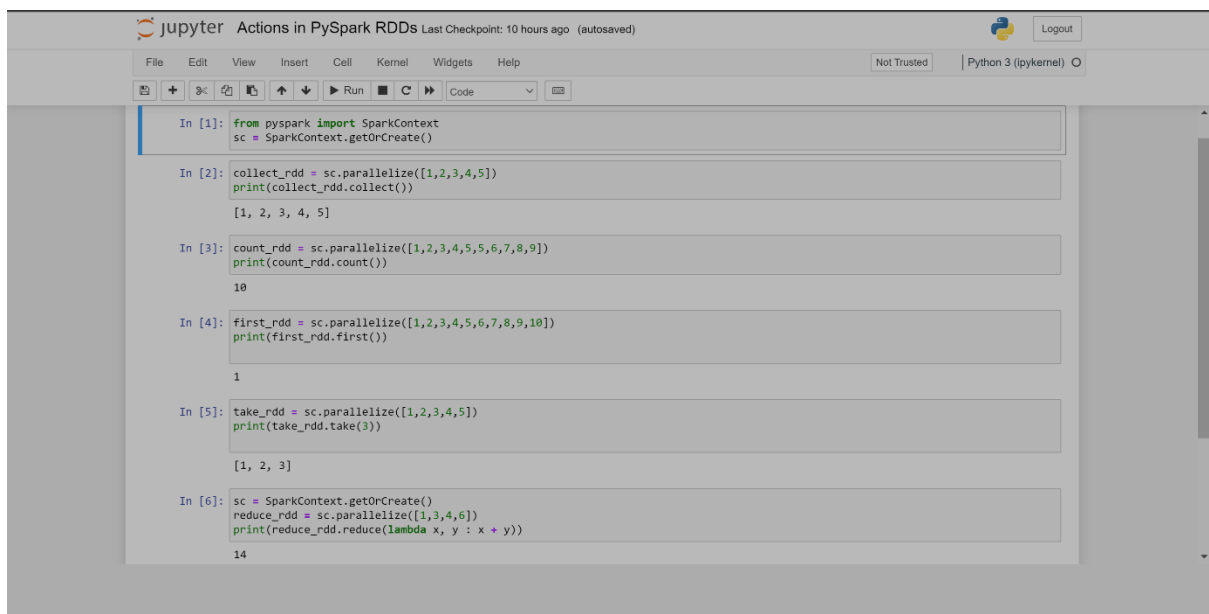


Name: Pradip Bochare

PySpark RDD Operations

- **Transformations:** Transformations are a kind of operation that takes an RDD as input and produces another RDD as output. Once a transformation is applied to an RDD, it returns a new RDD, the original RDD remains the same and thus are immutable.
- **Actions:** Actions are a kind of operation which are applied on an RDD to produce a single value. These methods are applied on a resultant RDD and produces a non-RDD value

