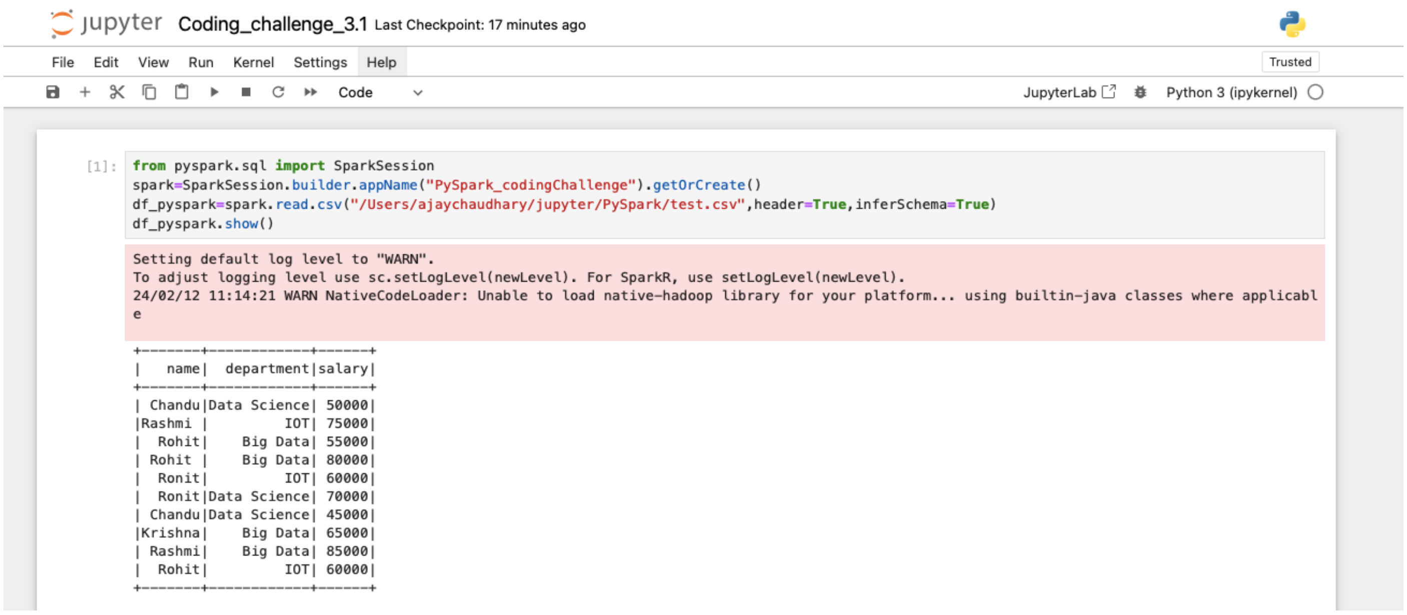
Coding Challenge-3
Pyspark & Spark

Name-Ajay Chaudhary
Batch-Data Engineering(Batch-1)

Execute Manipulating, Dropping, Sorting, Aggregations, Joining, GroupBy dataframes

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.

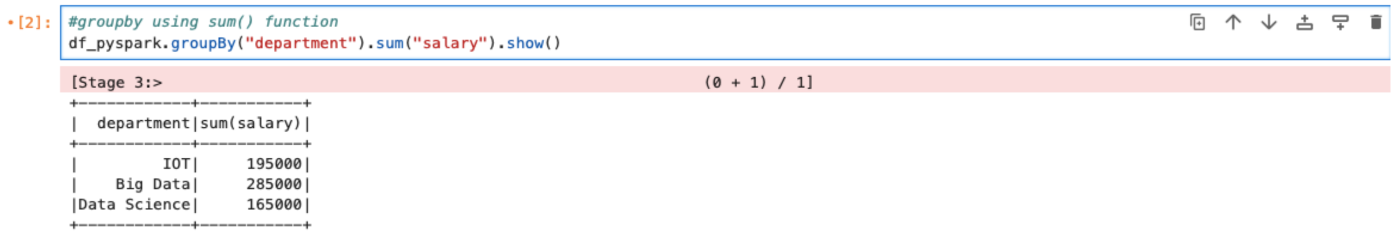

jupyter
Coding_challenge_3.1 Last Checkpoint: 17 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
[1]: from pyspark.sql import SparkSession
spark=SparkSession.builder.appName("PySpark_codingChallenge").getOrCreate()
df_pyspark=spark.read.csv("/Users/ajaychaudhary/jupyter/PySpark/test.csv",header=True,inferSchema=True)
df_pyspark.show()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/02/12 11:14:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicabl
e
```

