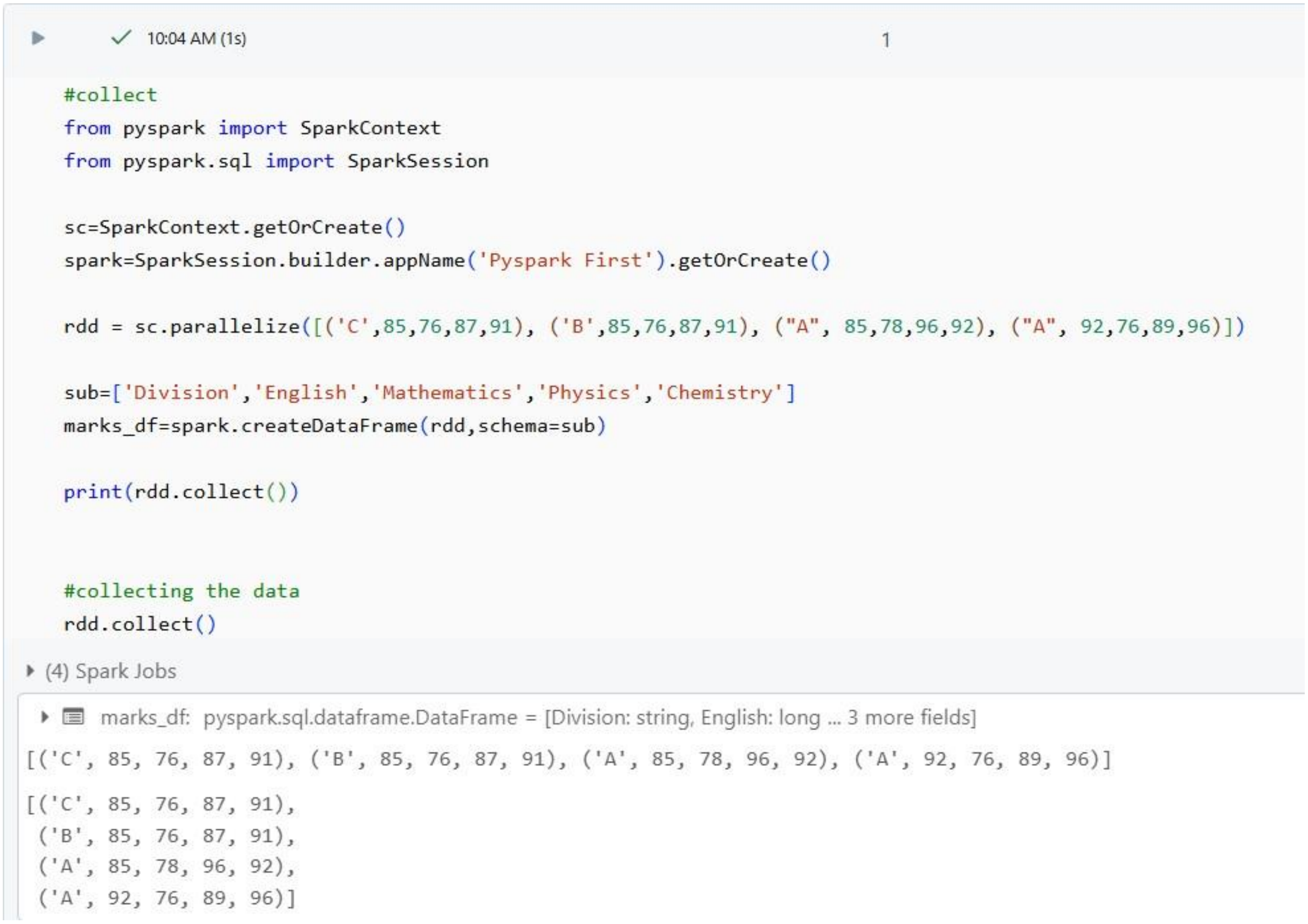
Apache Spark Day3 Assignment

Submitted by – Aniroop Gupta
DE Batch1

I. Transformation & actions in PySpark:

Execution Screenshots:

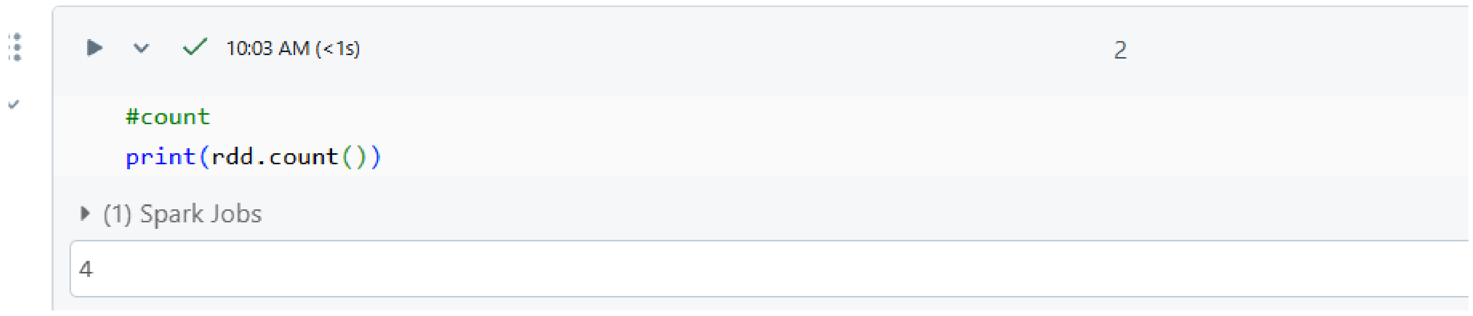
1.collect():



```
#collect
from pyspark import SparkContext
from pyspark.sql import SparkSession

sc=SparkContext.getOrCreate()
spark=SparkSession.builder.appName('Pyspark First').getOrCreate()

rdd = sc.parallelize([('C',85,76,87,91), ('B',85,76,87,91), ("A", 85,78,96,92), ("A", 92,76,89,96)])

sub=['Division','English','Mathematics','Physics','Chemistry']
marks_df=spark.createDataFrame(rdd,schema=sub)

print(rdd.collect())


#collecting the data
rdd.collect()
```

```
marks_df: pyspark.sql.dataframe.DataFrame = [Division: string, English: long ... 3 more fields]
[('C', 85, 76, 87, 91), ('B', 85, 76, 87, 91), ('A', 85, 78, 96, 92), ('A', 92, 76, 89, 96)]
[('C', 85, 76, 87, 91),
 ('B', 85, 76, 87, 91),
 ('A', 85, 78, 96, 92),
 ('A', 92, 76, 89, 96)]
```

