Introducing Spark

Spark Basics

Apache Spark is an open-source distributed general-purpose cluster-computing framework. You want to be using Spark if you are at a point where it does not makes sense to fit all your data on RAM and no longer makes sense to fit all your data on a local machine. On a high level, it is a unified analytics engine for Big Data processing, with built-in modules for streaming, SQL, machine learning, and graph processing. Spark is one of the latest technologies that is being used to quickly and easily handle Big Data and can interact with language shells like Scala, Python, and R.

What is DataBricks?

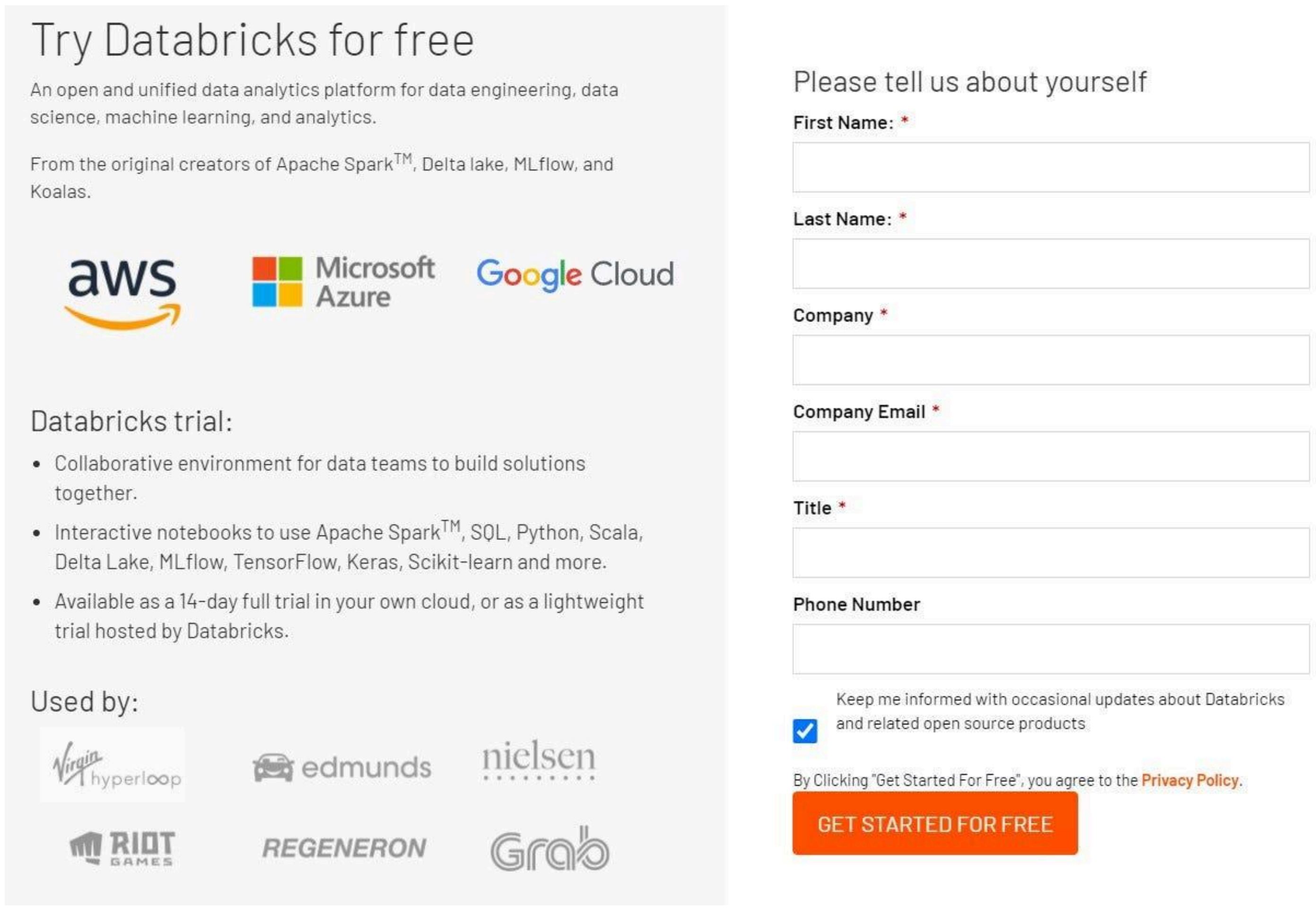
Press enter or click to view image in full size

Databricks is an industry-leading, cloud-based data engineering tool used for processing, exploring, and transforming Big Data and using the data with machine learning models. It is a tool that provides a fast and simple way to set up and use a cluster to analyze and model off of Big data. In a nutshell, it is the platform that will allow us to use PySpark (The collaboration of Apache Spark and Python) to work with Big Data. The version we will be using in this blog will be the community edition (completely free to use). Without further ado…

Let's Begin!