```
+-------+------------+------+
|   name|  department|salary|
+-------+------------+------+
| Chandu|Data Science| 50000|
|Rashmi |         IOT| 75000|
|  Rohit|    Big Data| 55000|
| Rohit |    Big Data| 80000|
|  Ronit|         IOT| 60000|
|  Ronit|Data Science| 70000|
| Chandu|Data Science| 45000|
|Krishna|    Big Data| 65000|
| Rashmi|    Big Data| 85000|
|  Rohit|         IOT| 60000|
+-------+------------+------+
```

groupBy() on Departments column of DataFrame and then the sum of salary for each department using sum() function.



```
[2]: #groupby using sum() function
df_pyspark.groupBy("department").sum("salary").show()
```

```
[Stage 3:>                                                          (0 + 1) / 1]
+------------+-----------+
|  department|sum(salary)|
+------------+-----------+
|         IOT|     195000|
|    Big Data|     285000|
|Data Science|     165000|
+------------+-----------+
```

Aggregations-

Performing min, max, mean, avg, and count using the groupBy function.

```
[3]: df_pyspark.groupBy("department").min("salary").show()
```

```
+------------+-----------+
|  department|min(salary)|
+------------+-----------+
|         IOT|      60000|
|    Big Data|      55000|
|Data Science|      45000|
+------------+-----------+
```

```
[4]: df_pyspark.groupBy("department").max("salary").show()
```

```
+------------+-----------+
|  department|max(salary)|
+------------+-----------+
|         IOT|      75000|
|    Big Data|      85000|
|Data Science|      70000|
+------------+-----------+
```

```
[5]: df_pyspark.groupBy("department").avg("salary").show()
```

```
+------------+-----------+
|  department|avg(salary)|
+------------+-----------+
|         IOT|    65000.0|
|    Big Data|    71250.0|
|Data Science|    55000.0|
+------------+-----------+
```

```
[6]: df_pyspark.groupBy("department").mean("salary").show()
```

```
+------------+-----------+
|  department|avg(salary)|
+------------+-----------+
|         IOT|    65000.0|
|    Big Data|    71250.0|
|Data Science|    55000.0|
+------------+-----------+
```

```
[7]: df_pyspark.groupBy("department").count().show()
```

```
+------------+-----+
|  department|count|
+------------+-----+
|         IOT|    3|
|    Big Data|    4|
|Data Science|    3|
+------------+-----+
```

```
[8]: df_pyspark.groupBy("department").pivot("Name").sum("salary").show()
```

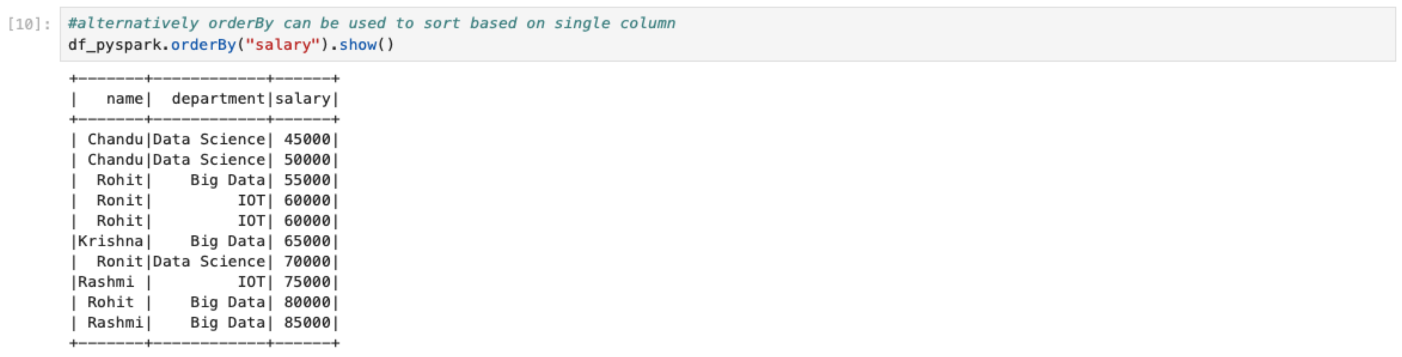
```
+------------+------+-------+------+-------+-----+------+-----+
|  department|Chandu|Krishna|Rashmi|Rashmi |Rohit|Rohit |Ronit|
+------------+------+-------+------+-------+-----+------+-----+
|         IOT|  NULL|   NULL|  NULL|  75000|60000|  NULL|60000|
|    Big Data|  NULL|  65000| 85000|   NULL|55000| 80000| NULL|
|Data Science| 95000|   NULL|  NULL|   NULL| NULL|  NULL|70000|
+------------+------+-------+------+-------+-----+------+-----+
```

Sort(): To sort a dataframe by using one or more columns, Default — ascending order

