000	20000	null	60000	null
109	Prakash	D03	60000	25000	20000	null	null

Command took 2.89 seconds -- by naresh_neelam@outlook.com at 10/16/2021, 11:18:07 PM on NN

23. Greatest and lease value of multiple columns

PROD_ID	PRICE_2017	PRICE_2018	PRICE_2019	PRICE_2020
PR0001	100	200	300	400
PR0002	50	200	300	100

PROD_ID	Max_Price	Min_Price
PR0001	400	100
PR0002	300	50

```
from pyspark.sql import functions as F

priceDf.select(priceDf.PROD_ID,
                    F.greatest(priceDf.PRICE_2017,priceDf.PRICE_2018,priceDf.PRICE_2019,priceDf.PRICE_2020).alias("Max_Price"),
                    F.least(priceDf.PRICE_2017,priceDf.PRICE_2018,priceDf.PRICE_2019,priceDf.PRICE_2020).alias("Min_Price")
               ).show()
```

```
from pyspark.sql import functions as F

priceDf.select(priceDf.PROD_ID,
                              F.greatest(priceDf.PRICE_2017,priceDf.PRICE_2018,priceDf.PRICE_2019,priceDf.PRICE_2020).alias("Max_Price"),
                              F.least(priceDf.PRICE_2017,priceDf.PRICE_2018,priceDf.PRICE_2019,priceDf.PRICE_2020).alias("Min_Price"),
                 ).show()
```

(1) Spark Jobs

```
+-------+---------+---------+
|PROD_ID|Max_Price|Min_Price|
+-------+---------+---------+
| PR0001|      400|      100|
| PR0002|      300|       50|
+-------+---------+---------+
```

Command took 1.43 seconds -- by naresh_neelam@outlook.com at 10/17/2021, 12:49:55 PM on NN

24. Horizontal Addition of values with 3 (n) times

PROD_ID	PRICE_2017
PR0001	100
PR0002	50

PROD_ID	Horz_Price
PR0001	100100100
PR0002	505050

```
from pyspark.sql import functions as F

priceDf.select(priceDf.PROD_ID, F.repeat(priceDf.PRICE_2017,3).alias("Horz_Price") ).show()
```

```
from pyspark.sql import functions as F

priceDf.select(priceDf.PROD_ID,
                                F.repeat(priceDf.PRICE_2017,3).alias("Horz_Price")
                ).show()
```

(1) Spark Jobs

```
+-------+----------+
|PROD_ID|Horz_Price|
+-------+----------+
| PR0001| 100100100|
| PR0002|    505050|
+-------+----------+
```

Command took 1.23 seconds -- by naresh_neelam@outlook.com at 10/17/2021, 1:09:24 PM on NN

25. Compare 2020 purchased data and 2019 purchased data. If product id and price is the same then ignore else display it.

Purchase_2020.csv

PRODUCT_ID	PURCHASE_PRICE
PRD0001	100
PRD0002	200
PRD0003	300
PRD0004	350
PRD0005	500

Purchase_2019.csv

PRODUCT_ID	PURCHASE_PRICE
PRD0001	100
PRD0002	200
PRD0003	300
PRD0004	400

PRODUCT_ID	PURCHASE_PRICE
PRD0004	350
PRD0004	400
PRD0005	500

Solution : its basically A – B UNION B - A

```
(pur2020_df.subtract(pur2019_df) \
.union( \
pur2019_df.subtract(pur2020_df) \
)).show()
```

```
(pur2020_df.subtract(pur2019_df) \
.union(    \
pur2019_df.subtract(pur2020_df) \
)).show()
```

▸ (4) Spark Jobs

```
+----------+--------------+
|PRODUCT_ID|PURCHASE_PRICE|
+----------+--------------+
|   PRD0005|           500|
|   PRD0004|           350|
|   PRD0004|           400|
+----------+--------------+
```

26. Extract the array of elements into multiple rows

Products_Arrary.json

Product_Id	Product_Details	Product_Properties
PRD0001	["Mobile","Electronics"]	["Black","101 grams","5G"]
PRD0002	["Mobile","Electronics"]	["Blue","100 grams","4G"]
PRD0003	["Mobile",""]	["Black","101 grams","5G"]
PRD0004	["Mobile","Electronics"]	["Blue","100 grams",""]

Product_Id	Product_Properties
PRD0001	Black
PRD0001	101 grams
PRD0001	5G
PRD0002	Blue
PRD0002	100 grams
PRD0002	4G
PRD0003	Black
PRD0003	101 grams
PRD0003	5G
PRD0004	Blue
PRD0004	100 grams
PRD0004	

