LEARNING PROGRES REVIEW

DWI HANDOYO

DATA ENGINEER BATCH 9 WEEK 8

DigitalSkola

SPARK

Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size. It does not have its own storage system, but runs analytics on other storage systems like HDFS, or other popular stores like Amazon Redshift, Amazon S3, Couchbase, Cassandra, and others. Spark is suitable tools when working with big data which require fast data processing.

Definition of big data reflected in three characteritics know as 3V.
✦ Volume: Huge amount of data that is being generated on a daily basis.
✦ Variety: Refers to structured, unstructured, and semistructured data that is gathered from multiple sources.
✦ Velocity: The speed at which data is being created in real-time. Non-conventional technology is required to process it.

Issues in handling big data

- ✦ Single machine is not enough power and resources to process data in big volume
- ✦ Requires a cluster or group of computers (many machines).
- ✦ Need a framework to coordinate and manage the cluster

Spark is suitable for big data handling. It can manage and coordinate the execution of tasks on a single data located on a computer cluster. Apache Spark features:

- ✦ Distributed cluster computing framework
- ✦ Efficient in-memory computing for large data sets
- ✦ Extremely fast data processing framework
- ✦ Provides support for Java, Scala, Python, R, and SQL

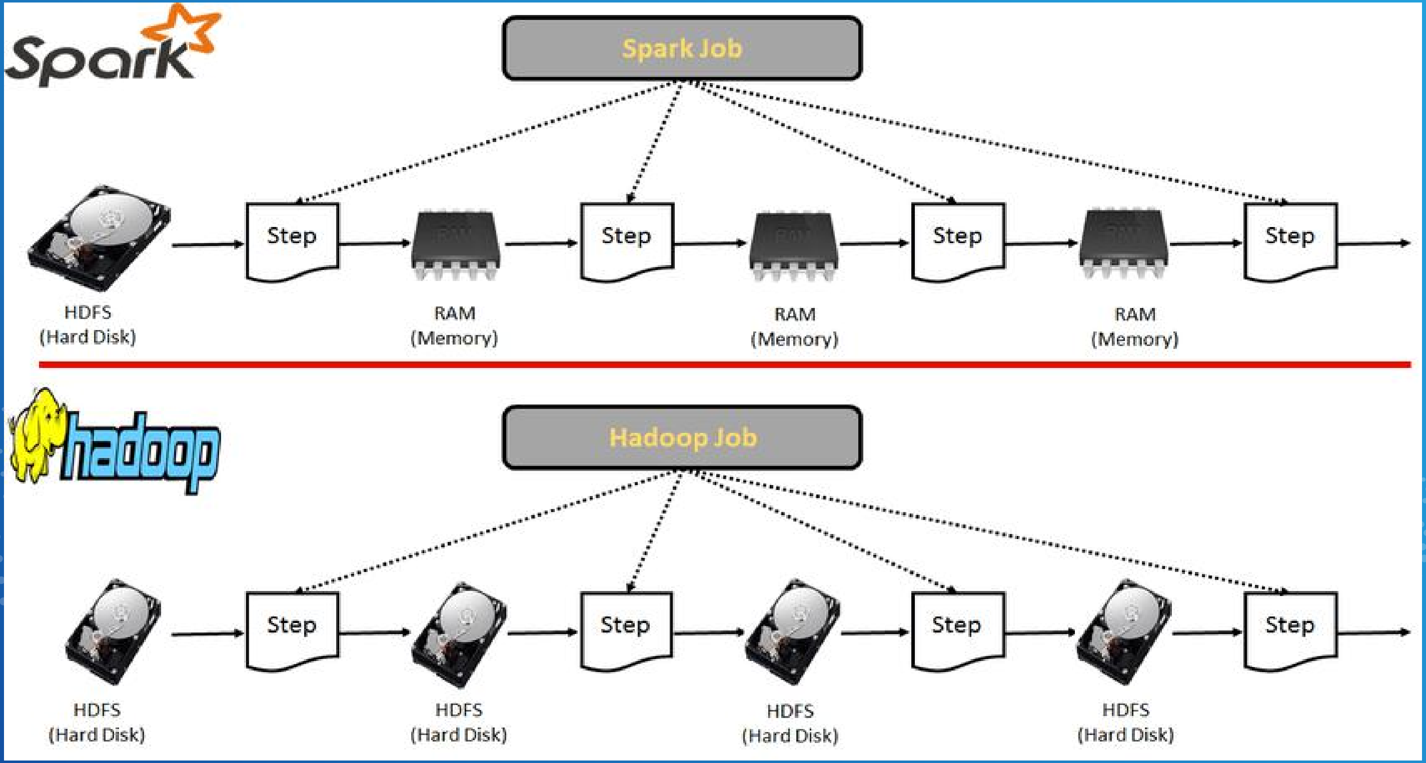
DigitalSkola

