


# Reading data with Pyspark Dataframes in Databricks

---

## Table of Contents (you can navigate directly by clicking on this points)

- [Introduction to Dataframes and Various File Formats](#)
- [Databricks UI](#)
- [How to create cluster in Databricks?](#)
- [How to upload files in Databricks?](#)
- [How to create tables in Databricks with UI?](#)
- [How to create notebook in Databricks?](#)
- [Reading data using csv format](#)
- [Reading data using json format](#)
- [Reading data using parquet format](#)
- [Reading data using table format](#)
- [Reading data from text files using csv and text format](#)

# Note → Arrow  is used to go back to “table of contents”.

---

Let's start.

## Introduction to Dataframes and Various File Formats



**Dataframe** - Dataframe is the most common structured API of Spark, it is simply a table with rows and columns.

**Schema of a Dataframe** - The list of columns and their types.

**Why we need Dataframes? (When we have already various options, e.g. -**

**Microsoft Excel)** – Spark Dataframe can span over thousands of computers, the reason behind this is –

- The data is either very large to get fit on a single machine. Or
- The time taken to perform the computation on a single machine is too long.

So, we simply put the data on multiple machines.

---

Let's talk about various formats (from which we are reading data using Dataframes) –

1) CSV - Values are separated by comma.

Example –

Name, Age

Chinky, 5

Minky, 7

- 2) JSON - Data is stored in key-value pairs.

Example –

```
{'Name': 'Chinky', 'Age': 5}
```

```
{'Name': 'Minky', 'Age': 7}
```

- 3) Parquet - Columnar storage format and provides efficient data compression. Also, query performance is increased over row-based data stores. It supports limited schema evolution. It is a split-able file format.

Example – Column based data store

Name	Chinky	Minky
Age	5	7

- 4) Text – Data where each record is delineated by a newline character.

Example –

My name is Chinky.

I am 5 years old.

- 5) Tables – The collection of rows and columns, where each column represents the field name and row contains the values of those fields.

Example –

Name	Age
Chinky	5
Minky	7

---

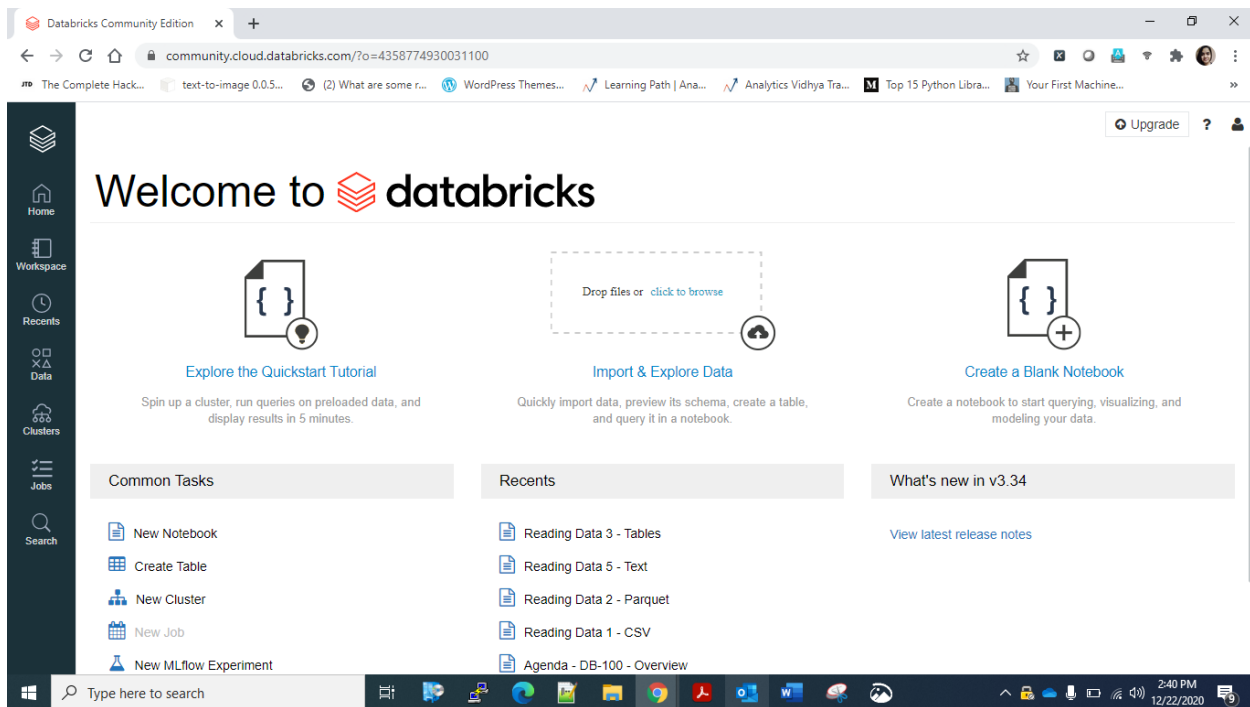
## Databricks UI



Let's discuss few things about databricks UI –

- [How to create cluster in Databricks?](#)
- [How to upload files in Databricks?](#)
- [How to create tables in Databricks with UI?](#)
- [How to create notebook in Databricks?](#)

Before going into these points, just make sure you have a databricks account (or go for a community edition). Once you have successfully created your account, the UI will look like this –

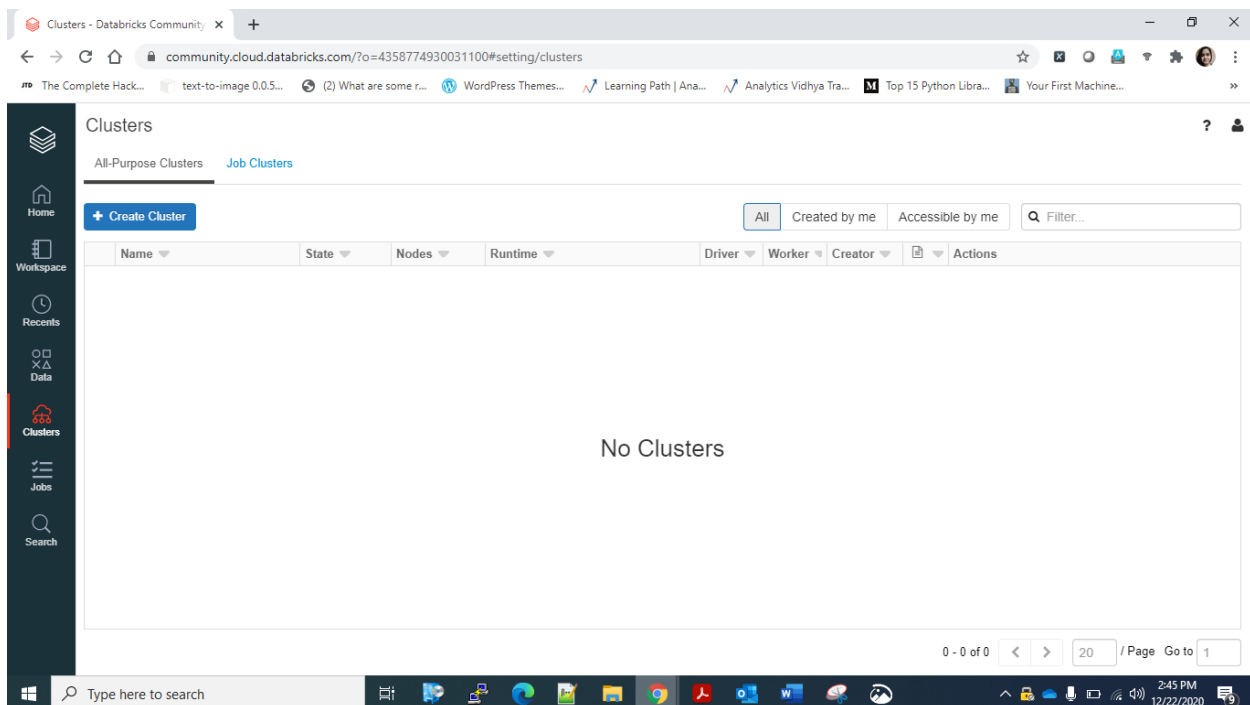


(Note – Your UI might look different than me, because my account is in community edition).

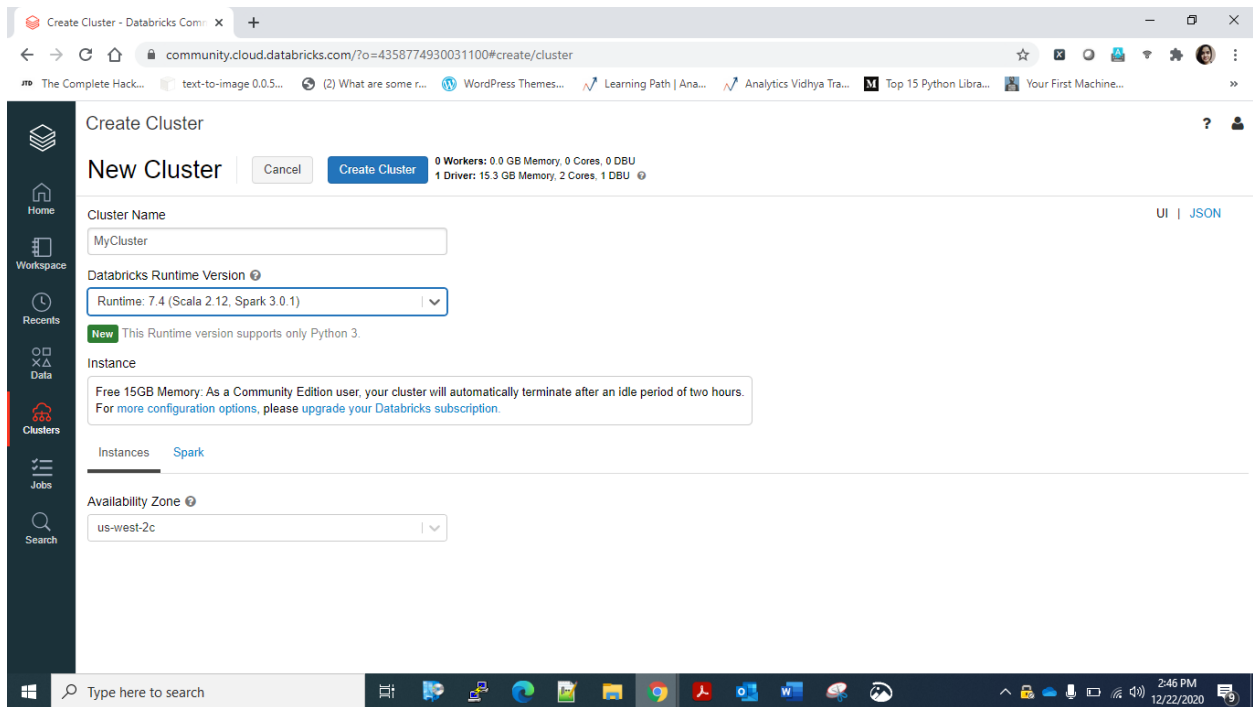
Let's start answering to our first question.

