

Spark Lab 1 Report

Introduction to PySpark and Spark SQL

Contents


Starting the Cluster	2
Task 1	3
Task 2	10

André Plancha
andre.plancha@hotmail.com

December 06th, 2025


Starting the Cluster

Dockerfile on the server

 Docker


```
1 FROM apache/spark:4.0.1
2 # switch to root to install packages
3 USER root
4 RUN pip install --no-cache-dir "pandas==2.3.2" "pyarrow==21.0.0"
5 # switch back to spark user
6 USER spark
```

compose.yaml on the server

 YAML

```
1 services:
2   spark:
3     build: .
4     hostname: apache-spark
5     ports:
6       - "7077:7077"      # Spark master port
7       - "8080:8080"      # Spark master web UI
8       - "8081:8081"      # Spark worker web UI
9       - "15002:15002"    # Spark Connect server port
10      - "4040:4040"      # Spark Connect web UI
11     command: >
12       bash -c "/opt/spark/sbin/start-master.sh;
13               /opt/spark/sbin/start-connect-server.sh;
14               /opt/spark/sbin/start-worker.sh spark://192.168.1.7:7077;
15               sleep infinity"
```

```
1 $ ls
2 compose.yaml  Dockerfile
3 $ docker compose up -d
4 [+] Running 1/1
5   ✓ Container spark-docker-spark-1  Started
   0.5s
```

 Shell

Task 1

Task1: Retail Store Insights

Scenario: You're a data analyst at a large retail store. The store sells a variety of products, including books and fruits. The management wants insights into sales patterns, customer preferences, product popularity, and potential promotions.

Objective: Analyze the provided dataset to extract meaningful insights and present them to the management.

Instructions:

1. Initialization:

- Set up your environment by initializing a Spark session. Name this session "RetailStoreInsights".

In [1]:

```
1 from pyspark.sql import SparkSession
2
3 spark = SparkSession.builder \
4     .remote("sc://192.168.1.7:15002") \
5     .appName("RetailStoreInsights") \
6     .getOrCreate()
7
8 # limit() shows a nice HTML table in Jupyter, while show() prints plain text
9 spark.conf.set('spark.sql.repl.eagerEval.enabled', True)
10
11 spark
```

Python

Out[1]:

```
1 <pyspark.sql.connect.session.SparkSession at 0x14c54de0b50>
```

2. Data Loading:

- Load the provided dataset into a DataFrame. This dataset will have O information about various products, including their type (e.g., fruit or book) and price. Create DataFrame based on bellow data:

...

3. Data Exploration:

- Familiarize yourself with the dataset:
 - Display the first 10 rows to understand the structure and content.

In [2]:

```
1 from pyspark.sql import Row
2
3 data = [
4     ('Ulysses', 'Book', 23.17, 16),
5     ('Apple', 'Fruit', 2.34, 8),
6     ('Pineapple', 'Fruit', 2.57, 1),
```

Python

```

7   ('Apple', 'Fruit', 2.43, 6),
8   ('To Kill a Mockingbird', 'Book', 24.14, 19),
9   ('To Kill a Mockingbird', 'Book', 11.18, 11),
10  ('Watermelon', 'Fruit', 3.35, 15),
11  ('Pride and Prejudice', 'Book', 24.99, 3),
12  ('To Kill a Mockingbird', 'Book', 21.82, 17),
13  ('Moby Dick', 'Book', 14.83, 20),
14  ('Pride and Prejudice', 'Book', 5.03, 16),
15  ('Jane Eyre', 'Book', 20.40, 8),
16  ('Moby Dick', 'Book', 5.55, 20),
17  ('Don Quixote', 'Book', 19.75, 17),
18  ('Watermelon', 'Fruit', 2.31, 9),
19  ('Hamlet', 'Book', 18.20, 12),
20  ('Mango', 'Fruit', 4.10, 7),
21  ('1984', 'Book', 16.75, 14),
22  ('Strawberry', 'Fruit', 1.90, 25),
23  ('War and Peace', 'Book', 22.50, 9),
24  ('Orange', 'Fruit', 3.05, 13),
25  ('The Great Gatsby', 'Book', 12.30, 10),
26  ('Peach', 'Fruit', 2.80, 11),
27  ('Grapes', 'Fruit', 2.60, 18),
28  ('Pride and Prejudice', 'Book', 9.50, 5)
29 ]
30
31 df = spark.createDataFrame([
32     Row(product_name=row[0], category=row[1], price=row[2], quantity=row[3])
33     for row in data
34 ], schema = 'product_name STRING, category STRING, price FLOAT, quantity
35             SHORT')
36 df.createOrReplaceTempView("retail_sales") # give it a name for sql
37 df.limit(10)

```

Out[2]:

```


1  +-----+-----+-----+
2  | product_name|category|price|quantity|
3  +-----+-----+-----+
4  |      Ulysses|      Book|23.17|      16|
5  |       Apple|      Fruit| 2.34|       8|
6  |   Pineapple|      Fruit| 2.57|       1|
7  |       Apple|      Fruit| 2.43|       6|
8  |To Kill a Mocking...|      Book|24.14|      19|
9  |To Kill a Mocking...|      Book|11.18|      11|
10 |      Watermelon|      Fruit| 3.35|      15|
11 |Pride and Prejudice|      Book|24.99|       3|
12 |To Kill a Mocking...|      Book|21.82|      17|
13 |      Moby Dick|      Book|14.83|      20|
14 +-----+-----+-----+

```

- Print the schema of the DataFrame to understand the data types and columns.

In [3]:

```
1 df.printSchema()
```

 Python


```
1 root
2 |-- product_name: string (nullable = true)
3 |-- category: string (nullable = true)
4 |-- price: float (nullable = true)
5 |-- quantity: short (nullable = true)
6
```

3. Data Analysis:

- Extract specific columns of interest using the select operation. For this task, focus on the product name and its price.
- Identify and display products that are priced above \$2.

In [4]:

```
1 spark.sql("""
2     select * from retail_sales
3     where price > 2
4     order by price
5 """)
```

 Python


Out[4]:

```
1 +-----+-----+-----+-----+
2 | product_name|category|price|quantity|
3 +-----+-----+-----+-----+
4 |      Watermelon|   Fruit| 2.31|      9|
5 |           Apple|   Fruit| 2.34|      8|
6 |           Apple|   Fruit| 2.43|      6|
7 |      Pineapple|   Fruit| 2.57|      1|
8 |           Grapes|   Fruit|  2.6|     18|
9 |           Peach|   Fruit|  2.8|     11|
10 |          Orange|   Fruit| 3.05|     13|
11 |      Watermelon|   Fruit| 3.35|     15|
12 |           Mango|   Fruit|  4.1|      7|
13 |Pride and Prejudice|   Book| 5.03|     16|
14 |      Moby Dick|   Book| 5.55|     20|
15 |Pride and Prejudice|   Book|  9.5|      5|
16 |To Kill a Mocking...|   Book|11.18|     11|
17 |   The Great Gatsby|   Book| 12.3|     10|
18 |      Moby Dick|   Book|14.83|     20|
19 |           1984|   Book|16.75|     14|
20 |          Hamlet|   Book| 18.2|     12|
21 |      Don Quixote|   Book|19.75|     17|
22 |      Jane Eyre|   Book| 20.4|      8|
23 |To Kill a Mocking...|   Book|21.82|     17|
24 +-----+-----+-----+-----+
25 only showing top 20 rows
```

- Group the data by product type (e.g., fruit or book) and determine the count for each category.

In [5]:

```
1 # https://spark.apache.org/docs/latest/sql-pipe-syntax.html
2 spark.sql("""
```

 Python

```

3   from retail_sales
4   |> aggregate count(*) as category_count
5     group by category
6   """)

```

Out[5]:

```

1  +-----+-----+
2  |category|category_count|
3  +-----+-----+
4  |   Book|           15|
5  |  Fruit|           10|
6  +-----+-----+

```


- Calculate the average price of all products in the dataset.

In [6]:

```

1  spark.sql("""
2    from retail_sales
3    |> aggregate avg(price) as avg_price
4      group by product_name
5    |> set avg_price = round(avg_price, 2)
6    """)

```

 Python

Out[6]:

```

1  +-----+-----+
2  |      product_name|avg_price|
3  +-----+-----+
4  |      Pineapple|      2.57|
5  |To Kill a Mocking...|     19.05|
6  |      Ulysses|     23.17|
7  |      Apple|      2.38|
8  |      Jane Eyre|     20.4|
9  |      Moby Dick|     10.19|
10 |      Watermelon|      2.83|
11 |Pride and Prejudice|     13.17|
12 |      1984|     16.75|
13 |      Mango|       4.1|
14 |      Don Quixote|     19.75|
15 |      Hamlet|     18.2|
16 |      Orange|      3.05|
17 |      Peach|       2.8|
18 |The Great Gatsby|     12.3|
19 |      Grapes|       2.6|
20 |      Strawberry|      1.9|
21 |War and Peace|     22.5|
22 +-----+-----+

```


- The store is considering a promotion where they offer a 10% discount on all products. Add a new column to the DataFrame that calculates the discounted price for each product.

In [7]:

```

1  spark.sql("""
2    from retail_sales

```

 Python

```

3  |> extend price - (price * 0.1) as discounted_price
4  |> set discounted_price = round(discounted_price, 2)
5  |> select product_name, discounted_price, price as original_price
6  """)

```

Out[7]:

product_name	discounted_price	original_price
Ulysses	20.85	23.17
Apple	2.11	2.34
Pineapple	2.31	2.57
Apple	2.19	2.43
To Kill a Mocking...	21.73	24.14
To Kill a Mocking...	10.06	11.18
Watermelon	3.01	3.35
Pride and Prejudice	22.49	24.99
To Kill a Mocking...	19.64	21.82
Moby Dick	13.35	14.83
Pride and Prejudice	4.53	5.03
Jane Eyre	18.36	20.4
Moby Dick	5.0	5.55
Don Quixote	17.78	19.75
Watermelon	2.08	2.31
Hamlet	16.38	18.2
Mango	3.69	4.1
1984	15.08	16.75
Strawberry	1.71	1.9
War and Peace	20.25	22.5

only showing top 20 rows

- Using SQL, perform the following operations:
 - Determine the total number of products sold (including duplicates).


note: this operations seemed to be a bit out of place, so I interpreted them as best as I could.

In [8]:

```

1  spark.sql("""
2    from retail_sales
3    |> aggregate sum(quantity) as n_sold_total
4    """)

```

 Python

Out[8]:

n_sold_total
310


- Calculate the total sales (sum of prices) for each product category (e.g., fruits vs. books).

In [9]:

```

1  spark.sql("""

```

 Python

```

2   from retail_sales
3   |> aggregate sum(quantity) as n_sold
4     group by category
5   """)

```

Out[9]:

```

1  +-----+-----+
2  |category|n_sold|
3  +-----+-----+
4  |   Book|   197|
5  |   Fruit|   113|
6  +-----+-----+


```

In [10]:

```

1  spark.sql("""
2    from retail_sales
3    |> aggregate sum(price * quantity) as revenue
4      group by category
5    """)

```

 Python

Out[10]:

```

1  +-----+-----+
2  |category|      revenue|
3  +-----+-----+
4  |   Book|3211.2000007629395|
5  |   Fruit| 300.3599935770035|
6  +-----+-----+

```


- (Optional) Identify the products based on the frequency of their occurrence in the dataset. (To find out which products appear most often in the dataset.)

In [11]:

```

1  spark.sql("""
2    from retail_sales
3    |> aggregate sum(quantity) as n_sold
4      group by category, product_name
5    |> order by n_sold desc
6    """)

```

 Python

Out[11]:

```

1  +-----+-----+-----+
2  |category|product_name|n_sold|
3  +-----+-----+-----+
4  |   Book|To Kill a Mocking...|   47|
5  |   Book|      Moby Dick|   40|
6  |   Fruit|    Strawberry|   25|
7  |   Fruit|    Watermelon|   24|
8  |   Book|Pride and Prejudice|   24|
9  |   Fruit|        Grapes|   18|
10 |   Book|    Don Quixote|   17|
11 |   Book|      Ulysses|   16|
12 |   Fruit|        Apple|   14|
13 |   Book|        1984|   14|
14 |   Fruit|       Orange|   13|


```



```
15 |   Book|           Hamlet|   12|
16 |  Fruit|           Peach|   11|
17 |   Book|  The Great Gatsby|   10|
18 |   Book|    War and Peace|    9|
19 |   Book|      Jane Eyre|    8|
20 |  Fruit|           Mango|    7|
21 |  Fruit|      Pineapple|    1|
22 +-----+-----+-----+
```

In [12]:

```
1 spark.stop()
```

 Python

Task 2

Scenario: You're a data engineering intern at a tech company. As part of your training, you're given a basic PySpark script that demonstrates the use of Python UDFs and Pandas UDFs. Your task is to modify the script by implementing a new transformation function.

Instructions:

1. Initialization:

- Start a Spark session named "UDFTransformation".

In [1]:

```
1 from pyspark.sql import SparkSession
2
3 spark = SparkSession.builder \
4     .remote("sc://192.168.1.7:15002") \
5     .appName("UDFTransformation") \
6     .config("spark.sql.ansi.enabled", "false") \
7     .config("spark.sql.execution.pythonUDF.arrow.enabled", "true") \
8     .getOrCreate()
9
10 # limit() shows a nice HTML table in Jupyter, while show() prints plain text
11 spark.conf.set('spark.sql.repl.eagerEval.enabled', True)
12
13 spark
```

Out[1]:

```
1 <pyspark.sql.connect.session.SparkSession at 0x22751278d30>
```

2. Using Python Native Function as UDF:

- Define a Python function that multiplies the input value by 3.
- Convert this function into a Spark UDF.
- Create a DataFrame with a series of numbers: [(4,), (5,), (6,)].

In [2]:

```
1 from pyspark.sql.functions import udf
2
3 @udf(returnType='int')
4 def mult_by_3(s: int) -> int:
5     return s * 3
6
7 df = spark.createDataFrame([(4, ), (5, ), (6, )], ['value'])
8 df
```


Out[2]:

```
1 +-----+
2 |value|
3 +-----+
4 |    4|
5 |    5|
```

```
6 |    6|
7 +-----+
```

- Apply the UDF to the DataFrame to triple each number.
- Display the updated DataFrame.

```
In [3]: 1 dff = df.withColumn('value_x3', mult_by_3(df.value))
        2 dff
```


 Python

```
Out[3]: 1 +-----+-----+
        2 |value|value_x3|
        3 +-----+-----+
        4 |    4|    12|
        5 |    5|    15|
        6 |    6|    18|
        7 +-----+-----+
```

3. Using Pandas UDF:

- Define a Pandas UDF that subtracts 2 from each value in the input series.
- Apply this UDF to the DataFrame from step 2.
- Display the updated DataFrame with the subtracted values.

```
In [4]: 1 import pandas as pd
        2 import pyspark.pandas as ps
        3 from pyspark.sql.functions import pandas_udf
        4
        5 @pandas_udf("int")
        6 def sub_2(s: pd.Series) -> pd.Series:
        7     return s - 2
        8
        9 dffs = dff.withColumn('value_minus_2', sub_2(dff.value))
       10 dffs
```

 Python

```
c:\Users\plancha\spark-lab1\venv\lib\site-packages\pyspark\pandas\__init__.py:43: UserWarning:
1 'PYARROW_IGNORE_TIMEZONE' environment variable was not set. It is required to set this environment
  variable to '1' in both driver and executor sides if you use pyarrow>=2.0.0. pandas-on-Spark will
  set it for you but it does not work if there is a Spark context already launched.
2 warnings.warn(
```


```
Out[4]: 1 +-----+-----+-----+
        2 |value|value_x3|value_minus_2|
        3 +-----+-----+-----+
        4 |    4|    12|         2|
        5 |    5|    15|         3|
        6 |    6|    18|         4|
        7 +-----+-----+-----+
```

4. Cleanup:

- Stop the Spark session.

In [5]:

```
1 spark.stop()
```

 Python