❖ Actions in PySpark RDDs



A screenshot of a Jupyter Notebook titled "Actions in PySpark RDDs". The notebook interface includes a top bar with "jupyter" logo, the title, and a "Last Checkpoint: 10 hours ago (autosaved)" message. Below the top bar is a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook content consists of six code cells, each starting with "In [n]:". The code in these cells demonstrates various PySpark RDD actions: creating a SparkContext, parallelizing lists, collecting, counting, taking, and reducing. The output of each cell is displayed below the code. The notebook is running on Python 3 (ipykernel) and has a "Not Trusted" status.

```
In [1]: from pyspark import SparkContext
sc = SparkContext.getOrCreate()

In [2]: collect_rdd = sc.parallelize([1,2,3,4,5])
print(collect_rdd.collect())

[1, 2, 3, 4, 5]

In [3]: count_rdd = sc.parallelize([1,2,3,4,5,5,6,7,8,9])
print(count_rdd.count())

10

In [4]: first_rdd = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
print(first_rdd.first())

1

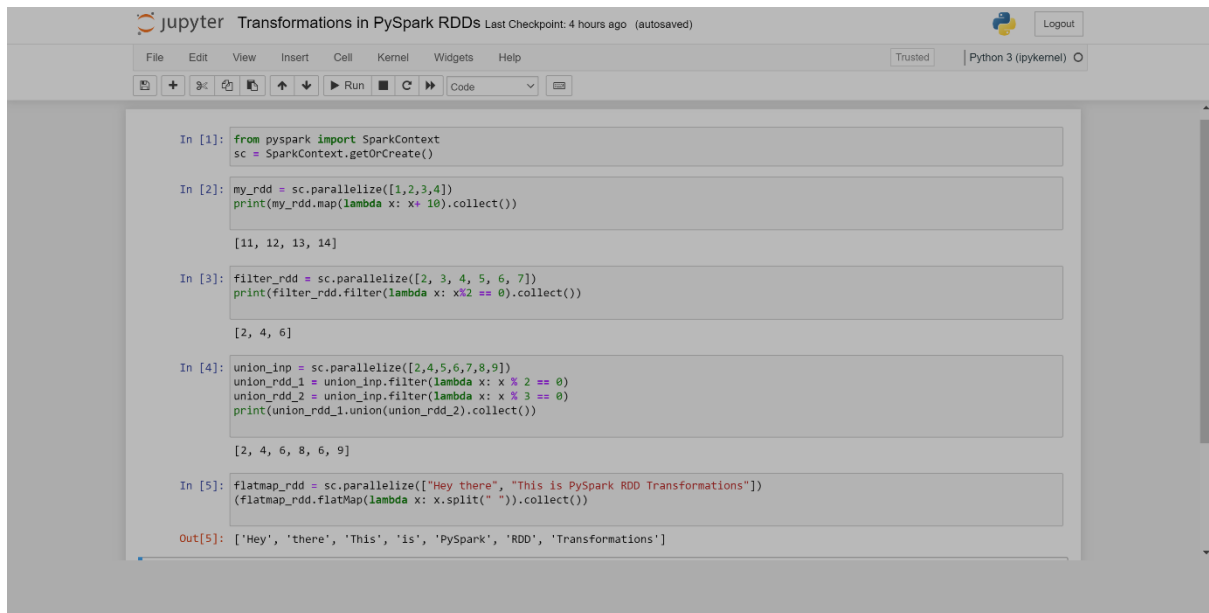
In [5]: take_rdd = sc.parallelize([1,2,3,4,5])
print(take_rdd.take(3))

[1, 2, 3]

In [6]: sc = SparkContext.getOrCreate()
reduce_rdd = sc.parallelize([1,3,4,6])
print(reduce_rdd.reduce(lambda x, y : x + y))

14
```