## How to create cluster in Databricks? -

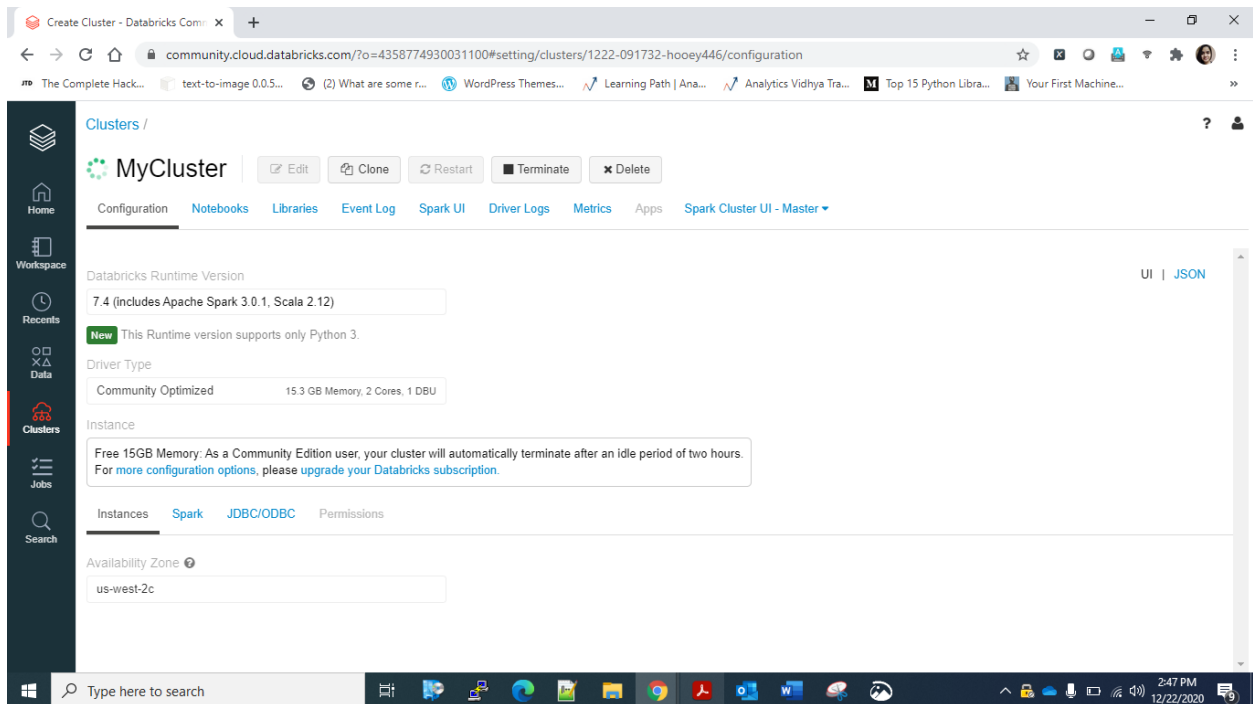
In the right side of the screen, you will find “Clusters” option, click on that. The page will look like this, once you clicked on that button –



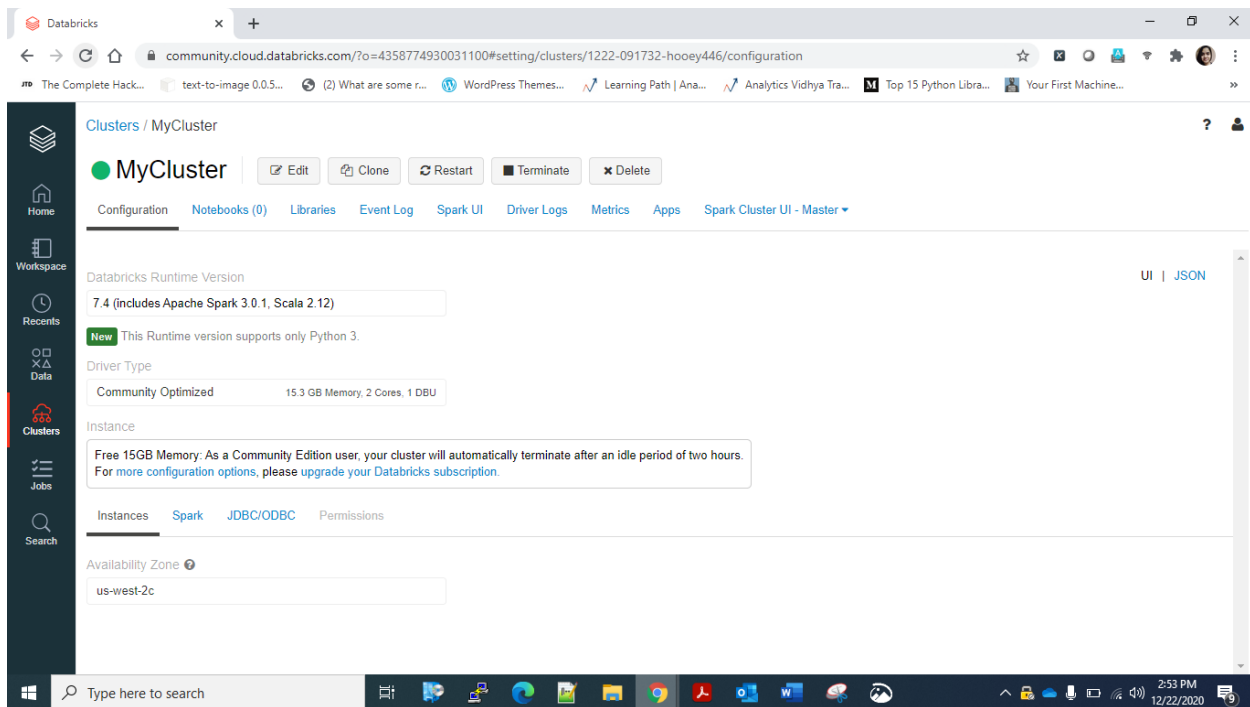
Now, click on “+ Create Cluster” button.



Fill your cluster name and hit the button “Create Cluster”.



Wait for that green “dot-dot” circle to become a full round circle. Just like the below screen.

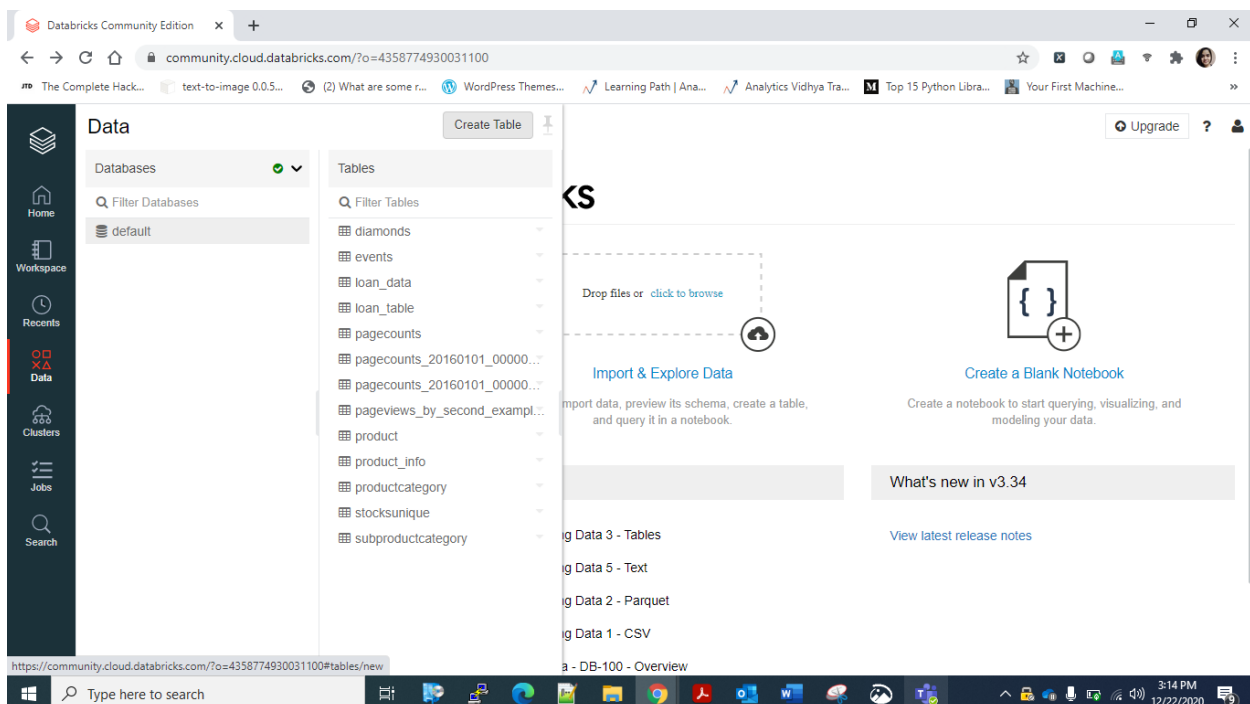


Now you are ready to use your cluster.

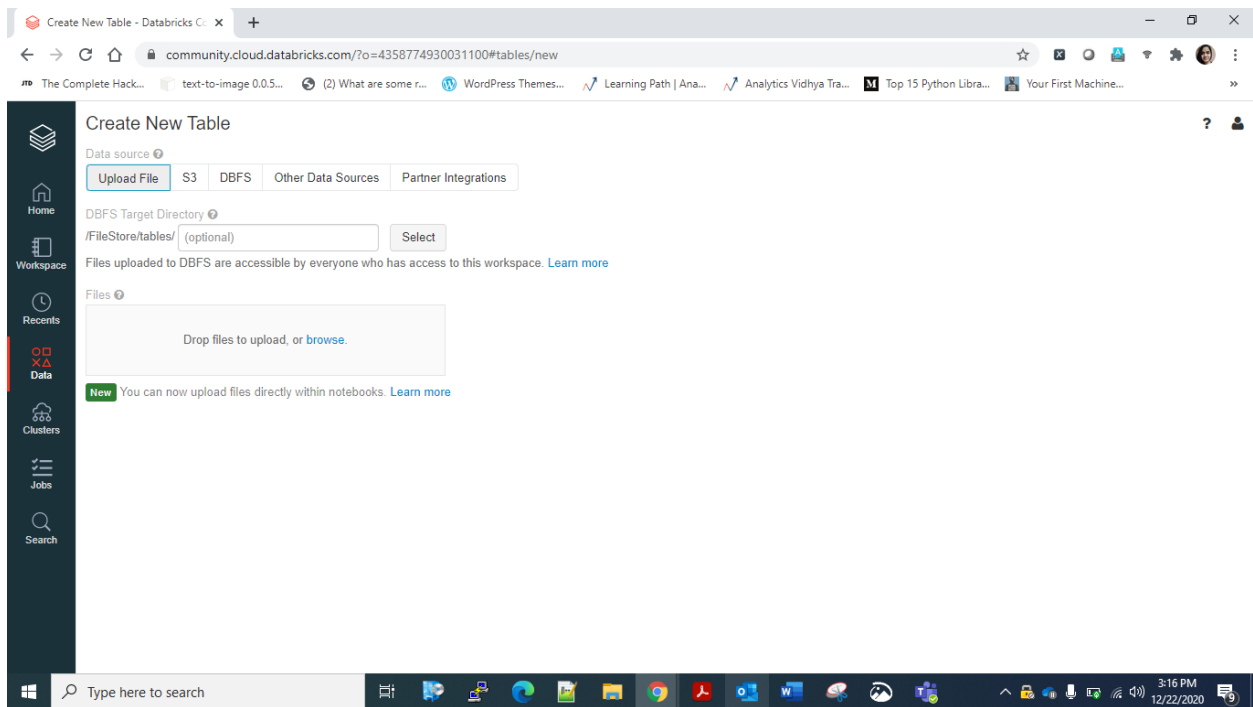
Let's go to second question.

