

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 overthousands 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 –

The screenshot shows the Databricks Community Edition interface. On the left is a dark sidebar with icons for Home, Workspace, Recents, Data, Clusters (which is selected), Jobs, and Search. The main content area has a title "Welcome to databricks". It features three cards: "Explore the Quickstart Tutorial" (spin up a cluster, run queries on preloaded data, and display results in 5 minutes), "Import & Explore Data" (quickly import data, preview its schema, create a table, and query it in a notebook), and "Create a Blank Notebook" (create a notebook to start querying, visualizing, and modeling your data). Below these are sections for "Common Tasks" (New Notebook, Create Table, New Cluster, New Job, New MLflow Experiment), "Recents" (Reading Data 3 - Tables, Reading Data 5 - Text, Reading Data 2 - Parquet, Reading Data 1 - CSV, Agenda - DB-100 - Overview), and "What's new in v3.34" (View latest release notes). At the bottom is a search bar and a taskbar.

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

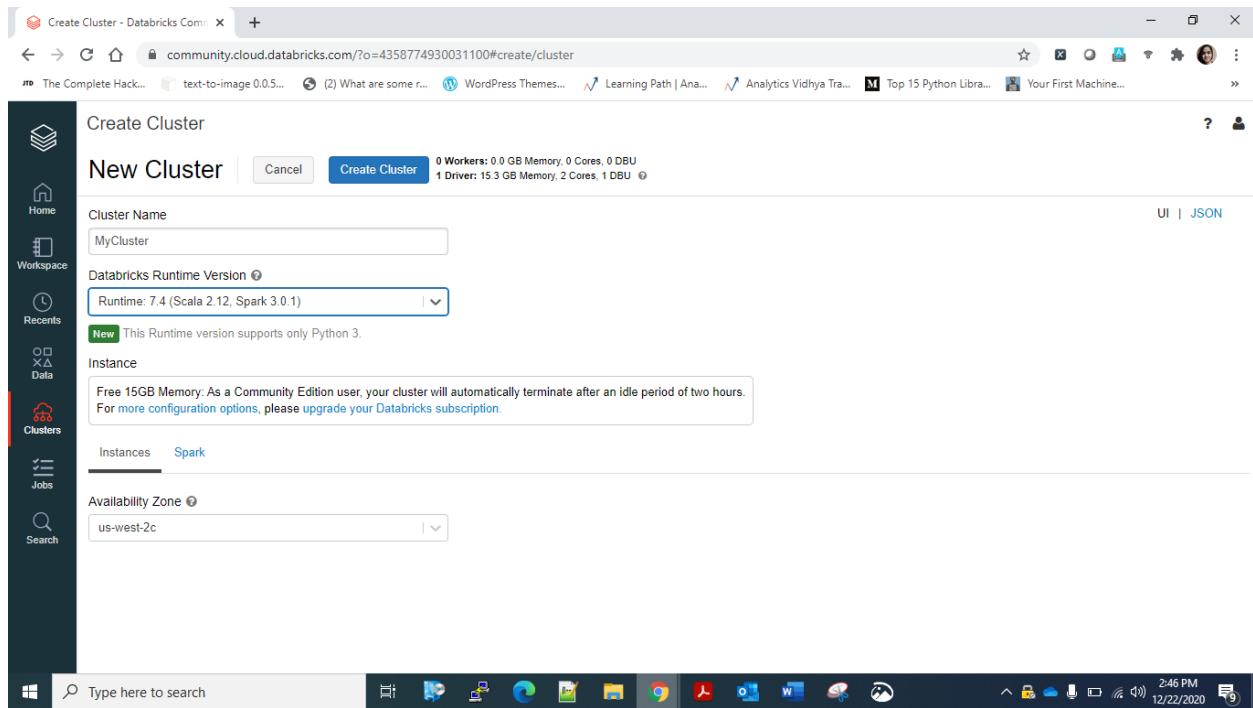
Let's start answering to our first question.