❖ Transformations in PySpark RDDs



A Jupyter notebook titled "Transformations in PySpark RDDs" showing a series of code cells demonstrating RDD transformations. The notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations, and a status bar indicating "Python 3 (ipykernel)".

```
In [1]: from pyspark import SparkContext
sc = SparkContext.getOrCreate()

In [2]: my_rdd = sc.parallelize([1,2,3,4])
print(my_rdd.map(lambda x: x*10).collect())

[11, 12, 13, 14]

In [3]: filter_rdd = sc.parallelize([2, 3, 4, 5, 6, 7])
print(filter_rdd.filter(lambda x: x%2 == 0).collect())

[2, 4, 6]

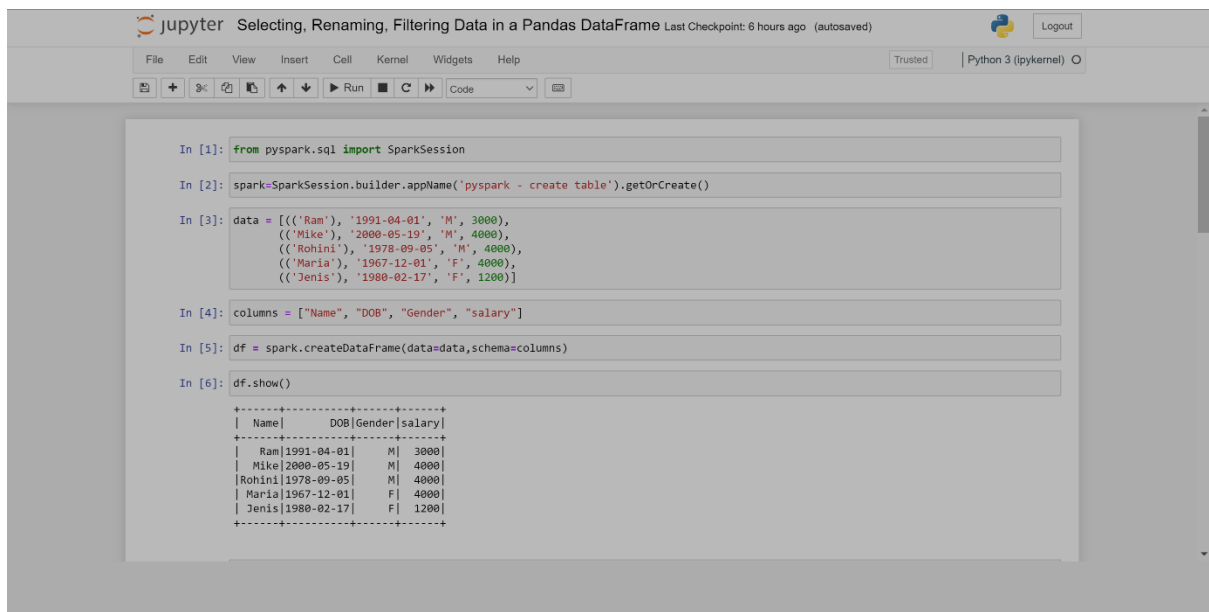
In [4]: union_inp = sc.parallelize([2,4,5,6,7,8,9])
union_rdd_1 = union_inp.filter(lambda x: x % 2 == 0)
union_rdd_2 = union_inp.filter(lambda x: x % 3 == 0)
print(union_rdd_1.union(union_rdd_2).collect())

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

In [5]: flatmap_rdd = sc.parallelize(["Hey there", "This is PySpark RDD Transformations"])
(flatmap_rdd.flatMap(lambda x: x.split(" ")).collect())

Out[5]: ['Hey', 'there', 'This', 'is', 'PySpark', 'RDD', 'Transformations']
```

+ Selecting, Renaming, Filtering Data in a Pandas DataFrame



A Jupyter notebook titled "Selecting, Renaming, Filtering Data in a Pandas DataFrame" showing code for creating a DataFrame from a list of tuples and displaying it. The notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations, and a status bar indicating "Python 3 (ipykernel)".

```
In [1]: from pyspark.sql import SparkSession

In [2]: spark=SparkSession.builder.appName('pyspark - create table').getOrCreate()

In [3]: data = [(['Ram'], '1991-04-01', 'M', 3000),
                (['Mike'], '2000-05-19', 'M', 4000),
                (['Rohini'], '1978-09-05', 'M', 4000),
                (['Maria'], '1967-12-01', 'F', 4000),
                (['Jenis'], '1980-02-17', 'F', 1200)]

In [4]: columns = ["Name", "DOB", "Gender", "salary"]

In [5]: df = spark.createDataFrame(data=data,schema=columns)

In [6]: df.show()

+-----+-----+-----+-----+
| Name |      DOB | Gender | salary |
+-----+-----+-----+-----+
| Ram  | 1991-04-01 | M      | 3000   |
| Mike | 2000-05-19 | M      | 4000   |
| Rohini | 1978-09-05 | M      | 4000   |
| Maria | 1967-12-01 | F      | 4000   |
| Jenis | 1980-02-17 | F      | 1200   |
+-----+-----+-----+-----+
```

Method 1: Using withColumnRenamed()

We will use of withColumnRenamed() method to change the column names of pyspark data frame.

