Execute Manipulating, Dropping, Sorting, Aggregations, Joining, GroupBy in Pyspark DataFrames

- ## Manipulating :

PySpark DataFrames are immutable, which means that they cannot be changed once they are created. However, there are a number of ways to manipulate DataFrames.

First we are initiating the spark session and creating a dataframe.

```python
# Execute Manipulating, Droping, Sorting, Aggregations, Joining, GroupeBy in DataFrames
import pyspark
from pyspark.sql import SparkSession

# Initializing Spark Session
spark = SparkSession.builder.appName("Manipulation in dataframes").getOrCreate()
# Creating dataframe
data = [
    ('Mitushi',22,'F',1000),
    ('Vishesh',24,'M',2000),
    ('Tanisha',22,'F',3000),
    ('Zamran',38,'M',5000)
]
columns = ["Name",'Age','Gender',"Salary"]
df = spark.createDataFrame(data=data,schema=columns)
df.show()
```

```
▸ (3) Spark Jobs
▸ df: pyspark.sql.dataframe.DataFrame = [Name: string, Age: long ... 2 more fields]
+-------+---+------+------+
|   Name|Age|Gender|Salary|
+-------+---+------+------+
|Mitushi| 22|     F|  1000|
|Vishesh| 24|     M|  2000|
|Tanisha| 22|     F|  3000|
| Zamran| 38|     M|  5000|
+-------+---+------+------+
```

1. **Adding new columns** : Using withColumn() method we can add new column to existing dataframe and using lit() function from pyspark.sql.functions module we can assign a constant value to the new column.

   Syntax : dataframe.withColumn(column_name, lit(constant_value))

Here we added a new column “ Country “ with value “India”.

Cmd 2

```python
# Adding new column
# using withColumn() with lit()
from pyspark.sql.functions import lit

df_newcolumn = df.withColumn('Country',lit("India"))
df_newcolumn.show()
```

(3) Spark Jobs

df_newcolumn: pyspark.sql.dataframe.DataFrame = [Name: string, Age: long ... 3 more fields]

```
+-------+---+------+------+-------+
|   Name|Age|Gender|Salary|Country|
+-------+---+------+------+-------+
|Mitushi| 22|     F|  1000|  India|
|Vishesh| 24|     M|  2000|  India|
|Tanisha| 22|     F|  3000|  India|
| Zamran| 38|     M|  5000|  India|
+-------+---+------+------+-------+
```

2. **Filtering Data using filter()** : We can manipulate the data in a way that we are filtering specific values from the data. We can use filter() to filter the data as shown below. Here we are filtering the rows where the age > 22.
   Syntax : dataframe.filter(condition)

```python
# Filtering Data using filter()

df_filter = df.filter(df['Age']> 22)
df_filter.show()
```

(3) Spark Jobs

df_filter: pyspark.sql.dataframe.DataFrame = [Name: string, Age: long ... 2 more fields]

```
+-------+---+------+------+
|   Name|Age|Gender|Salary|
+-------+---+------+------+
|Vishesh| 24|     M|  2000|
| Zamran| 38|     M|  5000|
+-------+---+------+------+
```

3. **Selecting columns from the dataframe** : We can select single or multiple columns using select() method. As shown below we are selecting column " Name" from the dataframe.

Syntax : dataframe.select(dataframe[column_name1], dataframe[column_name2]..)

```
# Selecting Columns
df.select(df['Name']).show()
```

(3) Spark Jobs

```
+-------+
|   Name|
+-------+
|Mitushi|
|Vishesh|
|Tanisha|
| Zamran|
+-------+
```

• Dropping

Dropping in Pyspark can be done to handle null values, or dropping not required columns. Drop() method can be used to drop columns, null values, duplicate values.

We are using drop() method to drop the rows containing null values.Firstly we are reading a csv file containing some null values. Then using .na.drop() we are dropping them.

Syntax : dataframe.na.drop(how='any',thresh=integer,subset=column_name)

```python
# Execute Dropping
csv_file = spark.read.csv('/FileStore/tables/jobs_in_data___Copy.csv',header=True,inferSchema=True)
csv_file.show()
```