1. The first step we must do to use Databricks is:
2. Create an account. **You can visit https://databricks.com/try-databricks**, or click this link to go create an account — if you already have one, then feel free to skip this step!

Press enter or click to view image in full size


Try Databricks for free

An open and unified data analytics platform for data engineering, data science, machine learning, and analytics.

From the original creators of Apache Spark™, Delta lake, MLflow, and Koalas.

Databricks trial:

- Collaborative environment for data teams to build solutions together.
- Interactive notebooks to use Apache Spark™, SQL, Python, Scala, Delta Lake, MLflow, TensorFlow, Keras, Scikit-learn and more.
- Available as a 14-day full trial in your own cloud, or as a lightweight trial hosted by Databricks.

Used by:

Please tell us about yourself

First Name: *

Last Name: *

Company *

Company Email *

Title *

Phone Number

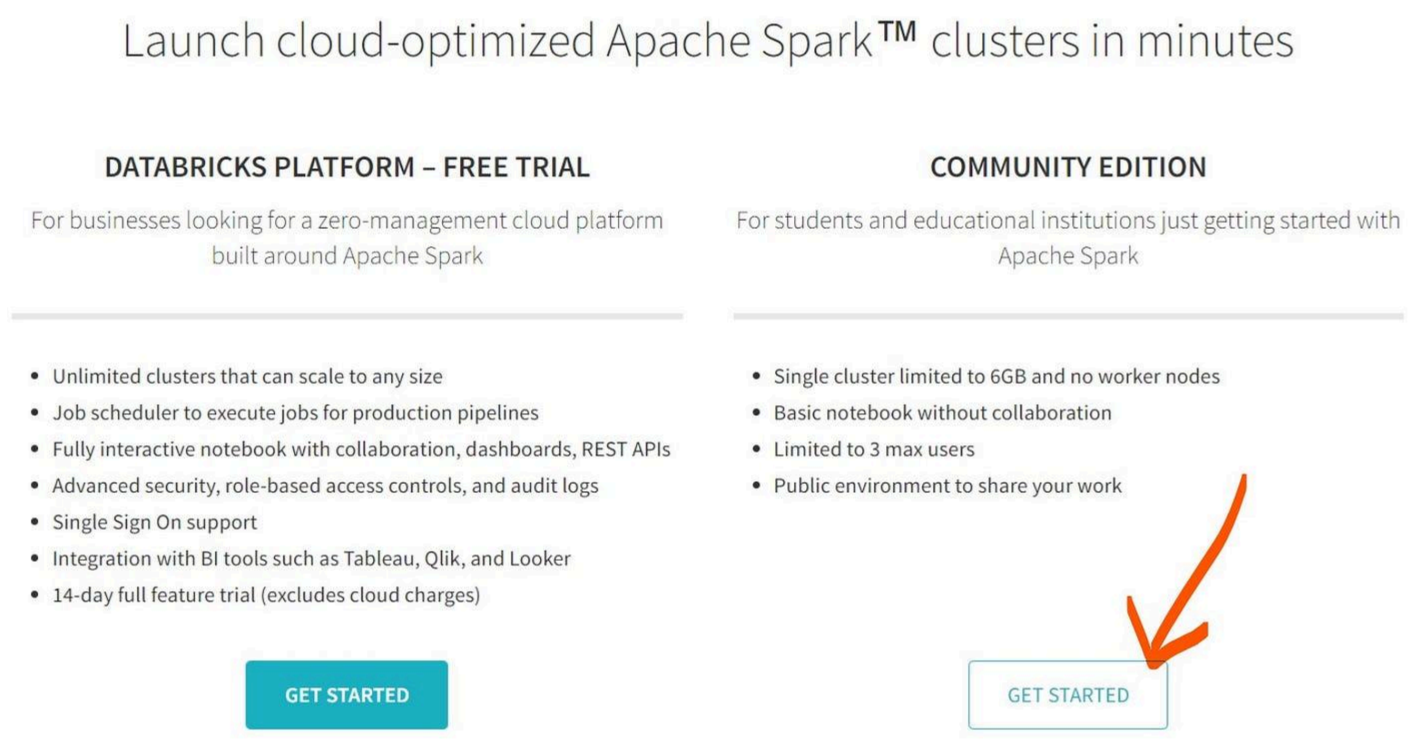
[✓] Keep me informed with occasional updates about Databricks and related open source products

By Clicking "Get Started For Free", you agree to the Privacy Policy.

Creating an account

Once you have entered in your information, it will ask you to select which version of Databricks you want to work with along with email address verification. I **highly recommend using the Community Edition** because the Databricks Community Edition is free of charge. You do not pay for the platform nor do you incur AWS costs.

