Kavya Gundala

26-12-2023

**Apache Spark:**

* Four main reasons form Apache Spark official website are good enough to convince to use Spark.

1. **Speed:**

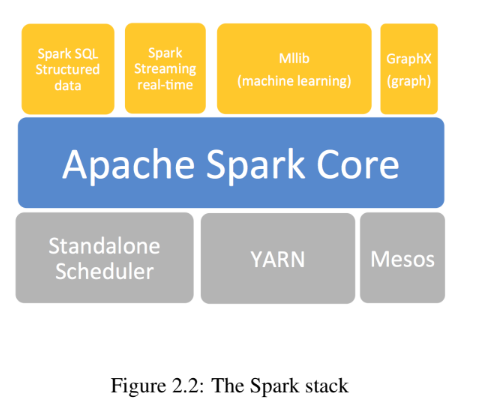
Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. Apache Spark has an advanced DAG execution engine that supports acyclic data flow and in-memory computing.

1. **Ease of Use:**

Write applications quickly in Java, Scala, Python, R. Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python and R shells.

1. **Generality:**

Combine SQL, streaming, and complex analytics. Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.



1. **Runs Everywhere:**

Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.



**Spark with Python(PySpark):**

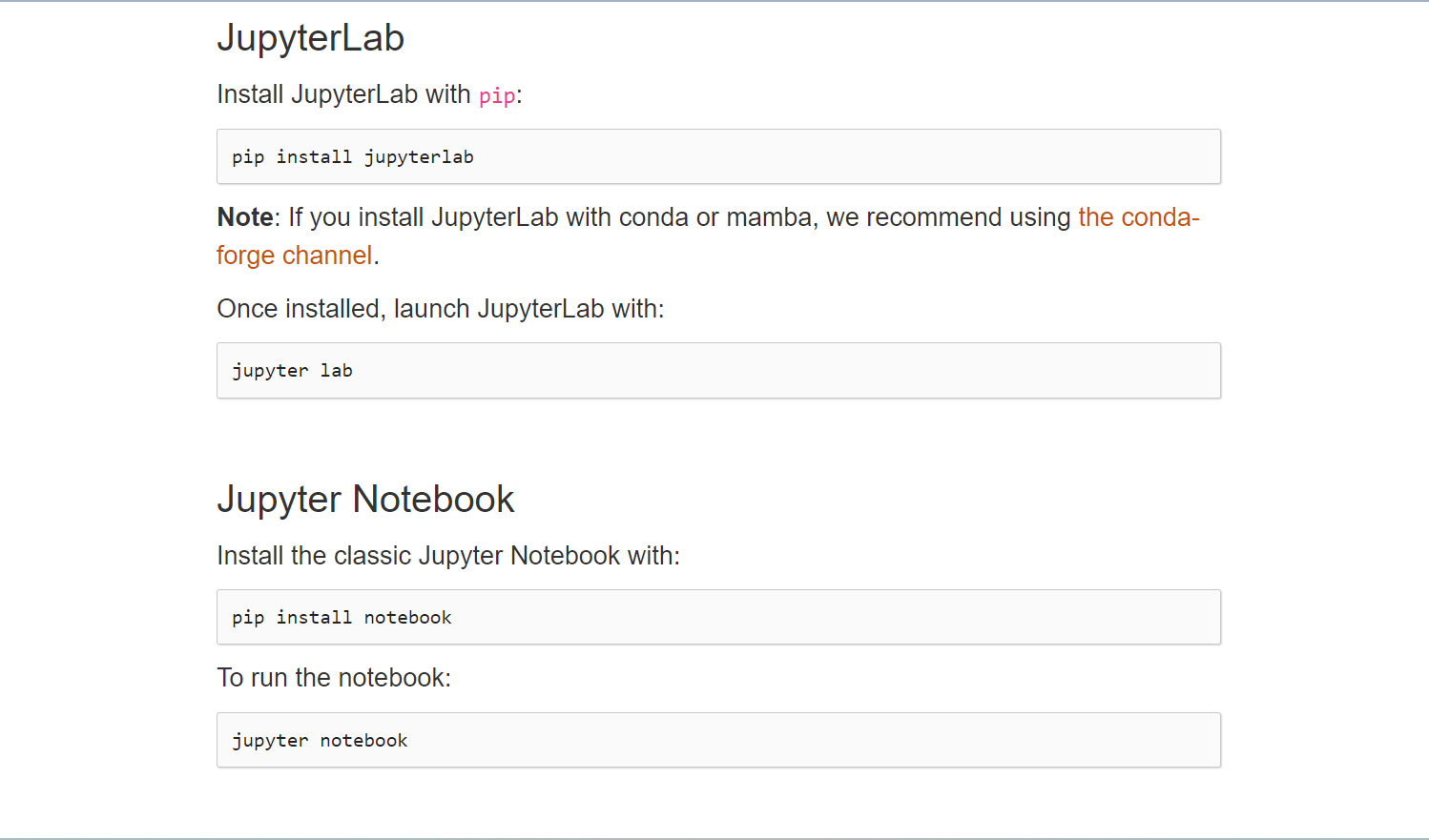
* Apache Spark is written in Scala programming language. PySpark has been released in order to support the collaboration of Apache Spark and Python, it actually is a Python API for Spark. In addition, PySpark, helps you interface with Resilient Distributed Datasets (RDDs) in Apache Spark and Python programming language. This has been achieved by taking advantage of the Py4j library.



