HOW TO USE JUPYTER NOTEBOOKS WITH APACHE SPARK

Visualize and interact with data using Apache Spark and Jupyter Notebook

Apache Spark is an open-source, fast unified analytics engine developed at UC Berkeley for big data and machine learning. Spark utilizes in-memory caching and optimized query execution to provide a fast and efficient big data processing solution.

Moreover, Spark can easily support multiple workloads ranging from batch processing, interactive querying, real-time analytics to machine learning and graph processing. All these capabilities have led to Spark becoming a leading data analytics tool.

From a developer perspective, one of the best attributes of Spark is its support for multiple languages. Unlike many other platforms with limited options or requiring users to learn a platform-specific language, Spark supports all leading data analytics languages such as R, SQL, Python, Scala, and Java. Spark offers developers the freedom to select a language they are familiar with and easily utilize any tools and services supported for that language when developing.

When considering Python, Jupyter Notebooks is one of the most popular tools available for a developer. Yet, how can we make a Jupyter Notebook work with Apache Spark? In this post, we will see how to utilize Jupyter, Spark and PySpark to create an Apache Spark installation that can carry out data analytics through your familiar Jupyter Notebook interface.

(This tutorial is part of our Apache Spark Guide. Use the right-hand menu to navigate.)

How Jupyter Notebooks work

Jupyter is an interactive computational environment managed by Jupyter Project and distributed under the modified BSB license. A Notebook is a shareable document that combines both inputs and outputs to a single file. These notebooks can consist of:

- Code
- Mathematical equations
- Narrative text
- Visualizations
- Statistical modeling
- Other rich media

The beauty of a notebook is that it allows developers to develop, visualize, analyze, and add any kind of information to create an easily understandable and shareable single file. This approach is highly useful in data analytics as it allows users to include all the information related to the data within a specific notebook.

Jupyter supports over 40 programming languages and comes in two formats:

- The classic Jupyter notebook
- The JupyterLab

JupyterLab is the next-gen notebook interface that further enhances the functionality of Jupyter to create a more flexible tool that can be used to support any workflow from data science to machine learning. Jupyter also supports Big data tools such as Apache Spark for data analytics needs.

(Read our comprehensive intro to Jupyter Notebooks.)

How to connect Jupyter with Apache Spark

If you need to use Spark in Jupyter Notebook, Scala is the ideal language to interact with Apache Spark as it is written in Scala.

However, most developers prefer to use a language they are familiar with, such as Python. Jupyter supports both Scala and Python. However, Python is the more flexible choice in most cases due to its robustness, ease of use, and the availability of libraries like pandas, scikit-learn, and TensorFlow. While projects like almond allow users to add Scala to Jupyter, we will focus on Python in this post.

(See why Python is the language of choice for machine learning.)

PySpark for Apache Spark & Python

Python connects with Apache Spark through PySpark. It allows users to write Spark applications using the Python API and provides the ability to interface with the Resilient Distributed Datasets (RDDs) in Apache Spark. PySpark allows Python to interface with JVM objects using the Py4J library. Furthermore, PySpark supports most Apache Spark features such as Spark SQL, DataFrame, MLib, Spark Core, and Streaming.

Configuring PySpark with Jupyter and Apache Spark

Before configuring PySpark, we need to have Jupyter and Apache Spark installed. In this section, we will cover the simple installation procedures for using Spark in Jupyter Notebook. You can follow this Jupyter Notebooks for Data Analytics guide for detailed instructions on installing Jupiter, and you can follow the official documentation of Spark to set it up in your local environment.

Installing Spark

You will need Java, Scala, and Git as prerequisites for installing Spark. We can install them using the following command:

```
sudo apt install default-jdk scala git -y
```

Then, get the latest Apache Spark version, extract the content, and move it to a separate directory using the following commands.

```
wget https://dlcdn.apache.org/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz
tar xf spark-*
sudo mv spark-3.2.0-bin-hadoop3.2 /opt/spark
```

RESULT

```
❯ wget https://dlcdn.apache.org/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz
--2021-11-15 18:27:46--  https://dlcdn.apache.org/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz
Resolving dlcdn.apache.org (dlcdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 300965906 (287M) [application/x-gzip]
Saving to: 'spark-3.2.0-bin-hadoop3.2.tgz'

spark-3.2.0-bin-hadoop3.2.tgz 100%[=============================================>] 287.02M  11.3MB/s    in 26s

2021-11-15 18:28:12 (10.9 MB/s) - 'spark-3.2.0-bin-hadoop3.2.tgz' saved [300965906/300965906]

❯ tar xf spark-*
❯ sudo mv spark-3.2.0-bin-hadoop3.2 /opt/bin
```

