Coding Challenge-3
Pyspark & Spark

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

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

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|
+------+-----+---------------+-----------+-----------+------+------+
```

Applying Functions in a Pandas DataFrame

```
[ ]: #Applying function on pandas
```

```
[10]: import pandas as pd
data = [["John", "", "Johnson", 30, "M", 60000],
        ["Matthew", "River", "", 50, "M", 70000],
        ["Richard", "", "Walker", 42, "", 400000],
        ["Sophia", "Grace", "Smith", 38, "F", 500000],
        ["David", "Michael", "Taylor", 45, None, 0]]

columns=['First Name','Middle Name','Last Name','Age','Gender','Salary']
# Create the pandas DataFrame
pandasDF=pd.DataFrame(data=data, columns=columns)
# print dataframe.
print(pandasDF)
```

```
  First Name Middle Name Last Name  Age Gender  Salary
0       John              Johnson   30      M   60000
1    Matthew       River            50      M   70000
2    Richard                Walker   42         400000
3     Sophia       Grace     Smith   38      F  500000
4      David     Michael    Taylor   45   None       0
```

.count()

– Returns the count of each column (the count includes only non-null values).

```
[11]: print(pandasDF.count())
```

```
First Name     5
Middle Name    5
Last Name      5
Age            5
Gender         4
Salary         5
dtype: int64
```

.head(n)– Returns first n rows from the top.

```
[14]: print(pandasDF.head(2))
```

```
  First Name Middle Name Last Name  Age Gender  Salary
0       John               Johnson   30      M   60000
1    Matthew       River            50      M   70000
```

The following function utilizes the pandas library to manipulate a DataFrame by adding 10 to the values in column 'B'.

```
[14]: import pandas as pd

# Sample data
data = {'A': [1, 2, 3, 4, 5],
        'B': [10, 20, 30, 40, 50]}

# Create DataFrame
df = pd.DataFrame(data)

# Define a function to add the 10 in the values
def add_values(x):
    return x+10

# Apply the function to column 'B'
df['B'] = df['B'].apply(add_values)

# Display the modified DataFrame
print(df)
```

```
   A   B
0  1  20
1  2  30
2  3  40
3  4  50
4  5  60
```