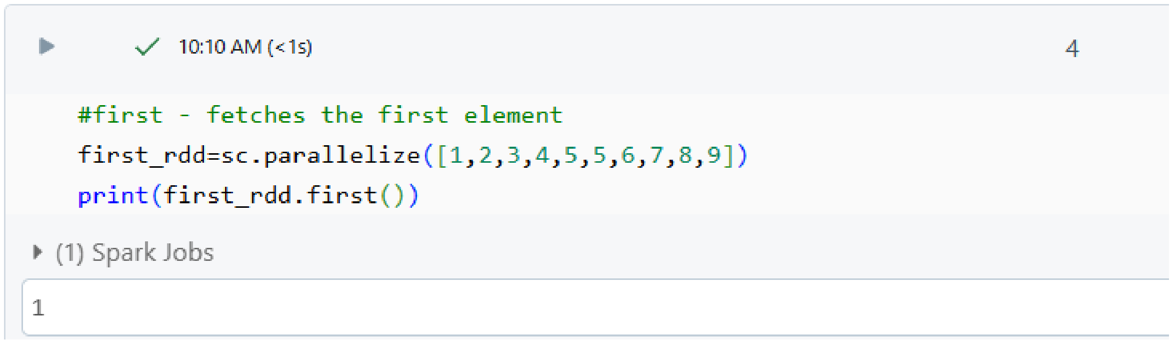
2. count():



```
#count
print(rdd.count())
```

```
4
```

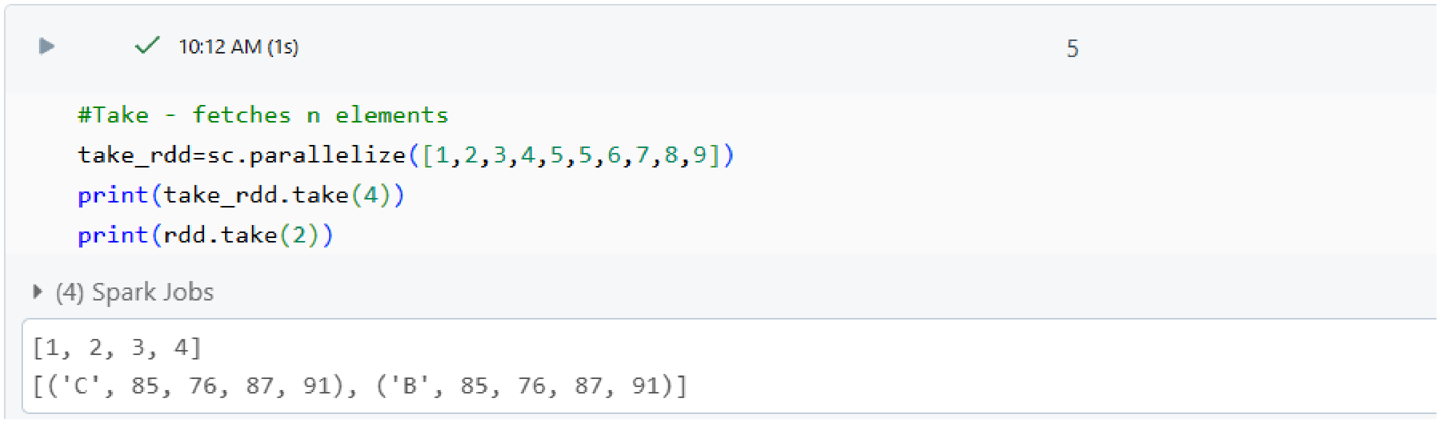
3. first():



```
#first - fetches the first element
first_rdd=sc.parallelize([1,2,3,4,5,5,6,7,8,9])
print(first_rdd.first())
```

```
1
```

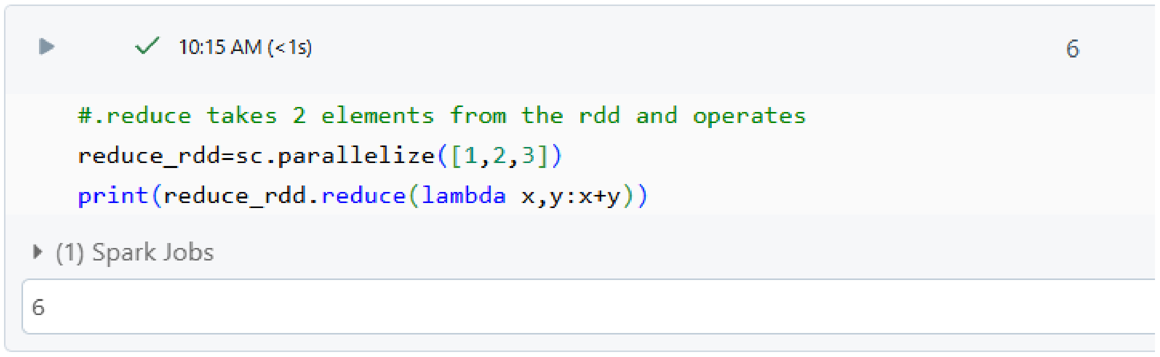
4. take():



```
#Take - fetches n elements
take_rdd=sc.parallelize([1,2,3,4,5,5,6,7,8,9])
print(take_rdd.take(4))
print(rdd.take(2))
```

```
[1, 2, 3, 4]
[('C', 85, 76, 87, 91), ('B', 85, 76, 87, 91)]
```

