

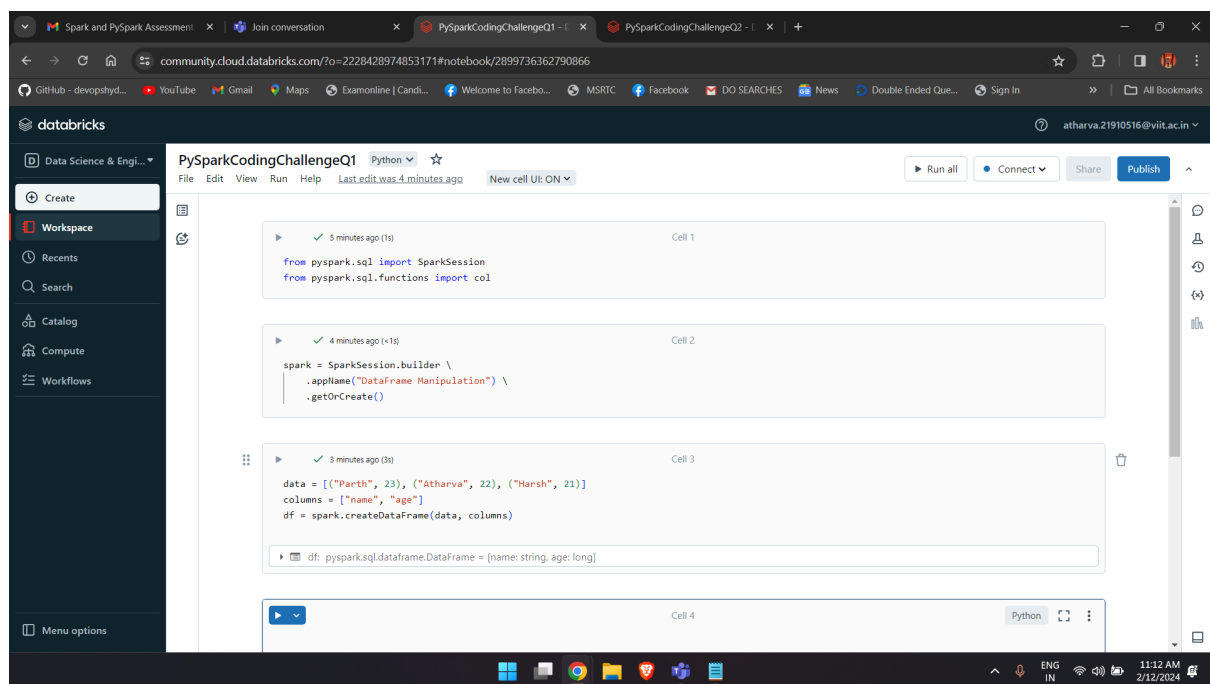
Name : Parth Nandedkar
Date : 12 Feb 2024
Topics : PySpark
Batch : Data Engineering Batch-1

Q1. Execute Manipulating, Dropping, Sorting, Aggregations, Joining, GroupBy DataFrames.

In Apache Spark, a DataFrame is a distributed collection of data organised into named columns, similar to a table in a relational database or a spreadsheet in a spreadsheet application.

We can perform different operations on it using Spark.

For now I am using DataBricks to execute the commands.



In the above screenshot I've imported a module `SparkSession` to generate a Spark Session so that further activities can be done.

