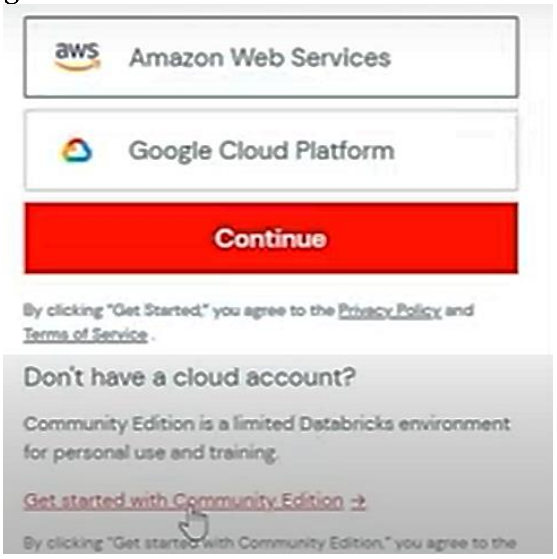
PRACTICAL-2

Sign-up process for the platform <u>https://www.databricks.com/</u>

STEPS:

1. Go to https://www.databricks.com/ and Click on "Login"
2. Click on "Sign-up" and fill-up the forms and click "Continue". Another page will open.
3. Choose a Cloud provider. Don't select the cloud provider. Below you can see the "Don't have a cloud account" under this click on "Get started with Community edition" See below figure



By clicking "Get Started," you agree to the Privacy Policy and Terms of Service.

Don't have a cloud account?

Community Edition is a limited Databricks environment for personal use and training

Get started with Community Edition →

By clicking "Get started with Community Edition," you agree to the

4. When you follow the step-3, you will receive one email to start your trial from databricks. Click on that. It will redirect on community edition databricks site and you can see the reset password GUI. Reset the password first.
5. Next time onwards you have to open the site:
   https://community.cloud.databricks.com/login.html

Learning How to upload Data, Cluster creation, and write the Basic script of Apache Spark.

Open https://community.cloud.databricks.com/login.html

STEP-1

Create Cluster and connect it.

STEP-2

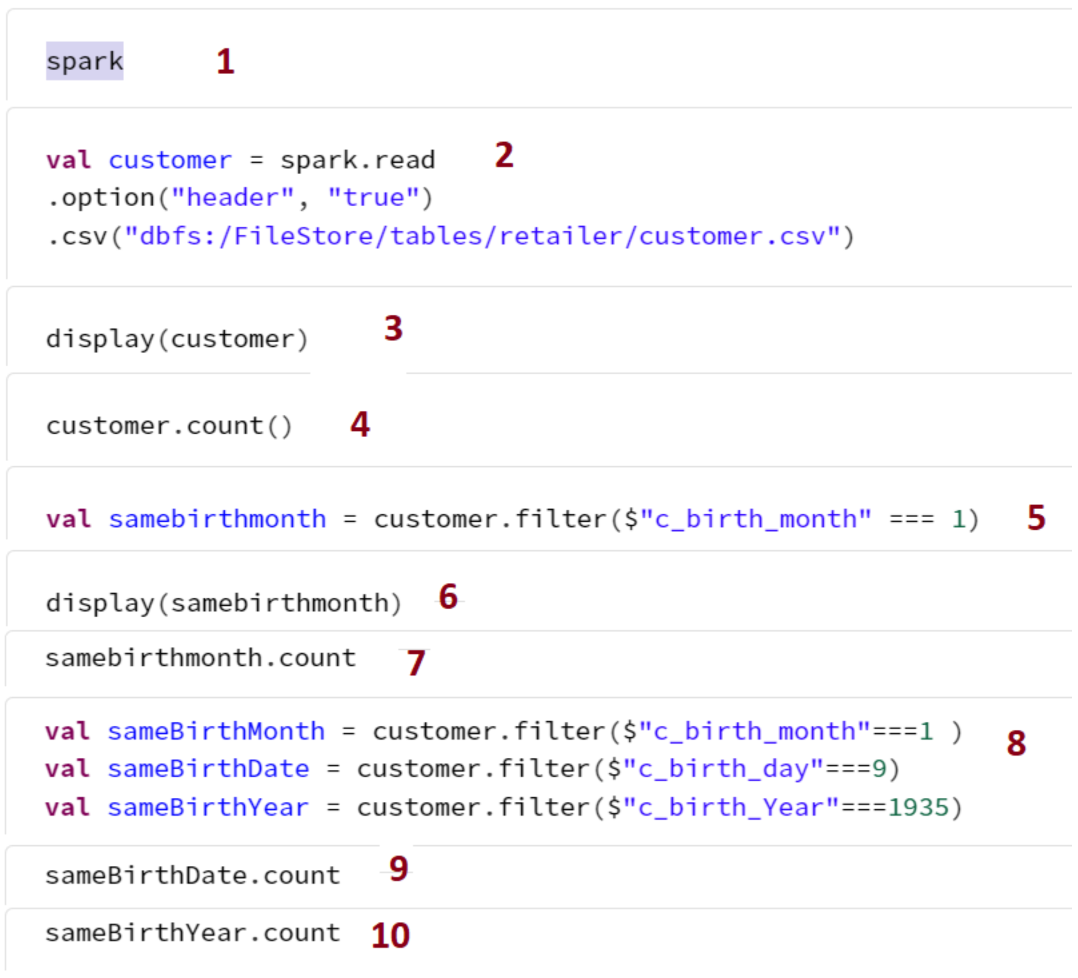
Upload the customer.csv file. Check DBFS. Keep remember the file path.