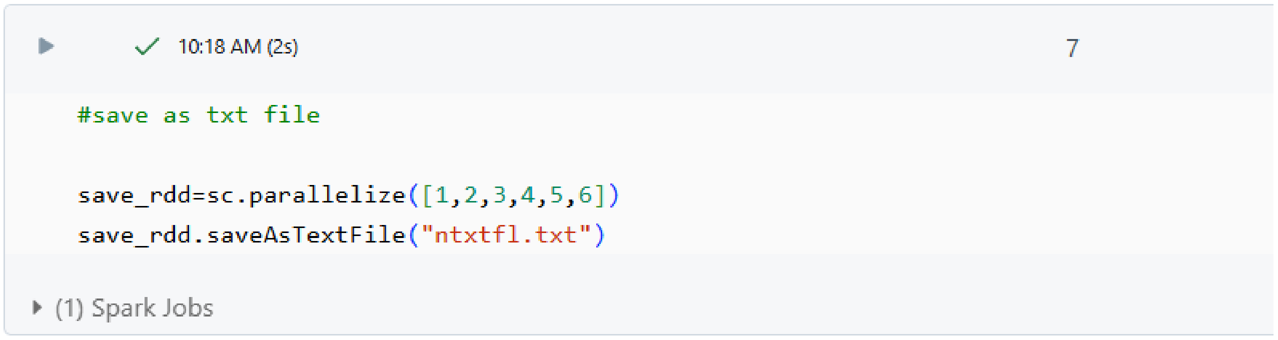
5. reduce():



```
#.reduce takes 2 elements from the rdd and operates
reduce_rdd=sc.parallelize([1,2,3])
print(reduce_rdd.reduce(lambda x,y:x+y))
```

```
6
```

6. saveAsTextFile():

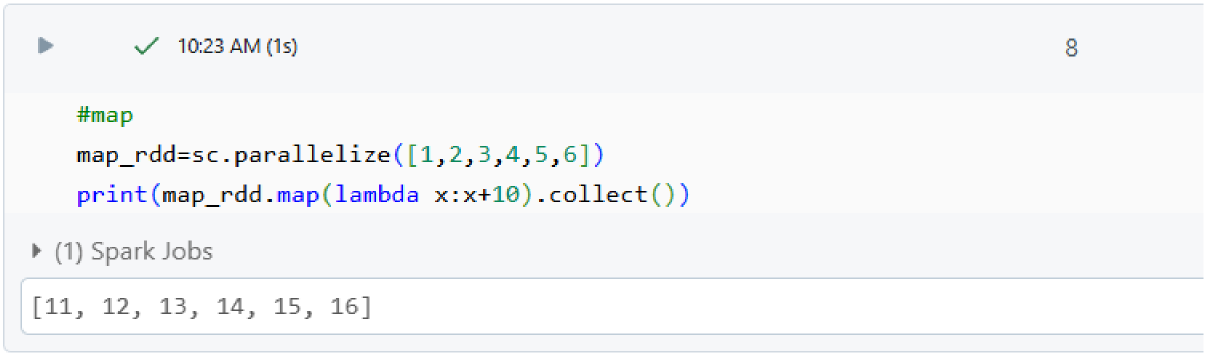


10:18 AM (2s) 7

```
#save as txt file

save_rdd=sc.parallelize([1,2,3,4,5,6])
save_rdd.saveAsTextFile("ntxtfl.txt")
```

(1) Spark Jobs

7. map():



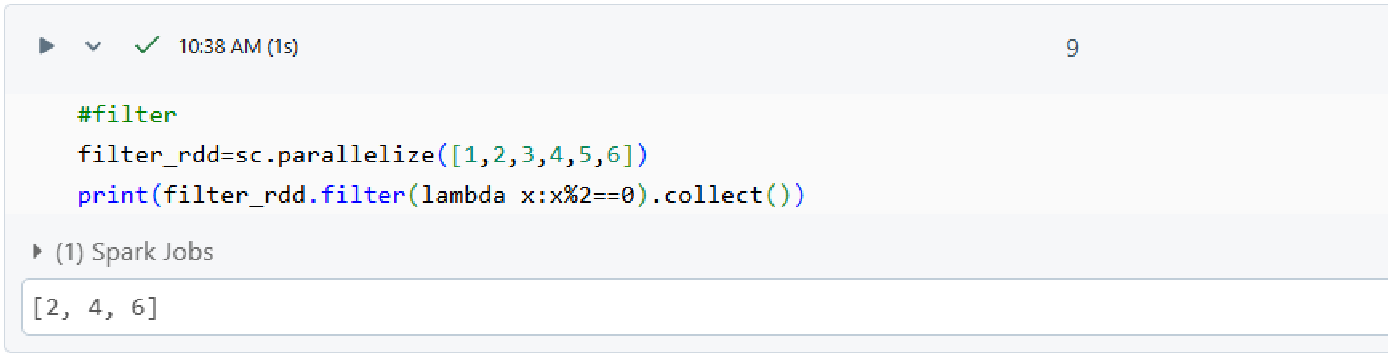
10:23 AM (1s) 8

```
#map
map_rdd=sc.parallelize([1,2,3,4,5,6])
print(map_rdd.map(lambda x:x+10).collect())
```

(1) Spark Jobs

```
[11, 12, 13, 14, 15, 16]
```

8. filter():



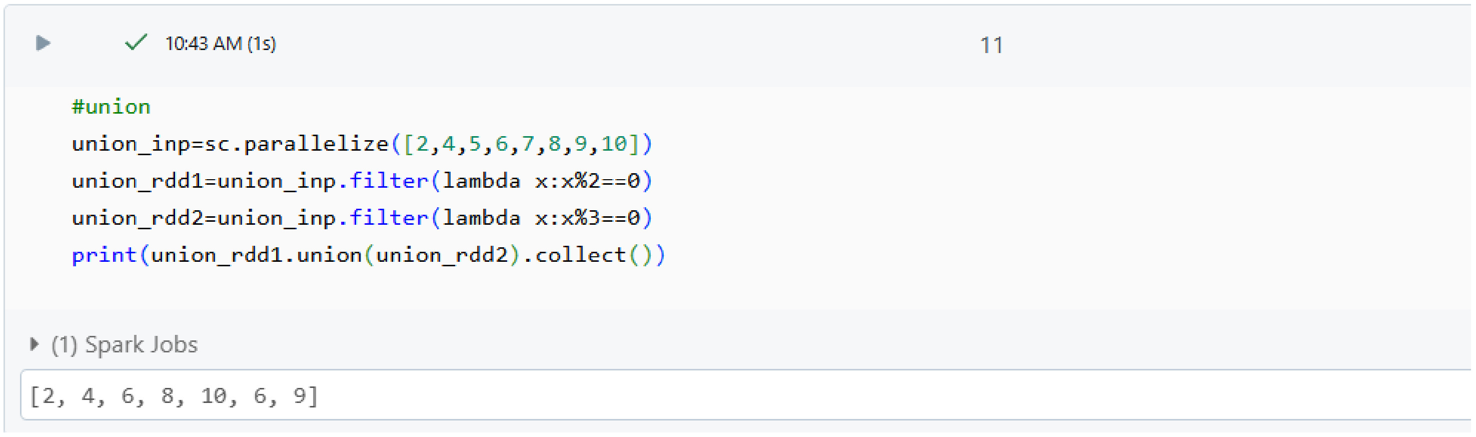
10:38 AM (1s) 9

```
#filter
filter_rdd=sc.parallelize([1,2,3,4,5,6])
print(filter_rdd.filter(lambda x:x%2==0).collect())
```