Syntax: DataFrame.withColumnRenamed(existing, new)

Parameters

- existingstr: Existing column name of data frame to rename.
- newstr: New column name.

Returns type: Returns a data frame by renaming an existing column.

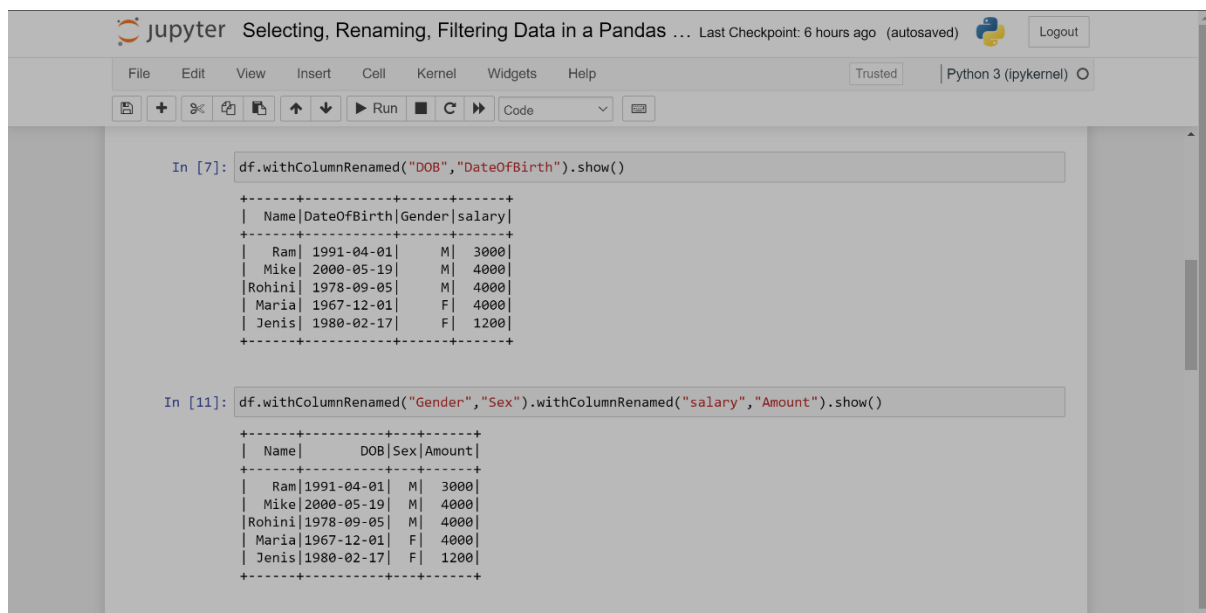
Method 2: Using selectExpr()

Renaming the column names using selectExpr() method

Syntax : DataFrame.selectExpr(expr)

Parameters :

expr : It's an SQL expression.



```
In [7]: df.withColumnRenamed("DOB", "DateOfBirth").show()

+-----+
| Name | DateOfBirth | Gender | salary |
+-----+
| Ram   | 1991-04-01  | M      | 3000   |
| Mike  | 2000-05-19  | M      | 4000   |
| Rohini | 1978-09-05  | M      | 4000   |
| Maria | 1967-12-01  | F      | 4000   |
| Jenis | 1980-02-17  | F      | 1200   |
+-----+

In [11]: df.withColumnRenamed("Gender", "Sex").withColumnRenamed("salary", "Amount").show()

+-----+
| Name | DOB | Sex | Amount |
+-----+
| Ram   | 1991-04-01 | M   | 3000   |
| Mike  | 2000-05-19 | M   | 4000   |
| Rohini | 1978-09-05 | M   | 4000   |
| Maria | 1967-12-01 | F   | 4000   |
| Jenis | 1980-02-17 | F   | 1200   |
+-----+
```

Method 3: Using select() method

Syntax: DataFrame.select(cols)

Parameters :

cols: List of column names as strings.