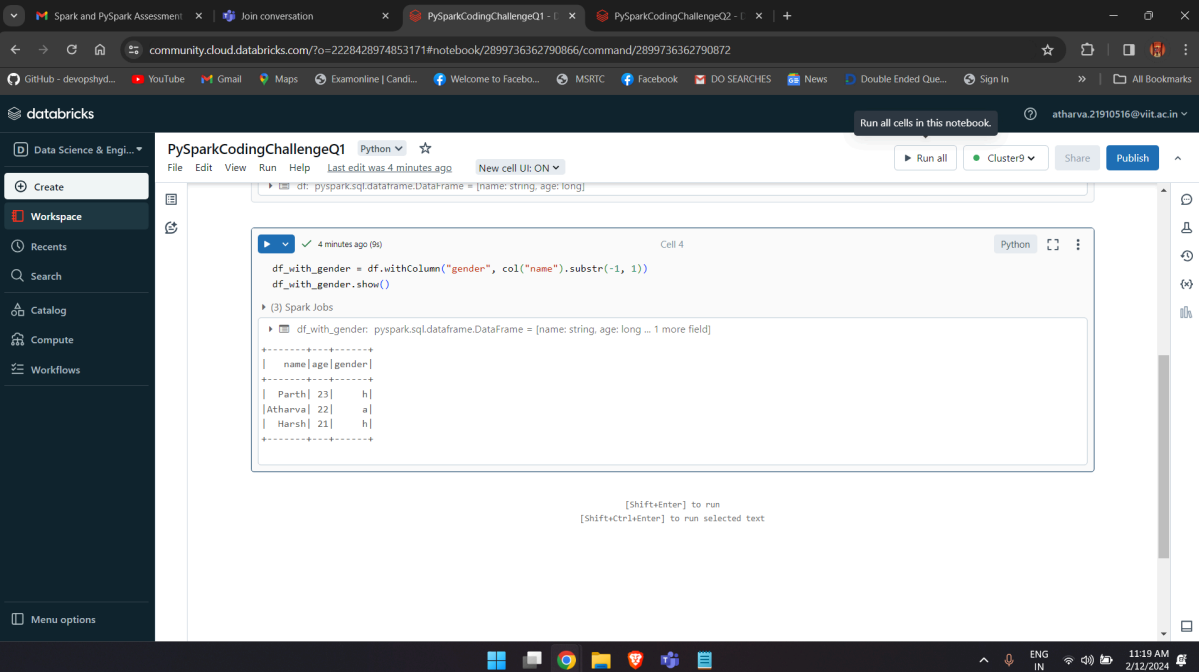
And for performing the rest of the applications the `sql.functions col` module is imported.

Data Frame named as **df** is created by using method `createDataFrame()`

List named as **data** is given as parameter and list named as **columns** is also given as a parameter to that function and the data frame is created.

Adding Column :

As there were only 2 columns **name** and **age** we are adding a new column named **gender**.



The screenshot shows a Databricks notebook interface. The notebook is titled 'PySparkCodingChallengeQ1'. The code in the cell is as follows:

```
df_with_gender = df.withColumn("gender", col("name").substr(-1, 1))
df_with_gender.show()
```

The output of the code is displayed below the code cell:

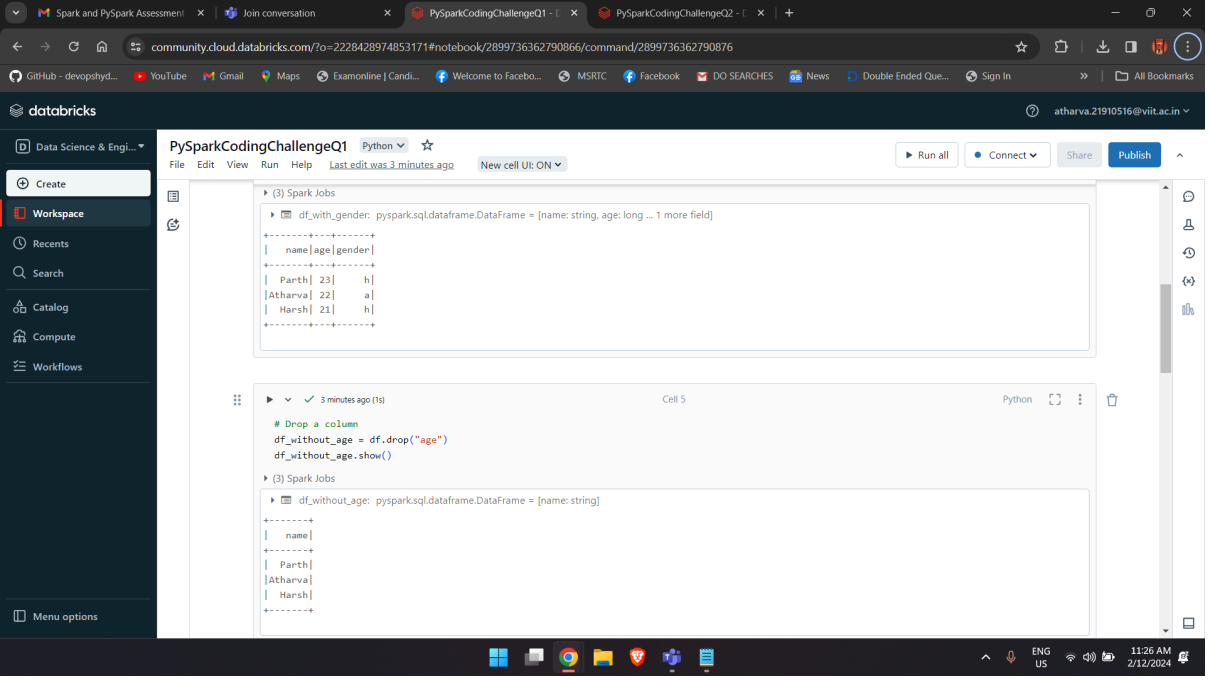
```
(3) Spark Jobs
└─ df_with_gender: pyspark.sql.dataframe.DataFrame = [name: string, age: long ... 1 more field]
┌── name | age | gender |
├───┬───┬───┤
│ Parth | 23 | h |
│ Atharva | 22 | a |
│ Harsh | 21 | h |
└───┬───┬───┘
```