How to create cluster in Databricks? -

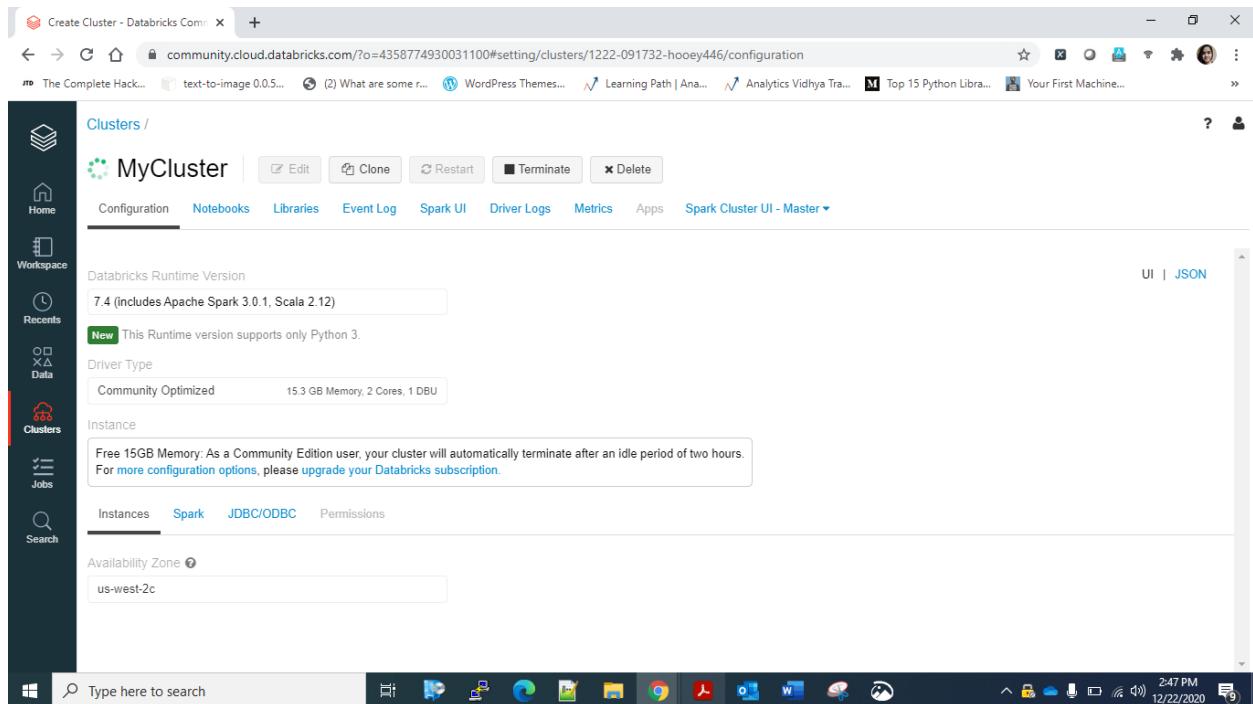
In the right side of the screen, you will find “Clusters” option, click on that. The page will look like this, once you clicked on that button –

The screenshot shows the "Clusters" page in the Databricks Community Edition. The sidebar is identical to the previous screenshot. The main area has a title "Clusters" and tabs for "All-Purpose Clusters" and "Job Clusters" (the latter is selected). A blue button "+ Create Cluster" is visible. Below is a table header with columns: Name, State, Nodes, Runtime, Driver, Worker, Creator, and Actions. The table body displays the message "No Clusters". At the bottom are pagination controls showing "0 - 0 of 0" and "Page Go to 1".

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.

The screenshot shows the Databricks UI for a cluster named 'MyCluster'. The left sidebar has 'Clusters' selected. The main area shows the configuration for 'MyCluster'. Key details include:

- Databricks Runtime Version:** 7.4 (includes Apache Spark 3.0.1, Scala 2.12)
- Driver Type:** Community Optimized (15.3 GB Memory, 2 Cores, 1 DBU)
- Instance:** Free 15GB Memory. A note states: "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."
- Availability Zone:** us-west-2c

At the bottom, there's a search bar and a taskbar with various icons.

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.

The screenshot shows the Databricks UI with the 'Data' option selected in the sidebar. The main area displays a list of tables and a central workspace for data management. Key features shown include:

- Create Table:** A prominent button at the top right of the workspace.
- Data Import:** A section titled "Import & Explore Data" with a placeholder "Drop files or click to browse" and a "Cloud" icon.
- Notebook Creation:** A button labeled "Create a Blank Notebook" with a notebook icon.
- What's new:** A section titled "What's new in v3.34" with a link to "View latest release notes".

At the bottom, there's a search bar and a taskbar with various icons.

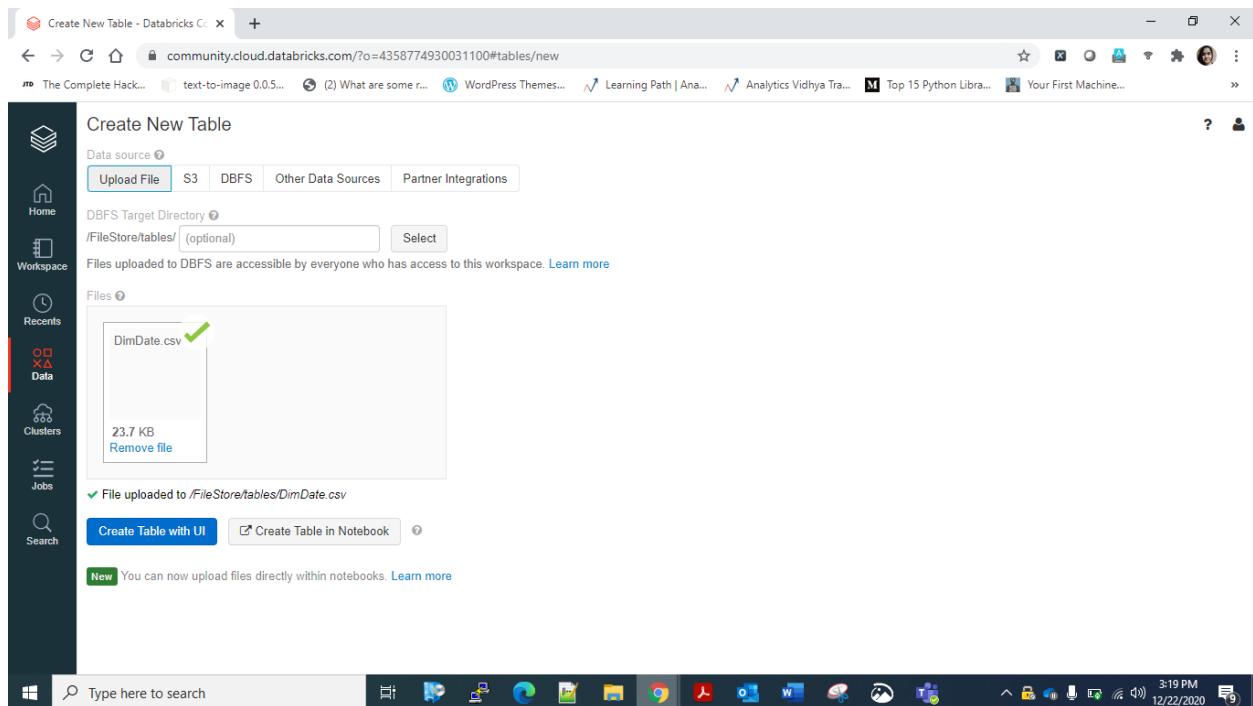
Click on “Create Table” button, the UI will look like this –