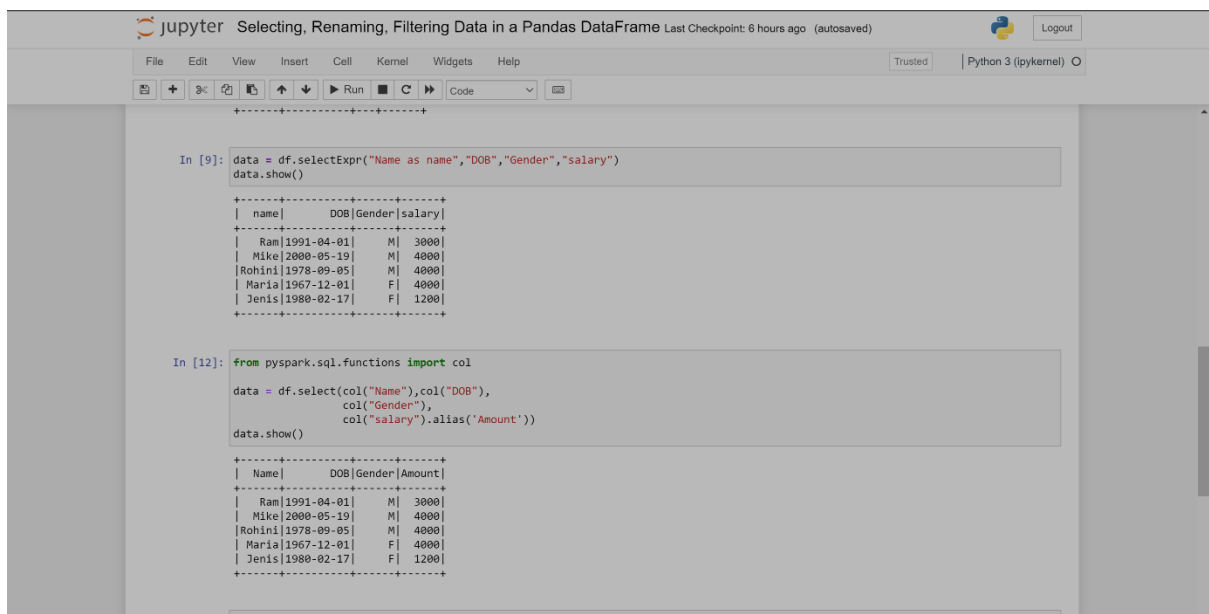
Return type: Selects the cols in the dataframe and returns a new DataFrame.

Method 4: Using toDF()

This function returns a new DataFrame that with new specified column names.

Syntax: toDF(*col)

Where, col is a new column name



The screenshot shows a Jupyter Notebook interface with the title "Selecting, Renaming, Filtering Data in a Pandas DataFrame". The notebook is running on Python 3 (ipykernel). The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and saving. The notebook contains two code cells. The first cell, labeled "In [9]:", uses the `df.selectExpr()` method to select columns and rename them. The second cell, labeled "In [12]:", uses the `df.select()` method with the `col()` function to select columns and alias the salary column as 'Amount'.

```
In [9]: data = df.selectExpr("Name as name","DOB","Gender","salary")
data.show()

+-----+-----+-----+
| name | DOB | Gender | salary |
+-----+-----+-----+
| Ram | 1991-04-01 | M | 3000 |
| Mike | 2000-05-19 | M | 4000 |
| Rohini | 1978-09-05 | M | 4000 |
| Maria | 1967-12-01 | F | 4000 |
| Jenis | 1980-02-17 | F | 1200 |
+-----+-----+-----+
```

```
In [12]: from pyspark.sql.functions import col

data = df.select(col("Name"),col("DOB"),
                 col("Gender"),
                 col("salary").alias('Amount'))
data.show()

+-----+-----+-----+
| Name | DOB | Gender | Amount |
+-----+-----+-----+
| Ram | 1991-04-01 | M | 3000 |
| Mike | 2000-05-19 | M | 4000 |
| Rohini | 1978-09-05 | M | 4000 |
| Maria | 1967-12-01 | F | 4000 |
| Jenis | 1980-02-17 | F | 1200 |
+-----+-----+-----+
```