The output shows a DataFrame with three columns: name, age, and gender. The gender column contains the last character of the name column.

By using the `withColumn()` method we created a new column **gender** and we used `show()` method to see if changes are reflected or not.

Dropping Column :

Columns of unnecessary data can be dropped as per the application need here to demonstrate we can drop age column.



The screenshot shows a Databricks workspace interface. The notebook is titled "PySparkCodingChallengeQ1" and is in Python mode. The first cell, labeled "Spark Jobs", contains the following code:

```
df_with_gender: pyspark.sql.dataframe.DataFrame = [name: string, age: long ... 1 more field]
| name | age | gender |
|-----|-----|-----|
| Parth | 23 | h |
| Atharva | 22 | a |
| Harsh | 21 | h |
```

The second cell, labeled "Cell 5", contains the following code:

```
# Drop a column
df_without_age = df.drop("age")
df_without_age.show()
```

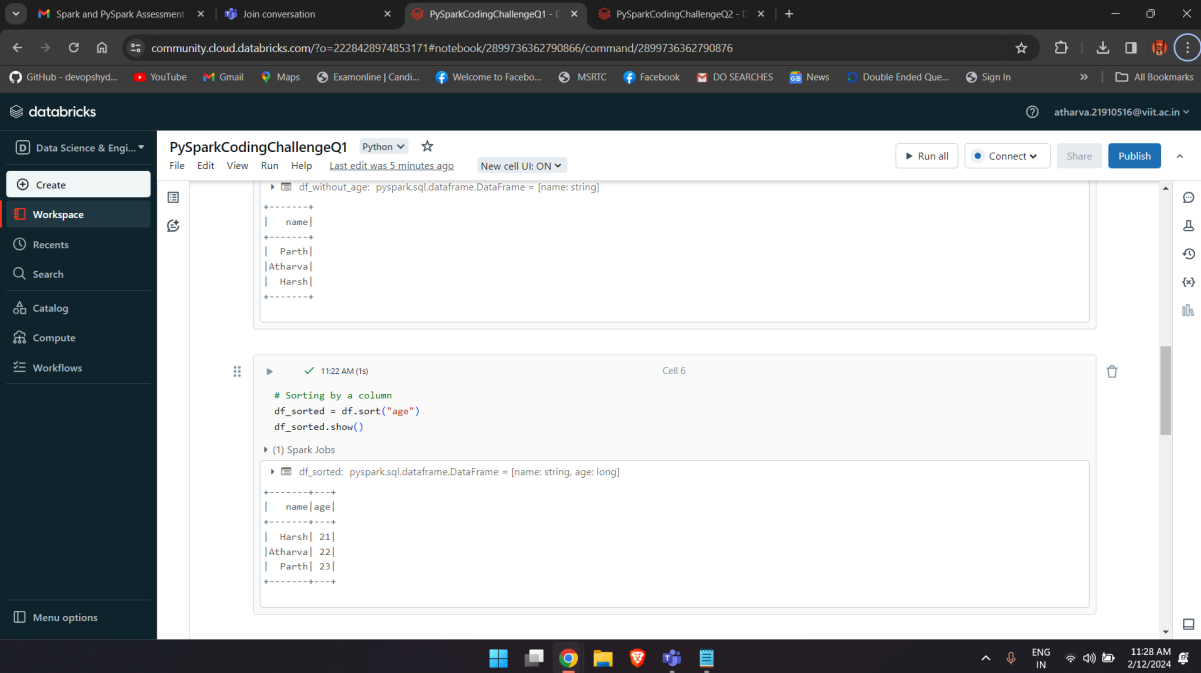
The output of the second cell shows the DataFrame after dropping the 'age' column:

```
df_without_age: pyspark.sql.dataframe.DataFrame = [name: string]
| name |
|-----|
| Parth |
| Atharva |
| Harsh |
```

By using **drop()** method we dropped **age** column and to see weather chanes are reflected or not we are using **show()** method.