(3) Spark Jobs

csv_file: pyspark.sql.dataframe.DataFrame = [work_year: integer, job_title: string ... 4 more fields]

```
+---------+--------------------+------+-----------------+------------+----------------+
|work_year|           job_title|salary|employee_residence|work_setting|company_location|
+---------+--------------------+------+-----------------+------------+----------------+
|     2023|Data DevOps Engineer| 88000|          Germany|      Hybrid|         Germany|
|     2023|      Data Architect|186000|    United States|   In-person|   United States|
|     2023|      Data Architect| 81800|             null|   In-person|   United States|
|     2023|      Data Scientist|212000|    United States|   In-person|   United States|
|     2023|                null| 93300|    United States|   In-person|   United States|
|     2023|      Data Scientist|130000|    United States|      Remote|   United States|
|     2023|      Data Scientist|100000|    United States|      Remote|            null|
|     null|Machine Learning ...|  null|    United States|   In-person|   United States|
|     2023|Machine Learning ...|138700|    United States|        null|   United States|
|     2023|       Data Engineer|210000|    United States|      Remote|   United States|
|     2023|                null|168000|    United States|      Remote|   United States|
|     2023|Machine Learning ...|224400|    United States|   In-person|   United States|
|     2023|Machine Learning ...|138700|    United States|   In-person|   United States|
|     2023|      Data Scientist| 35000|   United Kingdom|   In-person|  United Kingdom|
|     2023|      Data Scientist| 30000|   United Kingdom|   In-person|  United Kingdom|
|     2023|        Data Analyst| 95000|             null|   In-person|   United States|
|     2023|        Data Analyst| 75000|    United States|   In-person|   United States|
|     2023|      Data Scientist|300000|    United States|   In-person|   United States|
```

```python
# dropping column
csv_file.drop(csv_file['work_year'],csv_file['work_setting']).show()
```

(1) Spark Jobs

```
+--------------------+------+-----------------+----------------+
|           job_title|salary|employee_residence|company_location|
+--------------------+------+-----------------+----------------+
|Data DevOps Engineer| 88000|          Germany|         Germany|
|      Data Architect|186000|    United States|   United States|
|      Data Architect| 81800|             null|   United States|
|      Data Scientist|212000|    United States|   United States|
|                null| 93300|    United States|   United States|
|      Data Scientist|130000|    United States|   United States|
|      Data Scientist|100000|    United States|            null|
|Machine Learning ...|  null|    United States|   United States|
|Machine Learning ...|138700|    United States|   United States|
|       Data Engineer|210000|    United States|   United States|
|                null|168000|    United States|   United States|
|Machine Learning ...|224400|    United States|   United States|
|Machine Learning ...|138700|    United States|   United States|
|      Data Scientist| 35000|   United Kingdom|  United Kingdom|
|      Data Scientist| 30000|   United Kingdom|  United Kingdom|
|        Data Analyst| 95000|             null|   United States|
|        Data Analyst| 75000|    United States|   United States|
|      Data Scientist|300000|    United States|   United States|
```

Command took 0.65 seconds -- by mitushivishrgpv@gmail.com at 2/12/2024, 12:32:12 PM on Test

Cmd 6

```
# dropping na values
csv_file.na.drop().show()
```

(1) Spark Jobs

```
+---------+--------------------+------+-----------------+------------+----------------+
|work_year|           job_title|salary|employee_residence|work_setting|company_location|
+---------+--------------------+------+-----------------+------------+----------------+
|     2023|Data DevOps Engineer| 88000|          Germany|      Hybrid|         Germany|
|     2023|      Data Architect|186000|    United States|   In-person|   United States|
|     2023|      Data Scientist|212000|    United States|   In-person|   United States|
|     2023|      Data Scientist|130000|    United States|      Remote|   United States|
|     2023|       Data Engineer|210000|    United States|      Remote|   United States|
|     2023|Machine Learning ...|224400|    United States|   In-person|   United States|
|     2023|Machine Learning ...|138700|    United States|   In-person|   United States|
|     2023|      Data Scientist| 35000|   United Kingdom|   In-person|  United Kingdom|
|     2023|      Data Scientist| 30000|   United Kingdom|   In-person|  United Kingdom|
|     2023|        Data Analyst| 75000|    United States|   In-person|   United States|
|     2023|      Data Scientist|300000|    United States|   In-person|   United States|
+---------+--------------------+------+-----------------+------------+----------------+
```

• Sorting

Sorting of data can be done using sort() or orderBymethod. Here we are sorting the data using the sort() method on column “Name” with descending order.
Again we are sorting data using orderBy() method to sort data on the basis on age.

```
# Sorting
# using sort()
df.sort(df['Name'].desc()).show()
# using orderBy()
df.orderBy('Age').show()
```

(2) Spark Jobs

```
+-------+---+------+------+
|   Name|Age|Gender|Salary|
+-------+---+------+------+
| Zamran| 38|     M|  5000|
|Vishesh| 24|     M|  2000|
|Tanisha| 22|     F|  3000|
|Mitushi| 22|     F|  1000|
+-------+---+------+------+

+-------+---+------+------+
|   Name|Age|Gender|Salary|
+-------+---+------+------+
|Mitushi| 22|     F|  1000|
|Tanisha| 22|     F|  3000|
|Vishesh| 24|     M|  2000|
| Zamran| 38|     M|  5000|
+-------+---+------+------+
```

● Aggregations and GroupBy

Aggregations in PySpark are operations that perform computations on groups of rows in a DataFrame. PySpark provides various built-in functions for aggregations.