## How to upload files in Databricks? -

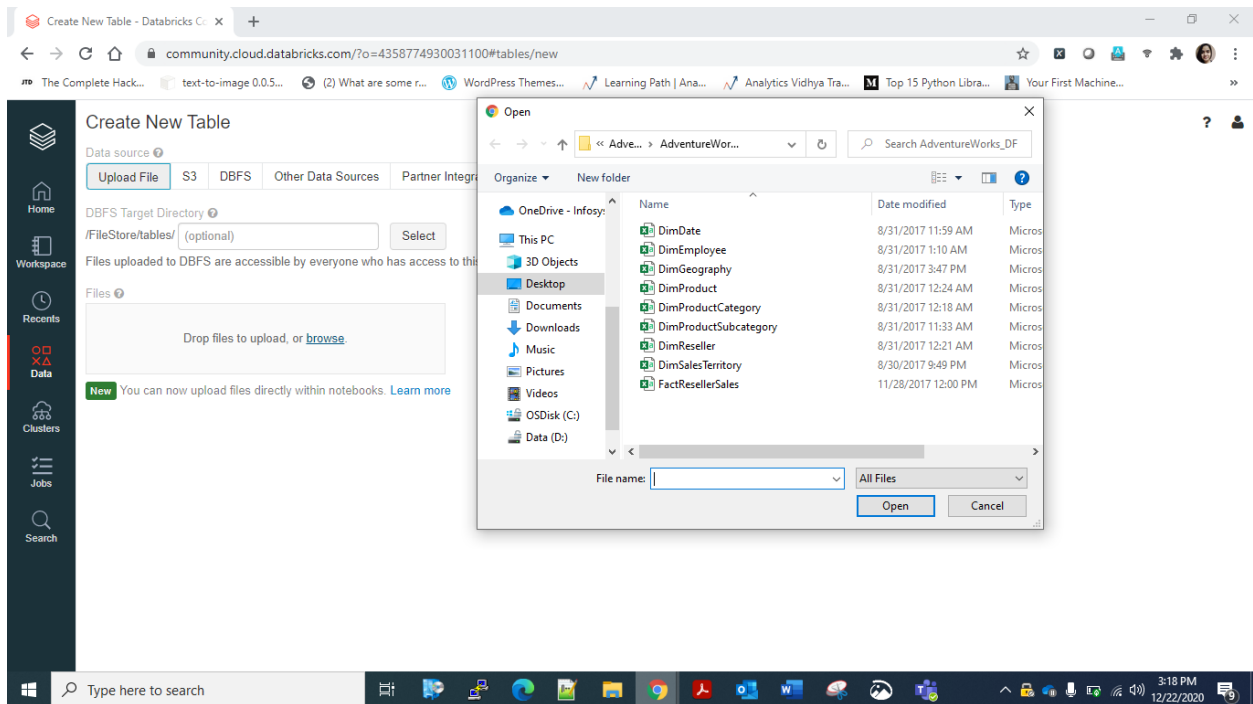
In the right side of the screen, you will find "Data" option, click on that.



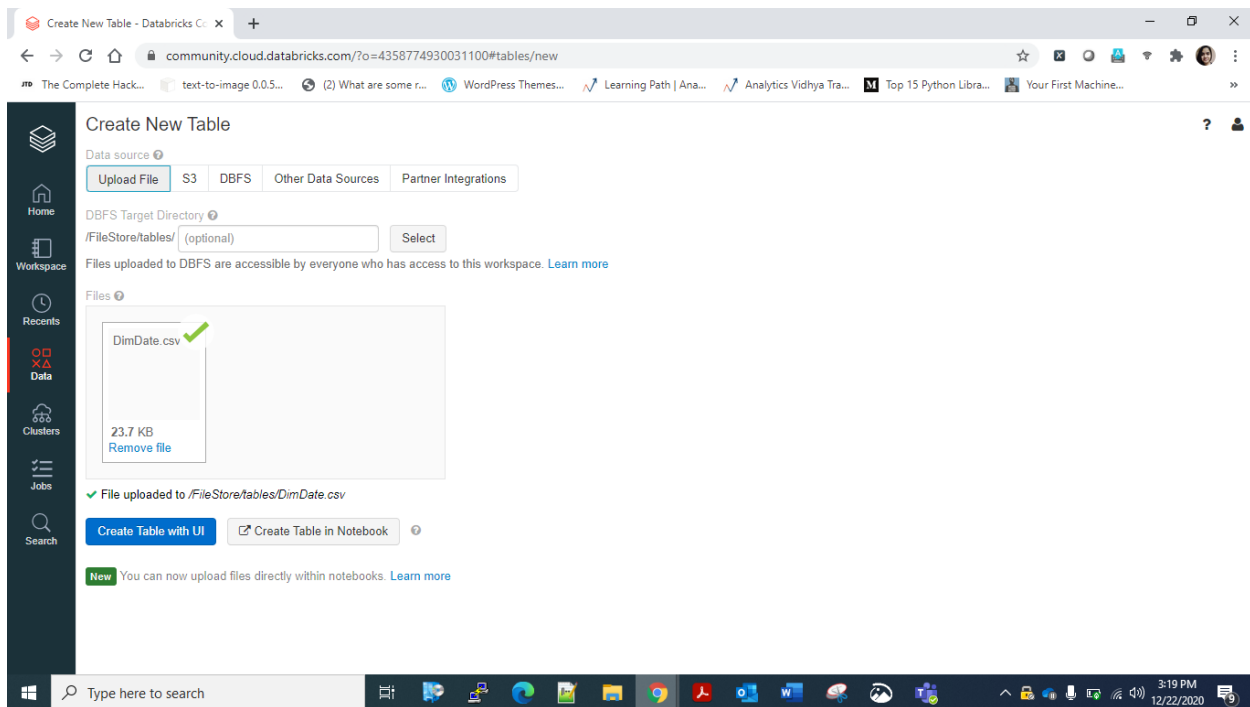
Click on “Create Table” button, the UI will look like this –



Don't get confused with the title “Create New Table”, here we are only uploading files by clicking on “Drop files to upload or browse”.



Select your file(s) and click “Open”.

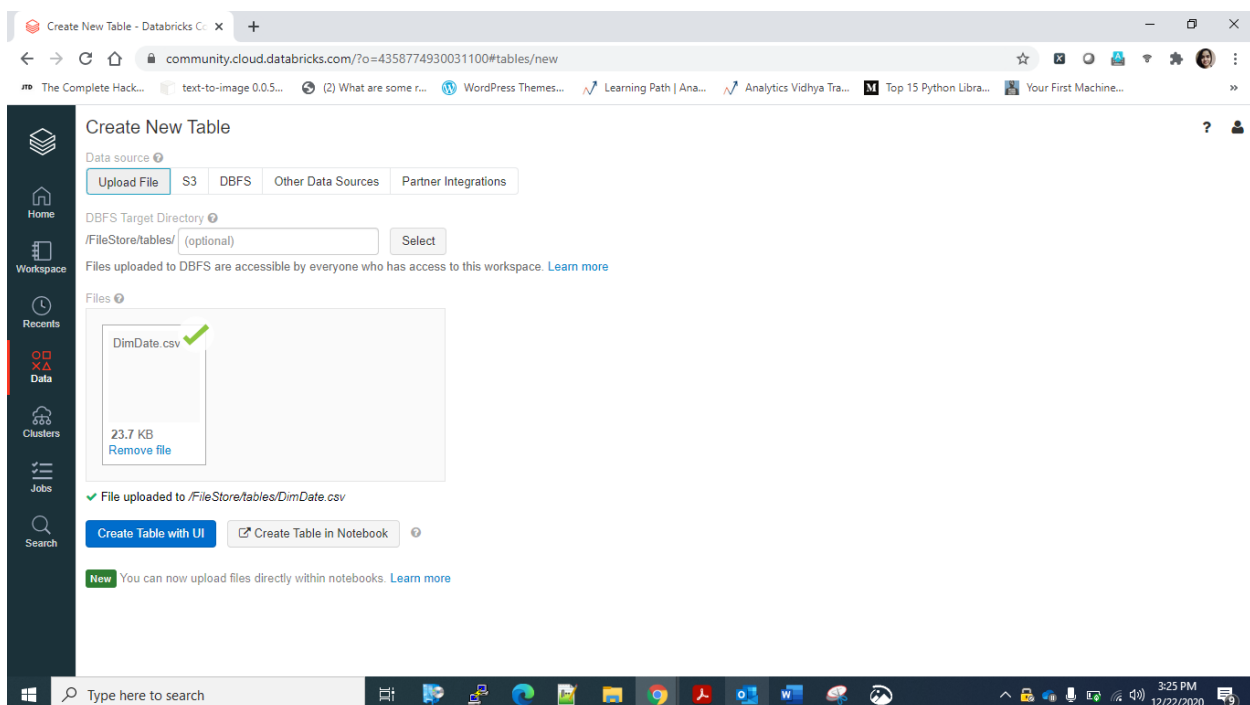


You can see that our file is successfully uploaded to `"/FileStore/tables/DimDate.csv"`