Press enter or click to view image in full size



Launch cloud-optimized Apache Spark™ clusters in minutes

DATABRICKS PLATFORM – FREE TRIAL

For businesses looking for a zero-management cloud platform built around Apache Spark

- Unlimited clusters that can scale to any size
- Job scheduler to execute jobs for production pipelines
- Fully interactive notebook with collaboration, dashboards, REST APIs
- Advanced security, role-based access controls, and audit logs
- Single Sign On support
- Integration with BI tools such as Tableau, Qlik, and Looker
- 14-day full feature trial (excludes cloud charges)

GET STARTED

COMMUNITY EDITION

For students and educational institutions just getting started with Apache Spark

- Single cluster limited to 6GB and no worker nodes
- Basic notebook without collaboration
- Limited to 3 max users
- Public environment to share your work

GET STARTED

Free Trial versus Community Edition Selection

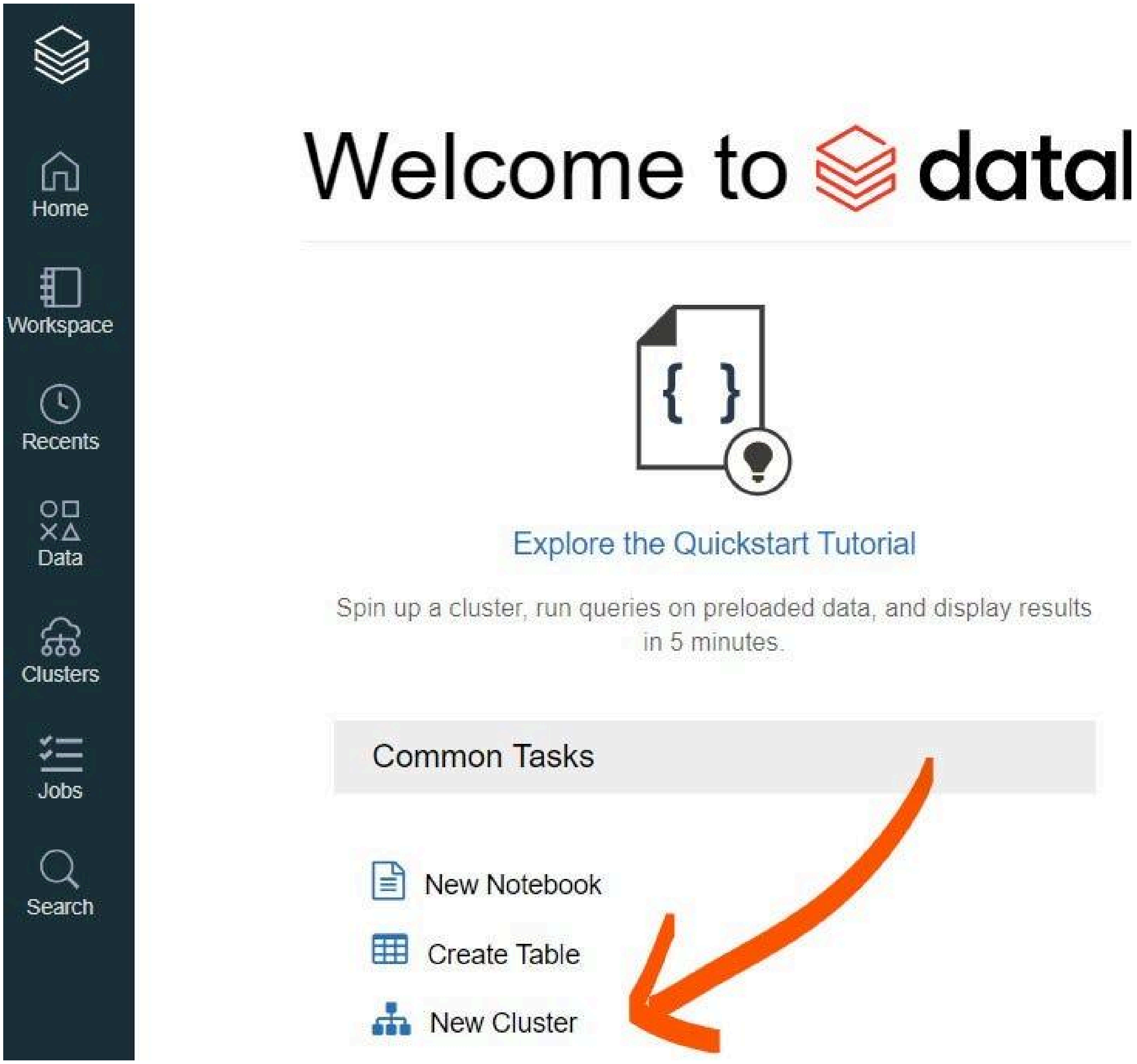
Once we have successfully created an account and chosen our preferred edition of the platform, the next step for us is to create a cluster.