The screenshot shows the 'Create New Table' interface in Databricks. On the left is a sidebar with icons for Home, Workspace, Recents, Data (which is selected), Clusters, Jobs, and Search. The main area has tabs for 'Data source' (Upload File, S3, DBFS, Other Data Sources, Partner Integrations), 'DBFS Target Directory' (set to '/FileStore/tables/'), and a note about files being accessible to everyone. Below is a 'Files' section with a 'Drop files to upload, or browse.' button and a green 'New' message about uploading files directly from notebooks. The bottom navigation bar includes a search bar and various Windows system icons.

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".

This screenshot is similar to the previous one but includes a 'File Explorer' window overlaid on the 'Create New Table' interface. The window shows a list of files in the 'AdventureWorks_DBFS' folder, including DimDate, DimEmployee, DimGeography, DimProduct, DimProductCategory, DimProductSubcategory, DimReseller, DimSalesTerritory, and FactResellerSales. The 'Open' button at the bottom of the file browser is highlighted.

Select your file(s) and click “Open”.

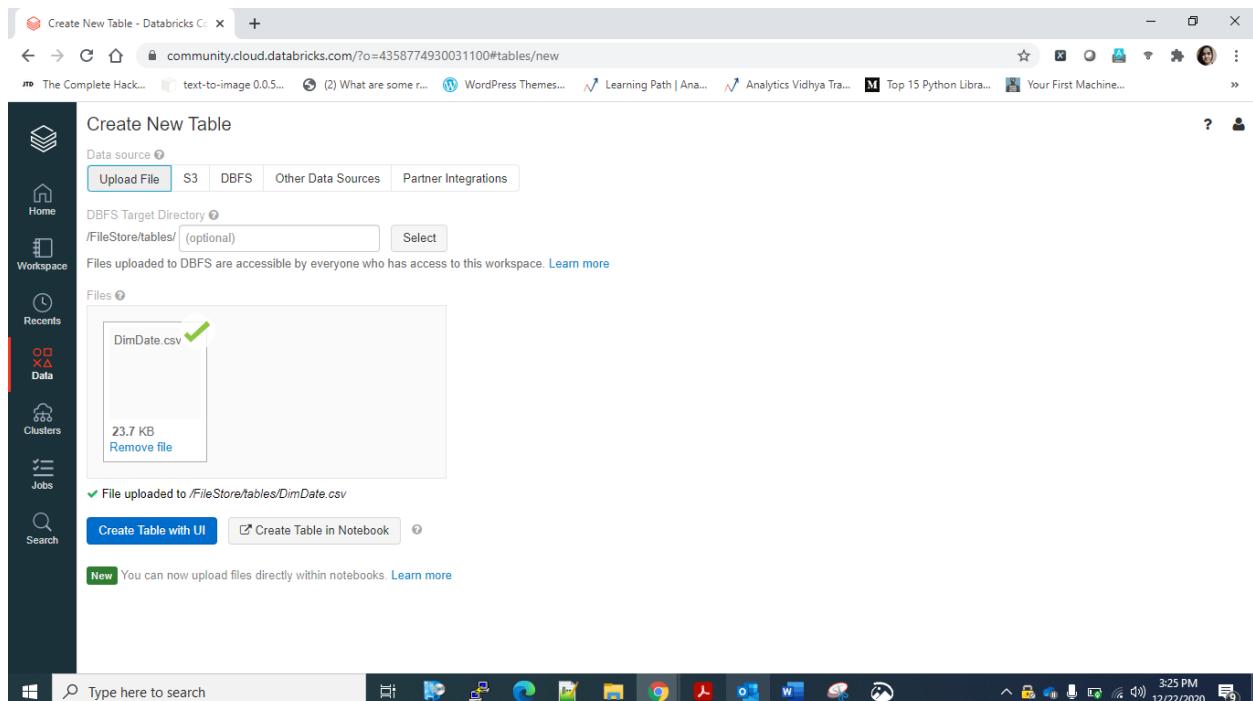


You can see that our file is successfully uploaded to “/FileStore/tables/DimDate.csv”

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.