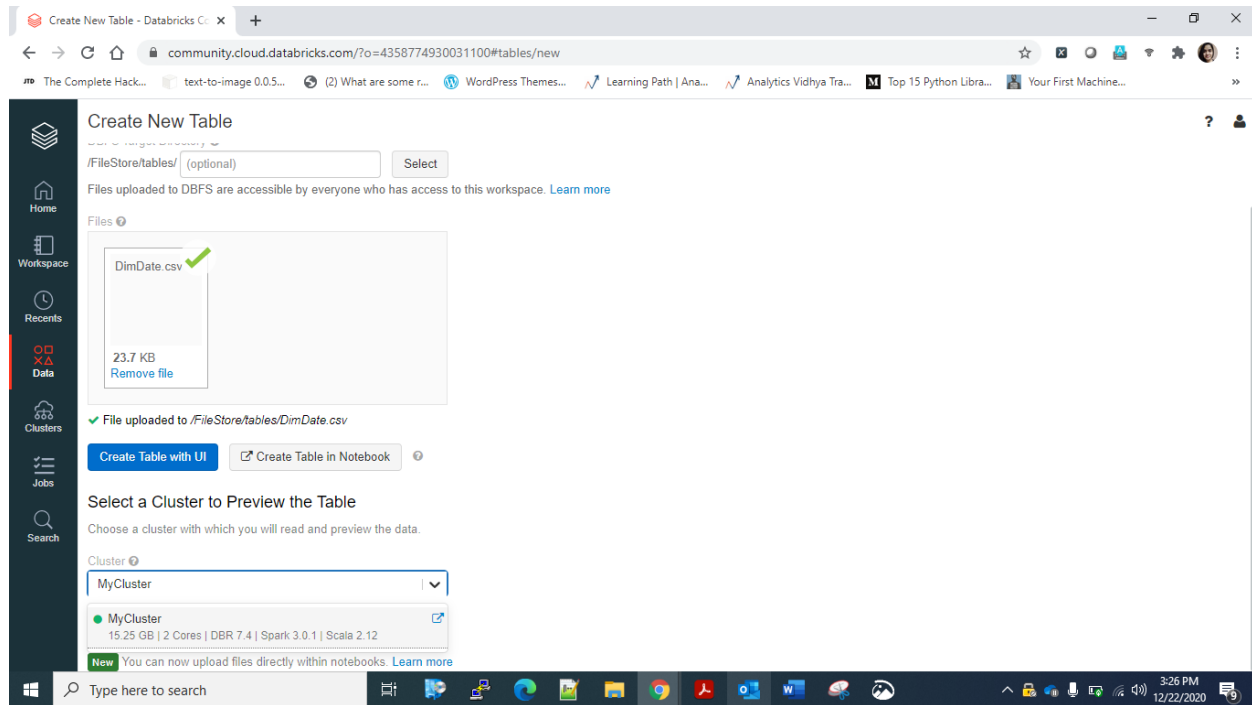
From the above image, I think you have got the idea how will we be going to create tables with UI. But if not, let me answer the next question.

## How to create tables in Databricks with UI? -

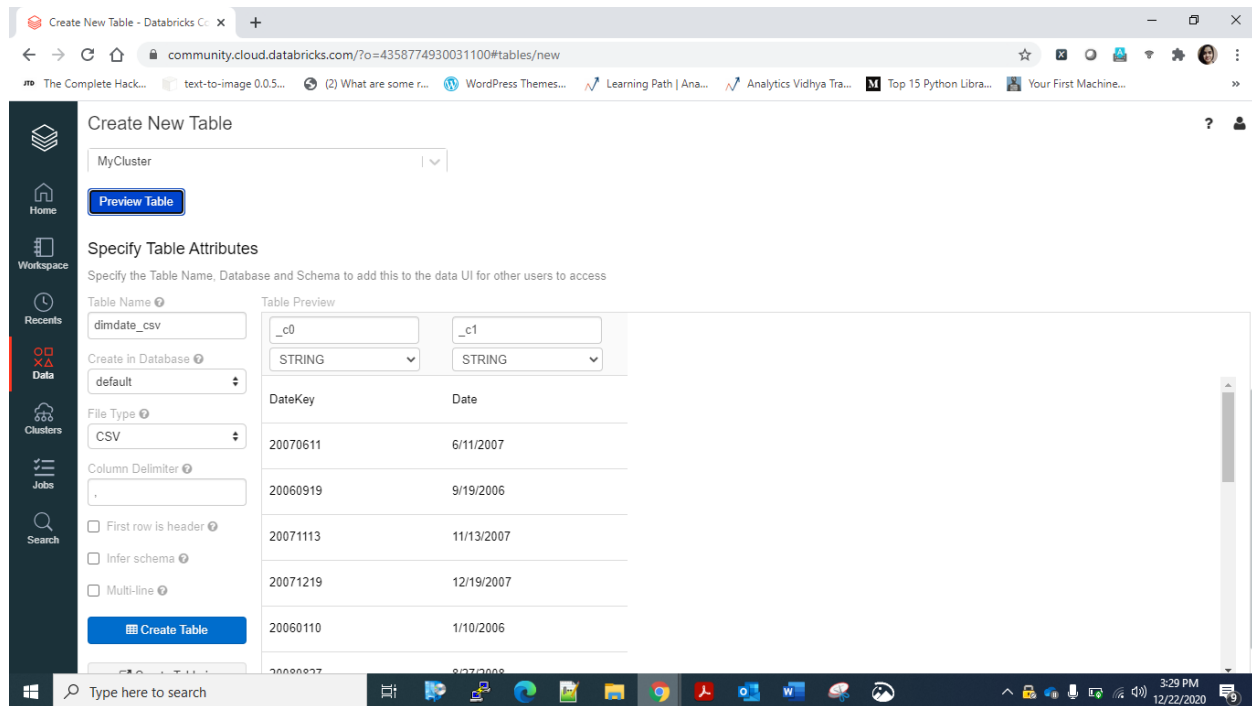
First, we will upload the file, and then create a table from it.



Now, click on “Create Table with UI” button. It will ask you to select a cluster to preview the table. Simply enter your cluster name.



Simply enter your cluster name. Hit the button “Preview Table”.



Screen will look like this. Now add the table name, database name(if you want the table in a particular database), column delimiter, then hit the checkbox if first row is a header, if you want



to infer schema and if data is expand to multi-lines. Once, you have done all these, the screen will look like this –

Create New Table - Databricks Co. x

community.cloud.databricks.com/?o=4358774930031100#tables/new

Create New Table

Preview Table

Specify Table Attributes

Specify the Table Name, Database and Schema to add this to the data UI for other users to access

Table Name

Create in Database

File Type

Column Delimiter

☒ First row is header

☒ Infer schema

☐ Multi-line

Create Table

Create Table in Notebook

Table Preview

DateKey	Date
20070611	6/11/2007
20060919	9/19/2006
20071113	11/13/2007
20071219	12/19/2007
20060110	1/10/2006
20080827	8/27/2008
20050709	7/9/2005

New You can now upload files directly within notebooks. [Learn more](#)

Let's done the table creation by hitting the button "Create table". You can verify the details of your table from this screen.

dimdate\_csv - Databricks Comm: x

community.cloud.databricks.com/?o=4358774930031100#table/default/dimdate\_csv

Table: dimdate\_csv

dimdate\_csv | Refresh

MyCluster

Schema:

	col_name	data_type	comment
1	DateKey	int	null
2	Date	string	null

Showing all 2 rows.

Sample Data:

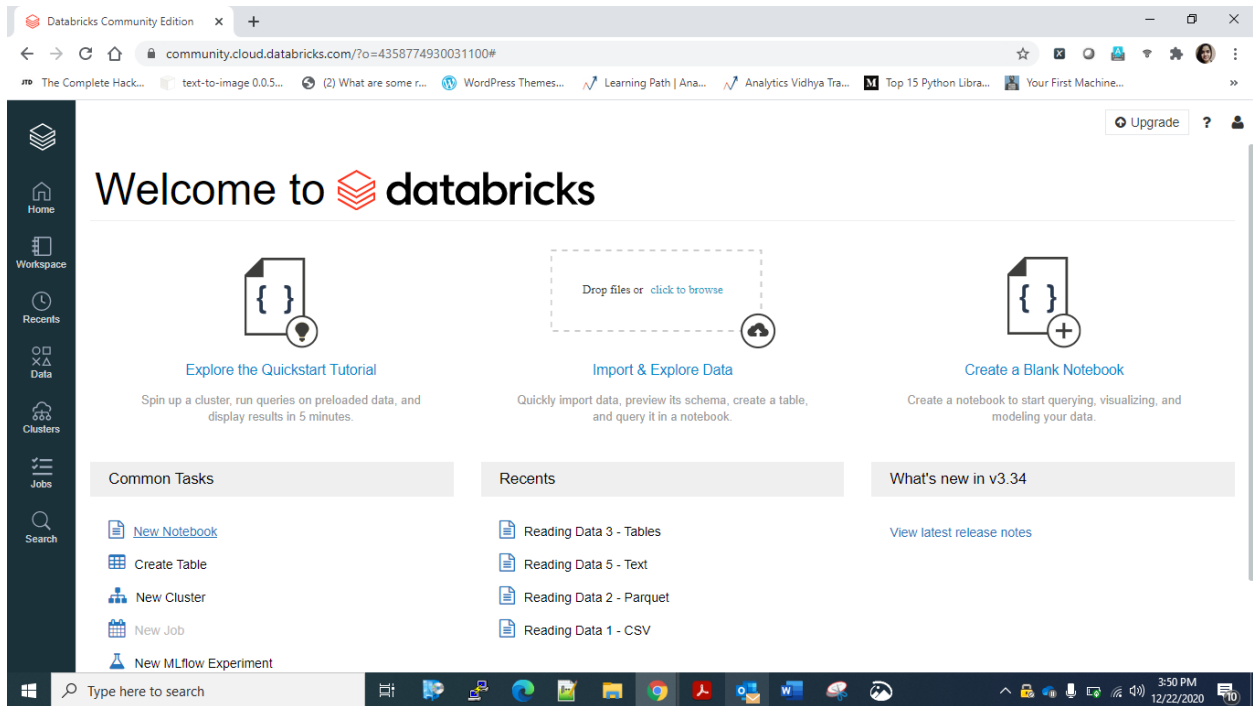
	DateKey	Date
1	20070611	6/11/2007
2	20060919	9/19/2006
3	20071113	11/13/2007
4	20071219	12/19/2007
5	20060110	1/10/2006
6	20080827	8/27/2008
7	20050709	7/9/2005
8	20061106	11/6/2006

Showing all 20 rows.