(1) Spark Jobs

```
[2, 4, 6]
```

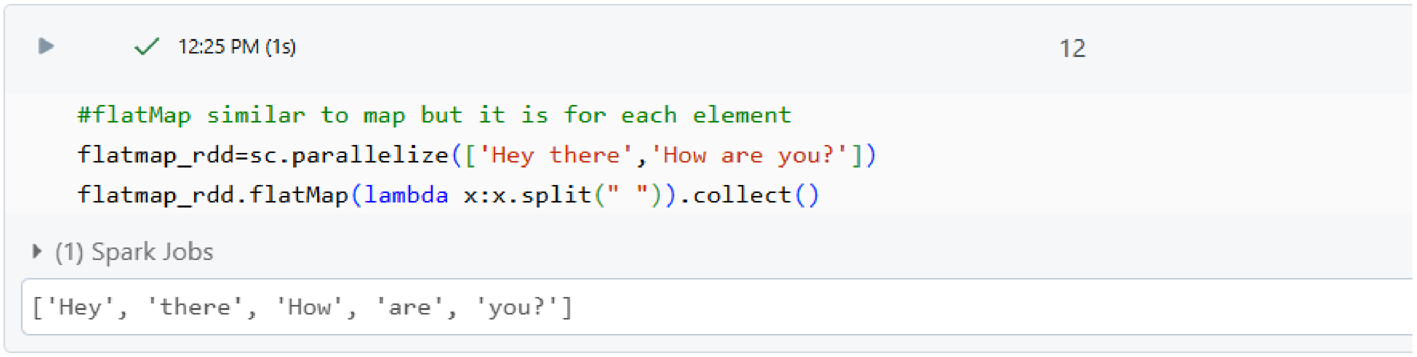
9. union():



```
#union
union_inp=sc.parallelize([2,4,5,6,7,8,9,10])
union_rdd1=union_inp.filter(lambda x:x%2==0)
union_rdd2=union_inp.filter(lambda x:x%3==0)
print(union_rdd1.union(union_rdd2).collect())
```

```
[2, 4, 6, 8, 10, 6, 9]
```

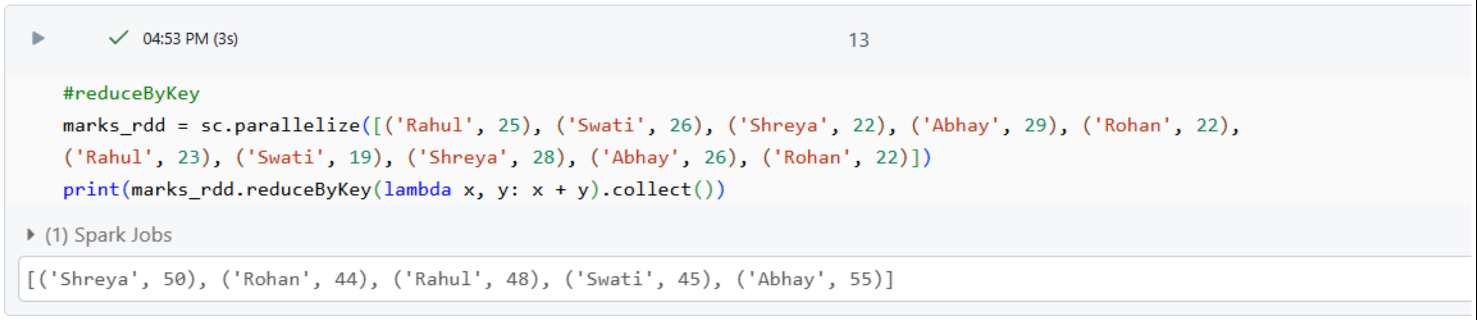
10. flatMap():



```
#flatMap similar to map but it is for each element
flatmap_rdd=sc.parallelize(['Hey there','How are you?'])
flatmap_rdd.flatMap(lambda x:x.split(" ")).collect()
```

```
['Hey', 'there', 'How', 'are', 'you?']
```

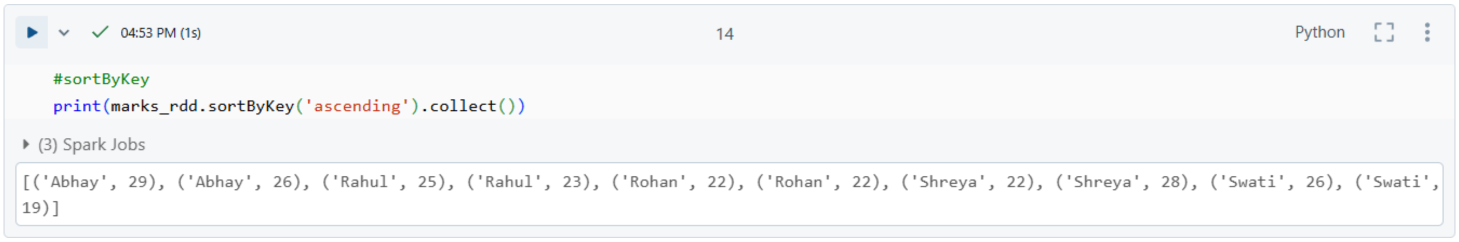
11. reduceByKey():



```
#reduceByKey
marks_rdd = sc.parallelize([('Rahul', 25), ('Swati', 26), ('Shreya', 22), ('Abhay', 29), ('Rohan', 22),
('Rahul', 23), ('Swati', 19), ('Shreya', 28), ('Abhay', 26), ('Rohan', 22)])
print(marks_rdd.reduceByKey(lambda x, y: x + y).collect())
```

```
[('Shreya', 50), ('Rohan', 44), ('Rahul', 48), ('Swati', 45), ('Abhay', 55)]
```