Solution

```
from pyspark.sql.functions import explode

preArray_Df.select(preArray_Df.Product_id,explode(preArray_Df.Product_Properties).alias("Product_Properties")).show()
```

```
from pyspark.sql.functions import explode

preArray_Df.select(preArray_Df.Product_id,explode(preArray_Df.Product_Properties).alias("Product_Properties")).show()
```

(1) Spark Jobs

```
+----------+------------------+
|Product_id|Product_Properties|
+----------+------------------+
|   PRD0001|             Black|
|   PRD0001|         101 grams|
|   PRD0001|                5G|
|   PRD0002|              Blue|
|   PRD0002|         100 grams|
|   PRD0002|                4G|
|   PRD0003|              Blue|
|   PRD0003|         100 grams|
|   PRD0003|                4G|
|   PRD0004|              Blue|
|   PRD0004|         100 grams|
|   PRD0004|                  |
+----------+------------------+
```

Explode wont work from multiple array types.

27. Joins

PySpark Key Word	Equivalent To SQL
`inner`	INNER JOIN
`cross`	CROSS JOIN
`outer, full, fullouter, full_outer`	OUTER JOIN
`left, leftouter, left_outer`	LEFT OUTER JOIN
`right, rightouter, right_outer`	RIGHT OUTER JOIN
`semi, leftsemi, left_semi`	same as INNER JOIN , but we don't have to select left table columns manually it will give always all the matching rows from left table only.
`anti, leftanti, left_anti`	opposite of INNER JOIN , it will display non matching records from left table only.

semi, leftsemi, left_semi – all are same

Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

Departments.csv

DEPT_ID	DEPT_NAME
D01	Administrator
D02	Manager
D03	Developer

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01

semi, leftsemi, left_semi – all are same

```
empdf.join(deptdf, empdf.DEPT_ID == deptdf.DEPT_ID,'leftsemi' ).show()
```

Did not selected any columns

(1) Spark Jobs

```
+---+-------+-------------+------+-------+
| ID|   NAME|DATE_OF_BIRTH|SALARY|DEPT_ID|
+---+-------+-------------+------+-------+
|101| Naresh|   11-11-1986| 50000|    D01|
|102| Prasad|   12-11-1986| 40000|    D02|
|103| Suresh|   13-11-1986| 20000|    D03|
|104|  Mohan|   14-11-1986| 10000|    D01|
|105|  Sadai|   15-11-1986| 70000|    D02|
|106|  Surya|   16-11-1986| 25000|    D03|
|107|  Nanda|   17-11-1986| 41000|    D01|
|108| Govind|   18-11-1986| 25600|    D02|
|109|Prakash|   19-11-1986| 60000|    D03|
|110|   Madu|   20-11-1986| 25000|    D01|
+---+-------+-------------+------+-------+
```

anti, leftanti, left_anti -- all are same

Employees.csv

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
101	Naresh	11-11-1986	50000	D01
102	Prasad	12-11-1986	40000	D02
103	Suresh	13-11-1986	20000	D03
104	Mohan	14-11-1986	10000	D01
105	Sadai	15-11-1986	70000	D02
106	Surya	16-11-1986	25000	D03
107	Nanda	17-11-1986	41000	D01
108	Govind	18-11-1986	25600	D02
109	Prakash	19-11-1986	60000	D03
110	Madu	20-11-1986	25000	D01
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

Departments.csv

DEPT_ID	DEPT_NAME
D01	Administrator
D02	Manager
D03	Developer

ID	NAME	DATE_OF_BIRTH	SALARY	DEPT_ID
	Kumar		1000	
	Surekha			
	Suraj	20-11-1986	25000	
	Komali			

anti, leftanti, left_anti -- all are same

```
empdf.join(deptdf, empdf.DEPT_ID == deptdf.DEPT_ID,'leftanti' ).show()
```

▸ (1) Spark Jobs

```
+----+-------+-------------+------+-------+
|  ID|   NAME|DATE_OF_BIRTH|SALARY|DEPT_ID|
+----+-------+-------------+------+-------+
|null|  Kumar|         null|  1000|   null|
|null|Surekha|         null|  null|   null|
|null|  Suraj|   20-11-1986| 25000|   null|
|null| Komali|         null|  null|   null|
|null|   null|         null|  null|   null|
|null|   null|         null|  null|   null|
+----+-------+-------------+------+-------+
```

28. Joins