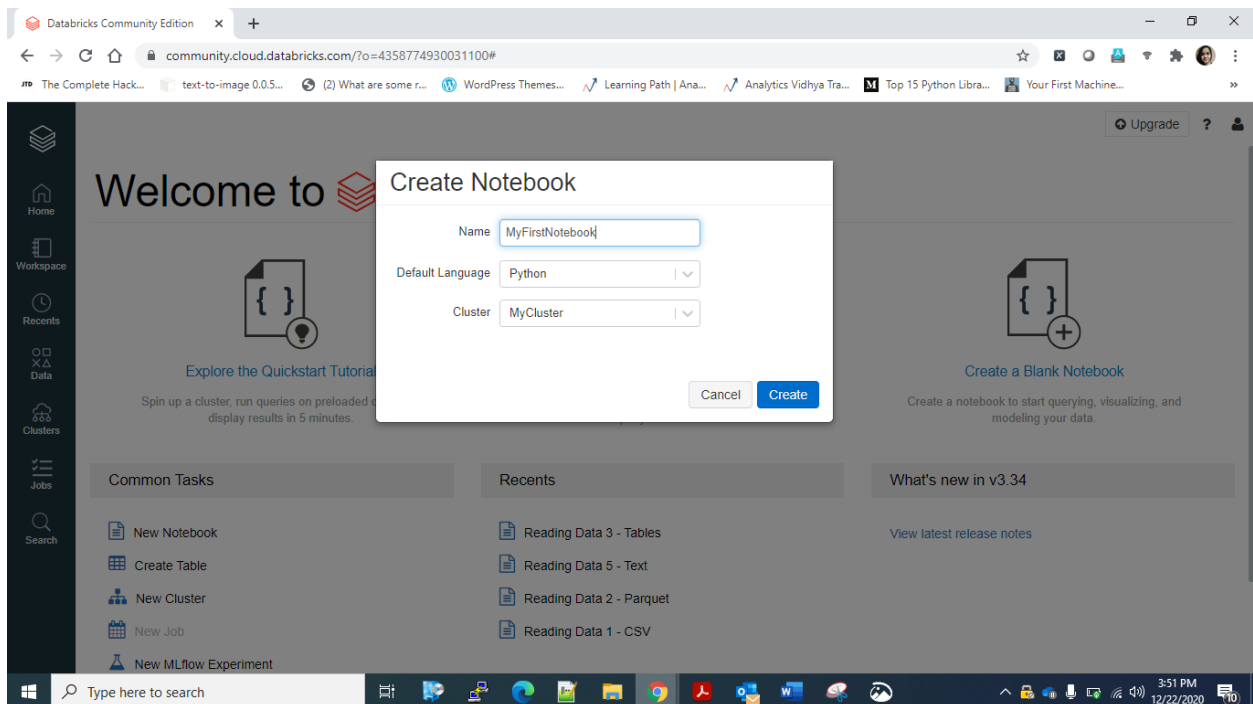
Let's go to next question.

## How to create notebook in Databricks with UI? -

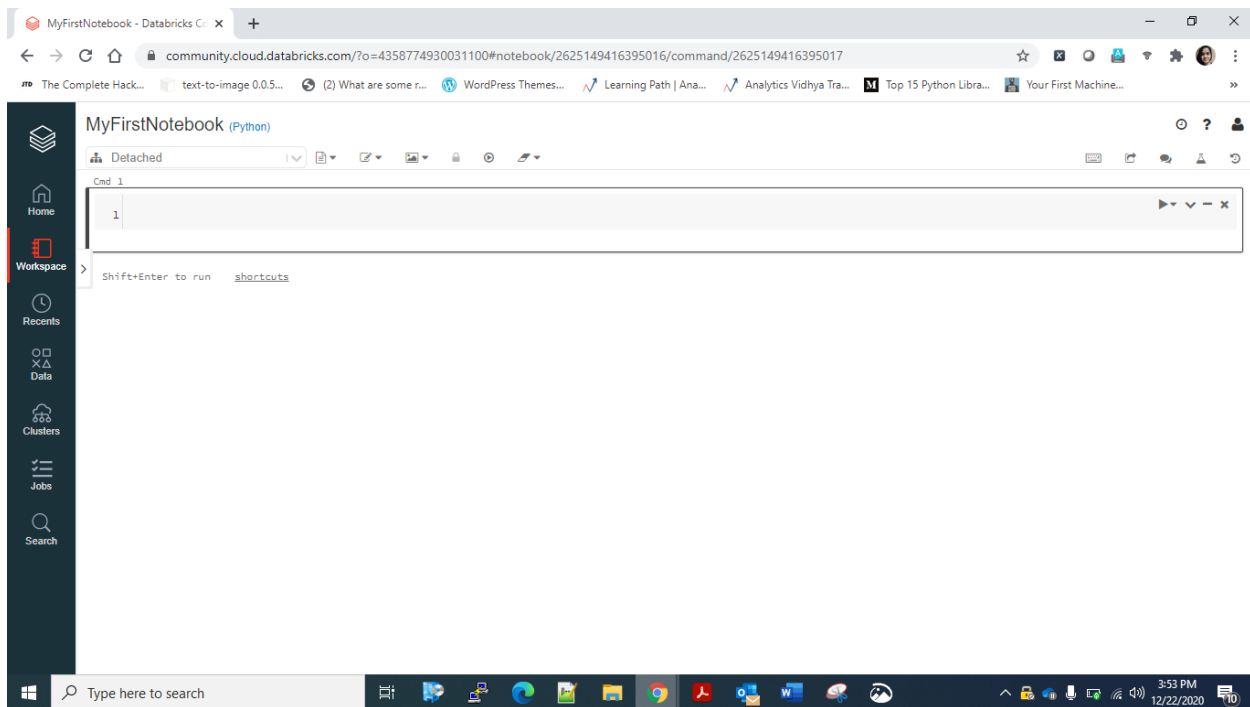
Click on “New Notebook” button.



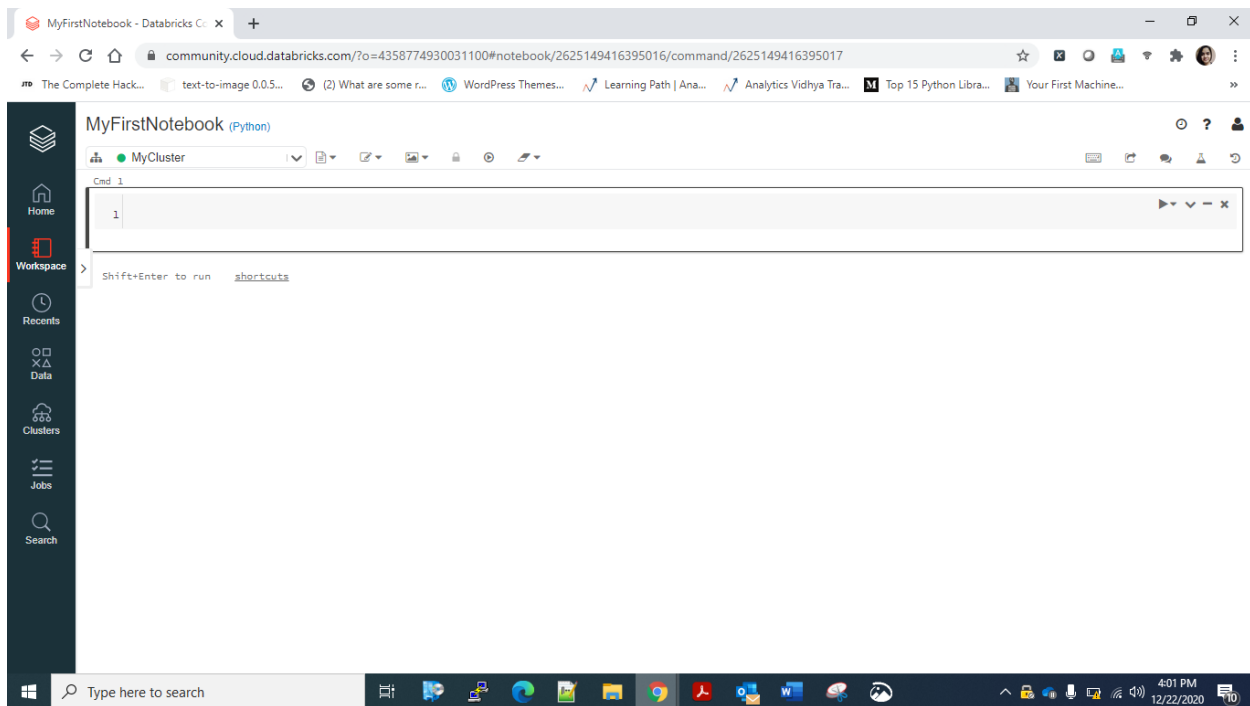
Enter your Notebook name and click the “Create” button.



Below your notebook name, you can see that our notebook is detached. In order to execute our commands, we need to attach the notebook to our cluster.



Once, we have attached our cluster, our screen will look like this –



Hence, we have successfully created a cluster, uploaded our files, created a table from a file and finally created a notebook and attached it to our cluster.

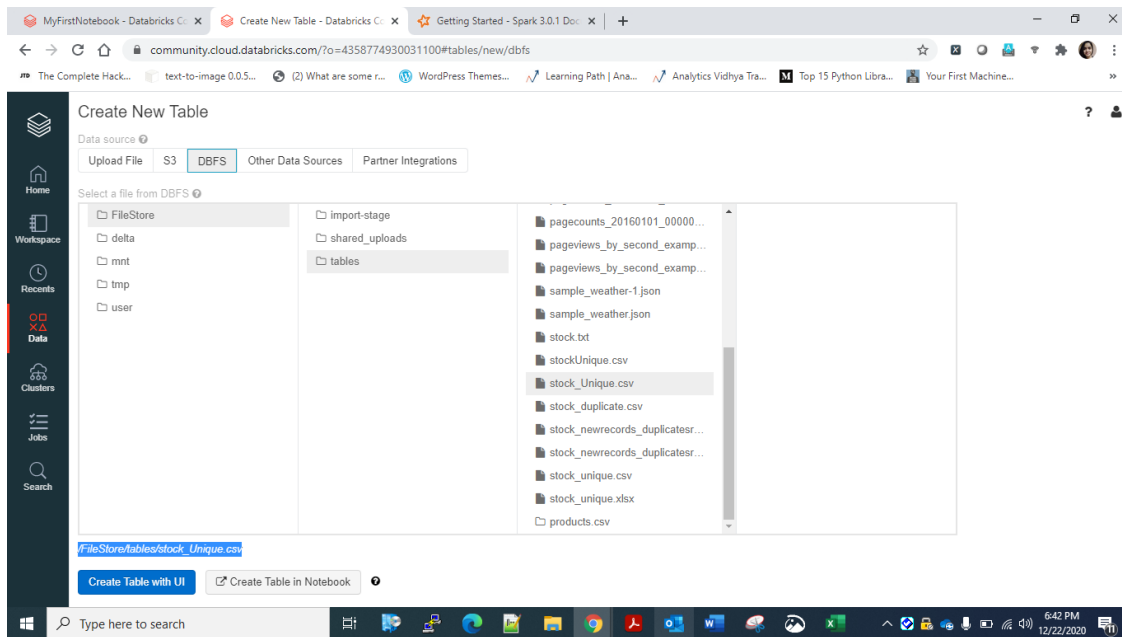
Let's get back to our main task – "Reading data using various formats in Pyspark Dataframes.