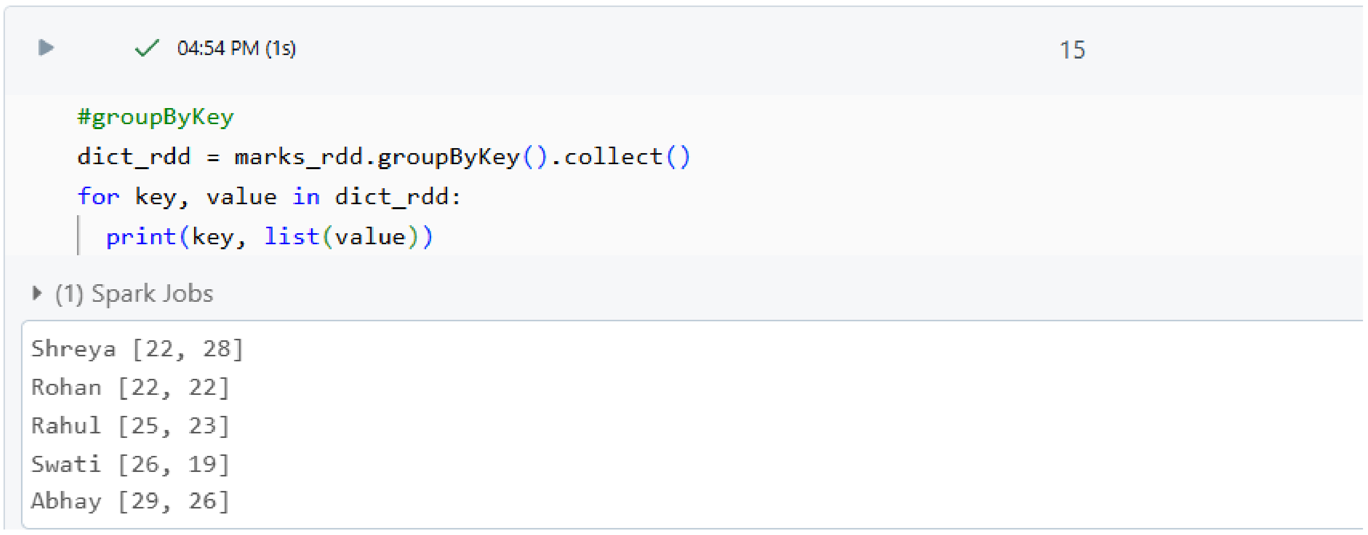
12. sortByKey():



```
#sortByKey
print(marks_rdd.sortByKey('ascending').collect())
```

```
[('Abhay', 29), ('Abhay', 26), ('Rahul', 25), ('Rahul', 23), ('Rohan', 22), ('Rohan', 22), ('Shreya', 22), ('Shreya', 28), ('Swati', 26), ('Swati',
19)]
```

13. groupByKey():

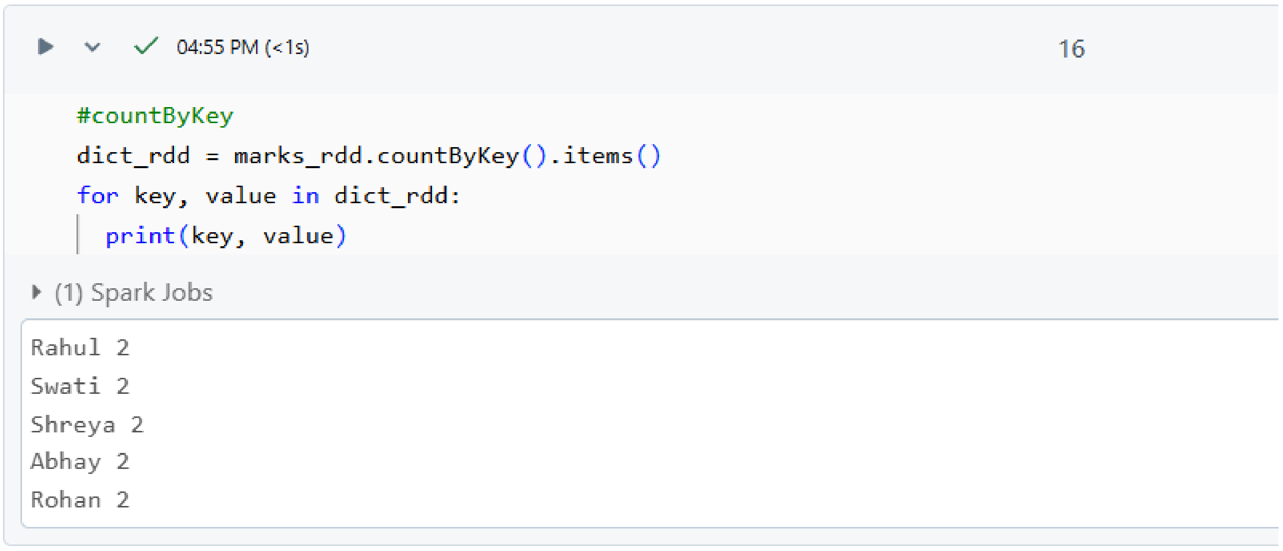

04:54 PM (1s) 15

```
#groupByKey
dict_rdd = marks_rdd.groupByKey().collect()
for key, value in dict_rdd:
    print(key, list(value))
```

(1) Spark Jobs

```
Shreya [22, 28]
Rohan [22, 22]
Rahul [25, 23]
Swati [26, 19]
Abhay [29, 26]
```

14. countByKey():


04:55 PM (<1s) 16

```
#countByKey
dict_rdd = marks_rdd.countByKey().items()
for key, value in dict_rdd:
    print(key, value)
```