```
[9]: df_pyspark.sort("salary").show()
```

```
+-------+------------+------+
|   name|  department|salary|
+-------+------------+------+
| Chandu|Data Science| 45000|
| Chandu|Data Science| 50000|
|  Rohit|    Big Data| 55000|
|  Ronit|         IOT| 60000|
|  Rohit|         IOT| 60000|
|Krishna|    Big Data| 65000|
|  Ronit|Data Science| 70000|
|Rashmi |         IOT| 75000|
| Rohit |    Big Data| 80000|
| Rashmi|    Big Data| 85000|
+-------+------------+------+
```

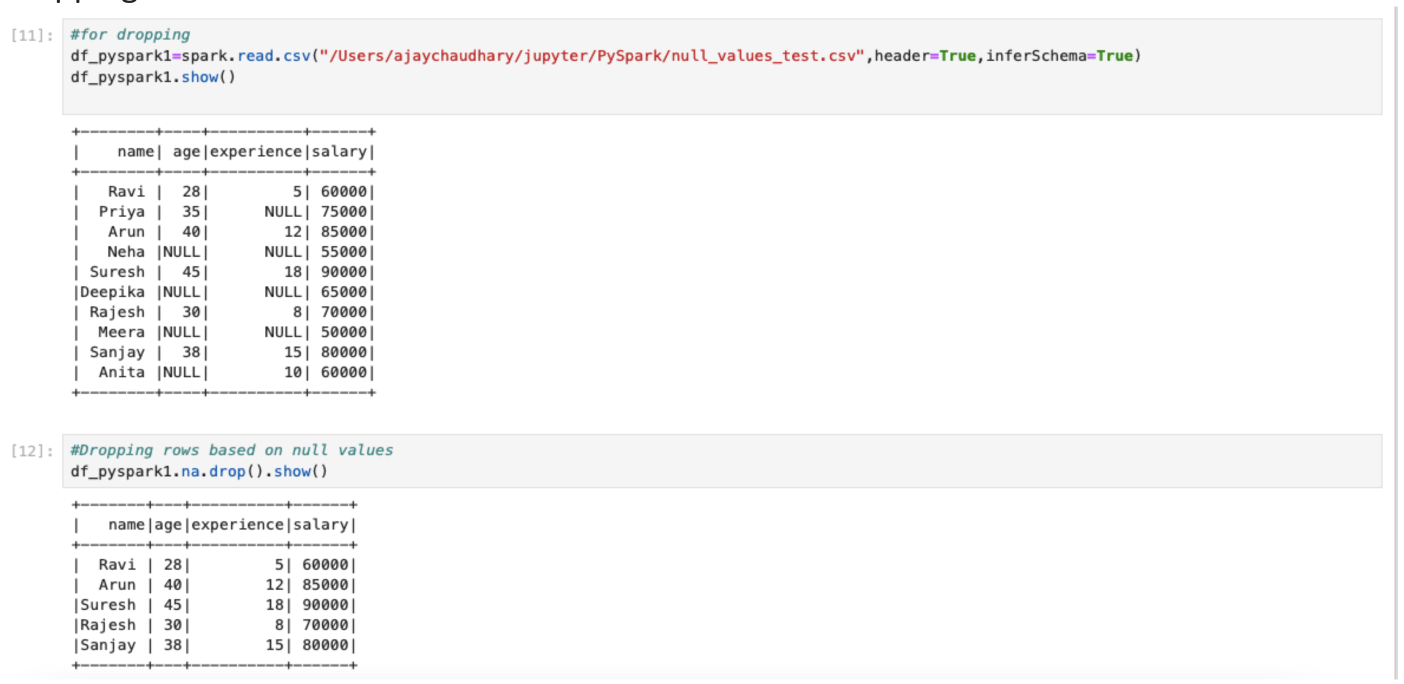
Alternatively **orderBy** can also be used to sort based on single column



```
[10]: #alternatively orderBy can be used to sort based on single column
      df_pyspark.orderBy("salary").show()
```