Here, we will discuss –

- [Reading data using csv format](#)
- [Reading data using json format](#)
- [Reading data using parquet format](#)
- [Reading data using table format](#)
- [Reading data from text files using csv and text format](#)

## Reading data using csv format

We will be working on stocks.csv file in this tutorial, let's copy the address (you can check in the Data section as shown below) –



Our stocks\_Unique file looks like this –

	A	B	C	D	E
1	1	11-Dec-20	1000	998	2
2	2	11-Dec-20	2000	1500	500
3	3	11-Dec-20	500	350	150
4	4	11-Dec-20	300	200	100
5	5	11-Dec-20	650	470	180
6	70	11-Dec-20	10000	9800	200
7	130	11-Dec-20	200	150	50
8	165	11-Dec-20	210	150	60
9	220	11-Dec-20	800	700	100
10	121	11-Dec-20	900	470	430
11					

Its schema is –

- Product\_id
- Sale\_Date
- Total Qty
- Available Qty
- Sale Qty

Let's read the data into Dataframe. (we added "dbfs:" in the file address because our file is stored in dbfs (databricks file system))

```
stocks = spark.read.csv("dbfs:/FileStore/tables/stock_Unique.csv")
```

```
display(stocks)
```

	_c0	_c1	_c2	_c3	_c4
1	1	11-Dec-20	1000	998	2
2	2	11-Dec-20	2000	1500	500
3	3	11-Dec-20	500	350	150
4	4	11-Dec-20	300	200	100
5	5	11-Dec-20	650	470	180
6	70	11-Dec-20	10000	9800	200
7	130	11-Dec-20	200	150	50

Here you can see – type of every column is string, try inferring schema now.

```
stocks = spark.read.csv("dbfs:/FileStore/tables/stock_Unique.csv", inferSchema=True)  
print(stocks.schema)
```

```
1 stocks = spark.read.csv("dbfs:/FileStore/tables/stock_Unique.csv", inferSchema=True) # to read data from csv file and inferring schema  
2 print(stocks.schema)
```

StructType(List(StructField(\_c0,IntegerType,true),StructField(\_c1,StringType,true),StructField(\_c2,IntegerType,true),StructField(\_c3,IntegerType,true),StructField(\_c4,IntegerType,true)))

Now, we can see it inferred the schema quite well. But you might get confused what is StructType and StructField right?

Let me answer it first. Whenever we define schema of a Dataframe, we represent row by StructType and column by StructField. And as row is list of various columns, the StructType is also the list of StructField.

**Note – inferring schema takes long time when we deal with large data, because it scans whole data. Hence providing schema is the best solution.**

Also, there is no name of any column and it is very hard to find what our columns represent. So, now try adding schema (column names and their types) to it.

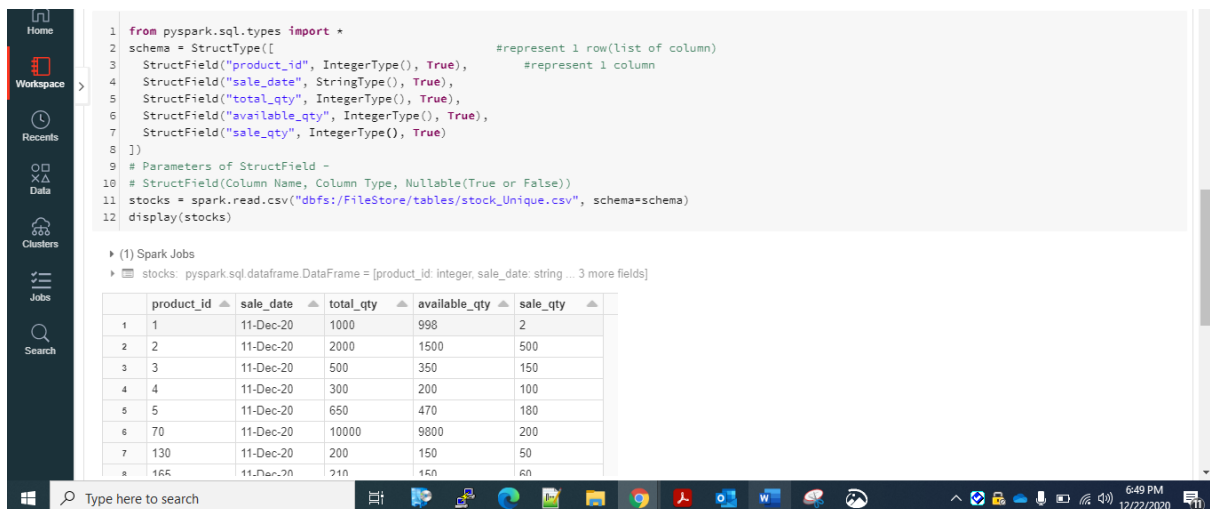
```
from pyspark.sql.types import *

schema = StructType([
    #represent 1 row(list of columns)
    StructField("product_id", IntegerType(), True),    #represent 1 column
    StructField("sale_date", StringType(), True),
    StructField("total_qty", IntegerType(), True),
    StructField("available_qty", IntegerType(), True),
    StructField("sale_qty", IntegerType(), True)
])

# Parameters of StructField -
# StructField(Column Name, Column Type, Nullable(True or False))

stocks = spark.read.csv("dbfs:/FileStore/tables/stock_Unique.csv", schema=schema)

display(stocks)
```



The screenshot shows a Databricks workspace interface. On the left is a sidebar with navigation icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area displays a code editor with the following Python code:

```
1 from pyspark.sql.types import *
2 schema = StructType([
3     StructField("product_id", IntegerType(), True),    #represent 1 row(list of column)
4     StructField("sale_date", StringType(), True),    #represent 1 column
5     StructField("total_qty", IntegerType(), True),
6     StructField("available_qty", IntegerType(), True),
7     StructField("sale_qty", IntegerType(), True)
8 ])
9 # Parameters of StructField -
10 # StructField(Column Name, Column Type, Nullable(True or False))
11 stocks = spark.read.csv("dbfs:/FileStore/tables/stock_Unique.csv", schema=schema)
12 display(stocks)
```

Below the code editor, a Spark job execution summary is shown, indicating that the job completed successfully. The output is a table with 8 rows and 5 columns:

	product_id	sale_date	total_qty	available_qty	sale_qty
1	1	11-Dec-20	1000	998	2
2	2	11-Dec-20	2000	1500	500
3	3	11-Dec-20	500	350	150
4	4	11-Dec-20	300	200	100
5	5	11-Dec-20	650	470	180
6	70	11-Dec-20	10000	9800	200
7	130	11-Dec-20	200	150	50
8	165	11-Dec-20	210	160	50

Pretty good output, right? Yeah!!

Note – if your file has a header, write command –

```
# stocks = spark.read.csv("file name", header=True, schema=schema)
```

So, with this, we learnt how to read data using csv format.

# Reading data using json format



We will be working on sample\_weather.json file in this tutorial. Our file address is – “dbfs:/FileStore/tables/sample\_file.json”. Let’s start loading data in the dataframe.

Note – the JSON reader also assumes...

- That there is one JSON object per line and
- it's delineated by a new line.

Our sample\_file.json data looks like –

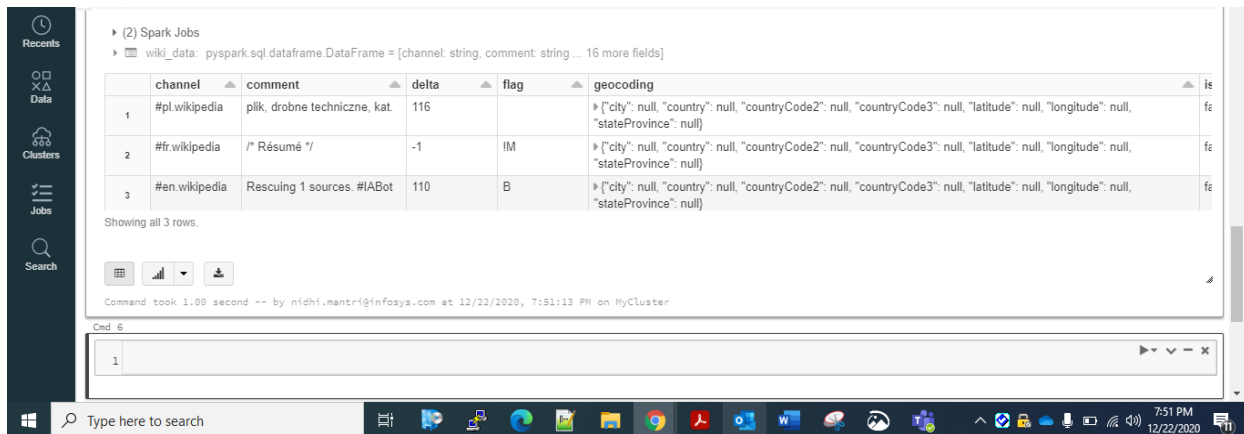
```
%fs head "dbfs:/FileStore/tables/sample_file.json"
```



Let’s try reading the data into dataframe.

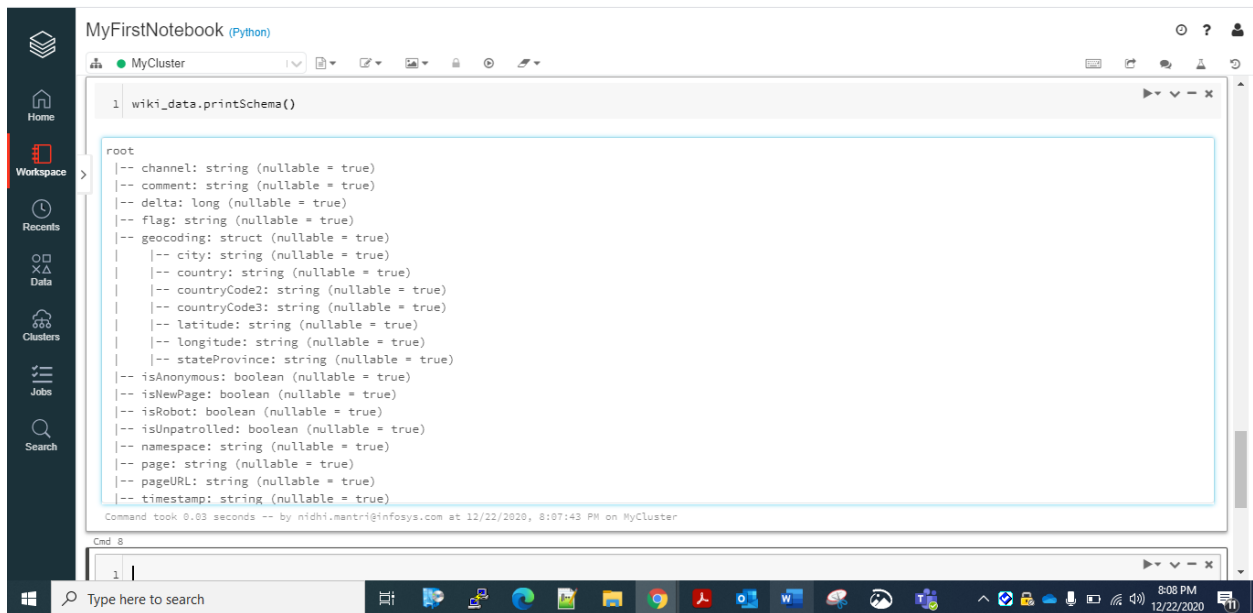
```
wiki_data = spark.read.json("dbfs:/FileStore/tables/sample_file.json")
```

```
display(wiki_data)
```