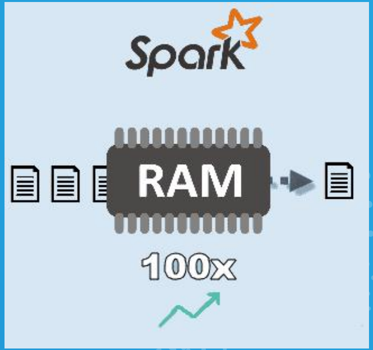
Comparison of Spark with other big data processing framework, ie. Hadoop:

Factors	Hadoop Map Reduce	Spark
Scalability	Easily scalable by adding nodes and disks for storage. Supports tens of thousands of nodes without a known limit.	A bit more challenging to scale because it relies on RAM for computations. Supports thousands of nodes in a cluster.
Fault Tolerant	A highly fault-tolerant system. Replicates the data across the nodes and uses them in case of an issue.	Tracks RDD block creation process, and then it can rebuild a dataset when a partition fails. Spark can also use a DAG to rebuild data across nodes.
Written in	Java	Scala
Speed	Faster than traditional system	100 times faster than Map Reduce
Ease of Use and Language Support	More difficult to use with less supported languages. Uses Java or Python for MapReduce apps.	More user friendly. Allows interactive shell mode. APIs can be written in Java, Scala, R, Python, Spark SQL.
Data Processing	Batch Processing	Batch and Real Time Processing
Data Store	Store data in disk	Store data in memory

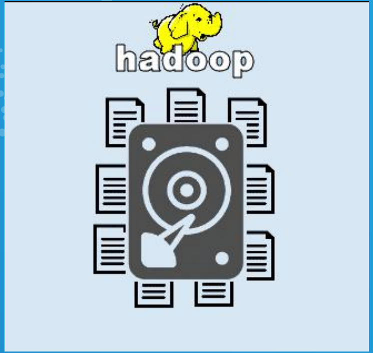
DigitalSkola

Comparison of big data processing framework, Spark vs Hadoop:





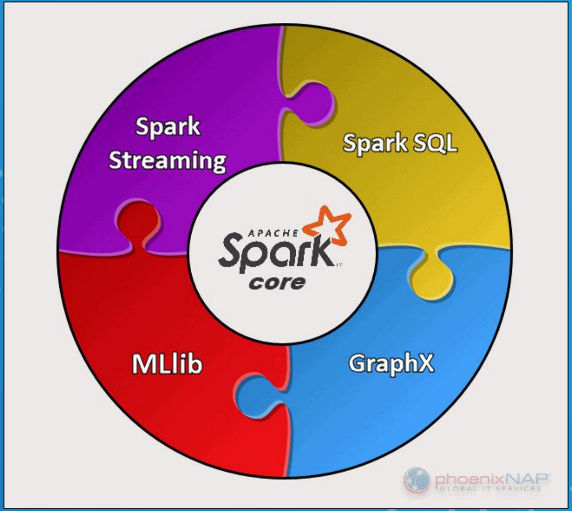
Read/write data on RAM

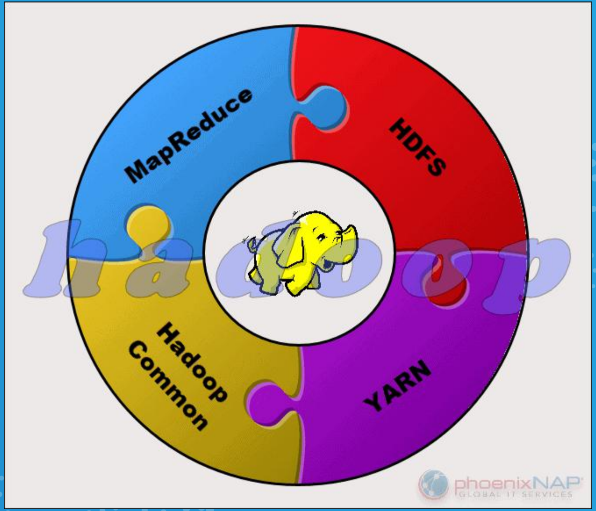


Read/write data on Harddisk

Spark Core Components

Comparison of components, Spark vs. Hadoop:





◉ Spark SQL

SQL database inside Spark.