```
+-------+------------+------+
|   name|  department|salary|
+-------+------------+------+
| Chandu|Data Science| 45000|
| Chandu|Data Science| 50000|
|  Rohit|    Big Data| 55000|
|  Ronit|         IOT| 60000|
|  Rohit|         IOT| 60000|
|Krishna|    Big Data| 65000|
|  Ronit|Data Science| 70000|
|Rashmi |         IOT| 75000|
| Rohit |    Big Data| 80000|
| Rashmi|    Big Data| 85000|
+-------+------------+------+
```

Dropping rows based on null values



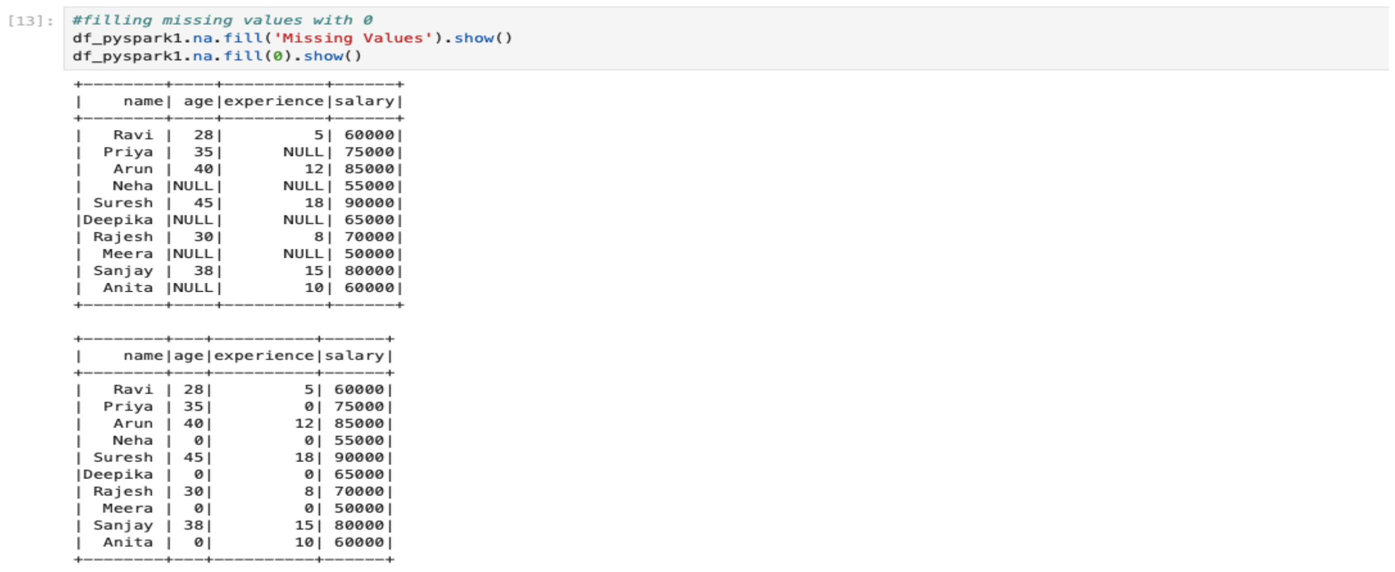
```
[11]: #for dropping
      df_pyspark1=spark.read.csv("/Users/ajaychaudhary/jupyter/PySpark/null_values_test.csv",header=True,inferSchema=True)
      df_pyspark1.show()
```

```
+--------+----+----------+------+
|    name| age|experience|salary|
+--------+----+----------+------+
|   Ravi |  28|         5| 60000|
|  Priya |  35|      NULL| 75000|
|   Arun |  40|        12| 85000|
|   Neha |NULL|      NULL| 55000|
| Suresh |  45|        18| 90000|
|Deepika |NULL|      NULL| 65000|
| Rajesh |  30|         8| 70000|
|  Meera |NULL|      NULL| 50000|
| Sanjay |  38|        15| 80000|
|  Anita |NULL|        10| 60000|
+--------+----+----------+------+
```

```
[12]: #Dropping rows based on null values
      df_pyspark1.na.drop().show()
```