Here, we can see in the schema that it gets all the column names and their default types(string and Boolean), but it did not infer the data types correctly. (ex. - values which are of type double are inferred as strings).

```
wiki_data.printSchema()
```

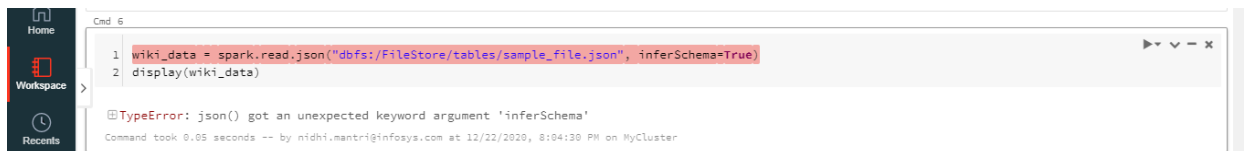


Let's infer the schema –

```
#/FileStore/tables/sample_weather.json
```

```
wiki_data = spark.read.json("dbfs:/FileStore/tables/sample_file.json", inferSchema=True)
```

```
display(wiki_data)
```



Oops!! We got an error, let's try providing user defined schema to it.

```
from pyspark.sql.types import *
```

```
jsonSchema = StructType([
```

```
    StructField("channel", StringType(), True),
```

```
    StructField("comment", StringType(), True),
```

```
    StructField("delta", IntegerType(), True),
```

```
    StructField("flag", StringType(), True),
```

```
    StructField("geocoding", StructType([
```

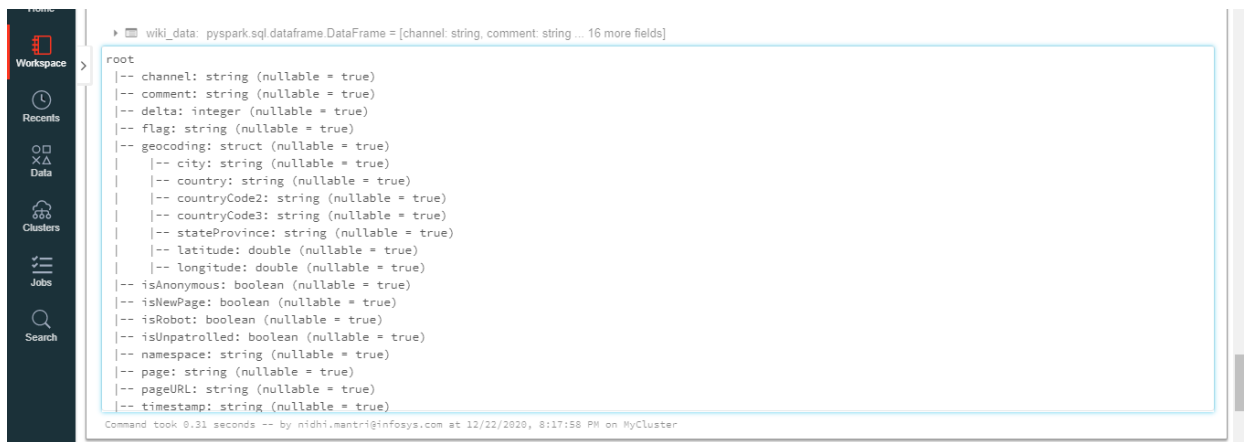
```
        StructField("city", StringType(), True),
```



```
StructField("country", StringType(), True),
StructField("countryCode2", StringType(), True),
StructField("countryCode3", StringType(), True),
StructField("stateProvince", StringType(), True),
StructField("latitude", DoubleType(), True),
StructField("longitude", DoubleType(), True)]]), True),
StructField("isAnonymous", BooleanType(), True),
StructField("isNewPage", BooleanType(), True),
StructField("isRobot", BooleanType(), True),
StructField("isUnpatrolled", BooleanType(), True),
StructField("namespace", StringType(), True),
StructField("page", StringType(), True),
StructField("pageURL", StringType(), True),
StructField("timestamp", StringType(), True),
StructField("url", StringType(), True),
StructField("user", StringType(), True),
StructField("userURL", StringType(), True),
StructField("wikipediaURL", StringType(), True),
StructField("wikipedia", StringType(), True)]]
```

Now, again read the data with schema provided.

```
wiki_data = spark.read.json("dbfs:/FileStore/tables/sample_file.json", schema=jsonSchema)
wiki_data.printSchema()
```



```
wiki_data: pyspark.sql.dataframe.DataFrame = [channel: string, comment: string ... 16 more fields]

root
|-- channel: string (nullable = true)
|-- comment: string (nullable = true)
|-- delta: integer (nullable = true)
|-- flag: string (nullable = true)
|-- geocoding: struct (nullable = true)
|   |-- city: string (nullable = true)
|   |-- country: string (nullable = true)
|   |-- countryCode2: string (nullable = true)
|   |-- countryCode3: string (nullable = true)
|   |-- stateProvince: string (nullable = true)
|   |-- latitude: double (nullable = true)
|   |-- longitude: double (nullable = true)
|-- isAnonymous: boolean (nullable = true)
|-- isNewPage: boolean (nullable = true)
|-- isRobot: boolean (nullable = true)
|-- isUnpatrolled: boolean (nullable = true)
|-- namespace: string (nullable = true)
|-- page: string (nullable = true)
|-- pageURL: string (nullable = true)
|-- timestamp: string (nullable = true)
```

Command took 0.21 seconds -- by nidhi.mantri@infosys.com at 12/22/2020, 8:17:58 PM on MyCluster

Yeah! We have done it right.

## Reading data using parquet format



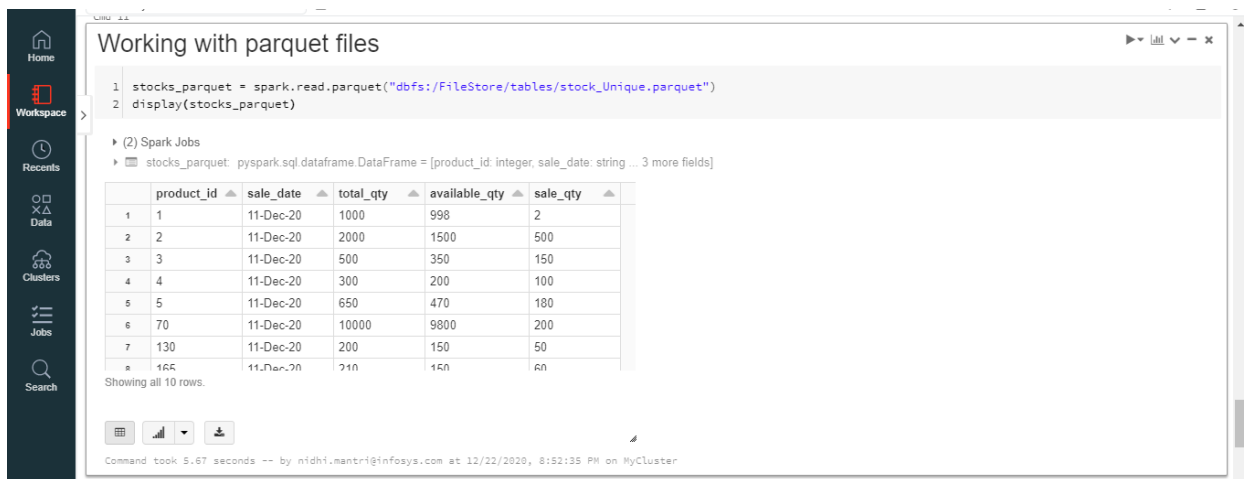
I have already created a parquet file of stocks\_Unique file.

Command I used to save the stocks dataframe into parquet file is –

```
stocks.write.parquet("dbfs:/FileStore/tables/stock_Unique.parquet")
```

Let's try reading data from it.

```
stocks_parquet = spark.read.parquet("dbfs:/FileStore/tables/stock_Unique.parquet")
display(stocks_parquet)
```



Working with parquet files

```
1 stocks_parquet = spark.read.parquet("dbfs:/FileStore/tables/stock_Unique.parquet")
2 display(stocks_parquet)
```

(2) Spark Jobs

stocks\_parquet: pyspark.sql.dataframe.DataFrame = [product\_id: integer, sale\_date: string ... 3 more fields]

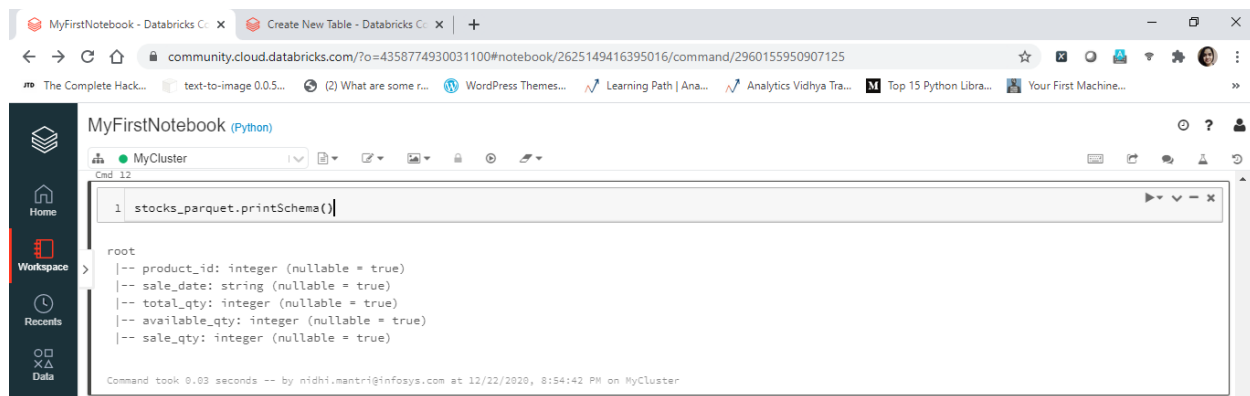
	product_id	sale_date	total_qty	available_qty	sale_qty
1	1	11-Dec-20	1000	998	2
2	2	11-Dec-20	2000	1500	500
3	3	11-Dec-20	500	350	150
4	4	11-Dec-20	300	200	100
5	5	11-Dec-20	650	470	180
6	70	11-Dec-20	10000	9800	200
7	130	11-Dec-20	200	150	50
8	165	11-Dec-20	210	150	60

Showing all 10 rows.