(1) Spark Jobs

```
Rahul 2
Swati 2
Shreya 2
Abhay 2
Rohan 2
```

Execution Summary: *Transformations & Actions in PySpark*

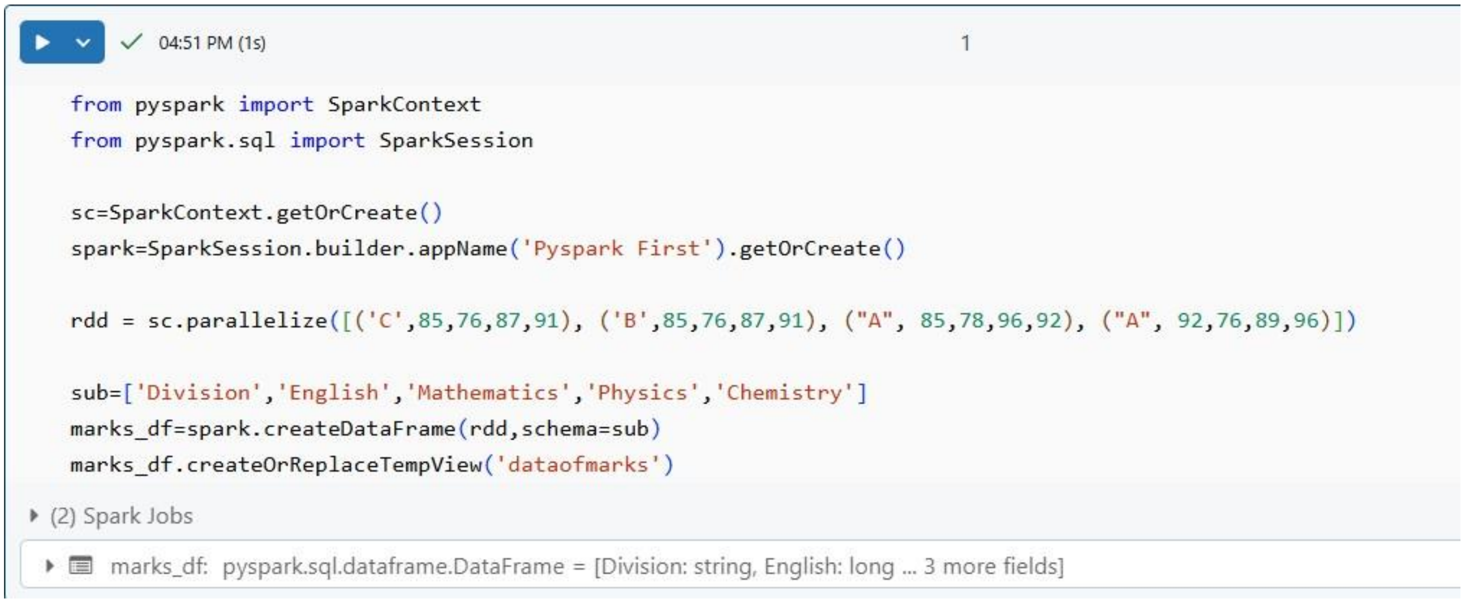
Transformations and actions are core concepts in PySpark that define how data is processed and results are obtained in distributed systems.

- **Transformations**:
  Transformations are operations applied to an RDD (Resilient Distributed Dataset) or DataFrame to produce a new dataset. These operations don't execute immediately but build a computation plan (DAG - Directed Acyclic Graph) that is executed only when an action is triggered. Examples of transformations include:
  - **map()**: Transforms each element of an RDD or DataFrame according to a function.
  - **filter()**: Filters elements based on a condition.
  - **union()**: Combines two RDDs or DataFrames.
  - **flatMap()**: Similar to map(), but flattens the output.
  - **reduceByKey()**, **groupByKey()**, **sortByKey()**: Specialized transformations for key-value pair RDDs that enable aggregation and sorting.
- **Actions**:
  Actions are operations that trigger the execution of the transformations applied on the data. They either return results to the driver or write them to an external storage system. Examples of actions include:
  - **collect()**: Brings all data to the driver as a list.
  - **count()**: Returns the number of elements in the dataset.
  - **first()**, **take()**: Retrieve one or a specified number of elements.
  - **reduce()**: Aggregates elements using a specified function.
  - **saveAsTextFile()**: Saves the output to a file in the distributed file system.

II. Pyspark view and temp view:

Execution Screenshots:

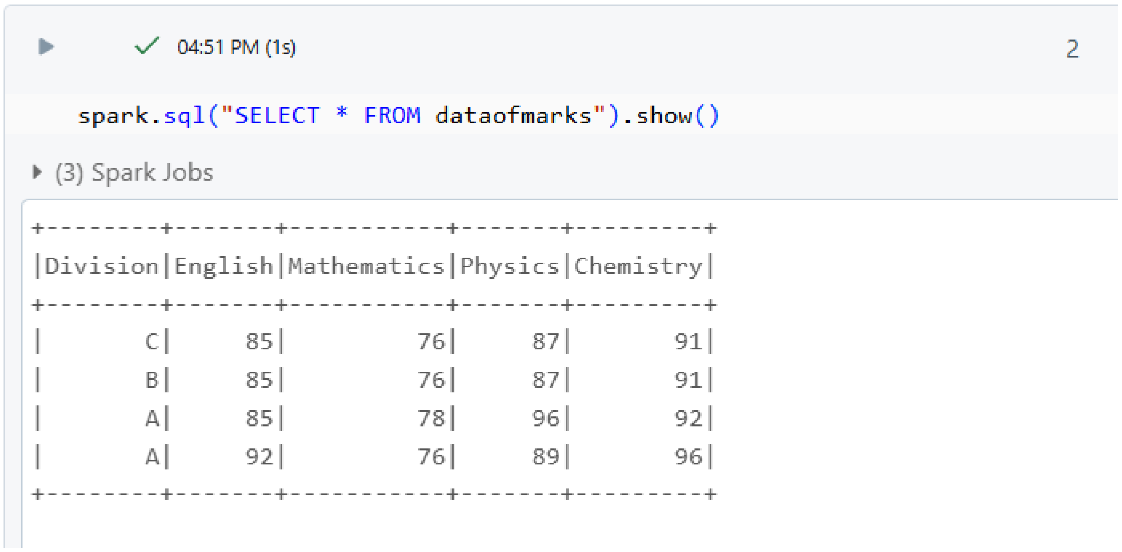
1.Creating a Temp View:



```
from pyspark import SparkContext
from pyspark.sql import SparkSession

sc=SparkContext.getOrCreate()
spark=SparkSession.builder.appName('Pyspark First').getOrCreate()

rdd = sc.parallelize([('C',85,76,87,91), ('B',85,76,87,91), ("A", 85,78,96,92), ("A", 92,76,89,96)])

sub=['Division','English','Mathematics','Physics','Chemistry']
marks_df=spark.createDataFrame(rdd,schema=sub)
marks_df.createOrReplaceTempView('dataofmarks')
```

```
(2) Spark Jobs
marks_df:  pyspark.sql.dataframe.DataFrame = [Division: string, English: long ... 3 more fields]
```