Then we can set up the environmental variables by adding them to the shell configuration file (Ex: .bashrc / .zshrc) as shown below.

```
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PYSPARK_PYTHON=/usr/bin/python3
```

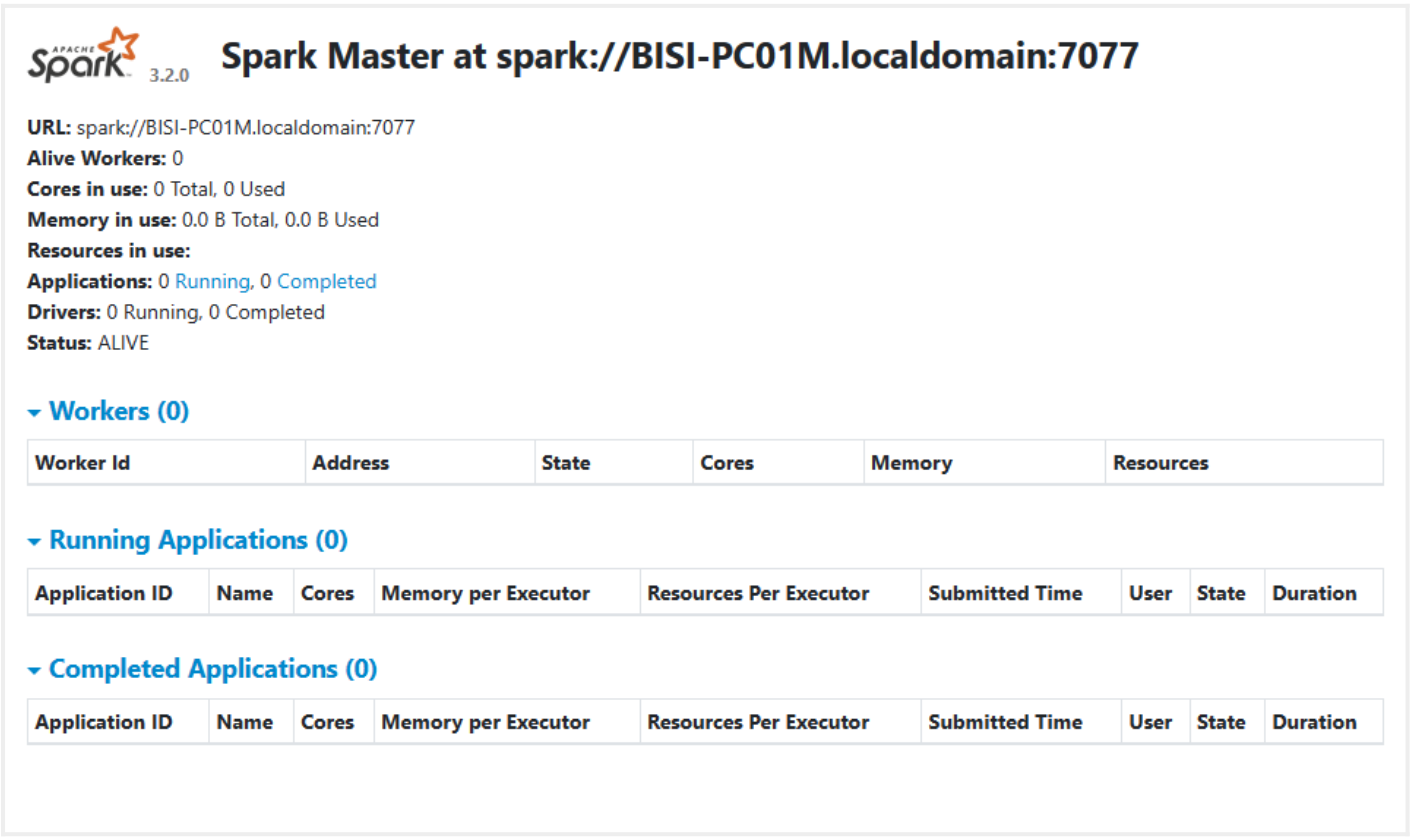
The **SPARK_HOME** variable indicates the Apache Spark installation, and **PATH** adds the Apache Spark (SPARK_HOME) to the system paths. After that, the **PYSPARK_PYTHON** variable points to the Python installation.