Creating A Cluster

A Databricks cluster is a set of computation resources and configurations on which you can run data engineering, data science, and data analytics workloads, such as production ETL pipelines, streaming analytics, ad-hoc analytics, and machine learning.

To create our first cluster, click on the “New Cluster” button:

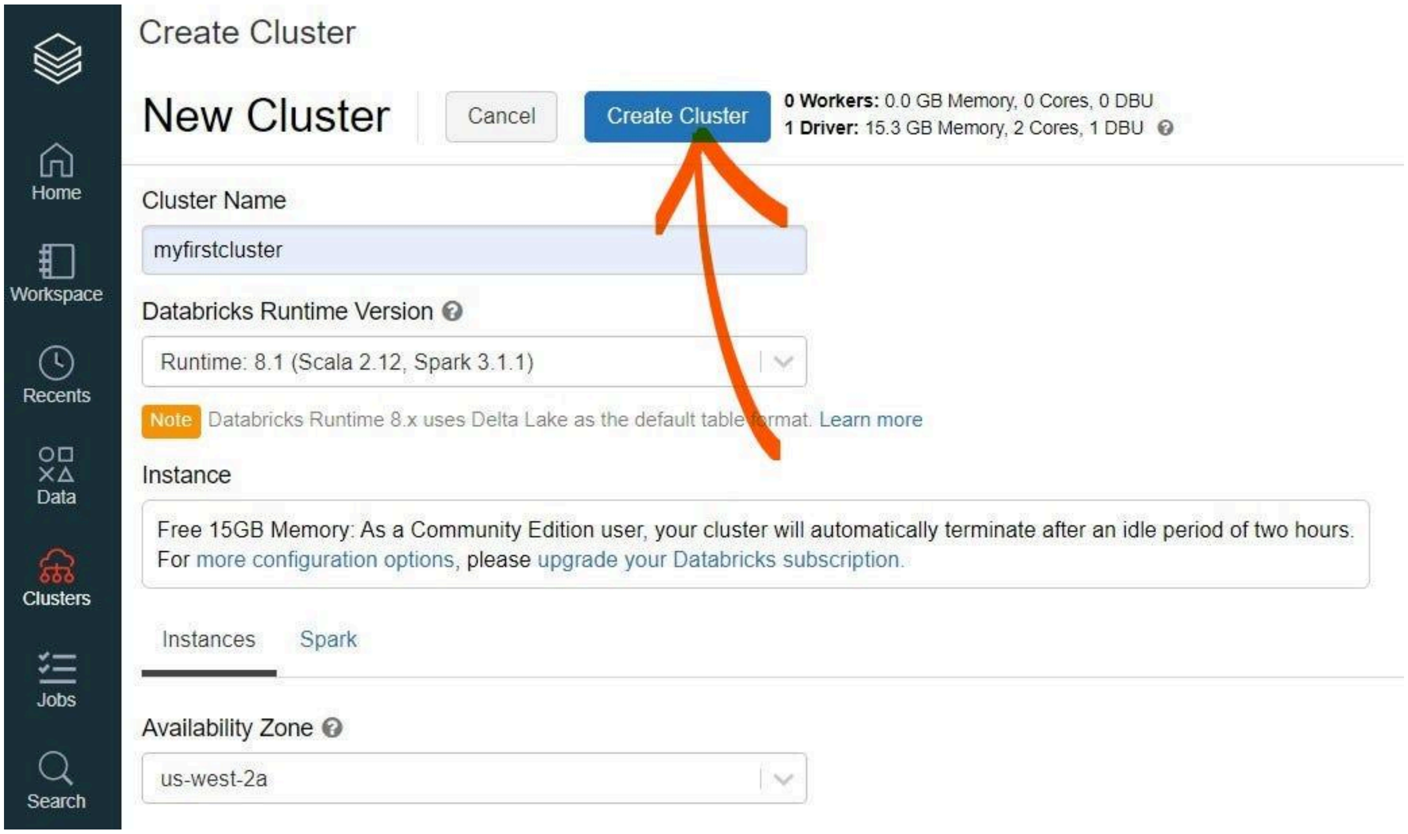
Press enter or click to view image in full size


Home
Workspace
Recents
Data
Clusters
Jobs
Search
Welcome to datal
Explore the Quickstart Tutorial
Spin up a cluster, run queries on preloaded data, and display results in 5 minutes.
Common Tasks
New Notebook
Create Table
New Cluster

“New Cluster” option below Common Tasks list

This will take us to a new page where we define the new cluster. Feel free to name the cluster whatever you like — I will name the new cluster “myfirstcluster”. I will leave the rest of the options alone and click on the “create cluster” button:

Press enter or click to view image in full size



Create Cluster

New Cluster

0 Workers: 0.0 GB Memory, 0 Cores, 0 DBU
1 Driver: 15.3 GB Memory, 2 Cores, 1 DBU

Cluster Name: myfirstcluster

Databricks Runtime Version: Runtime: 8.1 (Scala 2.12, Spark 3.1.1)

Note Databricks Runtime 8.x uses Delta Lake as the default table format. Learn more

Instance

Free 15GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. For more configuration options, please upgrade your Databricks subscription.