Command took 5.67 seconds -- by nidhi.mantri@infosys.com at 12/22/2020, 8:52:35 PM on MyCluster

You can see I did not provide the schema to it. Let's try printing the schema.

```
stocks_parquet.printSchema()
```



Here, we get the right schema, because parquet stores the schema (column names and their data types) in parquet files. It requires 1 job to read the schema from those files. And if we want to avoid that 1 job, we can provide schema (user defined) while reading data from parquet, which is shown below –

```
from pyspark.sql.types import *

schema = StructType([
    #represent 1 row(list of columns)
    StructField("product_id", IntegerType(), True),    #represent 1 column
    StructField("sale_date", StringType(), True),
    StructField("total_qty", IntegerType(), True),
    StructField("available_qty", IntegerType(), True),
    StructField("sale_qty", IntegerType(), True)
])

stocks_parquet =(spark.read
    .schema(schema)
    .parquet("dbfs:/FileStore/tables/stock_Unique.parquet"))

display(stocks_parquet)
```

(1) Spark Jobs

stocks\_parquet: pyspark.sql.dataframe.DataFrame = [product\_id: integer, sale\_date: string ... 3 more fields]

	product_id	sale_date	total_qty	available_qty	sale_qty
1	1	11-Dec-20	1000	998	2
2	2	11-Dec-20	2000	1500	500
3	3	11-Dec-20	500	350	150
4	4	11-Dec-20	300	200	100
5	5	11-Dec-20	650	470	180
6	70	11-Dec-20	10000	9800	200
7	130	11-Dec-20	200	150	50
8	165	11-Dec-20	210	150	60

Showing all 10 rows.

Command took 0.96 seconds -- by nidhi.mantri@infosys.com at 12/22/2020, 10:33:46 PM on MyCluster

Compare the spark jobs it took to run the commands (with or without schema). Here, we need to provide the schema before providing our file. (Try providing the schema as –

**`spark.read.parquet(filename, schema=schema))`**

So, with that, we have successfully read the data from parquet files. As we mentioned earlier, query performance is increased when we work with parquet files.

## Reading data from tables using table format

Let's try to read data into dataframe from dimdate\_csv table we created earlier. While reading data from tables, we don't need to provide schema, because we already provided it while creating tables.

**`dim_date = spark.read.table("dimdate_csv")`**

**`display(dim_date)`**

(1) Spark Jobs

dim\_date: pyspark.sql.dataframe.DataFrame = [DateKey: integer, Date: string]

	DateKey	Date
1	20070611	6/11/2007
2	20060919	9/19/2006
3	20071113	11/13/2007
4	20071219	12/19/2007
5	20060110	1/10/2006
6	20080827	8/27/2008
7	20050709	7/9/2005
8	20061106	11/6/2006

Showing the first 1000 rows.

Printing the schema to verify whether we get the right schema or not.

**`dim_date.printSchema()`**

```

Cmd 15
1 dim_date.printSchema()

root
 |-- DateKey: integer (nullable = true)
 |-- Date: string (nullable = true)

Command took 0.06 seconds -- by nidhi.mantri@infosys.com at 12/22/2020, 11:07:02 PM on MyCluster

Cmd 16

```

We got the right schema; hence we have successfully loaded the data from table into dataframe.

## Reading data from text files using csv and text format

Here we will discuss two types of text data. First, when our data is complete text, like there is no rows and columns, then we use text format. Second, when our data is like csv, tsv format, we use csv format.

Sample of both type of data is – (first data is simple raw text and second data is tab separated).

```

1 -----
2 Name - Nidhi Mantri
3 Emp # - 1101483
4 -----
5 Daily Learning - 1
6
7 1. Apache Spark Overview
8   - Short History of Apache Spark
9   - Who is Databricks?
10  - What is Apache Spark?
11  - Spark - A Unifying Engine
12  - The RDDs and The DataFrames
13  - Scala, Python, Java, R & SQL
14  - The Cluster: Drivers, Executors, Slots & Tasks
15  - Quick Note on Jobs & Stages
16  - Quick Note on Cluster Management
17

```

OrderID	OrderDate	CustomerName	PhoneNumber	DeliveryAddress
63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2 1319
63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2 1319
63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2 1319
63969	2020-12-01	Tailspin Toys (Sans Souci, SC)	(803) 555-0100	
63969	2020-12-01	Tailspin Toys (Sans Souci, SC)	(803) 555-0100	
63969	2020-12-01	Tailspin Toys (Sans Souci, SC)	(803) 555-0100	
63970	2020-12-01	Wingtip Toys (Birds, IL)	(217) 555-0100	

Let's read first type of data using text format –

```

text_df = spark.read.text("dbfs:/FileStore/tables/DailyLearning.txt")
display(text_df)

```

Reading data from text file

```

1 #/FileStore/tables/DailyLearning.txt
2 text_df = spark.read.text("dbfs:/FileStore/tables/DailyLearning.txt")
3 display(text_df)

```

(1) Spark Jobs

text\_df pyspark.sql.dataframe.DataFrame = [value: string]

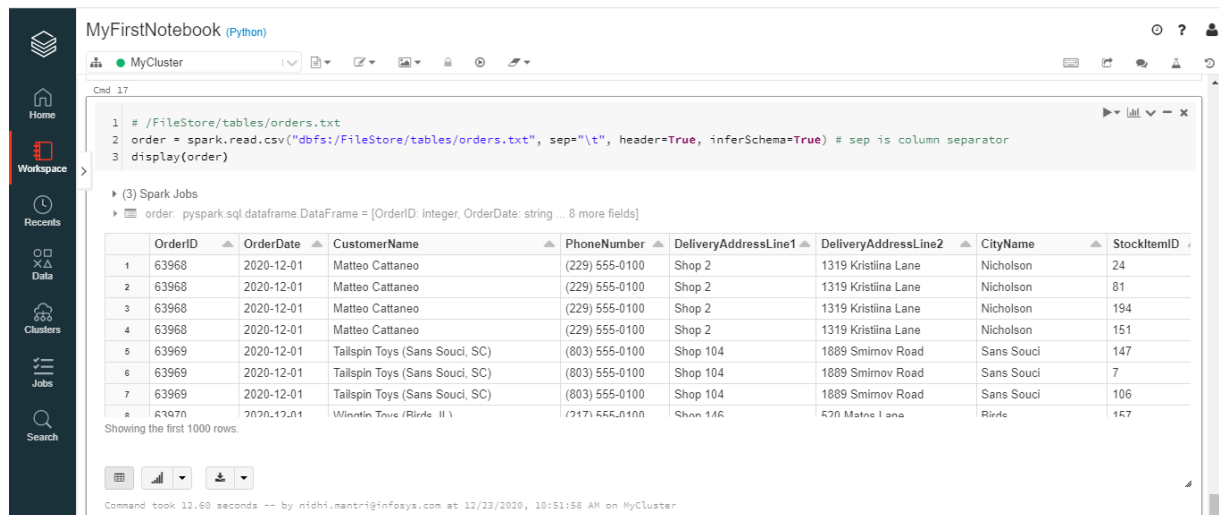
	value
1	-----
2	Name - Nidhi Mantri
3	Emp # - 1101483
4	-----
5	Daily Learning - 1
6	
7	1. Apache Spark Overview
8	- Short History of Apache Spark

Showing all 86 rows.

This text format will consider each new line data as a separate record, and to get some meaningful insights from such type of data, we need to process it.

Now, let's discuss second type of data – *why I mentioned earlier that we need csv format to read such type of data?* – Because we can't specify column separator type of things in text format. It only read data line by line. Therefore, we need csv format. Let's try it –

```
order = spark.read.csv("dbfs:/FileStore/tables/orders.txt", sep="\t", header=True, inferSchema=True) # sep is column separator
display(order)
```



Command took 12.60 seconds -- by nidhi.mantri@infosys.com at 12/23/2020, 10:51:58 AM on MyCluster

	OrderID	OrderDate	CustomerName	PhoneNumber	DeliveryAddressLine1	DeliveryAddressLine2	CityName	StockItemID
1	63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2	1319 Kristina Lane	Nicholson	24
2	63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2	1319 Kristina Lane	Nicholson	81
3	63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2	1319 Kristina Lane	Nicholson	194
4	63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2	1319 Kristina Lane	Nicholson	151
5	63969	2020-12-01	Tailspin Toys (Sans Souci, SC)	(803) 555-0100	Shop 104	1889 Smimov Road	Sans Souci	147
6	63969	2020-12-01	Tailspin Toys (Sans Souci, SC)	(803) 555-0100	Shop 104	1889 Smimov Road	Sans Souci	7
7	63969	2020-12-01	Tailspin Toys (Sans Souci, SC)	(803) 555-0100	Shop 104	1889 Smimov Road	Sans Souci	106

We got the correct table from this text file using csv format. Are you Still want to know? What will be the output when I use text format? Let me try it –

```
order = spark.read.text("dbfs:/FileStore/tables/orders.txt", sep="\t", header=True)
display(order)
```



Command took 0.30 seconds -- by nidhi.mantri@infosys.com at 12/23/2020, 10:57:44 AM on MyCluster

**TypeError:** text() got an unexpected keyword argument 'sep'

See, we got the error. Because text data has no separator, header kind of things. Let me remove this and see the output.

```
order = spark.read.text("dbfs:/FileStore/tables/orders.txt")
display(order)
```

MyFirstNotebook (Python)

MyCluster

Command took 0.30 seconds -- by nidhi.mantri@infosys.com at 12/23/2020, 10:57:44 AM on MyCluster

```
1 order = spark.read.text("dbfs:/FileStore/tables/orders.txt")
2 display(order)
```

▶ (1) Spark Jobs

▶ order: pyspark.sql.dataframe.DataFrame = [value: string]

	value
1	OrderID OrderDate CustomerName PhoneNumber DeliveryAddressLine1 DeliveryAddressLine2 CityName StockItemID Quantity UnitPrice
2	63968 2020-12-01 Matteo Cattaneo (229) 555-0100 Shop 2 1319 Kristina Lane Nicholson 24 8 13
3	63968 2020-12-01 Matteo Cattaneo (229) 555-0100 Shop 2 1319 Kristina Lane Nicholson 81 108 18
4	63968 2020-12-01 Matteo Cattaneo (229) 555-0100 Shop 2 1319 Kristina Lane Nicholson 194 96 4.1
5	63968 2020-12-01 Matteo Cattaneo (229) 555-0100 Shop 2 1319 Kristina Lane Nicholson 151 7 16
6	63969 2020-12-01 Tailspin Toys (Sans Souci, SC) (803) 555-0100 Shop 104 1889 Sminov Road Sans Souci 147 48 18
7	63969 2020-12-01 Tailspin Toys (Sans Souci, SC) (803) 555-0100 Shop 104 1889 Sminov Road Sans Souci 7 6 32

Showing the first 1000 rows.

Command took 0.69 seconds -- by nidhi.mantri@infosys.com at 12/23/2020, 10:59:57 AM on MyCluster

The output is just each new line as a separate record. So, using csv format for this type of cases is helpful and timesaving.

So, Yeahhh!!! With this we have successfully read the data using various formats(csv, json, table, parquet and text) in PySpark.

---

# Keep Learning!!!

- Nidhi Mantri

Specialist Programmer – Big Data