STEP-3

Open notebook and change the language "Scala/Spark"

STEP-4

Perform Following:



```
spark        1
```

```
val customer = spark.read      2
.option("header", "true")
.csv("dbfs:/FileStore/tables/retailer/customer.csv")
```

```
display(customer)      3
```

```
customer.count()     4
```

```
val samebirthmonth = customer.filter($"c_birth_month" === 1)     5
```

```
display(samebirthmonth)     6
```

```
samebirthmonth.count     7
```

```
val sameBirthMonth = customer.filter($"c_birth_month"===1 )     8
val sameBirthDate = customer.filter($"c_birth_day"===9)
val sameBirthYear = customer.filter($"c_birth_Year"===1935)
```

```
sameBirthDate.count     9
```

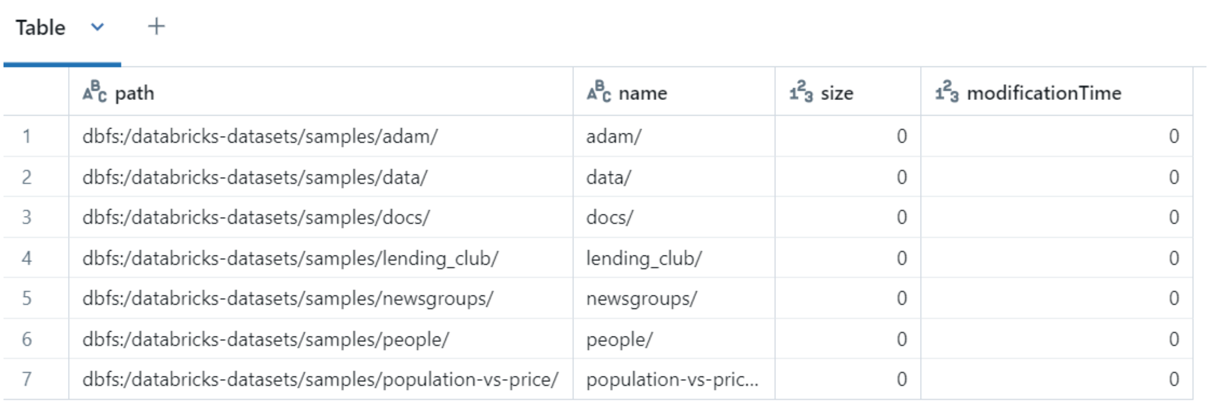
```
sameBirthYear.count     10
```

Working with sample dataset-databricks:

Run following in notebook

```
import pyspark
from pyspark.sql.functions import col
from pyspark.sql.types import IntegerType, FloatType
display(dbutils.fs.ls("/databricks-datasets/samples/"))
```

output:



Table

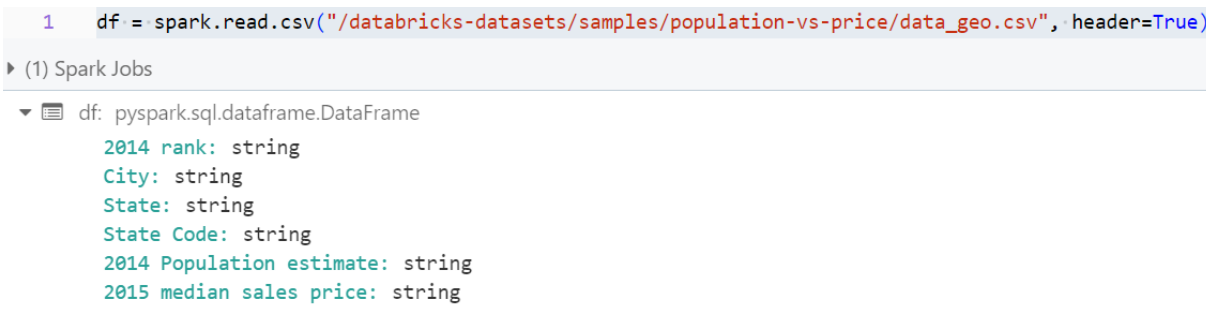
	path	name	size	modificationTime
1	dbfs:/databricks-datasets/samples/adam/	adam/	0	0
2	dbfs:/databricks-datasets/samples/data/	data/	0	0
3	dbfs:/databricks-datasets/samples/docs/	docs/	0	0
4	dbfs:/databricks-datasets/samples/lending_club/	lending_club/	0	0
5	dbfs:/databricks-datasets/samples/newsgroups/	newsgroups/	0	0
6	dbfs:/databricks-datasets/samples/people/	people/	0	0
7	dbfs:/databricks-datasets/samples/population-vs-price/	population-vs-pric...	0	0

We are going to be doing some basic exploration in the "population-vs-price/" sample dataset. So let's go ahead and define a variable called 'df' that will reference the dataframe in our notebook.

Code:

```
df = spark.read.csv("/databricks-datasets/samples/population-vs-
price/data_geo.csv", header=True)
```

output:



```
df = spark.read.csv("/databricks-datasets/samples/population-vs-price/data_geo.csv", header=True)
```

(1) Spark Jobs

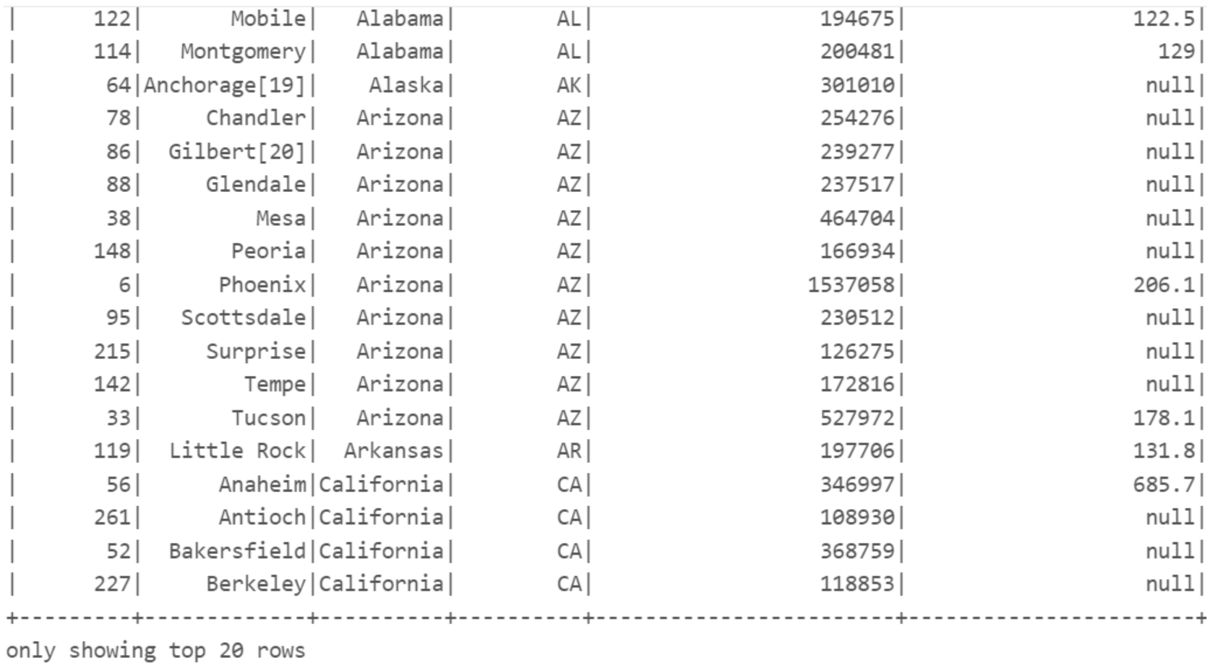
df: pyspark.sql.dataframe.DataFrame

```
2014 rank: string
City: string
State: string
State Code: string
2014 Population estimate: string
2015 median sales price: string
```

To show the data in table format use show() method (show default first 20 rows)

df.show()

output:

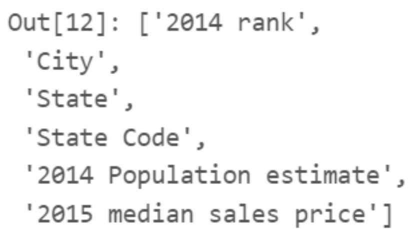


```
|       122|         Mobile|    Alabama|         AL|                   194675|                   122.5|
|       114|     Montgomery|    Alabama|         AL|                   200481|                     129|
|        64|Anchorage[19]|     Alaska|         AK|                   301010|                    null|
|        78|       Chandler|    Arizona|         AZ|                   254276|                    null|
|        86|    Gilbert[20]|    Arizona|         AZ|                   239277|                    null|
|        88|       Glendale|    Arizona|         AZ|                   237517|                    null|
|        38|           Mesa|    Arizona|         AZ|                   464704|                    null|
|       148|         Peoria|    Arizona|         AZ|                   166934|                    null|
|         6|        Phoenix|    Arizona|         AZ|                  1537058|                   206.1|
|        95|     Scottsdale|    Arizona|         AZ|                   230512|                    null|
|       215|       Surprise|    Arizona|         AZ|                   126275|                    null|
|       142|          Tempe|    Arizona|         AZ|                   172816|                    null|
|        33|         Tucson|    Arizona|         AZ|                   527972|                   178.1|
|       119|    Little Rock|   Arkansas|         AR|                   197706|                   131.8|
|        56|        Anaheim|California|         CA|                   346997|                   685.7|
|       261|        Antioch|California|         CA|                   108930|                    null|
|        52|    Bakersfield|California|         CA|                   368759|                    null|
|       227|       Berkeley|California|         CA|                   118853|                    null|
+----------+---------------+-----------+-----------+-------------------------+------------------------+
only showing top 20 rows
```

See the column names use:

df.columns

output:



```
Out[12]: ['2014 rank',
 'City',
 'State',
 'State Code',
 '2014 Population estimate',
 '2015 median sales price']
```

Note: Notice that many of the column names contain spaces; this is not ideal for us if we want to implement SQL to create queries from this dataframe. To change the column names, we can implement the ".withColumnRenamed()" method:

df.withColumnRenamed('2014 rank', '2014_rank')