Instances Spark

Availability Zone: us-west-2a

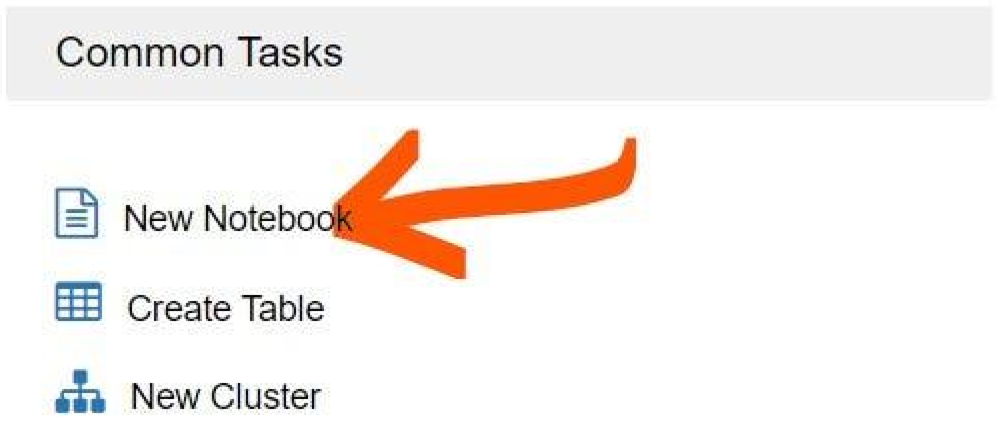
Defining and Creating Our Cluster

Note that creating the cluster may take a second to run, so please be patient. In the event the cluster fails to instantiate, you may try changing the availability zone in the lower option. If you are using the Community Edition, the cluster will terminate after 120 minutes of inactivity — and you will not be able to restart the cluster once it has been terminated. A way to sidestep this problem is to create a clone of the terminated cluster each time, or you can also create a new cluster. In my opinion, it is a small price to pay for a free edition of Databricks.

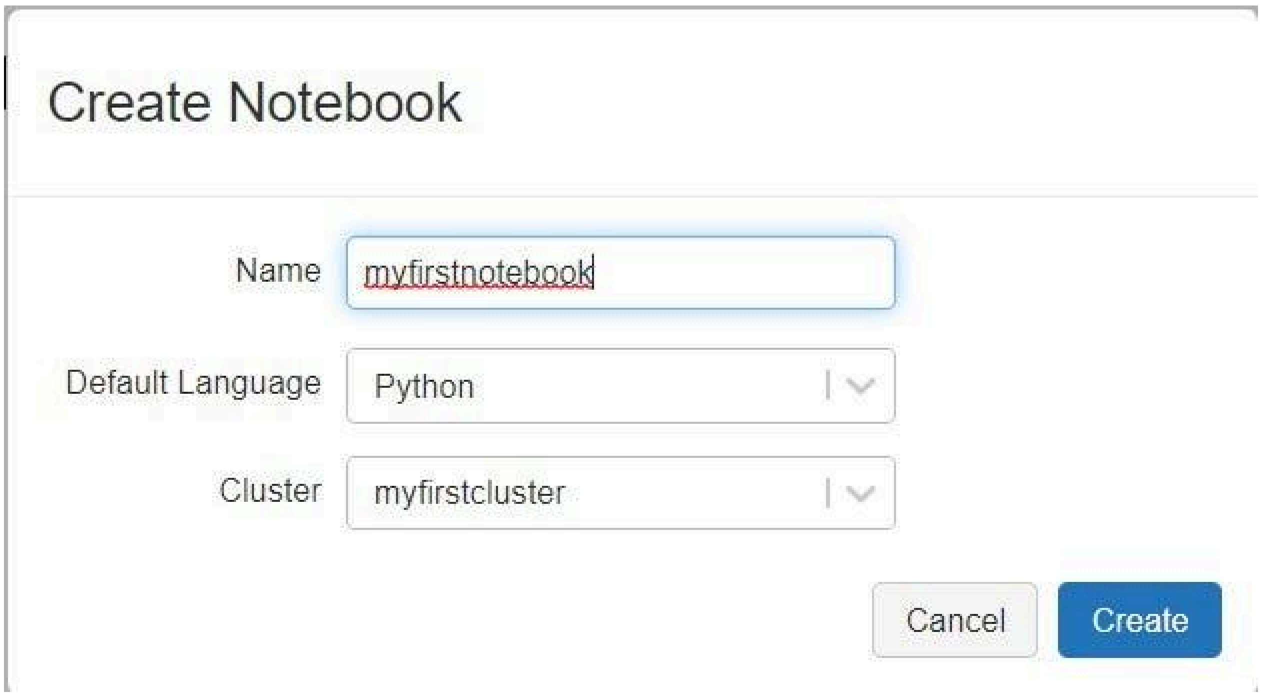
Creating a New Notebook

Once we have our cluster up and running, we can now create a new notebook!

Simply click on the top left Databricks icon and click on “New Notebook” underneath the “Common Tasks” list:



All we need to do to instantiate the notebook is to give it a name (I gave mine the name “myfirstnotebook”), select the language (I chose Python), and select the active cluster we created. Now, all we need to do is hit the “Create” button:


Create Notebook

Name: myfirstnotebook

Default Language: Python

Cluster: myfirstcluster

Creating a new notebook

Selecting A Sample Dataset

Now that our notebook has been created and successfully attached to our cluster, we can finally begin to have some fun! The first thing we want to do in this notebook is import the necessary libraries. So let's begin with importing PySpark:

```
import pyspark

from pyspark.sql.functions import col

from pyspark.sql.types import IntegerType, FloatType
```

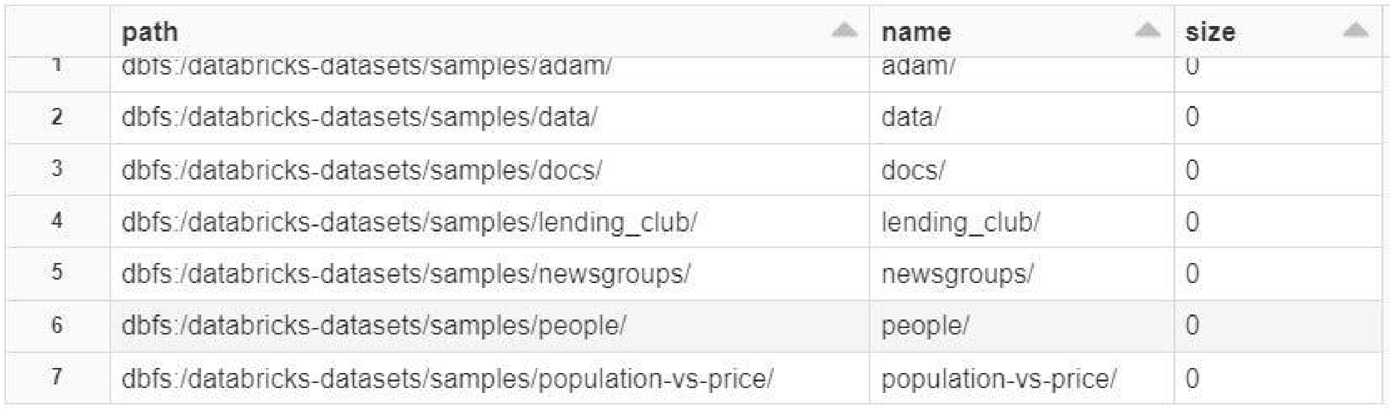
For this notebook, we will not be uploading any datasets into our Notebook.

Instead, we will be selecting a sample dataset that Databricks provides for us to mess around with. We can view the different sample datasets by typing in:

```
# A list of folders containing sample datasets we can use
```

```
display(dbutils.fs.ls("/databricks-datasets/samples/")
```

Press enter or click to view image in full size



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

A list of sample datasets Databricks provides

Exploring the Data

In this blog, 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.

```
# Loading in a sample table into the dataframe
```

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

Instead of calling df.head() to view the first 5 rows, we instead will call df.show() to view the dataframe. By default, the .show() method displays the top 20 rows of a dataframe.

```
# To view the first 20 rows of the df
```

```
df.show()
```

```
# OR we can add an integer into the parentheses to view a specific
```

```
# number of rows
```

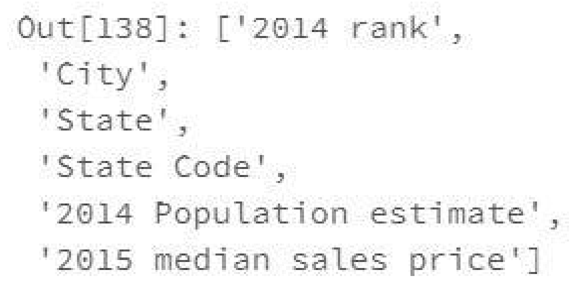
```
df.show(5)
```

To view the column names within the dataframe, we can call "*df.columns*" — this will return a list of the column names within the dataframe:

```
# Viewing the column names
```

```
df.columns
```

Press enter or click to view image in full size



```
Out[138]: ['2014 rank',
 'City',
 'State',
 'State Code',
 '2014 Population estimate',
 '2015 median sales price']
```

A list of the column names

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:

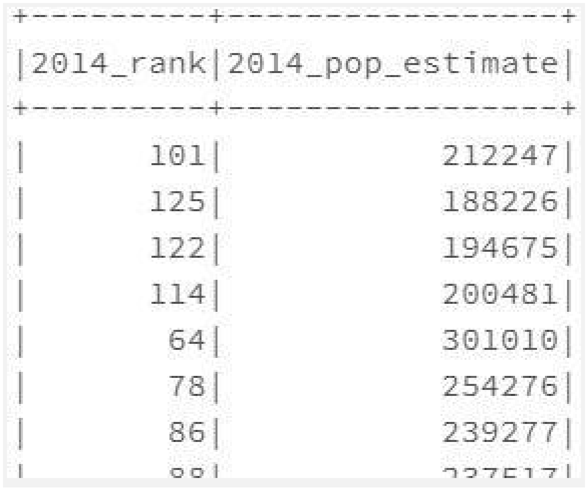
```
df.withColumnRenamed('2014 rank', '2014_rank')
```

Note 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:

```
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')
```

Great! if we want to view selected columns within df2 to view, we can say:

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



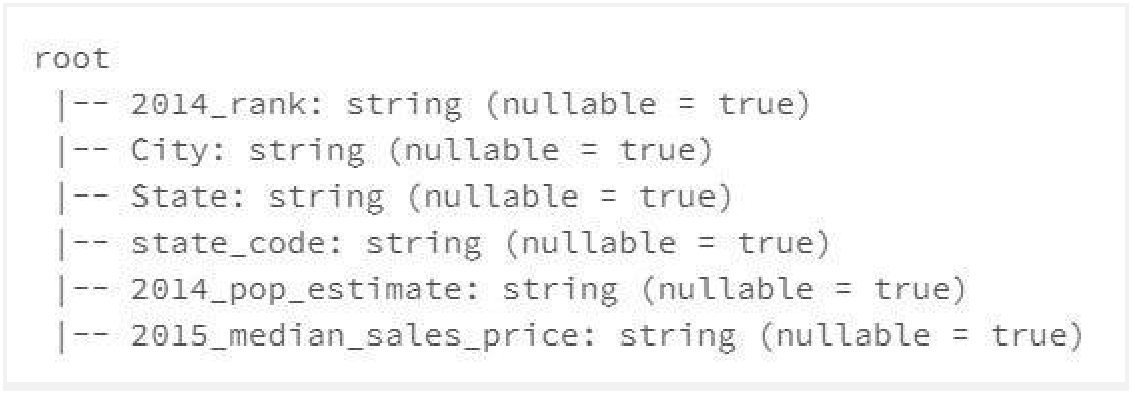
```
+---------+-----------------+
|2014_rank|2014_pop_estimate|
+---------+-----------------+
|      101|           212247|
|      125|           188226|
|      122|           194675|
|      114|           200481|
|       64|           301010|
|       78|           254276|
|       86|           239277|
```

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.

```
# Printing out the schema of the dataframe
```

```
df2.printSchema()
```



```
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)
```

Viewing the schema of df2

Oh no, we notice that all of our columns contain string values — even the columns that are supposed to contain numerical values! A way we can manually adjust the type of values within a column is somewhat similar to how we handled adjusting the names of the columns: using the "*.withColumn()*" method and chaining on the "*.cast()*" method. Before we initiate this on multiple columns at once, let's break down one example:

```
df2.withColumn("2014_rank", col("2014_rank").cast(IntegerType()))
```

In the above example, we are saying:

- With this selected column: "2014_rank"

- Make a new column called “2014_rank” (replacing the old column)
- This new column will contain the old column’s values, but we will recast them as integer types

Like the previous method we used, “.withColumnRenamed”, the change is only temporary unless we create a new variable to hold the changed dataframe. Just like the previous method, we can chain multiple columns at once (it looks a little messy in the code block below):

```
df3 = df2.withColumn(“2014_rank”,col(“2014_rank”).cast(IntegerType()))\

.withColumn(“2014_pop_estimate”,
col(“2014_pop_estimate”).cast(IntegerType()))\

.withColumn(‘2015_median_sales_price’,
col(‘2015_median_sales_price’).cast(FloatType()))
```

Using SQL Syntax

Before we end this tutorial, let’s finally run some SQL querying on our dataframe! For SQL to work correctly, we need to make sure df3 has a table name. To do this, we simply say:

```
# Giving our df3 the table name 'pop_price'
```

```
df3.createOrReplaceTempView(‘pop_price’)
```

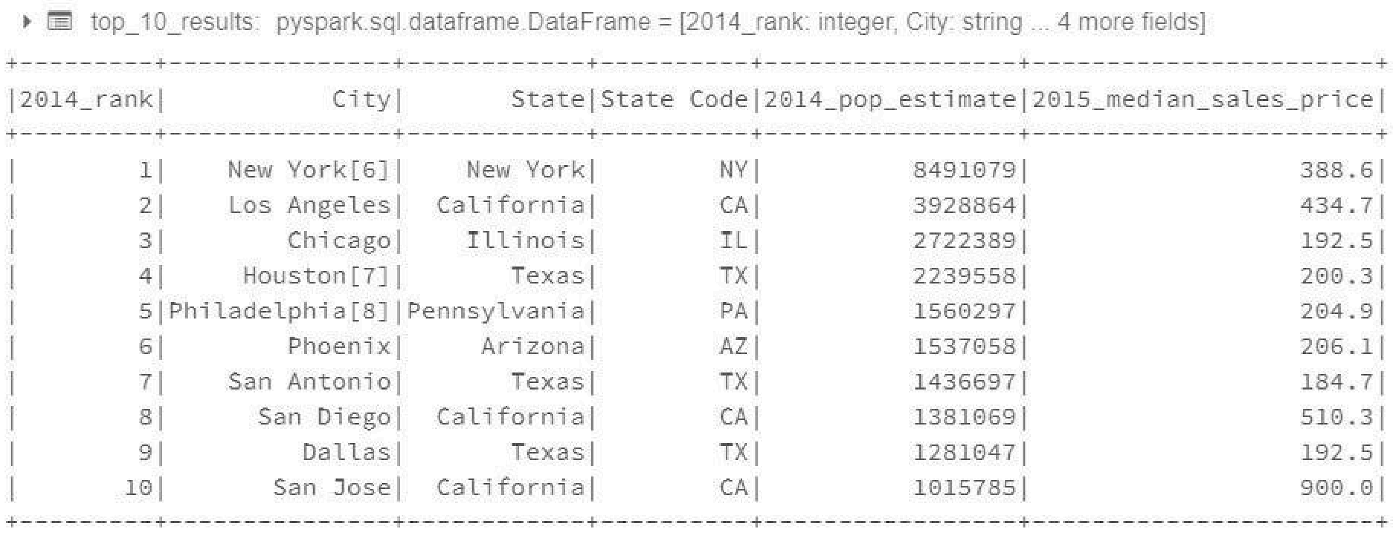
Now we can finally run a SQL query! It is extremely simple to run a SQL query in PySpark. Let’s run a basic query to see how it works:

```
# Viewing the top 10 cities based on the '2014_rank' column
```

```
top_10_results = spark.sql("""SELECT * FROM pop_price

                              WHERE 2014_rank <= 10

                              SORT BY 2014_rank ASC""")
```

```
top_10_results.show()
```

Press enter or click to view image in full size



```
▸ top_10_results:  pyspark.sql.dataframe.DataFrame = [2014_rank: integer, City: string ... 4 more fields]
+---------+---------------+------------+----------+-----------------+----------------------+
|2014_rank|           City|       State|State Code|2014_pop_estimate|2015_median_sales_price|
+---------+---------------+------------+----------+-----------------+----------------------+
|        1|    New York[6]|    New York|        NY|          8491079|                 388.6|
|        2|    Los Angeles|  California|        CA|          3928864|                 434.7|
|        3|        Chicago|    Illinois|        IL|          2722389|                 192.5|
|        4|     Houston[7]|       Texas|        TX|          2239558|                 200.3|
|        5|Philadelphia[8]|Pennsylvania|        PA|          1560297|                 204.9|
|        6|        Phoenix|     Arizona|        AZ|          1537058|                 206.1|
|        7|    San Antonio|       Texas|        TX|          1436697|                 184.7|
|        8|      San Diego|  California|        CA|          1381069|                 510.3|
|        9|         Dallas|       Texas|        TX|          1281047|                 192.5|
|       10|       San Jose|  California|        CA|          1015785|                 900.0|
+---------+---------------+------------+----------+-----------------+----------------------+
```

Top 10 2014 ranks

When we query from our dataframe using "*spark.sql()*", it returns a new dataframe within the conditions of the query. We simply save the queried results and then view those results using the "*.show()*" method. If you would like to see the notebook I used for this blog, you can click on the link below (it will be valid for 6 months from the day of this post):