The screenshot shows the 'Create New Table' interface in Databricks. On the left sidebar, the 'Data' icon is selected. In the main area, a file named 'DimDate.csv' is uploaded to the '/FileStore/tables/' directory. Below the file list, there is a message indicating the file was uploaded successfully. Two buttons are present: 'Create Table with UI' (highlighted in blue) and 'Create Table in Notebook'. A section titled 'Select a Cluster to Preview the Table' follows, with a dropdown menu set to 'MyCluster'. A note at the bottom says 'You can now upload files directly within notebooks. Learn more'. The bottom navigation bar includes a search bar and various system icons.

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

The screenshot shows the 'Create New Table' interface in Databricks. The 'Table Name' field is filled with 'dimdate_csv'. The 'Preview Table' button is highlighted in blue. Below it, the 'Specify Table Attributes' section is visible, containing fields for 'Table Name' (dimdate_csv), 'Create in Database' (default), 'File Type' (CSV), 'Column Delimiter' (,), and checkboxes for 'First row is header', 'Infer schema', and 'Multi-line'. A 'Table Preview' section shows a list of date entries from 2006 to 2007. At the bottom, there is a 'Create Table' button. The bottom navigation bar includes a search bar and various system icons.

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 –

The screenshot shows the 'Create New Table' interface in Databricks. On the left, there's a sidebar with icons for Home, Workspace, Recents, Data (which is selected), Clusters, Jobs, and Search. The main area is titled 'Create New Table' and has a sub-section 'Specify Table Attributes'. It asks for the 'Table Name' (dimdate_csv), 'Create in Database' (default), 'File Type' (CSV), and 'Column Delimiter' (,). There are checkboxes for 'First row is header' (checked) and 'Infer schema' (checked). Below these are two tables: 'Table Preview' and 'Sample Data'. The 'Table Preview' table has columns 'DateKey' and 'Date', with data rows: 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. The 'Sample Data' table also has columns 'DateKey' and 'Date', with data rows: 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. A note at the bottom says '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.

The screenshot shows the 'Table: dimdate_csv' view in Databricks. The sidebar is identical to the previous screenshot. The main area shows the table name 'dimdate_csv' and a 'Schema:' section with a table:

	col_name	data_type	comment
1	DateKey	int	null
2	Date	string	null

Below the schema is a message 'Showing all 2 rows.' and a 'Sample Data:' section with a table:

	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

Below the sample data is a message 'Showing all 20 rows.'

Let's go to next question.

How to create notebook in Databricks with UI? -

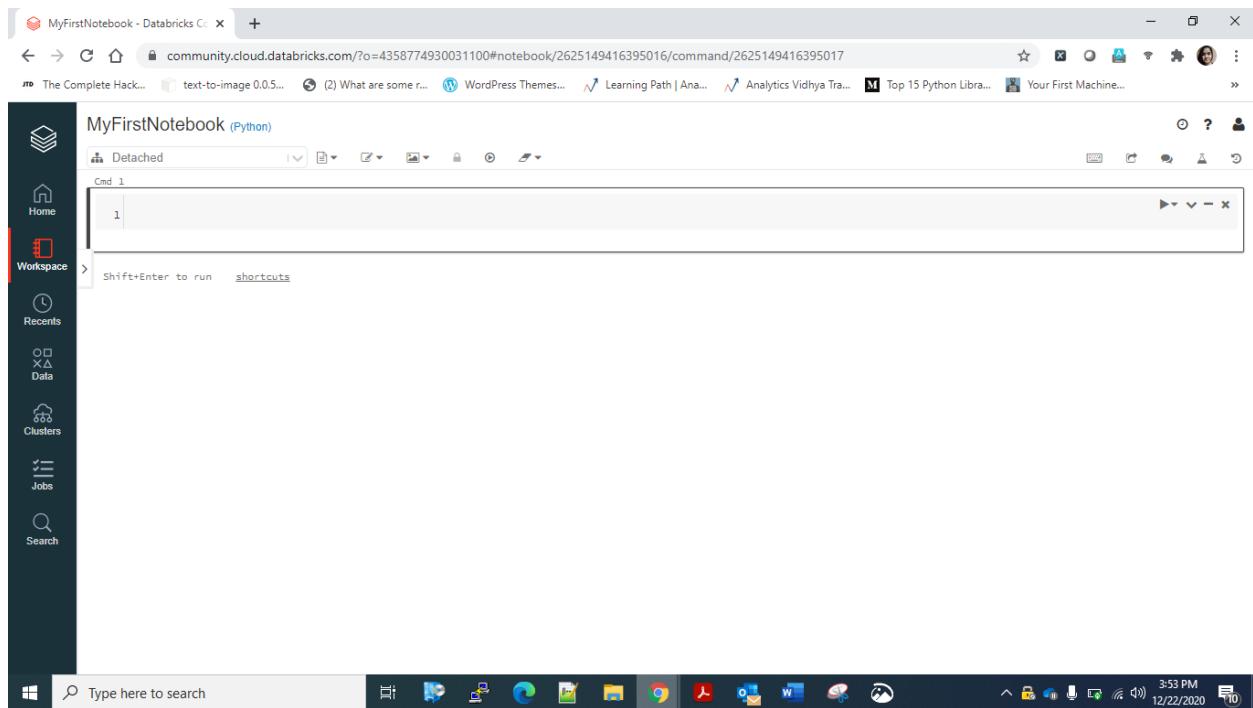
Click on “New Notebook” button.