```
+-------+---+----------+------+
|   name|age|experience|salary|
+-------+---+----------+------+
|  Ravi | 28|         5| 60000|
|  Arun | 40|        12| 85000|
|Suresh | 45|        18| 90000|
|Rajesh | 30|         8| 70000|
|Sanjay | 38|        15| 80000|
+-------+---+----------+------+
```

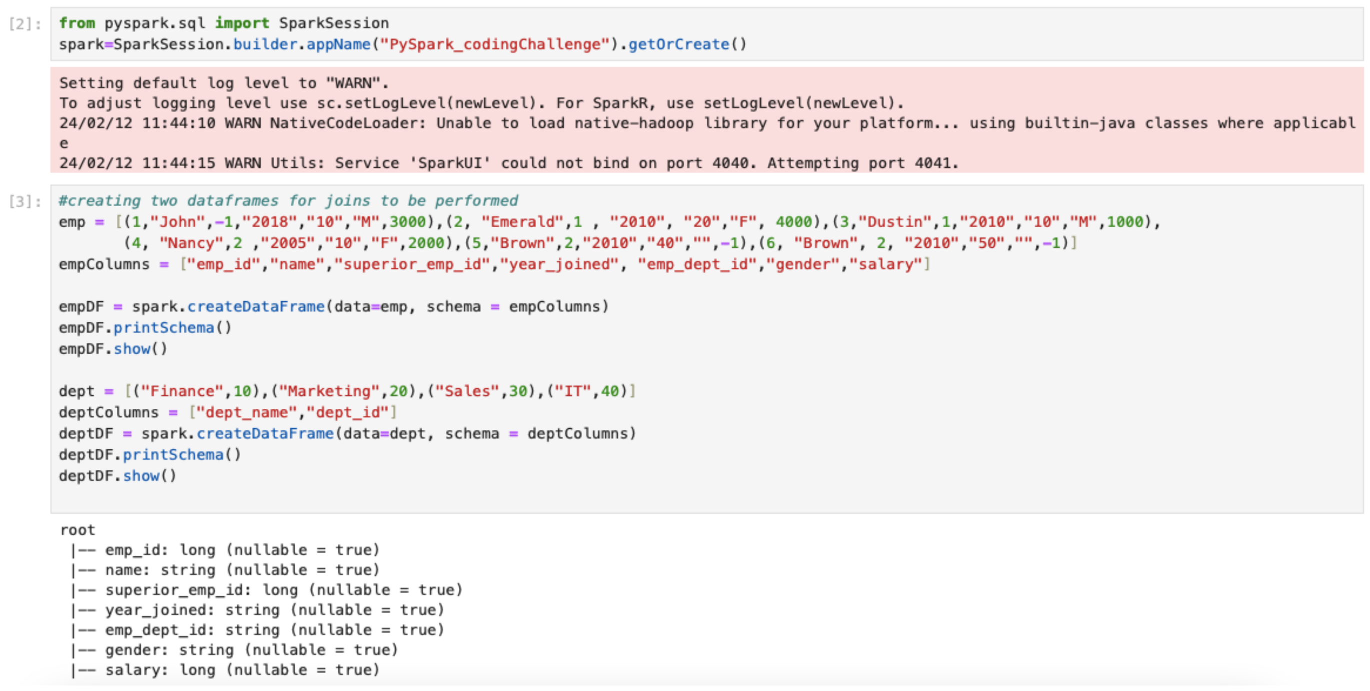
Filling missing values — Single Value



```
[13]: #filling missing values with 0
      df_pyspark1.na.fill('Missing Values').show()
      df_pyspark1.na.fill(0).show()
```

```
+--------+----+----------+------+
|    name| age|experience|salary|
+--------+----+----------+------+
|   Ravi |  28|         5| 60000|
|  Priya |  35|      NULL| 75000|
|   Arun |  40|        12| 85000|
|   Neha |NULL|      NULL| 55000|
| Suresh |  45|        18| 90000|
|Deepika |NULL|      NULL| 65000|
| Rajesh |  30|         8| 70000|
|  Meera |NULL|      NULL| 50000|
| Sanjay |  38|        15| 80000|
|  Anita |NULL|        10| 60000|
+--------+----+----------+------+

+--------+---+----------+------+
|    name|age|experience|salary|
+--------+---+----------+------+
|   Ravi | 28|         5| 60000|
|  Priya | 35|         0| 75000|
|   Arun | 40|        12| 85000|
|   Neha |  0|         0| 55000|
| Suresh | 45|        18| 90000|
|Deepika |  0|         0| 65000|
| Rajesh | 30|         8| 70000|
|  Meera |  0|         0| 50000|
| Sanjay | 38|        15| 80000|
|  Anita |  0|        10| 60000|
+--------+---+----------+------+
```

Creating dataframe for performing joins.