The screenshot shows a Jupyter Notebook interface with a blue header bar. The title bar reads "Jupyter Selecting, Renaming, Filtering Data in a pandas DataFrame" followed by "Last Checkpoint: 6 hours ago (autosaved)". The top navigation bar includes "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". On the right, there are icons for a Python 3 (ipykernel) environment and a "Logout" button.

The main content area displays a pandas DataFrame with the following data:

Ram	1991-04-01	M	3000
Mike	2000-05-19	M	4000
Rohini	1978-09-05	M	4000
Maria	1967-12-01	F	4000
Jenis	1980-02-17	F	1200

Below the DataFrame, a code cell contains the following Python code:

```
In [13]: Data_list = ["Emp Name", "Date of Birth",
                    "Gender-m/f", "Paid salary"]

new_df = df.toDF(*Data_list)
new_df.show()
```

The output of the code cell shows the DataFrame with the specified column names:

Emp Name	Date of Birth	Gender-m/f	Paid salary
Ram	1991-04-01	M	3000
Mike	2000-05-19	M	4000
Rohini	1978-09-05	M	4000
Maria	1967-12-01	F	4000
Jenis	1980-02-17	F	1200

At the bottom of the interface, there is an input prompt "In []:" followed by an empty text box.

Page No.
 Date

* PySpark RDD operations.

- ① Transformations
 - takes an RDD as input and produces another RDD as output. original RDD remains same and thus are immutable.
- ② Actions
 - Applied on an RDD to produce single value. produces a non-RDD value.

Transformations are applied on RDD to give another RDD while Actions are performed on RDD to give a non-RDD value.

→ Actions in PySpark RDD.

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
```

- ① .collect() Action.
 - Returns list of all elements of RDD.
 - collect_rdd = sc.parallelize([1,2,3,4,5])
 - print(collect_rdd.collect())
- ② .count() Action.
 - returns no. of elements of RDD
 - count_rdd = sc.parallelize([1,2,3,4,5,6,7,8,9])
 - print(count_rdd.count())

③ .first() Action

Returns first element of RDD.

```
first-rdd = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
print(first-rdd.first())
```

④ .take() Action

.take(n) returns n number of elements of RDD.

```
take-rdd = sc.parallelize([1,2,3,4,5])
print(take-rdd.take())
```

⑤ .reduce() Action

takes two elements from RDD and operates.

```
reduce-rdd = sc.parallelize([1,3,4,6])
print(reduce-rdd.reduce(lambda x,y: x+y))
```

⑥ .saveAsTextFile() Action

```
save-rdd = sc.parallelize([1,2,3,4,5,6])
save-rdd.saveAsTextFile('file.txt')
```


* Transformations in PySpark RDD

(1) .map()

maps value to elements of RDD

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

(2) .filter()

filtering elements from RDD

```
filter_rdd = sc.parallelize([2,3,4,5,6,7])  
print(filter_rdd.filter(lambda x: x%2==0).collect())
```

(3) .union()

```
union_inp = sc.parallelize([2,4,5,6,7,8,9])  
union_rdd_1 = union_inp.filter(lambda x: x%2==0)  
union_rdd_2 = union_inp.filter(lambda x: x%3==0)  
print(union_rdd_1.union(union_rdd_2).collect())
```

(4) .flatMap()

```
Flatmap_rdd = sc.parallelize(["Hey there", "This is PySpark"])  
(flatmap_rdd.flatMap(lambda x: x.split(" ")).collect())
```