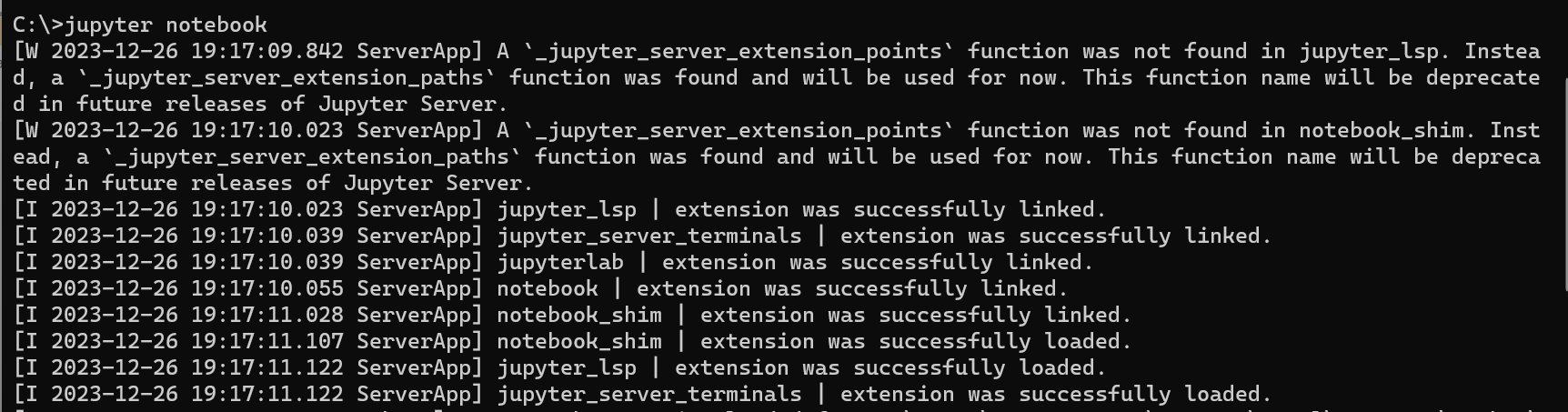
* Py4J is a popular library which is integrated within PySpark and allows python to dynamically interface with JVM objects. PySpark features quite a few libraries for writing efficient programs.