```
[2]: from pyspark.sql import SparkSession
     spark=SparkSession.builder.appName("PySpark_codingChallenge").getOrCreate()
```

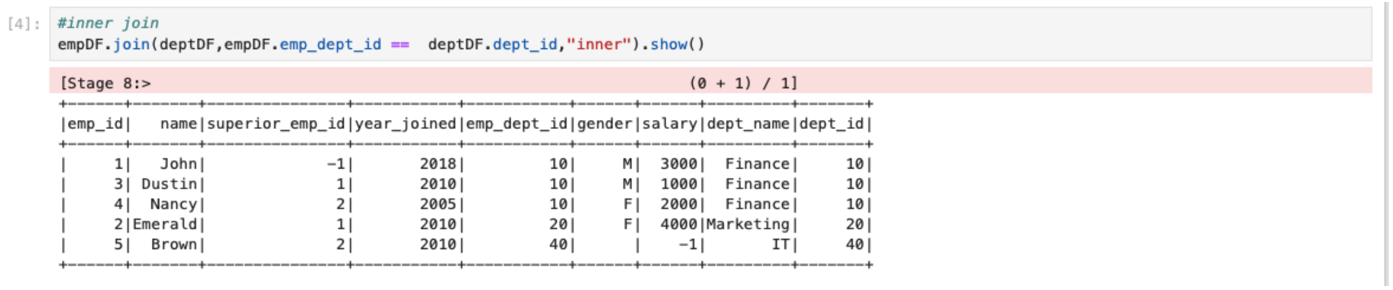
```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/02/12 11:44:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicabl
e
24/02/12 11:44:15 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
```

```
[3]: #creating two dataframes for joins to be performed
     emp = [(1,"John",-1,"2018","10","M",3000),(2, "Emerald",1 , "2010", "20","F", 4000),(3,"Dustin",1,"2010","10","M",1000),
            (4, "Nancy",2 ,"2005","10","F",2000),(5,"Brown",2,"2010","40","",-1),(6, "Brown", 2, "2010","50","",-1)]
     empColumns = ["emp_id","name","superior_emp_id","year_joined", "emp_dept_id","gender","salary"]

     empDF = spark.createDataFrame(data=emp, schema = empColumns)
     empDF.printSchema()
     empDF.show()

     dept = [("Finance",10),("Marketing",20),("Sales",30),("IT",40)]
     deptColumns = ["dept_name","dept_id"]
     deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
     deptDF.printSchema()
     deptDF.show()
```

```
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)
```

Inner join()

Join records when key column are matched and dropped when they are not matched

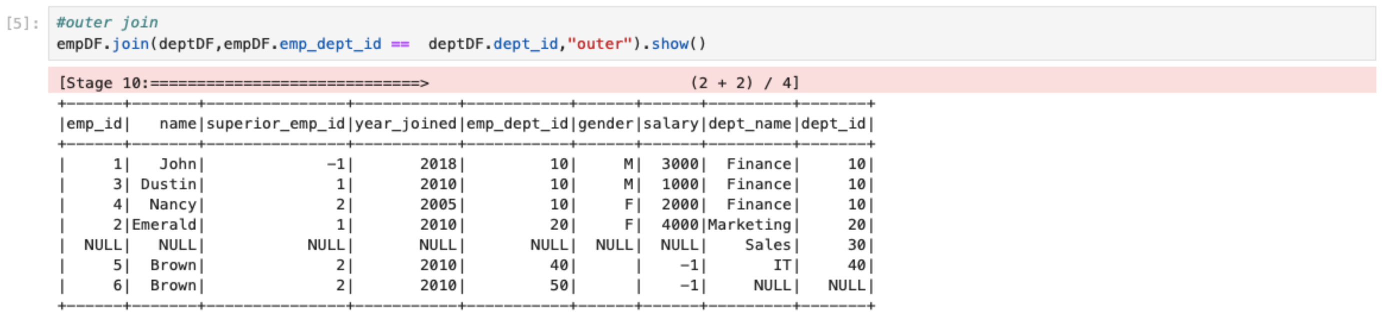


```
[4]: #inner join
     empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"inner").show()
```