We can use groupBy() method to group data then using aggregation methods like count(),sum(),min(),etc to find the aggregate results.

sum()

```
# GroupBy and Aggregations
csv_file.groupBy('job_title').sum('salary').show()
```

(2) Spark Jobs

```
+--------------------+-----------+
|           job_title|sum(salary)|
+--------------------+-----------+
|                null|     495300|
|Machine Learning ...|     363100|
|      Data Scientist|     807000|
|        Data Analyst|     170000|
|Data DevOps Engineer|      88000|
|      Data Architect|     267800|
|Machine Learning ...|     138700|
|       Data Engineer|     210000|
+--------------------+-----------+
```

min()

Cmd 10

```
csv_file.groupBy('job_title').min('salary').show()
```

(2) Spark Jobs

```
+--------------------+-----------+
|           job_title|min(salary)|
+--------------------+-----------+
|                null|      93300|
|Machine Learning ...|     138700|
|      Data Scientist|      30000|
|        Data Analyst|      75000|
|Data DevOps Engineer|      88000|
|      Data Architect|      81800|
|Machine Learning ...|     138700|
|       Data Engineer|     210000|
+--------------------+-----------+
```

avg()

Cmd 11

```
1    csv_file.groupBy('job_title').avg('salary').show()
```

(2) Spark Jobs

```
+--------------------+-----------+
|           job_title|avg(salary)|
+--------------------+-----------+
|                null|   165100.0|
|Machine Learning ...|   181550.0|
|      Data Scientist|   134500.0|
|        Data Analyst|    85000.0|
|Data DevOps Engineer|    88000.0|
|      Data Architect|   133900.0|
|Machine Learning ...|   138700.0|
|       Data Engineer|   210000.0|
+--------------------+-----------+
```

count()

```
1    csv_file.groupBy('job_title').count().show()
```

(2) Spark Jobs

```
+--------------------+-----+
|           job_title|count|
+--------------------+-----+
|                null|    3|
|Machine Learning ...|    2|
|      Data Scientist|    6|
|        Data Analyst|    2|
|Data DevOps Engineer|    1|
|      Data Architect|    2|
|Machine Learning ...|    2|
|       Data Engineer|    1|
+--------------------+-----+
```

- **Joins**

In PySpark, we can perform joins on DataFrames using the join() method. This method allows us to join two DataFrames based on one or more columns. Here's how we can perform joins in PySpark:

Inner join :

```
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"inner") .show()
```

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

Outer join :

```
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"outer").show()
#Or instead of outer we can give full Or fullouter
```

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

```
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"left").show()
# Or leftouter
```

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

Right join :

```
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"right").show()
# Or rightouter
```

```
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|    name|superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|     4|   Jones|              2|       2005|         10|     F|  2000|  Finance|     10|
|     3|Williams|              1|       2010|         10|     M|  1000|  Finance|     10|
|     1|   Smith|             -1|       2018|         10|     M|  3000|  Finance|     10|
|     2|    Rose|              1|       2010|         20|     M|  4000|Marketing|     20|
|  null|    null|           null|       null|       null|  null|  null|    Sales|     30|
|     5|   Brown|              2|       2010|         40|      |    -1|       IT|     40|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
```

Leftsemi join :

```
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftsemi").show()
```

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

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

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

● SparkSQL Joins

In Spark SQL, we can perform joins using SQL queries on DataFrames. This approach allows us to leverage SQL syntax to express join operations.

```python
# Spark SQL joins
empDF.createOrReplaceTempView("EmployeeView")
deptDF.createOrReplaceTempView("DeptView")
spark.sql("SELECT * from EmployeeView").show()
spark.sql("SELECT * from DeptView").show()
```

▸ (6) Spark Jobs

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

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

Inner Join :

```
spark.sql("SELECT * from DeptView DeptDF JOIN EmployeeView EmpDF ON empDF.emp_dept_id ==  deptDF.dept_id ").show()
```

(3) Spark Jobs

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

Left Join :

Cmd 22

```
spark.sql("SELECT * from DeptView DeptDF LEFT JOIN EmployeeView EmpDF ON empDF.emp_dept_id ==  deptDF.dept_id ").show()
```