Finally, run the ***start-master.sh*** command to start Apache Spark, and you will be able to confirm the successful installation by visiting http://localhost:8080/

Command

```
❯ start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-[redacted]-org.apache.spa
rk.deploy.master.Master-1-BISI-PC01M.out
```

Web UI



Installing Jupyter

Installing Jupyter is a simple and straightforward process. It can be installed directly via Python package manager using the following command:

```
pip install notebook
```

Installing PySpark

There's no need to install PySpark separately as it comes bundled with Spark. However, you also have the option of installing PySpark and the extra dependencies like Spark SQL or Pandas for Spark as a separate installation via the Python package manager.

You can directly launch PySpark by running the following command in the terminal.

```
pyspark
```

RESULT:

```
> pyspark
Python 3.8.10 (default, Sep 28 2021, 16:10:42)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
21/11/15 19:20:13 WARN Utils: Your hostname, BISI-PC01M resolves to a loopback address: 127.0.1.1; using 172.27.140.242
instead (on interface eth0)
21/11/15 19:20:13 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/opt/spark/jars/spark-unsafe_2.12-3.2.0.jar
) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

21/11/15 19:20:15 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.2.0
      /_/

Using Python version 3.8.10 (default, Sep 28 2021 16:10:42)
Spark context Web UI available at http://172.27.140.242:4040
Spark context available as 'sc' (master = local[*], app id = local-1636984216845).
SparkSession available as 'spark'.
>>>
```

How to integrate PySpark with Jupyter Notebook

The only requirement to get the Jupyter Notebook reference PySpark is to add the following environmental variables in your .bashrc or .zshrc file, which points PySpark to Jupyter.

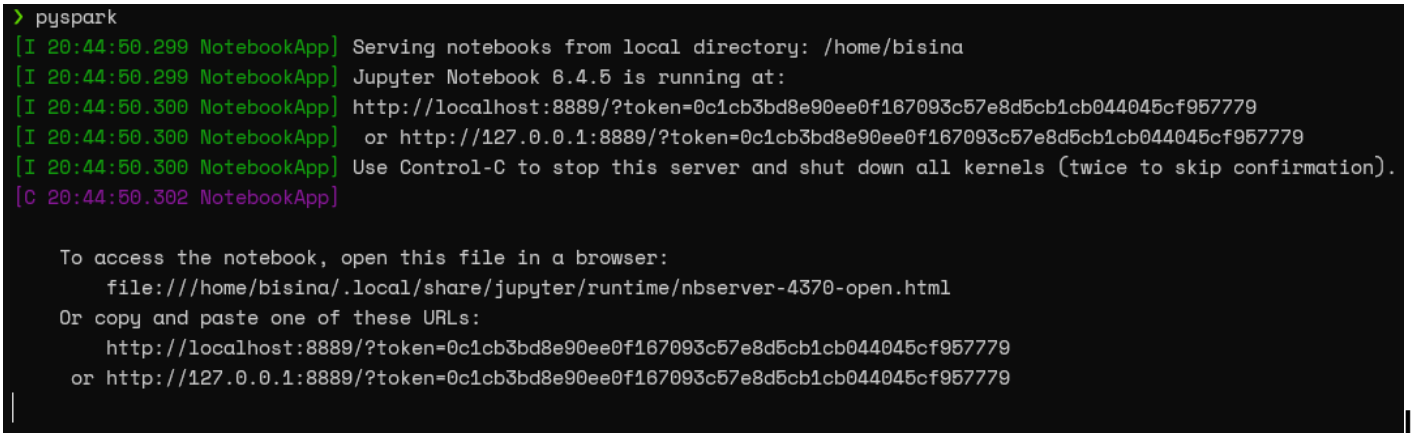
```
export PYSPARK_DRIVER_PYTHON='jupyter'
export PYSPARK_DRIVER_PYTHON_OPTS='notebook --no-browser --port=8889'
```

The **PYSPARK_DRIVER_PYTHON** points to Jupiter, while the **PYSPARK_DRIVER_PYTHON_OPTS** defines the options to be used when starting the notebook. In this case, it indicates the no-browser option and the port 8889 for the web interface.

With the above variables, your shell file should now include five environment variables required to power this solution.

```
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PYSPARK_PYTHON=/usr/bin/python3
export PYSPARK_DRIVER_PYTHON='jupyter'
export PYSPARK_DRIVER_PYTHON_OPTS='notebook --no-browser --port=8889'
```

Now, we can directly launch a Jupyter Notebook instance by running the ***pyspark*** command in the terminal.



Important note: Always make sure to refresh the terminal environment; otherwise, the newly added environment variables will not be recognized.

Now visit the provided URL, and you are ready to interact with Spark via the Jupyter Notebook.

Testing the Jupyter Notebook with PySpark

Since we have configured the integration by now, the only thing left is to test if all is working fine. So, let's run a simple Python script that uses Pyspark libraries and create a data frame with a test data set.

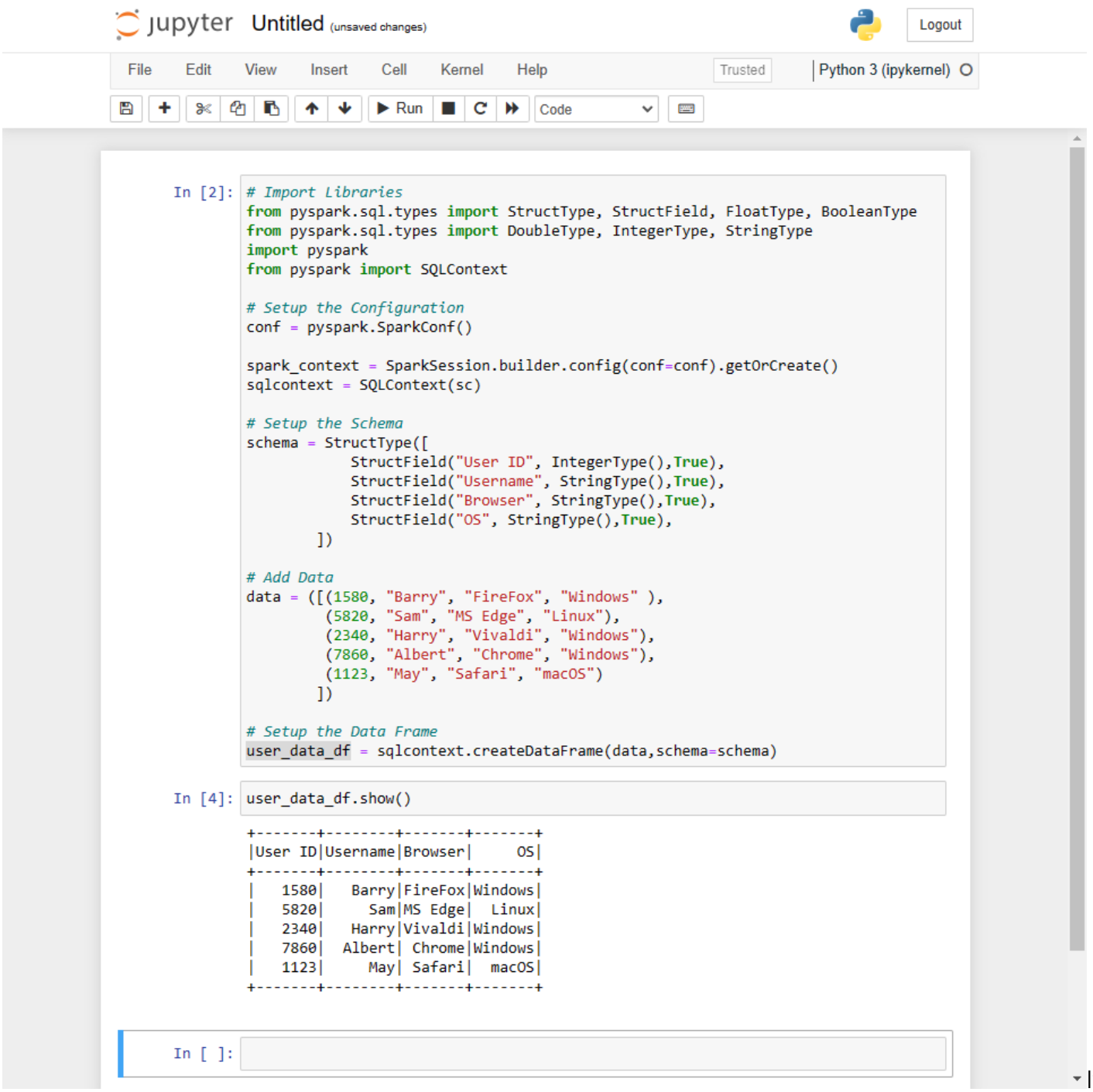
Create the data frame:

```
# Import Libraries
from pyspark.sql.types import StructType, StructField, FloatType, BooleanType
from pyspark.sql.types import DoubleType, IntegerType, StringType
import pyspark
from pyspark import SQLContext

# Setup the Configuration
conf = pyspark.SparkConf()

spark_context = SparkSession.builder.config(conf=conf).getOrCreate()
sqlcontext = SQLContext(sc)

# Setup the Schema
schema = StructType()

# Add Data
data = ()

# Setup the Data Frame
user_data_df = sqlcontext.createDataFrame(data,schema=schema)
```