```
[Stage 8:>                                                          (0 + 1) / 1]
+------+-------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|   name|superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+-------+---------------+-----------+-----------+------+------+---------+-------+
|     1|   John|             -1|       2018|         10|     M|  3000|  Finance|     10|
|     3| Dustin|              1|       2010|         10|     M|  1000|  Finance|     10|
|     4|  Nancy|              2|       2005|         10|     F|  2000|  Finance|     10|
|     2|Emerald|              1|       2010|         20|     F|  4000|Marketing|     20|
|     5|  Brown|              2|       2010|         40|      |    -1|       IT|     40|
+------+-------+---------------+-----------+-----------+------+------+---------+-------+
```

Outer join()

It returns all rows from both datasets, where join expression doesn’t match it returns null or respective columns.

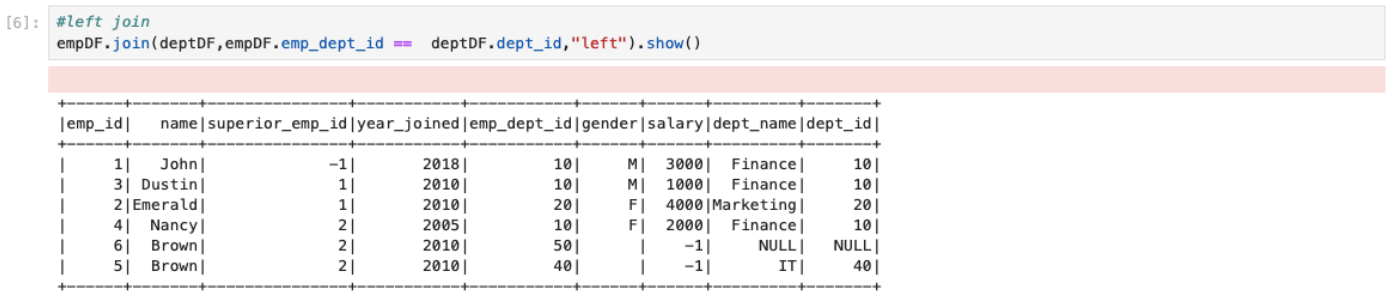


```
[5]: #outer join
     empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"outer").show()
```

```
[Stage 10:============================>                              (2 + 2) / 4]
+------+-------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|   name|superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+-------+---------------+-----------+-----------+------+------+---------+-------+
|     1|   John|             -1|       2018|         10|     M|  3000|  Finance|     10|
|     3| Dustin|              1|       2010|         10|     M|  1000|  Finance|     10|
|     4|  Nancy|              2|       2005|         10|     F|  2000|  Finance|     10|
|     2|Emerald|              1|       2010|         20|     F|  4000|Marketing|     20|
|  NULL|   NULL|           NULL|       NULL|       NULL|  NULL|  NULL|    Sales|     30|
|     5|  Brown|              2|       2010|         40|      |    -1|       IT|     40|
|     6|  Brown|              2|       2010|         50|      |    -1|     NULL|   NULL|
+------+-------+---------------+-----------+-----------+------+------+---------+-------+
```

Left join()

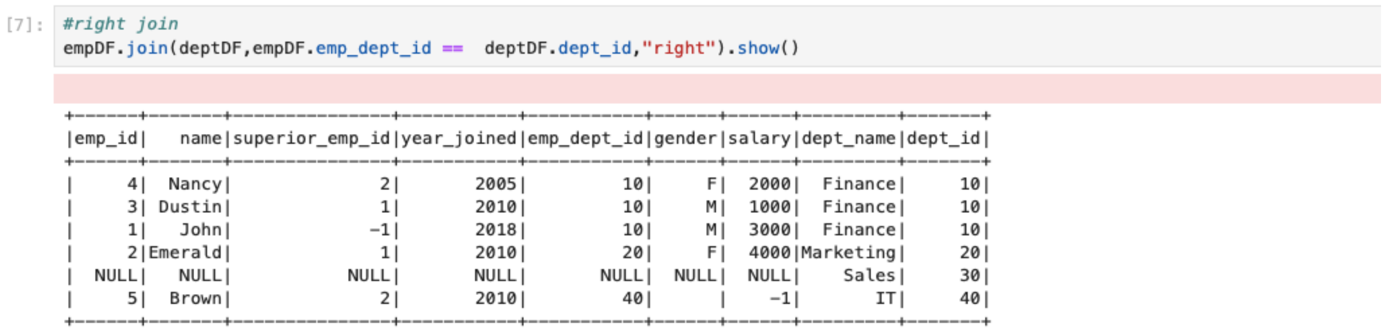
It returns all rows from left dataset regardless of match found on right dataset , when join doesn’t match it assigns null for that record.