◉ MLib

Libraries for Machine Learning purposes. In python, the libraries are pandas, scikitlearn, etc. to make practical machine learning scalable and easy

◉ GraphX

Spark API for graphs and graph-parallel computation. As an example in instagram, a follower has other followers, GraphX analyze connection of one entity to other entities.

◉ Spark Streaming

To process real-time data from various sources such as Kafka, Flume, and Amazon Kinesis. This processed data can be pushed out to file systems, databases, and live dashboards.

DigitalSkola

Spark Application Components

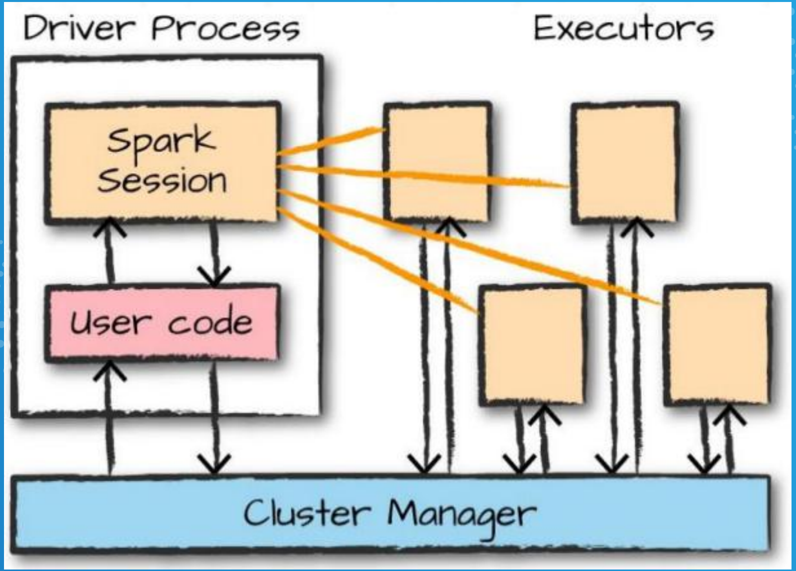
A Spark Application consists of a driver process and many executor processes.

◉ **Driver process: runs main(), resides on a single node, and does 3 things:**

- **Maintain information related to spark app**
- **Response to user or program input**
- **Analysis, distribution, scheduling tasks to executors**

◉ **Executors processes are in charge of running the given task. Running 2 things:**

- **Execute the code assigned by the driver**
- **Report the state of the compute back to the driver**



Spark uses a cluster manager to keep track of available resources. The process driver is responsible for executing code across executors

DigitalSkola

Spark Deployment Modes

◉ Cluster mode

Using collection of multiple machines, cluster mode is very suitable for use in production environments. Spark as cluster is difficult to build from scratch as well as its maintenance. Therefore services such as AWS and GCP provide the infrastructure that we can use. We simply submit the spark job and simply choose how many machines are needed.

◉ Local mode

Using single machine, like our personal laptop, local mode make it very easy for us to do testing, demos, and exploration. In local, installation can be done in below environments:

- Mac OS
- Windows
- Docker
- Conda

Cluster Manager Types

Synchronization of multiple computers can be managed in the following ways.

- Stand Alone
- Hadoop YARN
- Apache Mesos
- Kubernetes (AWS EMR)

Pyspark

PySpark is considered an interface for Apache Spark in Python. Through PySpark, we can write applications by using Python APIs. PySpark allows users to interact with Apache Spark without having to learn a different language like Scala.

Reasons for selecting pyspark:

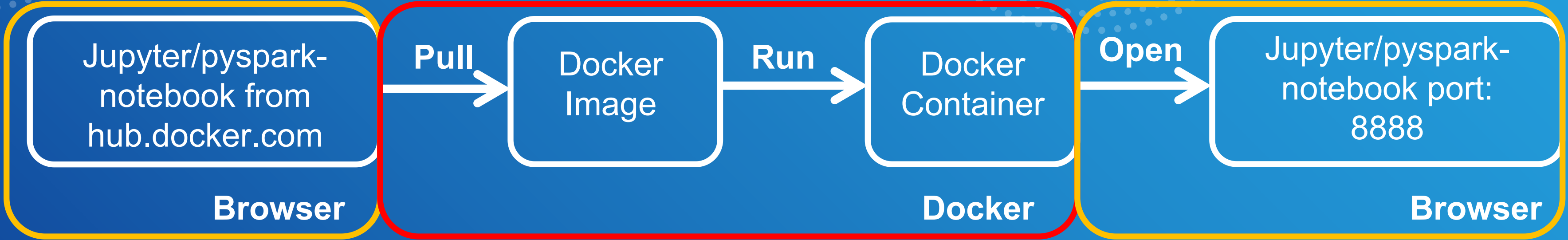
- Python is very easy to learn and implement.
- It provides simple and comprehensive API.
- With Python, the readability of code, maintenance, and familiarity is far better.
- It features various options for data visualization, which is difficult using Scala or Java.
- Python is widely used by Data Team (Data Analyst and Data Scientist)
- For those who already familiar with Python and libraries such as Pandas, then PySpark is a good language to create more scalable analyses and pipelines.