**PySpark with Jupyter:**

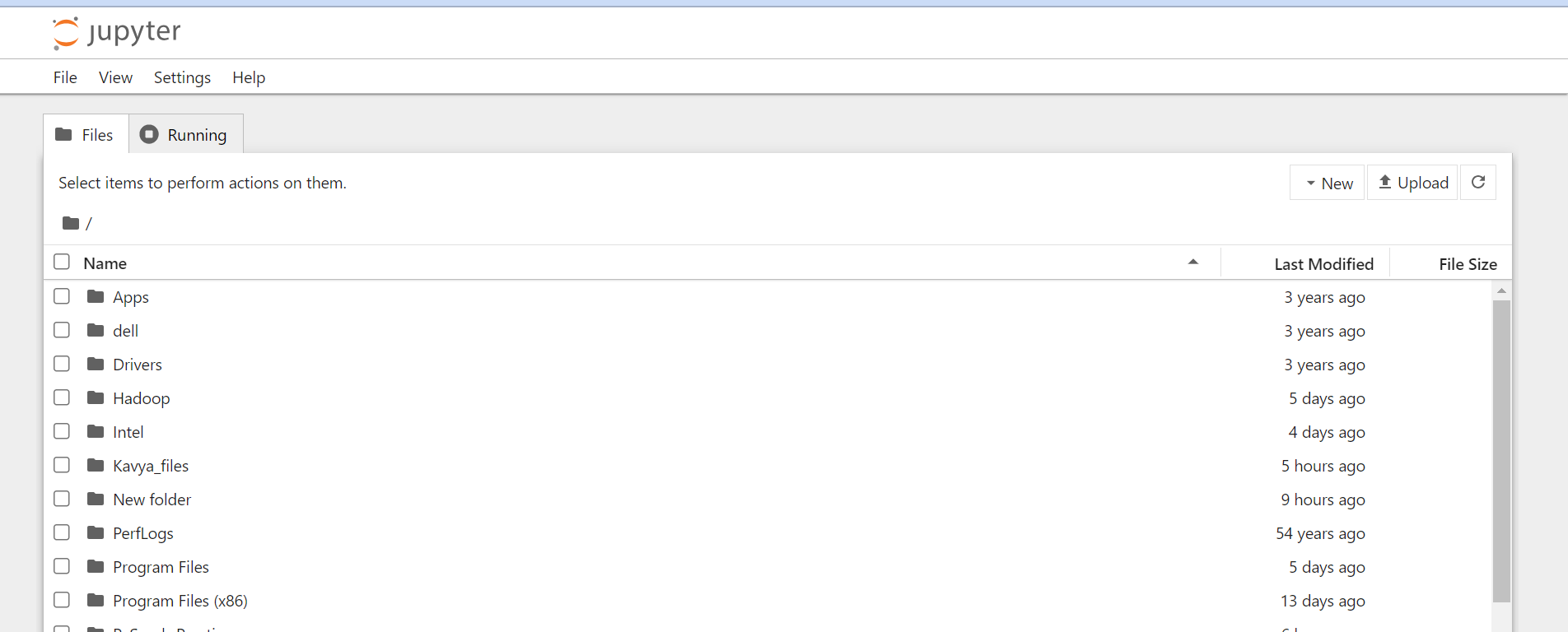
* Install JupyterLab, Jupyter Notebook using below commands in Command Prompt.



* Run Jupyter Notebook in cmd.



* In browser, localhost:8888/tree webpage will be opened.



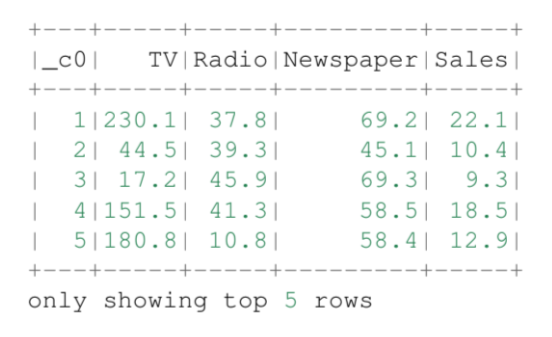
* Start Running PySpark commands in Jupyter notebook.
* Set up SparkSession