The screenshot shows the Databricks Community Edition interface. On the left is a dark sidebar with icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area has a title "Welcome to databricks". It features three cards: "Explore the Quickstart Tutorial" (Spin up a cluster, run queries on preloaded data, and display results in 5 minutes), "Import & Explore Data" (Quickly import data, preview its schema, create a table, and query it in a notebook), and "Create a Blank Notebook" (Create a notebook to start querying, visualizing, and modeling your data). Below these are sections for "Common Tasks" (New Notebook, Create Table, New Cluster, New Job, New MLflow Experiment), "Recents" (Reading Data 3 - Tables, Reading Data 5 - Text, Reading Data 2 - Parquet, Reading Data 1 - CSV), and "What's new in v3.34" (View latest release notes). At the bottom is a search bar and a taskbar with various icons.

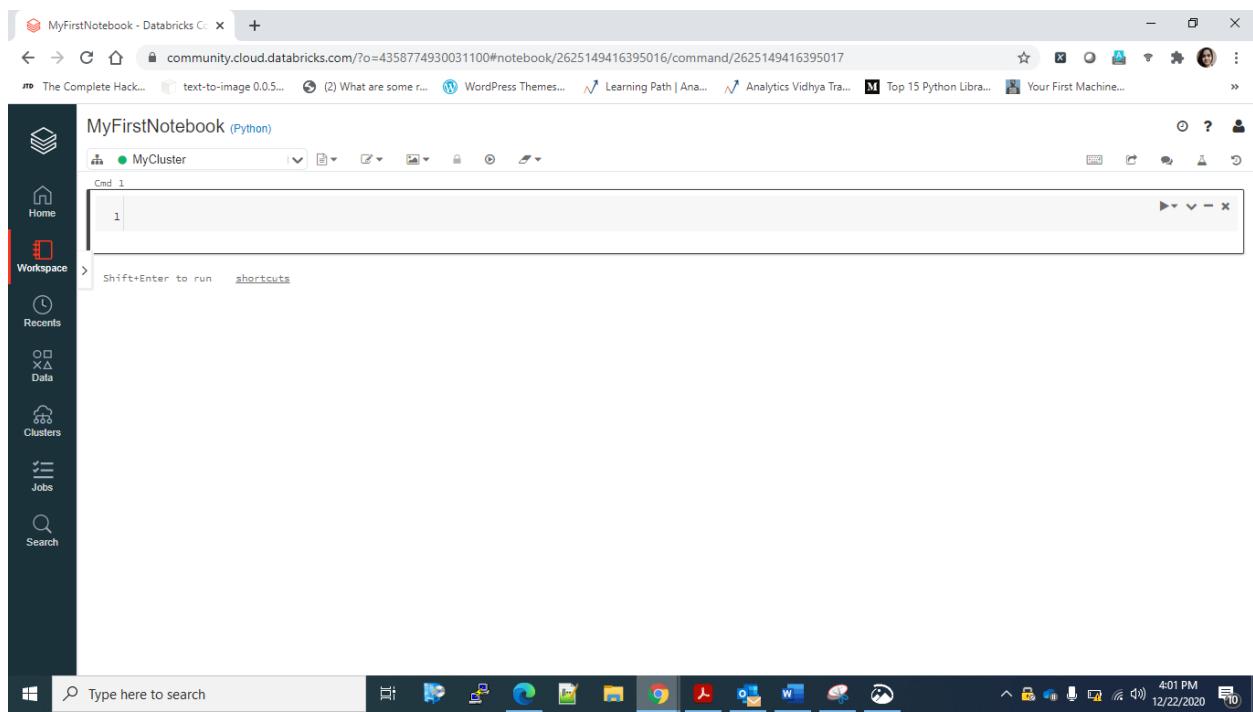
Enter your Notebook name and click the “Create” button.

This screenshot shows the "Create Notebook" dialog box overlaid on the Databricks UI. The dialog has fields for "Name" (MyFirstNotebook), "Default Language" (Python), and "Cluster" (MyCluster). There are "Cancel" and "Create" buttons at the bottom. The background shows the same workspace elements as the previous screenshot, including the sidebar and the "Create a Blank Notebook" card.

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.