Sorting Data :

According to application need we can sort data in the order by giving a particular column as reference as names cannot be sorted we can use age on the above example :



The screenshot shows a Databricks notebook interface. The top navigation bar includes the Databricks logo, a user profile, and various tool icons. The left sidebar contains a 'Workspace' tab and a 'Catalog' section. The main area displays a notebook titled 'PySparkCodingChallengeQ1'. It contains two code cells. The first cell defines a DataFrame with names: Parth, Atharva, and Harsh. The second cell sorts this DataFrame by the 'age' column and displays the result. The output shows the data sorted by age: Harsh (21), Atharva (22), and Parth (23).

```
df_without_age: pyspark.sql.dataframe.DataFrame = [name: string]
+-----+
| name |
+-----+
| Parth |
| Atharva |
| Harsh |
+-----+

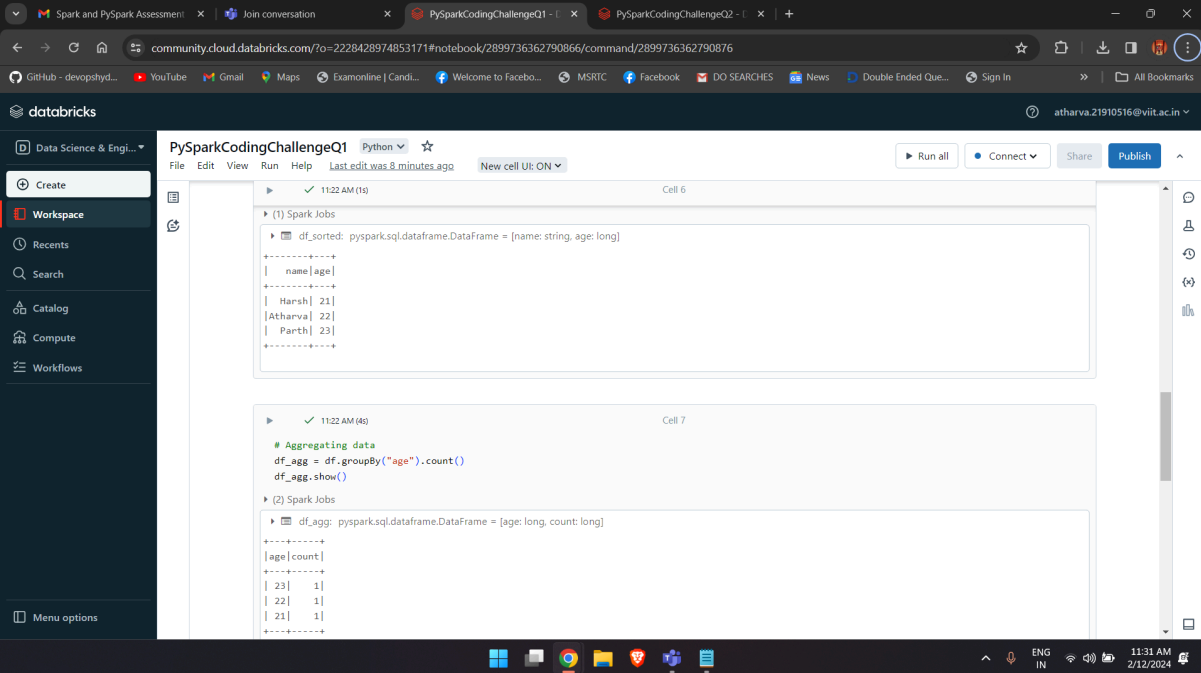
# Sorting by a column
df_sorted = df.sort("age")
df_sorted.show()

(1) Spark Jobs
df_sorted: pyspark.sql.dataframe.DataFrame = [name: string, age: long]
+-----+
| name | age |
+-----+
| Harsh | 21 |
| Atharva | 22 |
| Parth | 23 |
+-----+
```

According to age the data is sorted as **Harsh has age 21, Atharva 22 and Parth 23** accordingly the data is sorted as we used **sort()** method on data object.

Aggregating Data And Group By :

According to data we can distribute data in particular sets by using aggregations here on count method we are getting data to show how many people there are of same age .