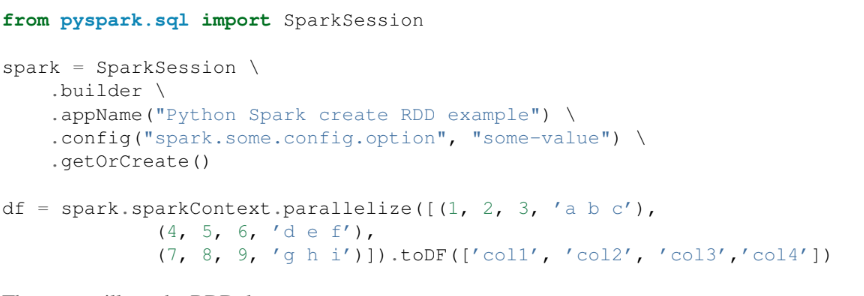
* Load the .csv datasets into Jupyter notebook.



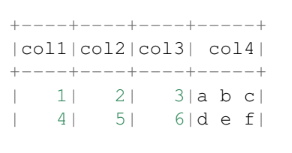
* df.show() – Showing First 5 rows from dataset.



* Create RDD(Resilient Distributed Dataset) using parallelize() function using below code:

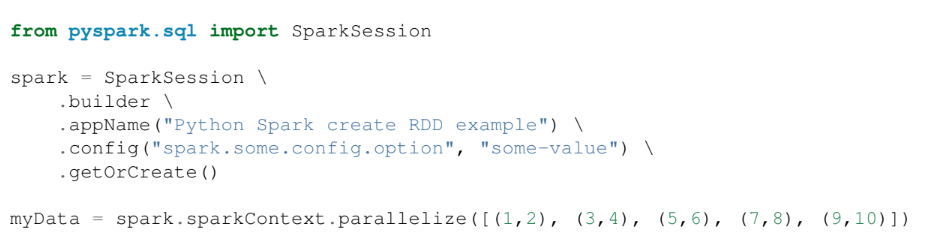


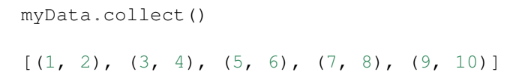




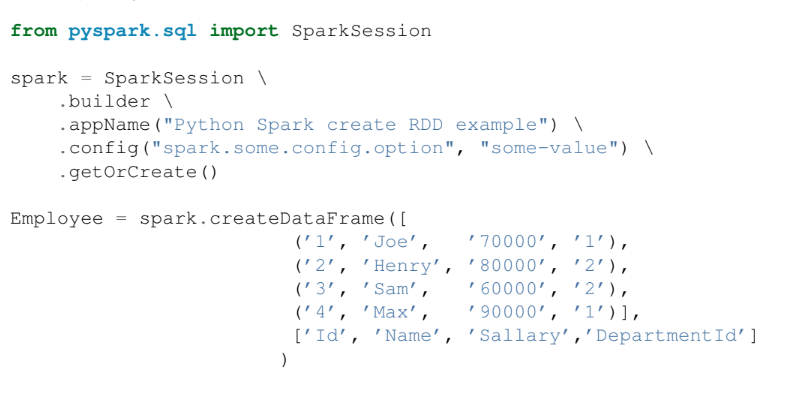


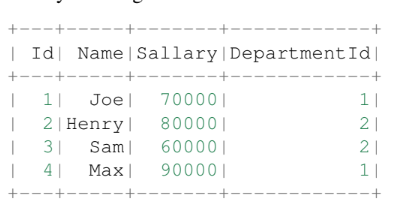
* Another example using SparkSession,





* By using createDataFrame() function:





* Example in PySpark to generate random number:



O/P: 3.14155616

**PySpark for ETL(EXTRACT, TRANSFORM, LOAD):**

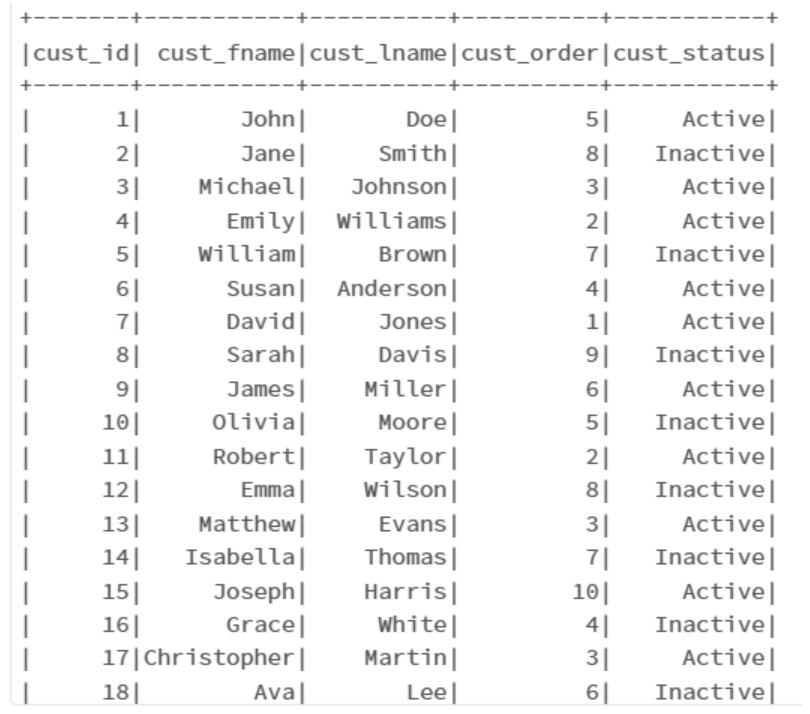
* PySpark, the Python API for Apache Spark, has revolutionized the way we handle Big Data. It’s an ETL powerhouse that combines the simplicity of Python with the scalability and performance of Spark.

**Why Choose PySpark for ETL?**

* Performance: PySpark leverages in-memory computing, making ETL processes faster than ever.
* Ease of Use: Python developers can seamlessly transition to PySpark due to its Pythonic syntax.
* Scalability: Handle massive datasets with ease, thanks to Spark’s distributed processing.
* Rich Ecosystem: PySpark integrates with popular tools and libraries, making it versatile for various data tasks.

**The PySpark ETL Workflow:**

* Extract: Retrieve data from various sources like databases, files, or APIs.
* Transform: Clean, aggregate, and manipulate data to fit your analysis needs.
* Load: Store the transformed data into a database or data warehouse for analysis.
* Source dataset:



* Start running PySpark for ETL:
* from pyspark.sql import SparkSession  
  from pyspark.sql.functions import col, concat, lit, floor, rand

# Initialize a Spark session  
spark = SparkSession.builder.appName(“ComplexETL”).getOrCreate()

# Define the external source and target paths  
source\_path = ‘your\_path’ # Update with your actual source file path  
target\_path = “your\_output\_path” # Update with your desired target file path

# Extract: Read data from an external CSV file  
df = spark.read.csv(source\_path, header=True,schema = ‘cust\_id int, first\_name string,last\_name string,cust\_order int,cust\_status string’)

# Transformation 1: Concatenate First and Last Names  
df = df.withColumn(“full\_name”, concat(col(“first\_name”), lit(“ “), col(“last\_name”)))

# Transformation 2: Calculate Net Salary (subtract 10% as taxes)  
df = df.withColumn(“net\_salary”, floor(lit(10000) + rand() \* lit(50)) )

#adding age column  
df = df.withColumn(“age”, floor(lit(20) + rand() \* lit(31)))

# Transformation 3: Filter by Age (age >= 30)  
df = df.filter(col(“age”) >= 30)

# Transformation 4: Group by Age and Calculate Average Salary  
avg\_salary\_by\_age = df.groupBy(“age”).agg({“net\_salary”: “avg”}).withColumnRenamed(“avg(salary)”, “avg\_salary”)

# Transformation 5: Sort by Age  
df = df.orderBy(“age”)

# Save the transformed data to an external CSV file  
df.write.csv(target\_path, mode=”overwrite”, header=True)

* After transforming the dataset will be:

