



EPAM Systems, RD Dep., RD Dep.

DATABRICKS OVERVIEW

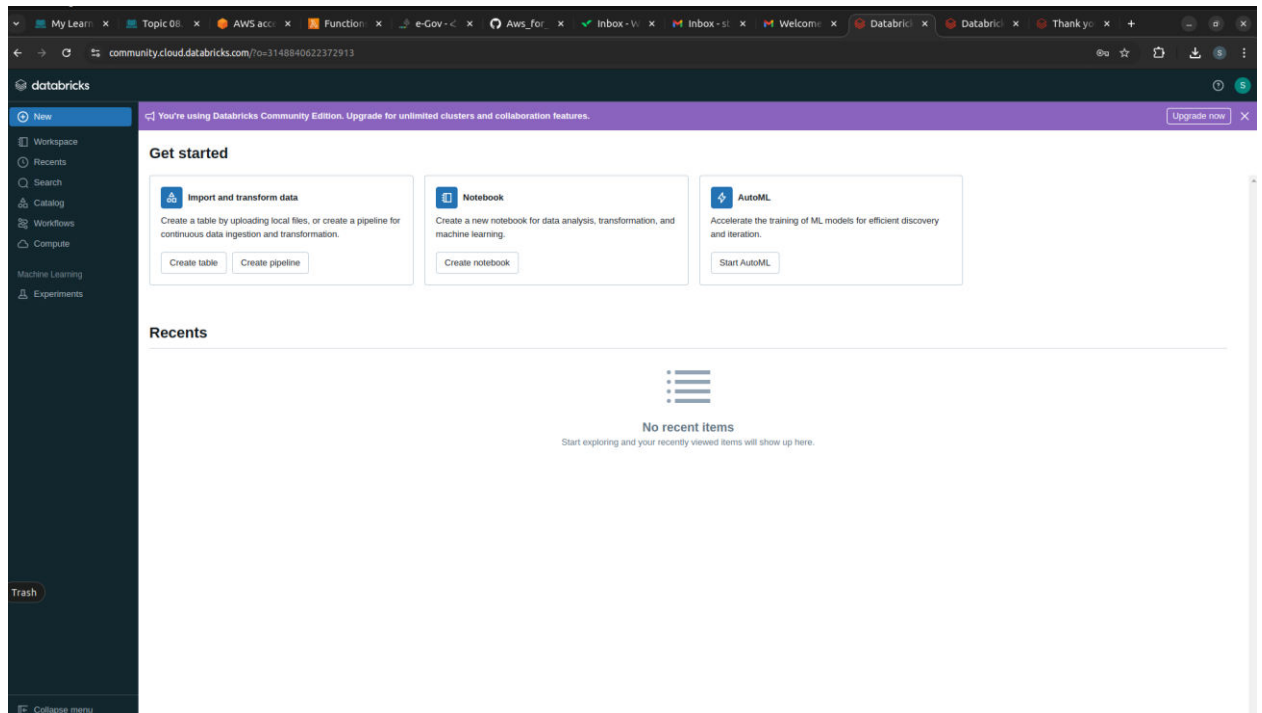
DQE

Legal Notice:

This document contains privileged and/or confidential information and may not be disclosed, distributed or reproduced without the prior written permission of EPAM®.

Confidential

Account created.



Dbc file imported in Databricks.

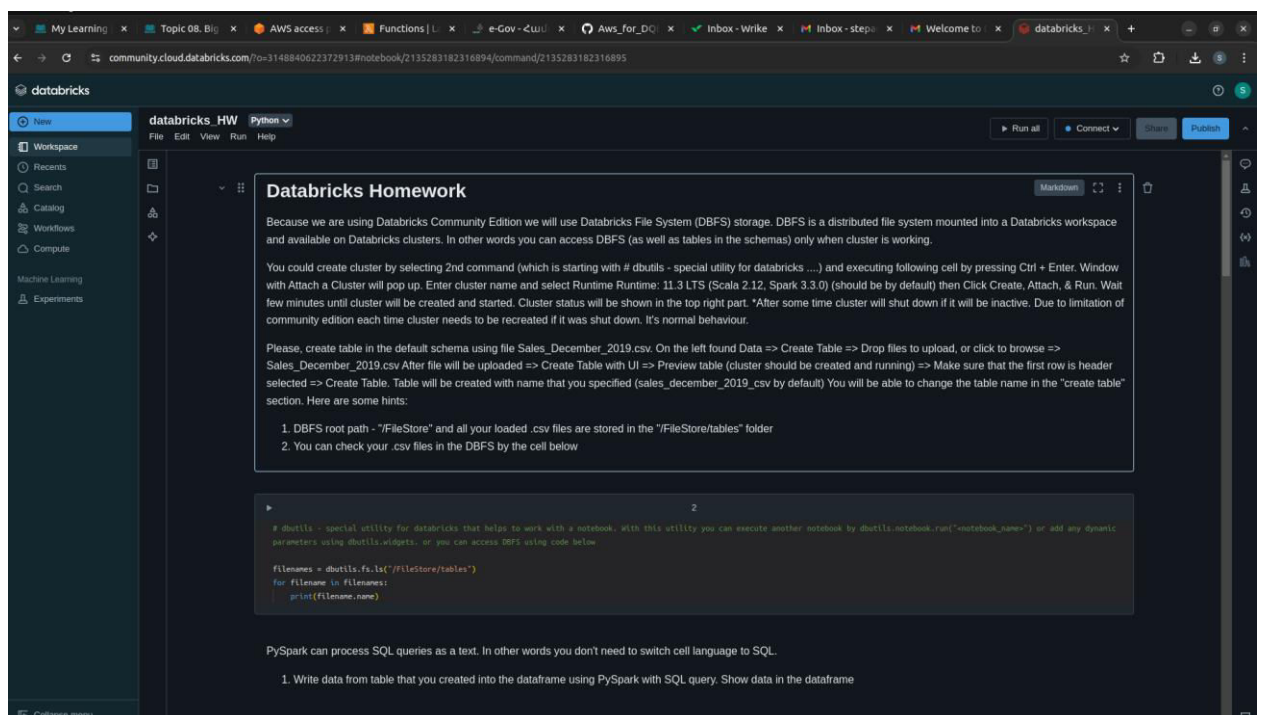
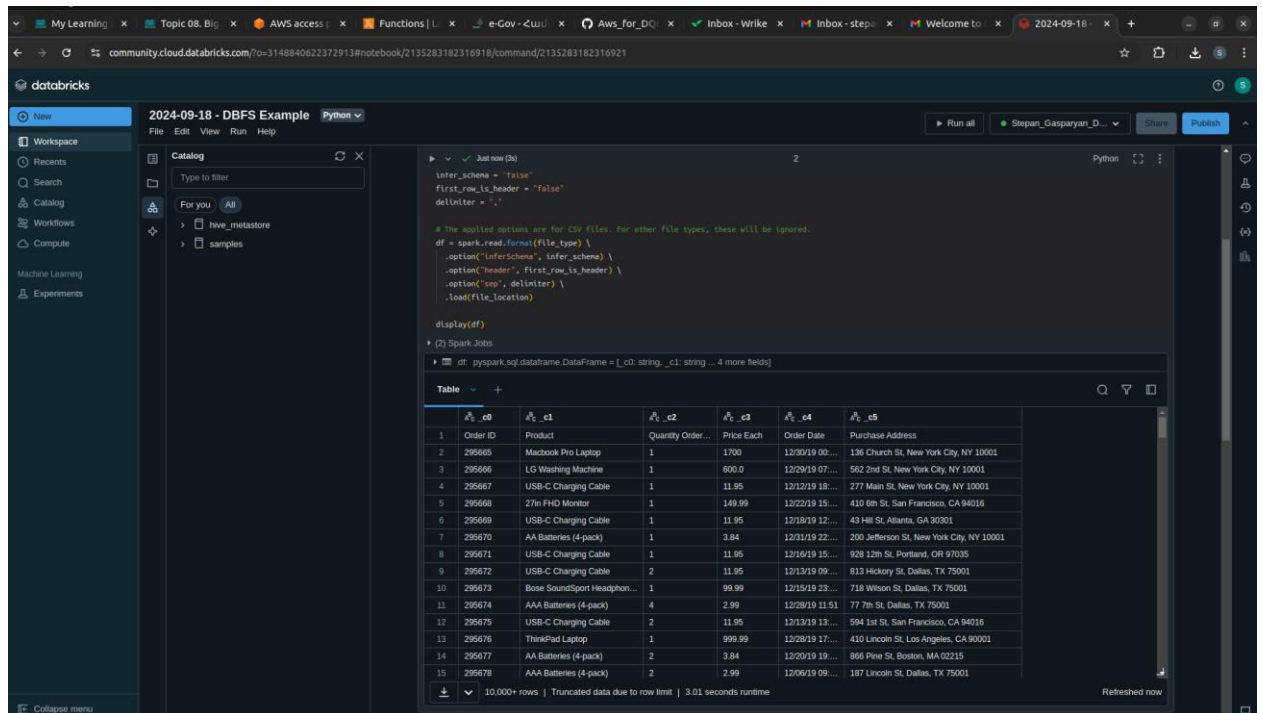


Table created from provided csv file .



The screenshot shows a Databricks workspace with a Python notebook titled "2024-09-18 - DBFS Example". The notebook contains the following code:

```
infer_schema = "false"
first_row_is_header = "false"
delimiter = ","

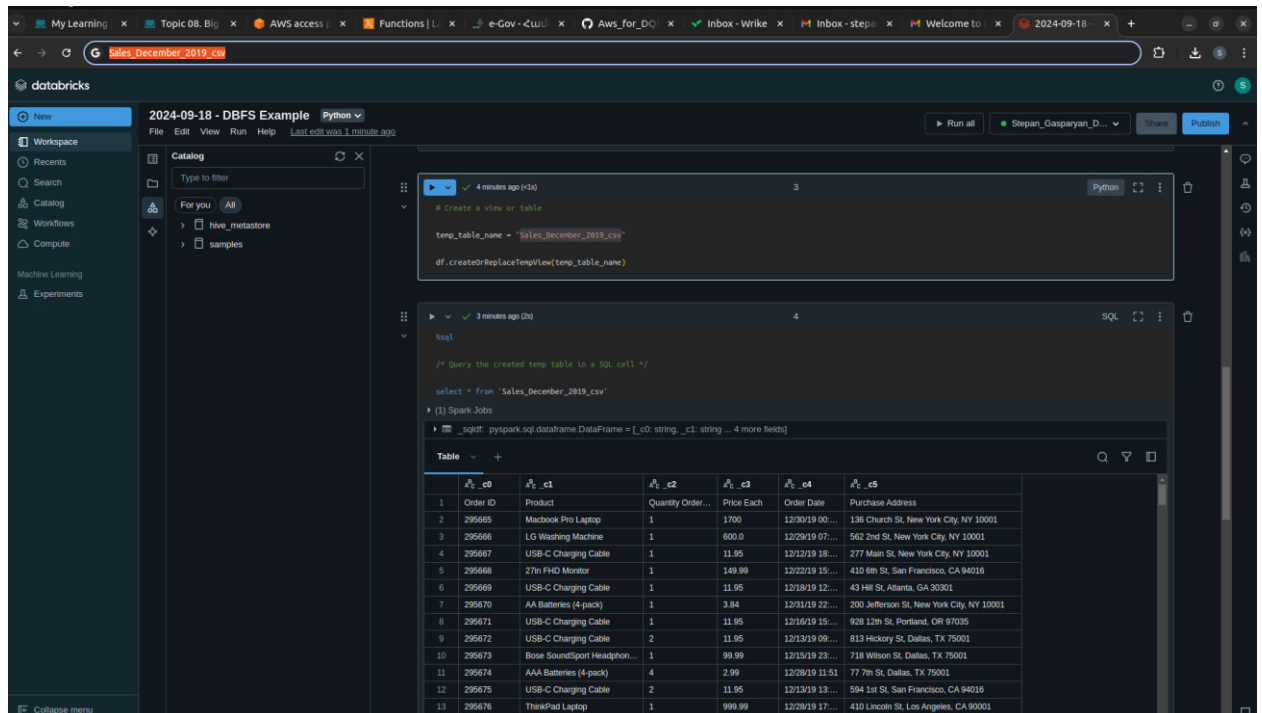
# The applied options are for CSV Files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
    .option("inferSchema", infer_schema) \
    .option("header", first_row_is_header) \
    .option("sep", delimiter) \
    .load(file_location)

display(df)
```

The output shows a Spark DataFrame with 15 rows and 6 columns. The columns are: Order ID, Product, Quantity Order, Price Each, Order Date, and Purchase Address. The data is as follows:

Order ID	Product	Quantity Order	Price Each	Order Date	Purchase Address
295665	Macbook Pro Laptop	1	1700	12/30/19 00...	136 Church St, New York City, NY 10001
295666	LG Washing Machine	1	600.0	12/29/19 07...	562 2nd St, New York City, NY 10001
295667	USB-C Charging Cable	1	11.95	12/12/19 18...	277 Main St, New York City, NY 10001
295668	27in FHD Monitor	1	149.99	12/22/19 15...	410 6th St, San Francisco, CA 94016
295669	USB-C Charging Cable	1	11.95	12/18/19 12...	43 Hill St, Atlanta, GA 30301
295670	AA Batteries (4-pack)	1	3.84	12/11/19 22...	200 Jefferson St, New York City, NY 10001
295671	USB-C Charging Cable	1	11.95	12/16/19 15...	928 12th St, Portland, OR 97035
295672	USB-C Charging Cable	2	11.95	12/13/19 09...	813 Hickory St, Dallas, TX 75001
295673	Bose SoundSport Headphon...	1	99.99	12/15/19 23...	718 Wilson St, Dallas, TX 75001
295674	AAA Batteries (4-pack)	4	2.99	12/28/19 11...	77 7th St, Dallas, TX 75001
295675	USB-C Charging Cable	2	11.95	12/13/19 13...	594 1st St, San Francisco, CA 94016
295676	ThinkPad Laptop	1	999.99	12/28/19 17...	410 Lincoln St, Los Angeles, CA 90001
295677	AA Batteries (4-pack)	2	3.84	12/20/19 19...	866 Pine St, Boston, MA 02215
295678	AAA Batteries (4-pack)	2	2.99	12/06/19 09...	187 Lincoln St, Dallas, TX 75001

TempView created with the name Sales_December_2019_csv ,see data inside.



The screenshot shows a Databricks workspace with a SQL notebook titled "2024-09-18 - DBFS Example". The notebook contains the following code:

```
# Create a view or table
temp_table_name = "Sales_December_2019_csv"
df.createOrReplaceTempView(temp_table_name)

/* Query the created temp table in a SQL cell */
select * from 'Sales_December_2019_csv'
```

The output shows a Spark DataFrame with 15 rows and 6 columns. The columns are: Order ID, Product, Quantity Order, Price Each, Order Date, and Purchase Address. The data is as follows:

Order ID	Product	Quantity Order	Price Each	Order Date	Purchase Address
295665	Macbook Pro Laptop	1	1700	12/30/19 00...	136 Church St, New York City, NY 10001
295666	LG Washing Machine	1	600.0	12/29/19 07...	562 2nd St, New York City, NY 10001
295667	USB-C Charging Cable	1	11.95	12/12/19 18...	277 Main St, New York City, NY 10001
295668	27in FHD Monitor	1	149.99	12/22/19 15...	410 6th St, San Francisco, CA 94016
295669	USB-C Charging Cable	1	11.95	12/18/19 12...	43 Hill St, Atlanta, GA 30301
295670	AA Batteries (4-pack)	1	3.84	12/11/19 22...	200 Jefferson St, New York City, NY 10001
295671	USB-C Charging Cable	1	11.95	12/16/19 15...	928 12th St, Portland, OR 97035
295672	USB-C Charging Cable	2	11.95	12/13/19 09...	813 Hickory St, Dallas, TX 75001
295673	Bose SoundSport Headphon...	1	99.99	12/15/19 23...	718 Wilson St, Dallas, TX 75001
295674	AAA Batteries (4-pack)	4	2.99	12/28/19 11...	77 7th St, Dallas, TX 75001
295675	USB-C Charging Cable	2	11.95	12/13/19 13...	594 1st St, San Francisco, CA 94016
295676	ThinkPad Laptop	1	999.99	12/28/19 17...	410 Lincoln St, Los Angeles, CA 90001

Now we are saving as a table for other users to see the data.

The screenshot shows a Databricks workspace with a notebook titled "2024-09-18 - DBFS Example". The notebook is in Python mode. The top section displays a table with product information:

id	product_name	quantity	price	timestamp	location
33	Wireless Headphones	1	11.99	12/15/19 16...	856 12th St, Atlanta, GA 30301
34	Lightning Charging Cable	1	14.95	12/26/19 18...	310 Pine St, Dallas, TX 75001
35	Lightning Charging Cable	1	14.95	12/14/19 06...	903 Lincoln St, Boston, MA 02215
36	Lightning Charging Cable	1	14.95	12/29/19 13...	253 Hickory St, San Francisco, CA 94016
37	ThinkPad Laptop	1	999.99	12/15/19 07...	742 River St, San Francisco, CA 94016
38	Vareebad Phone	1	400	12/13/19 14...	175 1st St, New York City, NY 10001
39	USB-C Charging Cable	2	11.95	12/13/19 14...	175 1st St, New York City, NY 10001

Below the table, a message states: "This result is stored as `sqldf` and can be used in other Python cells." The bottom section of the notebook shows a code cell with the following Python code:

```
# With this registered as a temp view, it will only be available to this particular notebook. If you'd like other users to be able to query this table, you can also create a table from the DataFrame.
# Once saved, this table will persist across cluster restarts as well as allow various users across different notebooks to query this data.
# To do so, choose your table name and uncomment the bottom line.

permanent_table_name = "sales_december_2019_csv"

df.write.format("parquet").saveAsTable(permanent_table_name)

# (1) Spark Jobs
```

Finally I published the notebook.

The screenshot shows the same Databricks workspace, but with a "Notebook Published" dialog box open. The dialog box contains the following text:

Notebook Published

The notebook was published successfully. Please copy the url and save it (it may take a minute or two for your updates to be publicly available). The link will remain valid for 6 months.

<https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902239c93eaa8714f173bcfc/3148840622372913/2135283182316918/4761590133845256/latest.html>

Done

Below the dialog box, a code cell shows the following Python code:

```
# File location and type
file_location = "/Filestore/tables/sales_december_2019_csv"
file_type = "csv"

# CSV options
infer_schema = "false"
first_row_is_header = "false"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
    .option("inferSchema", infer_schema) \
    .option("header", first_row_is_header) \
    .option("sep", delimiter) \
    .load(file_location)

display(df)

# (2) Spark Jobs
```

The bottom section of the notebook shows a table with product information:

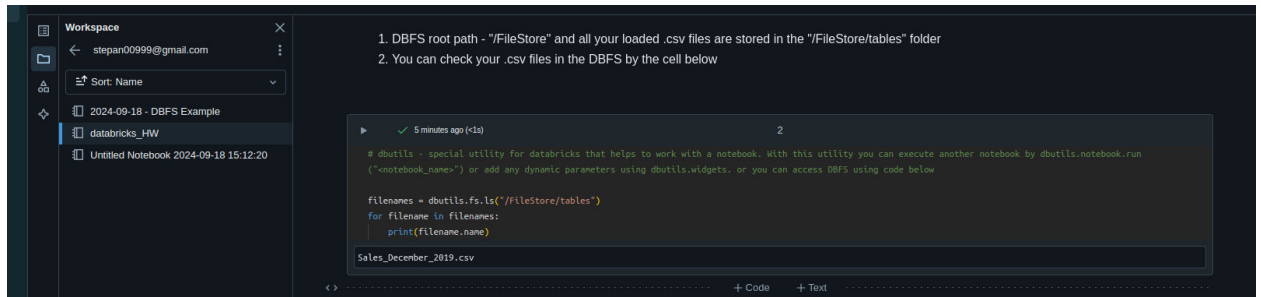
id	product_name	quantity	price	timestamp	location
34	Macbook Pro Laptop	1	1700	12/10/19 20...	311 Madison St, New York City, NY 10001
35	Base SoundSport Headphon...	1	99.99	12/24/19 07...	490 Spruce St, New York City, NY 10001

Link to notebook -

<https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902239c93eaa8714f173bcfc/3148840622372913/2135283182316918/4761590133845256/latest.html>

Now lets do HW file tasks.

1. We can see that we got the file in our file system.



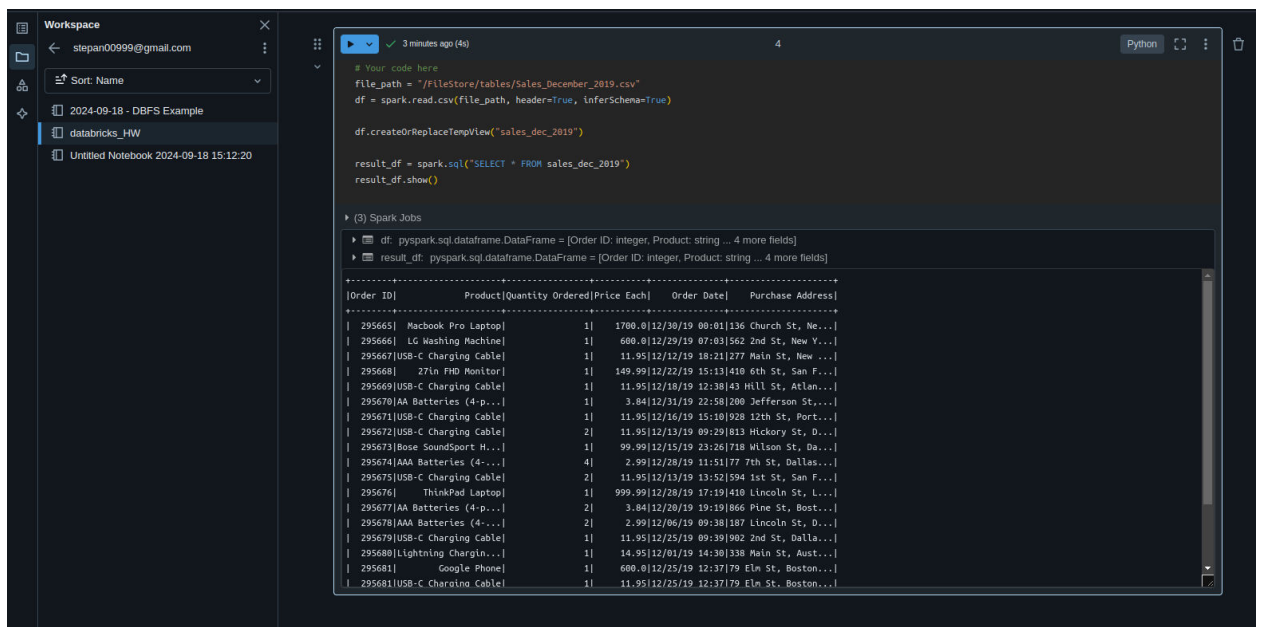
The screenshot shows a Databricks workspace interface. On the left, the 'Workspace' sidebar displays a file named 'Sales_December_2019.csv' under the 'databricks_HW' folder. The main area contains a code cell with the following Python code:

```
# dbutils - special utility for databricks that helps to work with a notebook. With this utility you can execute another notebook by dbutils.notebook.run
# ("notebook_name") or add any dynamic parameters using dbutils.widgets, or you can access DBFS using code below

filenames = dbutils.fs.ls("/FileStore/tables")
for filename in filenames:
    print(filename.name)
```

The output of the code cell shows the filename 'Sales_December_2019.csv'.

2. First we need to read our file then create a view. Next we use sqk inside a spark sql to extract all data.



The screenshot shows a Databricks workspace interface. On the left, the 'Workspace' sidebar displays a file named 'Sales_December_2019.csv' under the 'databricks_HW' folder. The main area contains a code cell with the following Python code:

```
# Your code here
file_path = "/FileStore/tables/Sales_December_2019.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)

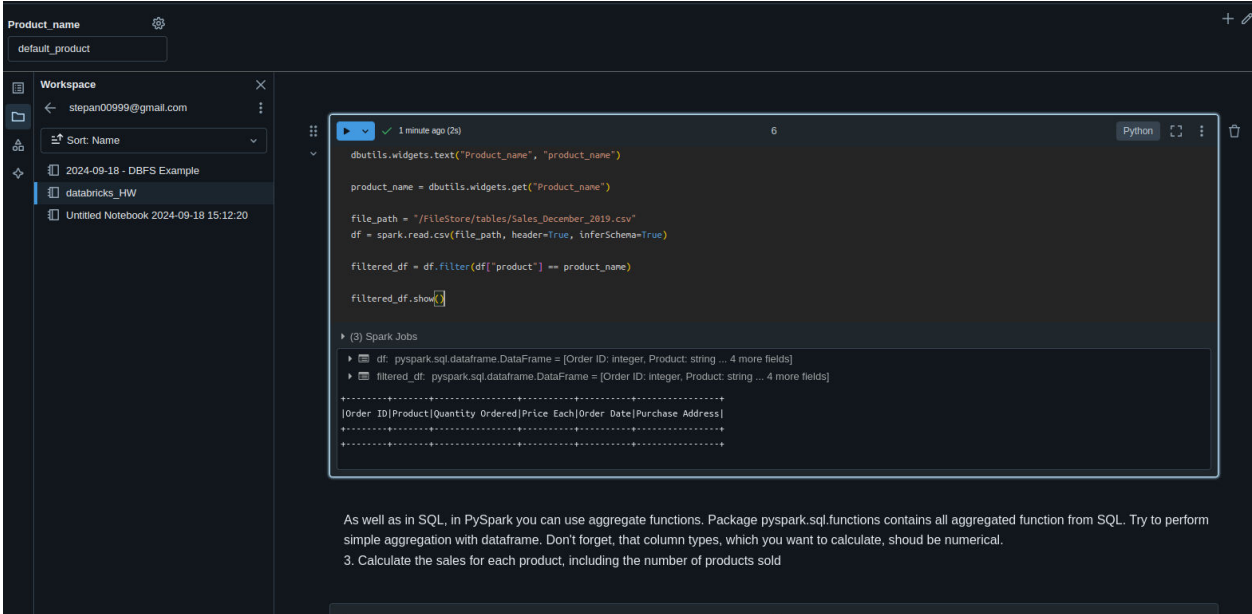
df.createOrReplaceTempView("sales_dec_2019")

result_df = spark.sql("SELECT * FROM sales_dec_2019")
result_df.show()
```

The output of the code cell shows the following table:

Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
295665	Macbook Pro Laptop	1	1700.0	12/30/19 00:01	136 Church St, Ne...
295666	LG Washing Machine	1	600.0	12/29/19 07:03	562 2nd St, New Y...
295667	USB-C Charging Cable	1	11.95	12/12/19 18:21	1277 Main St, New ...
295668	27In FHD Monitor	1	149.99	12/22/19 15:13	410 6th St, San F...
295669	USB-C Charging Cable	1	11.95	12/18/19 12:38	43 Hill St, Atlan...
295670	AA Batteries (4-p...	1	3.84	12/31/19 22:58	200 Jefferson St...
295671	USB-C Charging Cable	1	11.95	12/16/19 15:10	928 12th St, Port...
295672	USB-C Charging Cable	2	11.95	12/13/19 09:29	813 Hickory St, D...
295673	Bose SoundSport H...	1	99.99	12/15/19 23:26	718 Wilson St, Da...
295674	AAA Batteries (4-...	4	2.99	12/28/19 11:51	77 7th St, Dallas...
295675	USB-C Charging Cable	2	11.95	12/13/19 13:52	594 1st St, San F...
295676	ThinkPad Laptop	1	999.99	12/28/19 17:19	410 Lincoln St, L...
295677	AA Batteries (4-p...	2	3.84	12/20/19 19:19	866 Pine St, Bost...
295678	AA Batteries (4-...	2	2.99	12/06/19 09:38	187 Lincoln St, O...
295679	USB-C Charging Cable	1	11.95	12/25/19 09:39	902 2nd St, Dalla...
295680	Lightning Chargin...	1	14.95	12/01/19 14:30	330 Main St, Aust...
295681	Google Phone	1	600.0	12/25/19 12:37	79 Elm St, Boston...
295681	USB-C Charging Cable	1	11.95	12/25/19 12:37	79 Elm St, Boston...

3. Db util created with product_name value and used in df filtering.



The screenshot shows a Databricks workspace with a Python notebook. The left sidebar shows the workspace structure with a file named 'databricks_HW'. The main area displays a code cell that reads a CSV file and filters it based on the 'product_name' widget value. Below the code, the Spark Jobs output shows the filtered DataFrame with columns: Order ID, Product, Quantity Ordered, Price Each, Order Date, and Purchase Address. The output is a table with 10 rows of data.

```
Product_name
default_product

Workspace
stepan00999@gmail.com
Sort: Name
2024-09-18 - DBFS Example
databricks_HW
Untitled Notebook 2024-09-18 15:12:20

1 minute ago (2s) 6 Python

dbutils.widgets.text("Product_name", "product_name")

product_name = dbutils.widgets.get("Product_name")

file_path = "/FileStore/tables/Sales_December_2019.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)

filtered_df = df.filter(df["product"] == product_name)

filtered_df.show()
```

(3) Spark Jobs

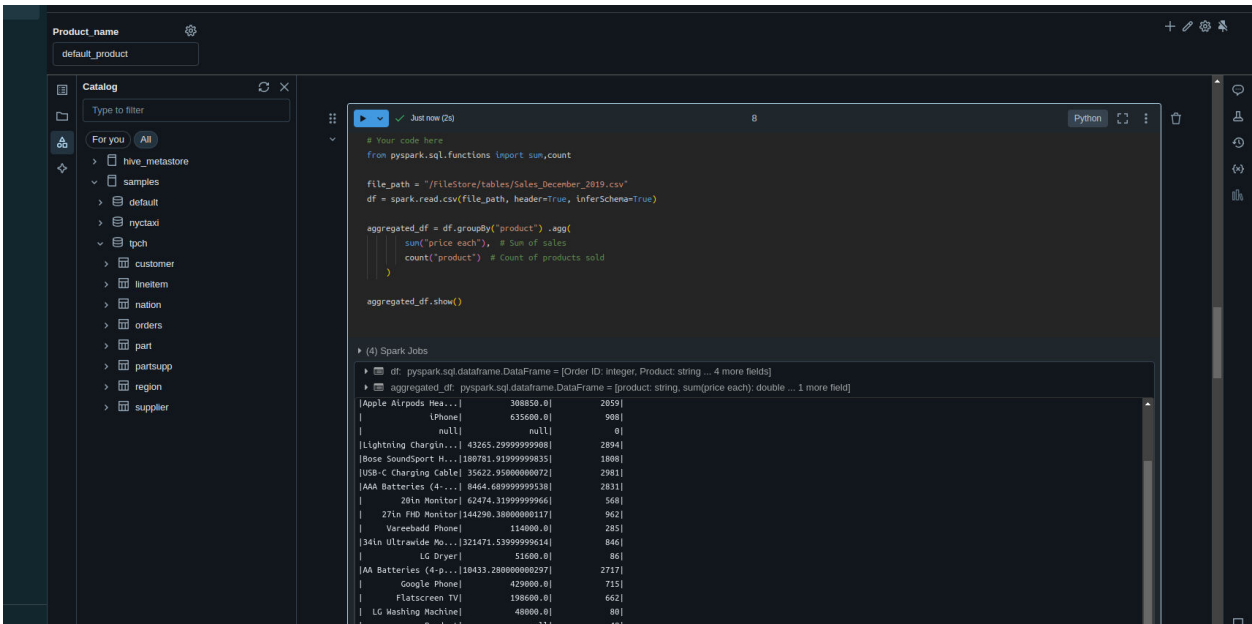
- df: pyspark.sql.dataframe.DataFrame = [Order ID: integer, Product: string ... 4 more fields]
- filtered_df: pyspark.sql.dataframe.DataFrame = [Order ID: integer, Product: string ... 4 more fields]

Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
1	Apple AirPods	308850.0	2859		
1	Phone	635600.0	908		
1	null	null	0		
1	Lightning Charge...	42825.299999999998	2894		
1	Bose Soundport He...	188781.91999999985	1888		
1	USB-C Charging Cable	35622.958000000721	2981		
1	AAA Batteries (4-...	8464.689999999538	2831		
1	20in Monitor	62474.31999999966	568		
1	27in FHD Monitor	144290.38000000117	962		
1	Vareebadd Phone	114800.0	285		
1	36in Ultrawide Mo...	321471.53999999616	846		
1	LG Dryer	52600.0	86		
1	AA Batteries (4-p...	10433.288000000207	2717		
1	Google Phone	428000.0	715		
1	Flatscreen TV	198600.0	662		
1	LG Washing Machine	48800.0	80		
1	Product	null	48		

As well as in SQL, in PySpark you can use aggregate functions. Package pyspark.sql.functions contains all aggregated function from SQL. Try to perform simple aggregation with dataframe. Don't forget, that column types, which you want to calculate, should be numerical.

3. Calculate the sales for each product, including the number of products sold

4. Grouped by product table:



The screenshot shows a Databricks workspace with a Python notebook. The left sidebar shows the workspace structure with a file named 'databricks_HW'. The main area displays a code cell that reads a CSV file and groups it by 'product' to calculate the sum of sales and the count of products sold. Below the code, the Spark Jobs output shows the aggregated DataFrame with columns: product, sum(price each), and count(product). The output is a table with 10 rows of data.

```
Product_name
default_product

Catalog
Type to filter
For you All
hive_metastore
samples
default
nyctaxi
tpch
customer
lineitem
nation
orders
part
partsupp
region
supplier

Just now (2s) 8 Python

# Your code here
from pyspark.sql.functions import sum,count

file_path = "/FileStore/tables/Sales_December_2019.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)

aggregated_df = df.groupby("product").agg(
    sum("price each"), # Sum of sales
    count("product") # Count of products sold
)

aggregated_df.show()
```

(4) Spark Jobs

- df: pyspark.sql.dataframe.DataFrame = [Order ID: integer, Product: string ... 4 more fields]
- aggregated_df: pyspark.sql.dataframe.DataFrame = [product: string, sum(price each): double ... 1 more field]

product	sum(price each)	count(product)
Apple AirPods	308850.0	2859
Phone	635600.0	908
null	null	0
Lightning Charge...	42825.299999999998	2894
Bose Soundport He...	188781.91999999985	1888
USB-C Charging Cable	35622.958000000721	2981
AAA Batteries (4-...	8464.689999999538	2831
20in Monitor	62474.31999999966	568
27in FHD Monitor	144290.38000000117	962
Vareebadd Phone	114800.0	285
36in Ultrawide Mo...	321471.53999999616	846
LG Dryer	52600.0	86
AA Batteries (4-p...	10433.288000000207	2717
Google Phone	428000.0	715
Flatscreen TV	198600.0	662
LG Washing Machine	48800.0	80
Product	null	48

5. The date column stored as string , I have changed the datatype to DATE:

The screenshot shows the Databricks workspace interface. On the left, the 'Catalog' sidebar is visible with a search bar and a list of databases including 'hive_metastore', 'samples', 'default', 'nyctaxi', 'tpch', 'customer', 'lineitem', 'nation', 'orders', 'part', 'partsupp', 'region', and 'supplier'. The main area displays two tasks:

Task 4: Show data profiles output for the new dataframe of table sales_december_2019_csv: row count, min and max value for each column. The code block shows:

```
from pyspark.sql.functions import to_date  
df = df.withColumn("Order Date", to_date(f.col("Order Date"), "MM/dd/yyyy"))  
df: pyspark.sql.dataframe.DataFrame = [Order ID: integer, Product: string ... 4 more fields]
```

Task 5: Add new column to the dataframe from previous task with any default value that you want. The code block shows:

```
#your code here  
df.printSchema()  
  
root  
 |-- Order ID: integer (nullable = true)  
 |-- Product: string (nullable = true)  
 |-- Quantity Ordered: integer (nullable = true)  
 |-- Price Each: double (nullable = true)  
 |-- Order Date: date (nullable = true)  
 |-- Purchase Address: string (nullable = true)
```

Below the code blocks, a note states: 'Temporary views are processed by cluster and always dropped when the session ends (when the cluster turns off).' and a task instruction: '6. Create temporary view from task 4 dataframe using PySpark and perform any select using SQL'.

See result

The screenshot shows the Databricks workspace interface with the same sidebar as the previous image. The main area displays the results of the tasks:

Task 4: The code block shows the same transformation as before, but the output now includes a 'summary' column:

```
df = df.withColumn("Order Date", to_date(col("Order Date"), "MM/dd/yyyy"))  
df_summary = df.summary()
```

The output shows two DataFrames:

- df:** pyspark.sql.dataframe.DataFrame with columns: Order ID: integer, Product: string, Quantity Ordered: integer, Price Each: double, Order Date: date, Purchase Address: string.
- df_summary:** pyspark.sql.dataframe.DataFrame with columns: summary: string, Order ID: string, Product: string, Quantity Ordered: string, Price Each: string, Purchase Address: string.

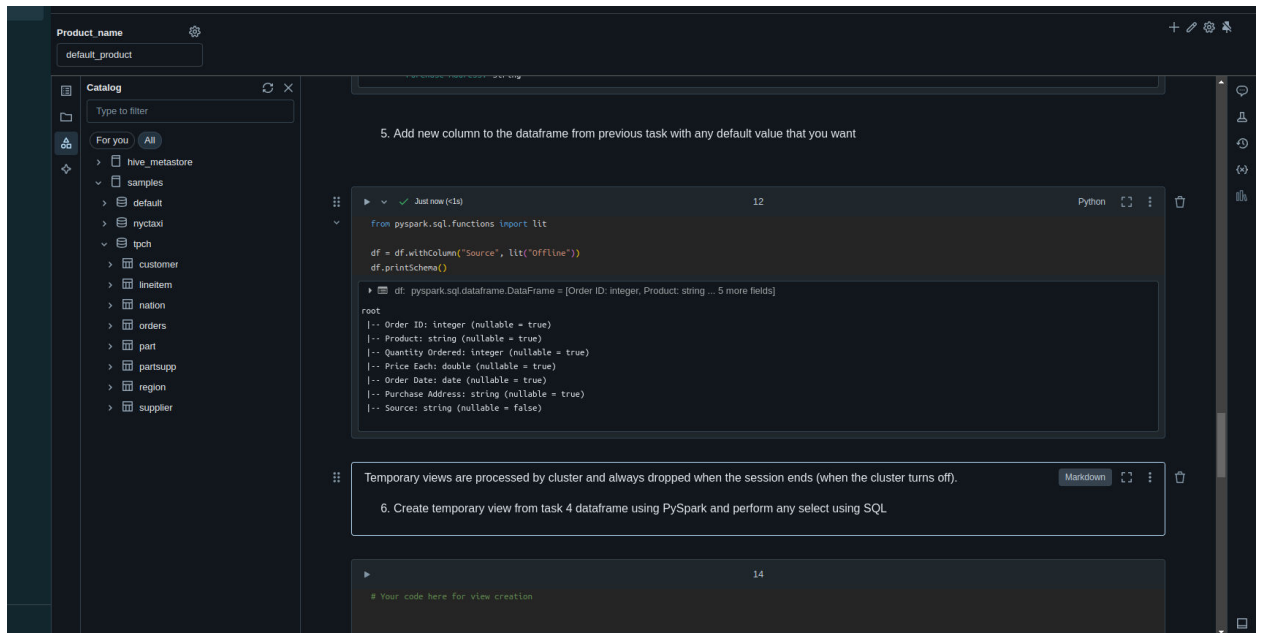
Task 5: The code block shows the same instruction as before, but the output now shows the schema of the dataframe:

```
#your code here  
df.printSchema()
```

The output shows the schema of the dataframe:

```
root  
 |-- Order ID: integer (nullable = true)  
 |-- Product: string (nullable = true)  
 |-- Quantity Ordered: integer (nullable = true)  
 |-- Price Each: double (nullable = true)  
 |-- Order Date: date (nullable = true)  
 |-- Purchase Address: string (nullable = true)
```


6. New column added with the default value:



Product_name
default_product

Catalog
Type to filter

For you **All**

- hive_metastore
- samples
- default
- nyctaxi
- tpch
- customer
- lineitem
- nation
- orders
- part
- partsupp
- region
- supplier

5. Add new column to the dataframe from previous task with any default value that you want

```
from pyspark.sql.functions import lit

df = df.withColumn("Source", lit("OffLine"))
df.printSchema()
```

df: pyspark.sql.dataframe.DataFrame = [Order ID: integer, Product: string ... 5 more fields]

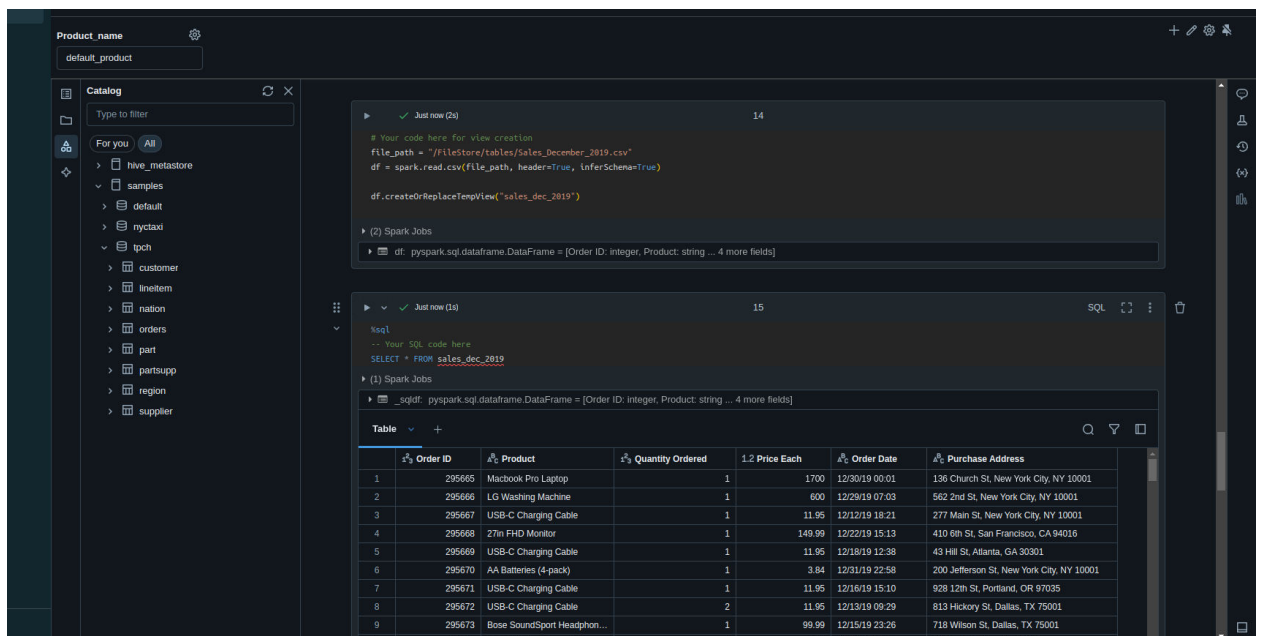
```
root
 |-- Order ID: integer (nullable = true)
 |-- Product: string (nullable = true)
 |-- Quantity Ordered: integer (nullable = true)
 |-- Price Each: double (nullable = true)
 |-- Order Date: date (nullable = true)
 |-- Purchase Address: string (nullable = true)
 |-- Source: string (nullable = false)
```

Temporary views are processed by cluster and always dropped when the session ends (when the cluster turns off).

6. Create temporary view from task 4 dataframe using PySpark and perform any select using SQL

Your code here for view creation

7. New table view created from csv file in the top block and then we just query that view using SQL.



Product_name
default_product

Catalog
Type to filter

For you **All**

- hive_metastore
- samples
- default
- nyctaxi
- tpch
- customer
- lineitem
- nation
- orders
- part
- partsupp
- region
- supplier

```
# Your code here for view creation
file_path = "/FileStore/tables/Sales_December_2019.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)

df.createOrReplaceTempView("sales_dec_2019")
```

(2) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [Order ID: integer, Product: string ... 4 more fields]

```
-- Your SQL code here
SELECT * FROM sales_dec_2019
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [Order ID: integer, Product: string ... 4 more fields]

Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
1	Macbook Pro Laptop	1	1700	12/30/19 00:01	136 Church St, New York City, NY 10001
2	LG Washing Machine	1	600	12/29/19 07:03	562 2nd St, New York City, NY 10001
3	USB-C Charging Cable	1	11.95	12/12/19 18:21	277 Main St, New York City, NY 10001
4	27in FHD Monitor	1	149.99	12/22/19 15:13	410 6th St, San Francisco, CA 94016
5	USB-C Charging Cable	1	11.95	12/18/19 12:38	43 Hill St, Atlanta, GA 30301
6	AA Batteries (4-pack)	1	3.84	12/31/19 22:58	200 Jefferson St, New York City, NY 10001
7	USB-C Charging Cable	1	11.95	12/16/19 15:10	928 12th St, Portland, OR 97035
8	USB-C Charging Cable	2	11.95	12/13/19 09:29	813 Hickory St, Dallas, TX 75001
9	Bose SoundSport Headphon...	1	99.99	12/15/19 23:26	718 Wilson St, Dallas, TX 75001

8. New parquet file created from csv:

The screenshot shows a Databricks workspace interface. On the left, the 'Catalog' sidebar is visible with a search bar and filters for 'hive_metastore' and 'samples'. The main area displays a Python notebook cell with the following code:

```
# Your code here
file_path = "/FileStore/tables/Sales_December_2019.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)

output_path = "/FileStore/tables/Sales_December_2019.parquet"

df.write.mode("overwrite").parquet(output_path)

filenames = dbutils.fs.ls("/FileStore/tables")
for filename in filenames:
    print(filename.name)
```

Below the code, the output shows the Spark job status and the resulting DataFrame structure:

```
(5) Spark Jobs
dt: pyspark.sql.dataframe.DataFrame = [Order ID: integer, Product: string ... 4 more fields]
Sales_December_2019.csv
Sales_December_2019.parquet/
```

Text annotations in the image explain the process: 'When we export dataframe to the any file, databricks creates folder with files which are divided into separate files of the same size.' and '7. Export dataframe from task 5 as .parquet file to the DBFS and show that databricks creates folder with .parquet files'.

9. Created temp table using parquet file:

The screenshot shows a Databricks workspace interface. On the left, the 'Catalog' sidebar is visible with a search bar and filters for 'hive_metastore' and 'samples'. The main area displays a SQL notebook cell with the following code:

```
-- Your SQL code here
CREATE OR REPLACE TEMP VIEW sales_data_table_from_parquet
USING parquet
OPTIONS (
  path "/FileStore/tables/Sales_December_2019.parquet"
);