Jupyter/Pyspark-Notebook

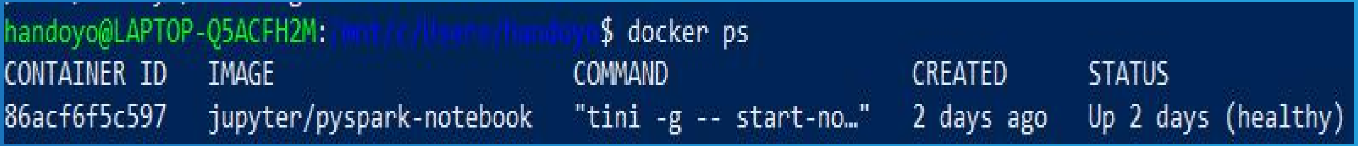
Jupyter Notebook is a popular application that enables us to edit, run and share Python code into a web view. It allows us to modify and re-execute parts of our code in a very flexible way. That's why Jupyter is a great tool to test and prototype programs. Jupyter Notebook running Python code. Jupyter/Pyspark Notebook is user interface platform to run pyspark interactively.

Installation of Jupyter Pyspark in Docker

- Pull jupyter/pyspark-notebook from hub.docker.com to create docker image
- Run docker container of jupyter/pyspark-notebook in powershell
- Open jupyter pyspark in browser for localhost:8888