2.Displaying the created view:



```
spark.sql("SELECT * FROM dataofmarks").show()
```

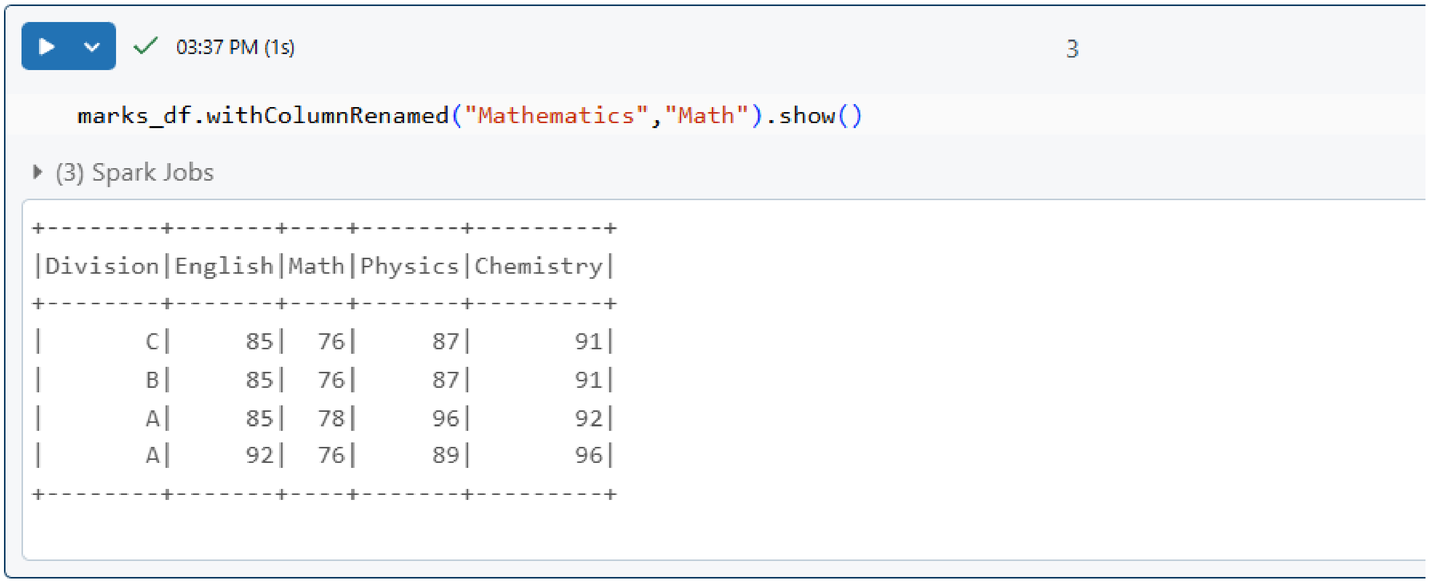
```
(3) Spark Jobs
+--------+-------+-----------+-------+---------+
|Division|English|Mathematics|Physics|Chemistry|
+--------+-------+-----------+-------+---------+
|       C|     85|         76|     87|       91|
|       B|     85|         76|     87|       91|
|       A|     85|         78|     96|       92|
|       A|     92|         76|     89|       96|
+--------+-------+-----------+-------+---------+
```

Execution Summary: *PySpark Views and Temporary Views*

PySpark temporary views allow DataFrames to be queried using SQL syntax. They are session-specific and created using the createOrReplaceTempView() method. Temporary views, like the dataofmarks example, provide a quick, dynamic way to run SQL queries on structured data through the spark.sql() function. These views simplify data manipulation with familiar SQL operations such as SELECT, WHERE, and GROUP BY. They are ideal for transient data processing without impacting persistent storage. While convenient, temporary views are session-scoped and must be recreated after a session ends. This feature bridges the gap between PySpark and SQL workflows efficiently.