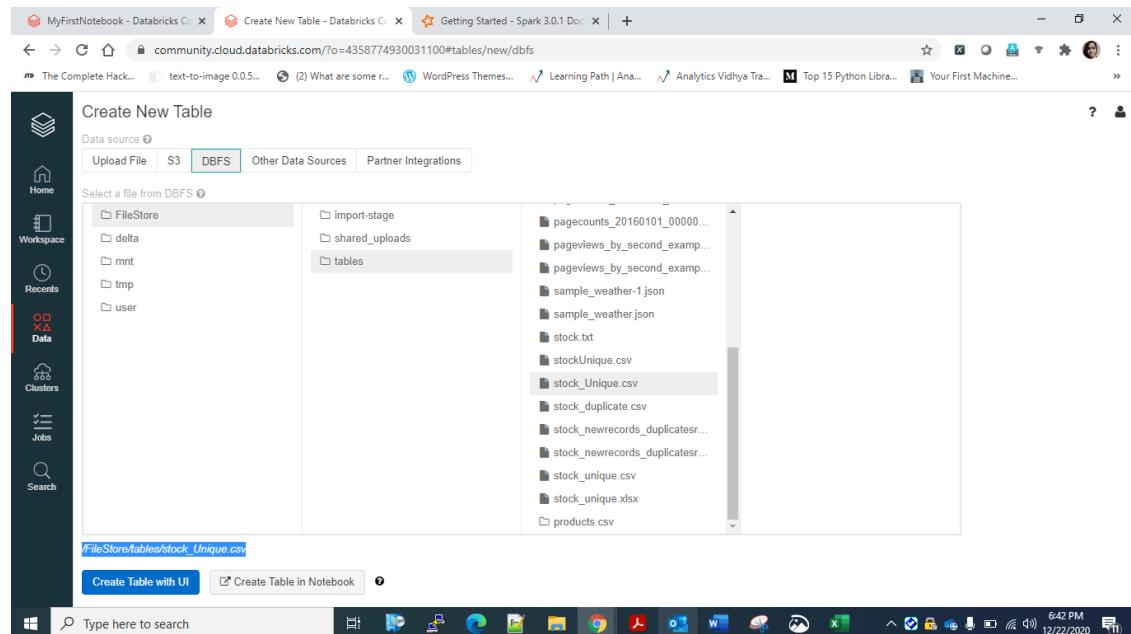
Let's get back to our main task – "Reading data using various formats in Pyspark Dataframes."

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) –



The screenshot shows the Databricks interface for creating a new table. On the left, there's a sidebar with 'Home', 'Workspace', 'Recents', 'Data' (which is currently selected), 'Clusters', 'Jobs', and 'Search'. The main area is titled 'Create New Table' and has tabs for 'Data source' (with 'DBFS' selected), 'Upload File', 'S3', 'Other Data Sources', and 'Partner Integrations'. Below the tabs, it says 'Select a file from DBFS'. A tree view shows 'FileStore' with subfolders 'delta', 'mnt', 'tmp', and 'user'. Under 'FileStore', 'tables' is expanded, showing files like 'import-stage', 'shared_uploads', 'pagecounts_20160101_00000...', 'pageviews_by_second_exampl...', 'sample_weather-1.json', 'sample_weather.json', 'stock.txt', 'stockUnique.csv', 'stocks_Unique.csv' (which is highlighted in blue), 'stock_duplicate.csv', 'stock_newrecords_duplicates...', 'stock_newrecords_duplicates...', 'stock_unique.csv', and 'stock_unique.xlsx'. At the bottom, there are buttons for 'Create Table with UI' and 'Create Table in Notebook'. The status bar at the bottom shows 'Type here to search' and various system icons.

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
8	165	11-Dec-20	210	150	60

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([
    StructField("product_id", IntegerType(), True),      #represent 1 row(list of columns)
    StructField("sale_date", StringType(), True),        #represent 1 column
    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 the Databricks workspace interface. On the left is the sidebar with icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area contains the Python code for reading the CSV file with a schema. Below the code, a table titled '(1) Spark Jobs' shows the resulting DataFrame named 'stocks'. The table has columns: product_id, sale_date, total_qty, available_qty, and sale_qty. The data is as follows:

	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

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"
```

```
Cmd 4
1 %fs head "dbfs:/FileStore/tables/sample_file.json"

{"isRobot":false,"channel":"#pl.wikipedia","timestamp":"2016-05-26T19:19:14.367Z","url":"https://pl.wikipedia.org/w/index.php?diff=45897271&oldid=42573382","isUnpatrolled":false,"page":"Banda bat'ki Knysza","wikipedia":"pl","wikipediaURL":"http://pl.wikipedia.org","comment":"plik, drobne techniczne, kąt. ","userURL":"http://pl.wikipedia.org/wiki/Czynjestrlogika","pageURL":"http://pl.wikipedia.org/wiki/Banda_bat'ki_Knysza","delta":16,"flag":null,"isNewPage":false,"isAnonymous":false,"geocoding":{"countryCode2":null,"city":null,"latitude":null,"longitude":null,"country":null,"stateProvince":null,"countryCode3":null},"user":"Czynjestrlogika","namespace":"article"}  
{"isRobot":false,"channel":"#fr.wikipedia","timestamp":"2016-05-26T19:19:14.515Z","url":"https://fr.wikipedia.org/w/index.php?diff=126531898&oldid=118306121&cid=188792265","isUnpatrolled":true,"page":"L'Éprise (roman)","wikipedia":"fr","wikipediaURL":"http://fr.wikipedia.org","comment":"/ Résumé */","userURL":"http://fr.wikipedia.org/wiki/User:Polipille","pageURL":"http://fr.wikipedia.org/wiki/La_Méprise_(roman)","delta":-1,"flag":IM,"isNewPage":false,"isAnonymous":false,"geocoding":{"countryCode2":null,"city":null,"latitude":null,"country":null,"longitude":null,"stateProvince":null,"countryCode3":null},"user":"Polipille","namespace":"article"}  
{"isRobot":true,"channel":"#en.wikipedia","timestamp":"2016-05-26T19:19:14.586Z","url":"https://en.wikipedia.org/w/index.php?diff=722229003&oldid=720692086","isUnpatrolled":false,"page":"Carlos Zeballos","wikipedia":"en","wikipediaURL":"http://en.wikipedia.org","comment":"Rescuing 1 sources. #IABot","userURL":"http://en.wikipedia.org/wiki/User:Cyberbot_II","pageURL":"http://en.wikipedia.org/wiki/Carlos_Zeballos","delta":10,"flag":B,"isNewPage":false,"isAnonymous":false,"geocoding":{"countryCode2":null,"city":null,"latitude":null,"country":null,"longitude":null,"stateProvince":null,"countryCode3":null},"user":"Cyberbot_II","namespace":"article"}  
Command took 0.45 seconds -- by nidi.mantri@infosys.com at 12/22/2020, 7:47:33 PM on MyCluster
```

Let's try reading the data into dataframe.

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

```
display(wiki_data)
```

	channel	comment	delta	flag	geocoding	is
1	#pl.wikipedia	plik, drobne techniczne, kąt.	116		["city": null, "country": null, "countryCode2": null, "countryCode3": null, "latitude": null, "longitude": null, "stateProvince": null]	fe
2	#fr.wikipedia	/* Résumé */	-1	IM	["city": null, "country": null, "countryCode2": null, "countryCode3": null, "latitude": null, "longitude": null, "stateProvince": null]	fe
3	#en.wikipedia	Rescuing 1 sources. #IABot	110	B	["city": null, "country": null, "countryCode2": null, "countryCode3": null, "latitude": null, "longitude": null, "stateProvince": null]	fe

Showing all 3 rows.

```
Cmd 6
1
```

Type here to search

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()
```

```
1 wiki_data.printSchema()

root
|-- channel: string (nullable = true)
|-- comment: string (nullable = true)
|-- delta: long (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)
|   |-- latitude: string (nullable = true)
|   |-- longitude: string (nullable = true)
|   |-- stateProvince: string (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 0.03 seconds -- by nidhi.mantri@infosys.com at 12/22/2020, 8:07:43 PM on MyCluster
```

Cmd 8

```
1 |
```

Type here to search

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)
```

```
Cmd 6
1 wiki_data = spark.read.json("dbfs:/FileStore/tables/sample_file.json", inferSchema=True)
2 display(wiki_data)

TypeError: json() got an unexpected keyword argument 'inferSchema'
Command took 0.05 seconds -- by nidhi.mantri@infosys.com at 12/22/2020, 8:04:30 PM on MyCluster
```

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()
```

The screenshot shows the Databricks workspace interface. On the left is a sidebar with icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area displays the schema of a DataFrame named 'wiki_data'. The schema includes fields such as channel, comment, delta, flag, geocoding, city, country, countryCode2, countryCode3, stateProvince, latitude, longitude, isAnonymous, isNewPage, isRobot, isUnpatrolled, namespace, page, pageURL, and timestamp. A status bar at the bottom indicates the command took 0.31 seconds.

```
|-- 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 0.31 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)
```

The screenshot shows the Databricks workspace interface. The main area displays the contents of the 'stocks_parquet' DataFrame. The schema includes product_id, sale_date, total_qty, available_qty, and sale_qty. The data shows 10 rows of sales information for various products on December 11, 2020. A status bar at the bottom indicates the command took 5.67 seconds.

	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

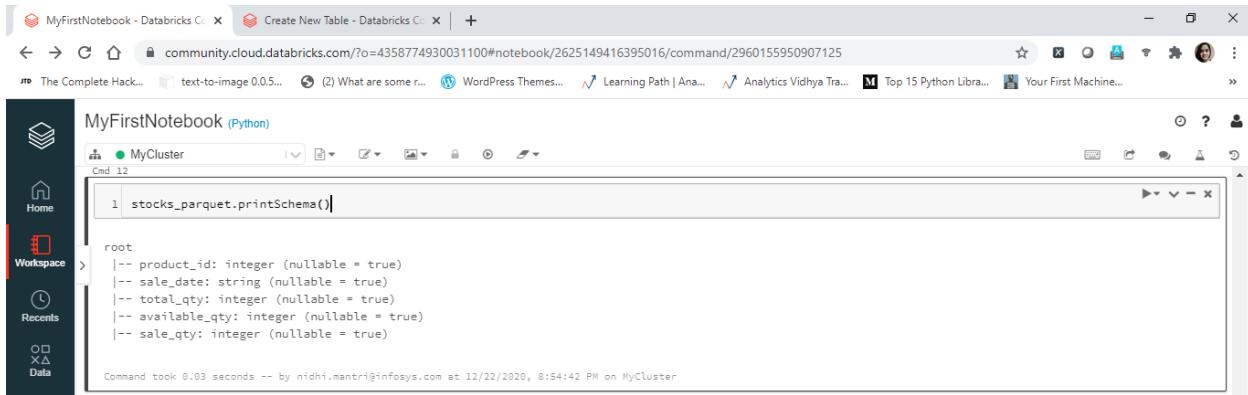
```
stocks_parquet = spark.read.parquet("dbfs:/FileStore/tables/stock_Unique.parquet")
stocks_parquet
# (2) Spark Jobs
stocks_parquet: pyspark.sql.dataframe.DataFrame = [product_id: integer, sale_date: string ... 3 more fields]

+-----+-----+-----+-----+-----+
|product_id|sale_date|total_qty|available_qty|sale_qty|
+-----+-----+-----+-----+-----+
|      1| 11-Dec-20|     1000|        998|       2|
|      2| 11-Dec-20|     2000|       1500|     500|
|      3| 11-Dec-20|      500|        350|     150|
|      4| 11-Dec-20|      300|        200|     100|
|      5| 11-Dec-20|      650|        470|     180|
|     70| 11-Dec-20|    10000|       9800|     200|
|     130| 11-Dec-20|      200|        150|      50|
|     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()
```