(6) Spark Jobs

dept_name	dept_id	emp_id	name	superior_emp_id	year_joined	emp_dept_id	gender	salary
Finance	10	4	Jones	2	2005	10	F	2000
Finance	10	3	Williams	1	2010	10	M	1000
Finance	10	1	Smith	-1	2018	10	M	3000
Marketing	20	2	Rose	1	2010	20	M	4000
Sales	30	null	null	null	null	null	null	null
IT	40	5	Brown	2	2010	40		-1

Right Join :

Cmd 23

```
spark.sql("SELECT * from DeptView DeptDF RIGHT JOIN EmployeeView EmpDF ON empDF.emp_dept_id ==  deptDF.dept_id ").show()
```

(6) Spark Jobs

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

Full Outer Join :

```
spark.sql("SELECT * from DeptView DeptDF FULL OUTER JOIN EmployeeView EmpDF ON empDF.emp_dept_id ==  deptDF.dept_id ").show()
```

(3) Spark Jobs

```
+---------+-------+------+--------+---------------+-----------+-----------+------+------+
|dept_name|dept_id|emp_id|    name|superior_emp_id|year_joined|emp_dept_id|gender|salary|
+---------+-------+------+--------+---------------+-----------+-----------+------+------+
|  Finance|     10|     1|   Smith|             -1|       2018|         10|     M|  3000|
|  Finance|     10|     3|Williams|              1|       2010|         10|     M|  1000|
|  Finance|     10|     4|   Jones|              2|       2005|         10|     F|  2000|
|Marketing|     20|     2|    Rose|              1|       2010|         20|     M|  4000|
|    Sales|     30|  null|    null|           null|       null|       null|  null|  null|
|       IT|     40|     5|   Brown|              2|       2010|         40|      |    -1|
|     null|   null|     6|   Brown|              2|       2010|         50|      |    -1|
+---------+-------+------+--------+---------------+-----------+-----------+------+------+
```

Left Anti Join :

```
spark.sql("SELECT * from DeptView DeptDF LEFT ANTI JOIN EmployeeView EmpDF ON empDF.emp_dept_id ==  deptDF.dept_id ").show()
```

(4) Spark Jobs

```
+---------+-------+
|dept_name|dept_id|
+---------+-------+
|    Sales|     30|
+---------+-------+
```

Left Semi Join :

```
spark.sql("SELECT * from DeptView DeptDF LEFT SEMI JOIN EmployeeView EmpDF ON empDF.emp_dept_id ==  deptDF.dept_id ").show()
```

(3) Spark Jobs

```
+---------+-------+
|dept_name|dept_id|
+---------+-------+
|  Finance|     10|
|Marketing|     20|
|       IT|     40|
+---------+-------+
```

● Applying Functions in a Pandas DataFrame

To apply Pandas functions directly to PySpark DataFrame columns, we can use the pandas_udf module introduced in PySpark 2.3 and later versions. This allows us to use Pandas-like syntax to apply functions to PySpark DataFrame columns efficiently.

In below example ,

- We initialize a SparkSession.
- We create a DataFrame df with sample data.
- We define a Pandas UDF named square_udf using the pandas_udf decorator, specifying the return type (DoubleType() in this case).
- Inside the UDF, we use Pandas-like syntax to apply a square function to each element in the input age_series.
- We use the withColumn() method to apply the Pandas UDF to the "Age" column in the DataFrame and create a new column named "AgeSquared".
- We display the resulting DataFrame using show().

```python
import pandas as pd
from pyspark.sql.functions import pandas_udf
from pyspark.sql.types import DoubleType

@pandas_udf(DoubleType())
def square_udf(age_series: pd.Series) -> pd.Series:
    return age_series.apply(lambda x: x ** 2)

# Apply the Pandas UDF to a column
df = df.withColumn("AgeSquared", square_udf(df["Age"]))
df.show()
```

(3) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [Name: string, Age: long ... 3 more fields]

```
+-------+---+------+------+----------+
|   Name|Age|Gender|Salary|AgeSquared|
+-------+---+------+------+----------+
|Mitushi| 22|     F|  1000|     484.0|
|Vishesh| 24|     M|  2000|     576.0|
|Tanisha| 22|     F|  3000|     484.0|
| Zamran| 38|     M|  5000|    1444.0|
+-------+---+------+------+----------+
```

```python
# Applying Functions in a Pandas DataFrame
import pyspark.pandas as ps
pandasdf = ps.DataFrame({'a': [1,2,3], 'b':[4,5,6]})
def pandas_plus(pser):
    return pser + 1
pandasdf.apply(pandas_plus)
```

▸ (2) Spark Jobs

	a	b
0	2	5
1	3	6
2	4	7