III. Selecting, Renaming and filtering data in a Pandas Dataframe:

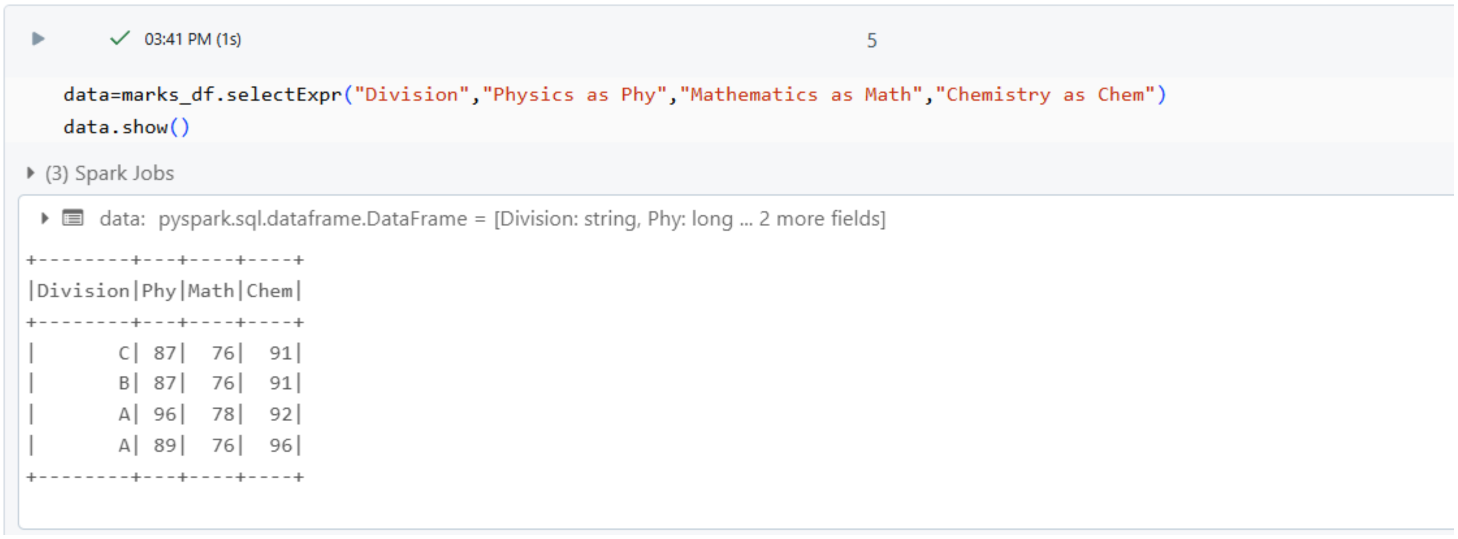
1.withColumnRenamed():



```
marks_df.withColumnRenamed("Mathematics","Math").show()
```

```
+--------+-------+----+-------+---------+
|Division|English|Math|Physics|Chemistry|
+--------+-------+----+-------+---------+
|       C|     85|  76|     87|       91|
|       B|     85|  76|     87|       91|
|       A|     85|  78|     96|       92|
|       A|     92|  76|     89|       96|
+--------+-------+----+-------+---------+
```

2.selectExpr():



03:41 PM (1s) 5

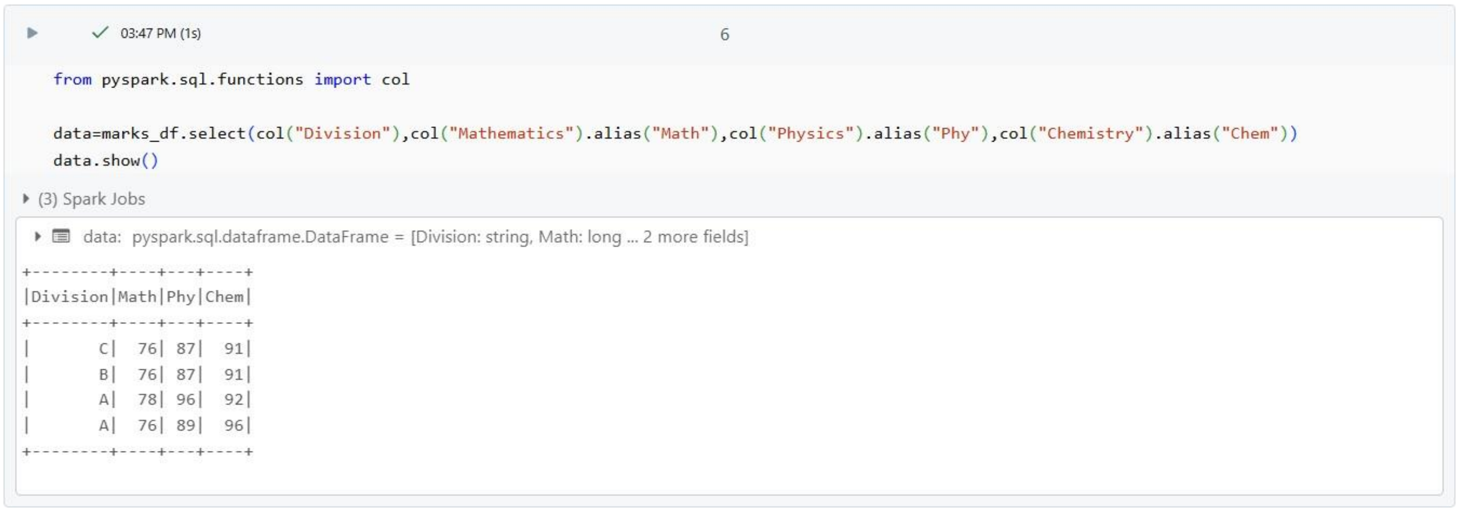
```
data=marks_df.selectExpr("Division","Physics as Phy","Mathematics as Math","Chemistry as Chem")
data.show()
```

(3) Spark Jobs

data: pyspark.sql.dataframe.DataFrame = [Division: string, Phy: long ... 2 more fields]

```
+--------+---+----+----+
|Division|Phy|Math|Chem|
+--------+---+----+----+
|       C| 87|  76|  91|
|       B| 87|  76|  91|
|       A| 96|  78|  92|
|       A| 89|  76|  96|
+--------+---+----+----+
```

3.select() (columns):



03:47 PM (1s) 6

```
from pyspark.sql.functions import col

data=marks_df.select(col("Division"),col("Mathematics").alias("Math"),col("Physics").alias("Phy"),col("Chemistry").alias("Chem"))
data.show()
```

(3) Spark Jobs

data: pyspark.sql.dataframe.DataFrame = [Division: string, Math: long ... 2 more fields]

```
+--------+----+---+----+
|Division|Math|Phy|Chem|
+--------+----+---+----+
|       C|  76| 87|  91|
|       B|  76| 87|  91|
|       A|  78| 96|  92|
|       A|  76| 89|  96|
+--------+----+---+----+
```

Execution Summary: *Selecting, Renaming, and Filtering Data in a PySpark DataFrame*

PySpark provides flexible methods for selecting, renaming, and filtering columns in a DataFrame, facilitating structured data manipulation:

- **Renaming Columns**:
    - withColumnRenamed("old_name", "new_name") renames a specific column.
    - In the example, the column Mathematics is renamed to Math.
- **Selecting Columns**:
    - select() allows specific columns to be retrieved. Using col() or .alias() enables renaming while selecting.
    - selectExpr() accepts SQL-like expressions to select and rename columns in a single step.

These methods enhance readability and organization of DataFrames, enabling dynamic transformation and preparation of data for analysis.