SELECT * FROM sales_data_table_from_parquet;
```

Below the code, the output shows the Spark job status and the resulting DataFrame structure:

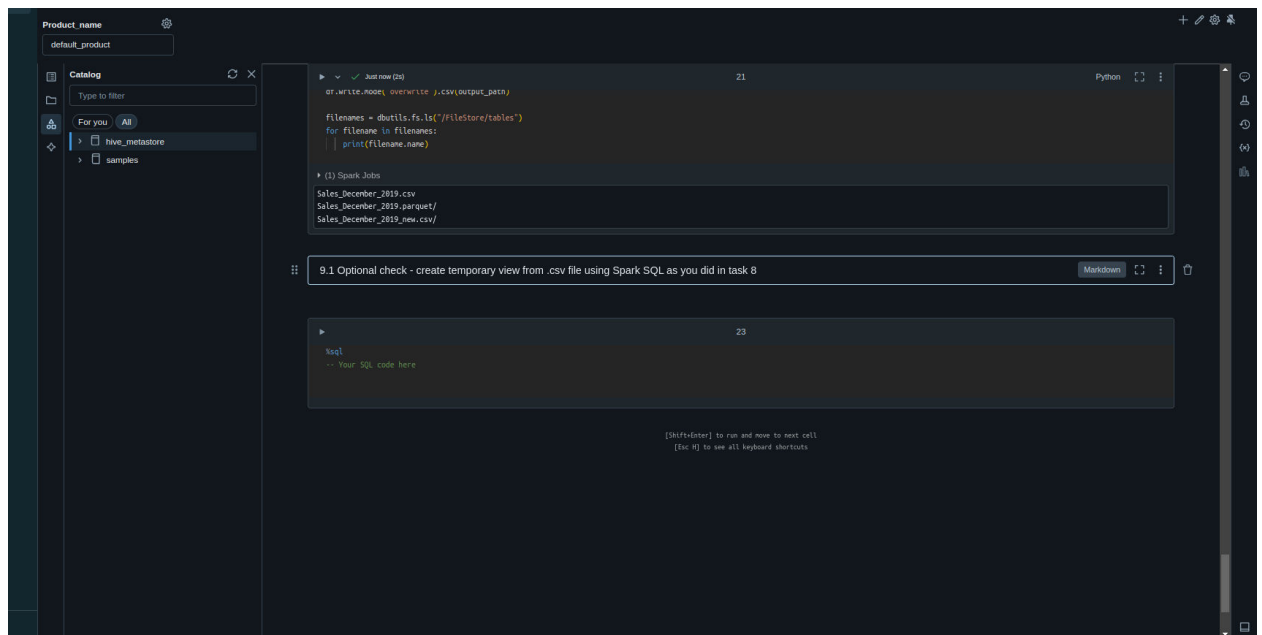
```
(2) Spark Jobs
dt: pyspark.sql.dataframe.DataFrame = [Order ID: integer, Product: string ... 4 more fields]
```

The output also displays a table view of the data:

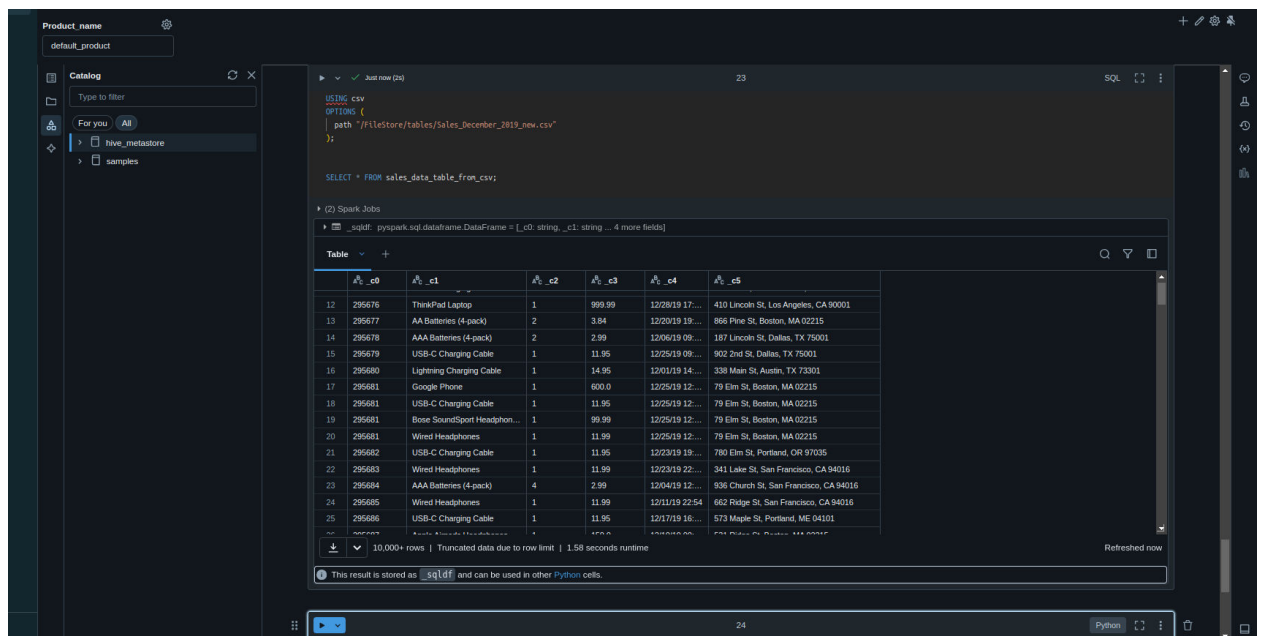
Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
1	Macbook Pro Laptop	1	1700	12/30/19 00:01	136 Church St, New York City, NY 10001
2	LG Washing Machine	1	600	12/29/19 07:03	562 2nd St, New York City, NY 10001
3	USB-C Charging Cable	1	11.95	12/12/19 18:21	277 Main St, New York City, NY 10001
4	27in PHD Monitor	1	149.99	12/22/19 15:13	410 6th St, San Francisco, CA 94016
5	USB-C Charging Cable	1	11.95	12/18/19 12:38	43 Hill St, Atlanta, GA 30301
6	AA Batteries (4-pack)	1	3.84	12/31/19 22:58	200 Jefferson St, New York City, NY 10001
7	USB-C Charging Cable	1	11.95	12/16/19 15:10	928 12th St, Portland, OR 97035
8	USB-C Charging Cable	2	11.95	12/13/19 09:29	813 Hickory St, Dallas, TX 75001
9	Bose SoundSport Headphon...	1	98.99	12/15/19 23:26	718 Wilson St, Dallas, TX 75001
10	AAA Batteries (4-pack)	4	2.99	12/28/19 11:51	77 7th St, Dallas, TX 75001
11	USB-C Charging Cable	2	11.95	12/13/19 13:52	594 1st St, San Francisco, CA 94016
12	ThinkPad Laptop	1	999.99	12/28/19 17:19	410 Lincoln St, Los Angeles, CA 90001
13	AA Batteries (4-pack)	2	3.84	12/20/19 19:19	866 Pine St, Boston, MA 02215
14	AAA Batteries (4-pack)	2	2.99	12/06/19 09:38	187 Lincoln St, Dallas, TX 75001
15	USB-C Charging Cable	1	11.95	12/25/19 09:39	902 2nd St, Dallas, TX 75001

Text annotations in the image explain the process: '8. Create table from .parquet files that was created in the previous cell using pure SQL'.

10. New csv file created from previous dataframe.



11. New temp table from new csv file we stored in file system.



Homework notebook published at link :

<https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bfcf/3148840622372913/2135283182316894/4761590133845256/latest.html>