The screenshot shows a Databricks notebook interface with two cells. Cell 6 displays a DataFrame with columns 'name' and 'age'. Cell 7 shows the aggregation of data by age using the 'count()' method.

```
df_sorted: pyspark.sql.dataframe.DataFrame = [name: string, age: long]
+-----+
| name | age |
+-----+
| Harsh | 21 |
| Atharva | 22 |
| Parth | 23 |
+-----+
```

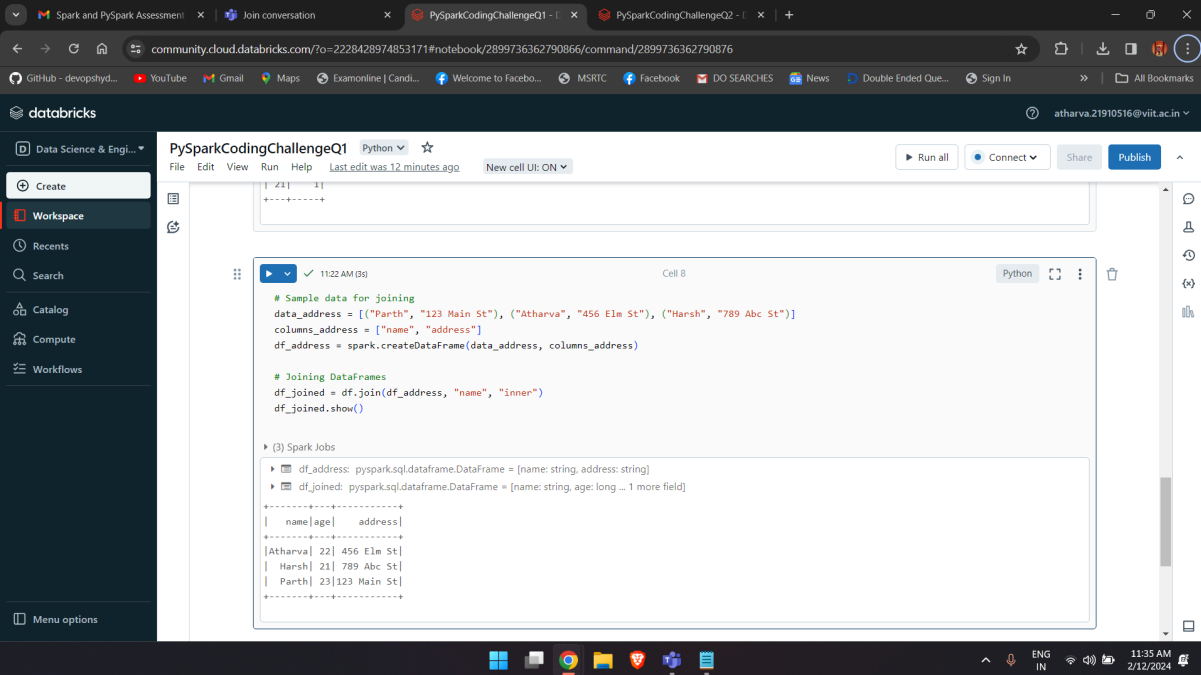
```
# Aggregating data
df_agg = df.groupBy("age").count()
df_agg.show()
```

```
df_agg: pyspark.sql.dataframe.DataFrame = [age: long, count: long]
+----+-----+
| age | count |
+----+-----+
| 23 | 1 |
| 22 | 1 |
| 21 | 1 |
+----+-----+
```

As there are 3 age groups 21,22,23 and each contains 1 person thats why the output shows 21 -1, 22-1, 23 -1.
Which is the right output.

Using Joins on data :

Joins can be used to get desired data from two different datasets combined based on the common condition so that the operation performing could be easy and data storing in different datasets to achieve normalisation can be achieved .



```
# Sample data for joining
data_address = [{"Parth", "123 Main St"}, {"Atharva", "456 Elm St"}, {"Harsh", "789 Abc St"}]
columns_address = ["name", "address"]
df_address = spark.createDataFrame(data_address, columns_address)

# Joining DataFrames
df_joined = df_address.join(df_address, "name", "inner")
df_joined.show()
```

(3) Spark Jobs

name	age	address
Atharva	22	456 Elm St
Harsh	21	789 Abc St
Parth	23	123 Main St

Here in the above example we added one more data frame which includes the same names but the second column is of Address.

As Names are common by using **join()** method we can connect df1 and df2.

As all the 3 names are common all the names , ages and address accordingly are displayed.

Here we used **inner** join as we need to put the right address to the right person so only common values will be joined.