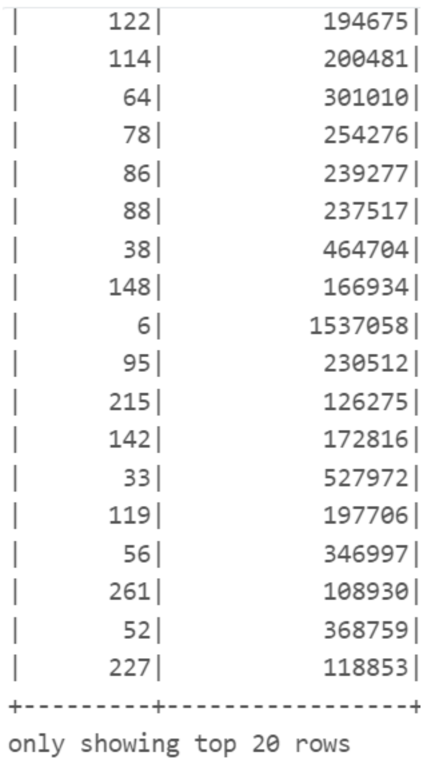
NOTE: Note that we must create a new variable (df2) to hold these changes in a new dataframe. If we were to simply "df.withColumnRenamed…", (as we did above) it would only be a temporary change — there is no "inplace=True" parameter. We can also chain these all at once for each column name we want to be changed:

```
df = spark.read.csv("/databricks-datasets/samples/population-vs-
price/data_geo.csv", header=True)
df2 = df.withColumnRenamed('2014 rank', '2014_rank')\
.withColumnRenamed('State Code', 'state_code')\
.withColumnRenamed('2014 Population estimate', '2014_pop_estimate')\
.withColumnRenamed('2015 median sales price', '2015_median_sales_price')
```

If we want to view selected columns within df2 to view, we can say:

```
df2.select(['2014_rank', '2014_pop_estimate']).show()
```

output:



```
|      122|           194675|
|      114|           200481|
|       64|           301010|
|       78|           254276|
|       86|           239277|
|       88|           237517|
|       38|           464704|
|      148|           166934|
|        6|          1537058|
|       95|           230512|
|      215|           126275|
|      142|           172816|
|       33|           527972|
|      119|           197706|
|       56|           346997|
|      261|           108930|
|       52|           368759|
|      227|           118853|
+---------+-----------------+
only showing top 20 rows
```

This would show us only the values of the first 20 rows for the selected columns. Now let's view the types of values within each column. A way we can do this is by using the method ".printSchema()" on our df2 variable.

```
df2.printSchema()
```

output:

```
root
 |-- 2014_rank: string (nullable = true)
 |-- City: string (nullable = true)
 |-- State: string (nullable = true)
 |-- state_code: string (nullable = true)
 |-- 2014_pop_estimate: string (nullable = true)
 |-- 2015_median_sales_price: string (nullable = true)
```

We can run a SQL query! It is extremely simple to run a SQL query in PySpark. Let's run a basic query to see how it works:

```
df2.createOrReplaceTempView('pop_price')
results = spark.sql("SELECT * FROM pop_price")
results.show()
```

Note: For SQL to work correctly, we need to make sure df3 has a table name. To do this, we simply say: df2.createOrReplaceTempView('pop_price')

Output:

```
|      122|          Mobile|   Alabama|         AL|           194675|                  122.5|
|      114|      Montgomery|   Alabama|         AL|           200481|                    129|
|       64|   Anchorage[19]|    Alaska|         AK|           301010|                   null|
|       78|        Chandler|   Arizona|         AZ|           254276|                   null|
|       86|     Gilbert[20]|   Arizona|         AZ|           239277|                   null|
|       88|        Glendale|   Arizona|         AZ|           237517|                   null|
|       38|            Mesa|   Arizona|         AZ|           464704|                   null|
|      148|          Peoria|   Arizona|         AZ|           166934|                   null|
|        6|         Phoenix|   Arizona|         AZ|          1537058|                  206.1|
|       95|      Scottsdale|   Arizona|         AZ|           230512|                   null|
|      215|        Surprise|   Arizona|         AZ|           126275|                   null|
|      142|           Tempe|   Arizona|         AZ|           172816|                   null|
|       33|          Tucson|   Arizona|         AZ|           527972|                  178.1|
|      119|     Little Rock|  Arkansas|         AR|           197706|                  131.8|
|       56|         Anaheim|California|         CA|           346997|                  685.7|
|      261|         Antioch|California|         CA|           108930|                   null|
|       52|     Bakersfield|California|         CA|           368759|                   null|
|      227|        Berkeley|California|         CA|           118853|                   null|
+---------+----------------+----------+-----------+-----------------+-----------------------+
only showing top 20 rows
```