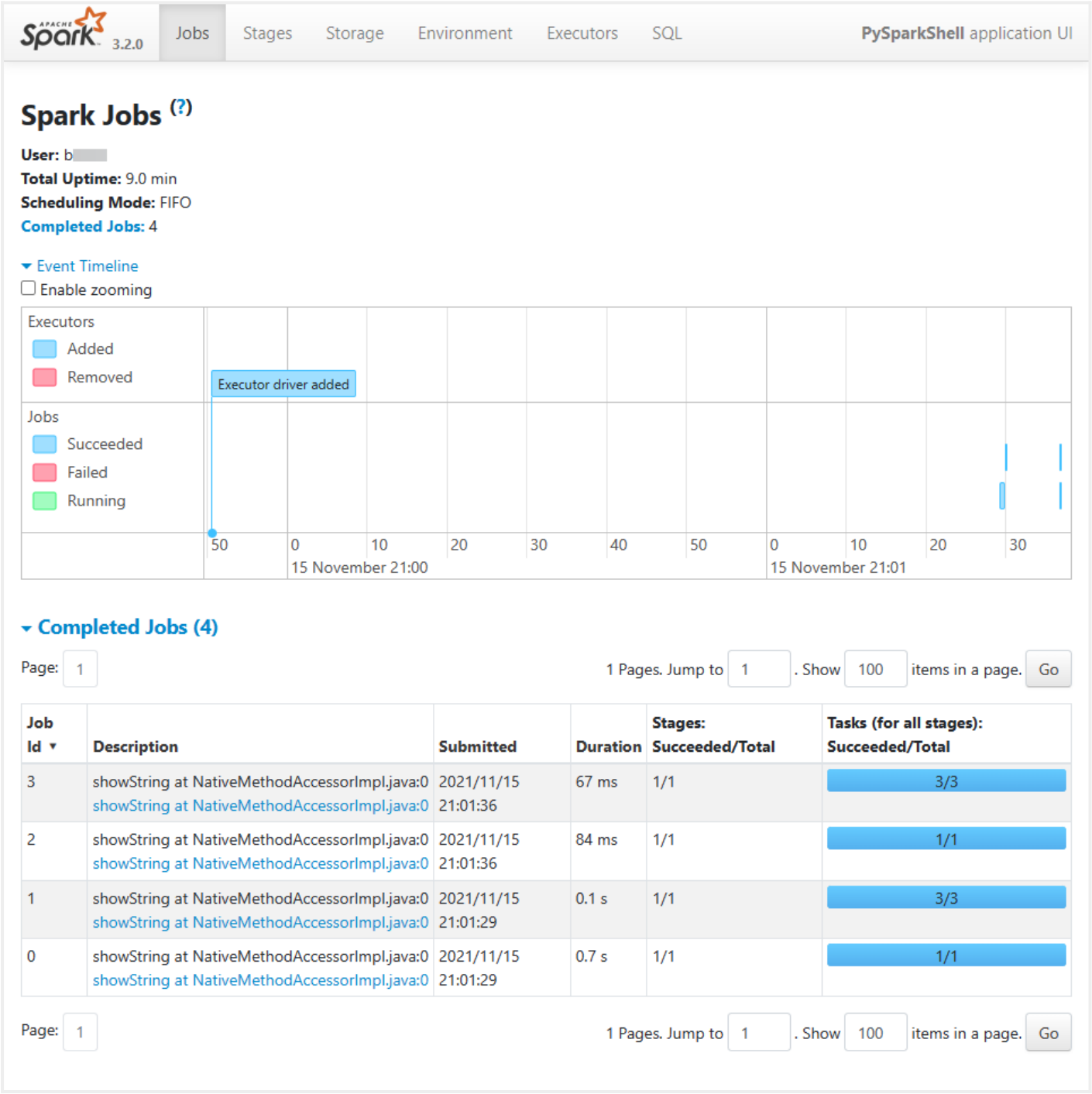
Print the data frame:

```
user_data_df.show()
```

RESULT:



If we look at the PySpark Web UI, which is accessible via port 4040, we can see the script execution job details as shown below.



The power of using Spark and Jupyter Notebook

Apache Spark is a powerful data analytics and big data tool. PySpark allows users to interact with Apache Spark without having to learn a different language like Scala. The combination of Jupyter Notebooks and Spark provides developers with a powerful and familiar development environment while harnessing the power of Apache Spark.