The screenshot shows a Databricks Notebook interface. The left sidebar has 'Home', 'Workspace' (selected), 'Recents', and 'Data'. The main area is titled 'MyFirstNotebook (Python)' and shows a command cell 'Cmd 12' with the code 'stocks_parquet.printSchema()' and its output:

```
1 stocks_parquet.printSchema()

root
 |-- product_id: integer (nullable = true)
 |-- sale_date: string (nullable = true)
 |-- total_qty: integer (nullable = true)
 |-- available_qty: integer (nullable = true)
 |-- sale_qty: integer (nullable = true)

Command took 0.03 seconds -- by nidhi.mentri@infosys.com at 12/22/2020, 8:54:42 PM on MyCluster
```

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([
    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.

Cmd 14

```
1
```

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

10:34 PM 12/22/2020

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

```

Type here to search

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	

File	Edit	Format	View	Help
orders - Notepad				
OrderID	OrderDate	CustomerName	PhoneNumber	DeliveryAddr
63968	2020-12-01	Matteo Cattaneo (229) 555-0100	Shop 2	1315
63968	2020-12-01	Matteo Cattaneo (229) 555-0100	Shop 2	1315
63968	2020-12-01	Matteo Cattaneo (229) 555-0100	Shop 2	1315
63968	2020-12-01	Matteo Cattaneo (229) 555-0100	Shop 2	1315
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

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)
```

The screenshot shows the Databricks workspace interface. On the left is the sidebar with icons for Home, Workspace (which is selected), Data, Clusters, Jobs, and Search. The main area is titled 'MyFirstNotebook (Python)' and shows a cluster named 'MyCluster'. A command cell labeled 'Cmd 17' contains the following code:

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

Below the code, it says '(3) Spark Jobs' and 'order: pyspark.sql.dataframe.DataFrame = [OrderID: integer, OrderDate: string ... 8 more fields]'. The main pane displays a table with 8 columns: OrderID, OrderDate, CustomerName, PhoneNumber, DeliveryAddressLine1, DeliveryAddressLine2, CityName, and StockItemID. The data consists of 8 rows, each representing an order. The last row is partially visible.

OrderID	OrderDate	CustomerName	PhoneNumber	DeliveryAddressLine1	DeliveryAddressLine2	CityName	StockItemID
1	63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2	1319 Kristina Lane	Nicholson
2	63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2	1319 Kristina Lane	Nicholson
3	63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2	1319 Kristina Lane	Nicholson
4	63968	2020-12-01	Matteo Cattaneo	(229) 555-0100	Shop 2	1319 Kristina Lane	Nicholson
5	63969	2020-12-01	Tailspin Toys (Sans Souci, SC)	(803) 555-0100	Shop 104	1889 Smirnov Road	Sans Souci
6	63969	2020-12-01	Tailspin Toys (Sans Souci, SC)	(803) 555-0100	Shop 104	1889 Smirnov Road	Sans Souci
7	63969	2020-12-01	Tailspin Toys (Sans Souci, SC)	(803) 555-0100	Shop 104	1889 Smirnov Road	Sans Souci
8	63970	2020-12-01	Wingstop (Riverte II)	(217) 555-0100	Shop 106	120 Matina Lane	Riverte

At the bottom of the cell, it says 'Showing the first 1000 rows.' and 'Command took 12.60 seconds -- by nidhi.mantri@infosys.com at 12/23/2020, 10:51:58 AM on MyCluster'.

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)
```

The screenshot shows the Databricks workspace interface. The sidebar is identical to the previous one. A command cell labeled 'Cmd 18' contains the following code:

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

Below the code, it says 'TypeError: text() got an unexpected keyword argument 'sep''. At the bottom, it says 'Command took 0.30 seconds -- by nidhi.mantri@infosys.com at 12/23/2020, 10:57:44 AM on MyCluster'.

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
Cmd 19
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
2 | Quantity UnitPrice
3 | 63968 2020-12-01 Matteo Cattaneo (229) 555-0100 Shop 2 1319 Kristina Lane Nicholson 24 8 13
4 | 63968 2020-12-01 Matteo Cattaneo (229) 555-0100 Shop 2 1319 Kristina Lane Nicholson 81 108 18
5 | 63968 2020-12-01 Matteo Cattaneo (229) 555-0100 Shop 2 1319 Kristina Lane Nicholson 194 96 4 1
6 | 63968 2020-12-01 Matteo Cattaneo (229) 555-0100 Shop 2 1319 Kristina Lane Nicholson 151 7 16
7 | 63969 2020-12-01 Tailspin Toys (Sans Souci, SC) (803) 555-0100 Shop 104 1889 Smirnov Road Sans Souci 147 48 18
8 | 63969 2020-12-01 Tailspin Toys (Sans Souci, SC) (803) 555-0100 Shop 104 1889 Smirnov Road Sans Souci 7 6 32
Showing the first 1000 rows.

Cmd 29
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