```
[6]: #left join
     empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"left").show()

+------+-------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|   name|superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+-------+---------------+-----------+-----------+------+------+---------+-------+
|     1|   John|             -1|       2018|         10|     M|  3000|  Finance|     10|
|     3| Dustin|              1|       2010|         10|     M|  1000|  Finance|     10|
|     2|Emerald|              1|       2010|         20|     F|  4000|Marketing|     20|
|     4|  Nancy|              2|       2005|         10|     F|  2000|  Finance|     10|
|     6|  Brown|              2|       2010|         50|      |    -1|     NULL|   NULL|
|     5|  Brown|              2|       2010|         40|      |    -1|       IT|     40|
+------+-------+---------------+-----------+-----------+------+------+---------+-------+
```

Right join()

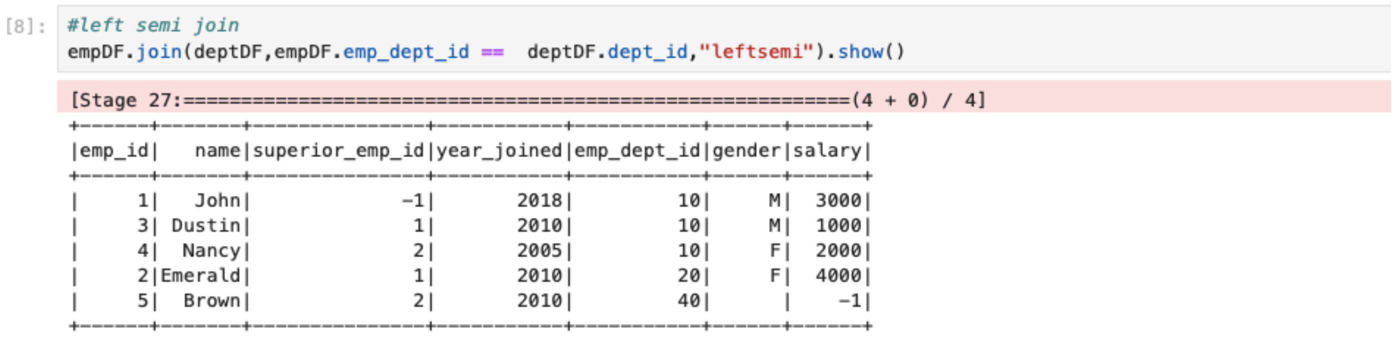
It returns all rows from right dataset regardless of match found on right dataset , when join doesn’t match it assigns null for that record.



```
[7]: #right join
     empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"right").show()

+------+-------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|   name|superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+-------+---------------+-----------+-----------+------+------+---------+-------+
|     4|  Nancy|              2|       2005|         10|     F|  2000|  Finance|     10|
|     3| Dustin|              1|       2010|         10|     M|  1000|  Finance|     10|
|     1|   John|             -1|       2018|         10|     M|  3000|  Finance|     10|
|     2|Emerald|              1|       2010|         20|     F|  4000|Marketing|     20|
|  NULL|   NULL|           NULL|       NULL|       NULL|  NULL|  NULL|    Sales|     30|
|     5|  Brown|              2|       2010|         40|      |    -1|       IT|     40|
+------+-------+---------------+-----------+-----------+------+------+---------+-------+
```

Left semi join()

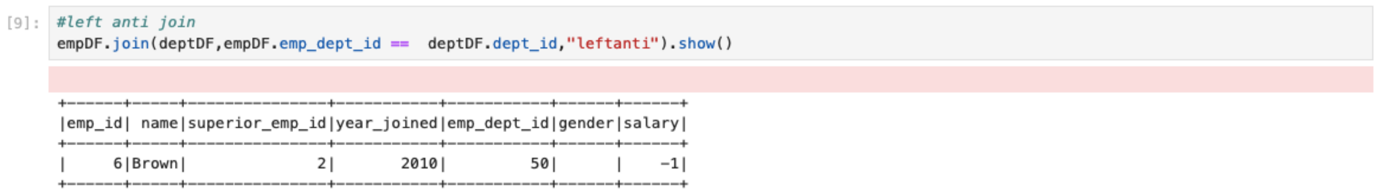
It returns columns from the only left dataset for the matched records in the right dataset on join expression.



```
[8]: #left semi join
     empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftsemi").show()

[Stage 27:==========================================================(4 + 0) / 4]
+------+-------+---------------+-----------+-----------+------+------+
|emp_id|   name|superior_emp_id|year_joined|emp_dept_id|gender|salary|
+------+-------+---------------+-----------+-----------+------+------+
|     1|   John|             -1|       2018|         10|     M|  3000|
|     3| Dustin|              1|       2010|         10|     M|  1000|
|     4|  Nancy|              2|       2005|         10|     F|  2000|
|     2|Emerald|              1|       2010|         20|     F|  4000|
|     5|  Brown|              2|       2010|         40|      |    -1|
+------+-------+---------------+-----------+-----------+------+------+
```

Left anti join()

It returns only columns from left dataset for non- matched records.



```
[9]: #left anti join
     empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftanti").show()

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