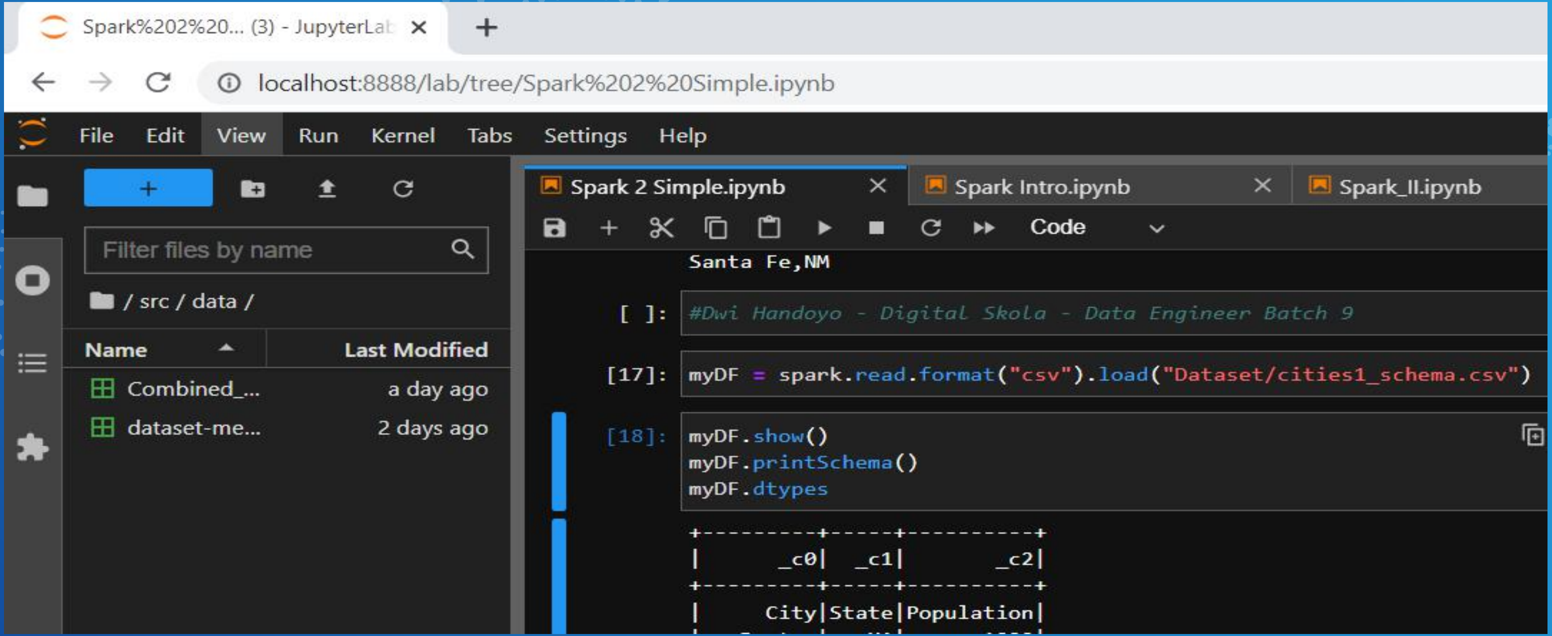

Jupyter/pyspark-notebook from hub.docker.com
Browser
Pull
Docker Image
Run
Docker Container
Docker
Open
Jupyter/pyspark-notebook port: 8888
Browser

Check if Jupyter/PySpark is running in Docker Container



```
handoyo@LAPTOP-Q5ACFH2M:/mnt/c/Users/handoyo$ docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED       STATUS
86acf6f5c597   jupyter/pyspark-notebook   "tini -g -- start-no…"   2 days ago    Up 2 days (healthy)
```

Open Jupyter/PySpark Notebook in Web Browser



localhost:8888/lab/tree/Spark%202%20Simple.ipynb

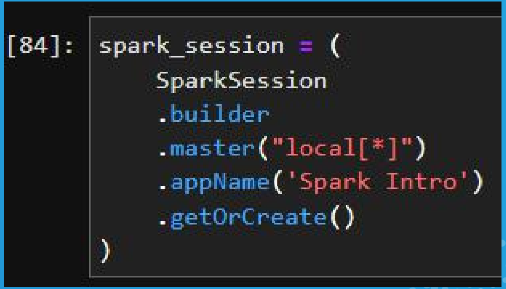
```
Santa Fe,NM

[ ]: #Dwi Handoyo - Digital Skola - Data Engineer Batch 9

[17]: myDF = spark.read.format("csv").load("Dataset/cities1_schema.csv")

[18]: myDF.show()
      myDF.printSchema()
      myDF.dtypes

+---------+-----+----------+
|      _c0|  _c1|       _c2|
+---------+-----+----------+
|     City|State|Population|
```

DATA ENGINEER BATCH 9 WEEK 7

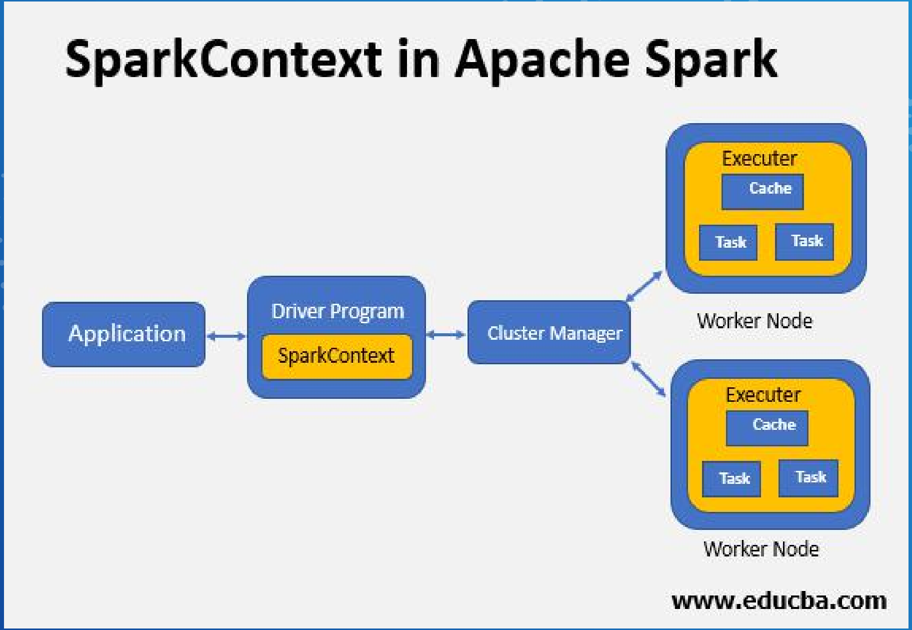
SPARK DATA PROCESSING

◉ Spark Session

Spark Session is the entry point to all the functionality in Spark, and is required if we want to build data frames in PySpark. We have to create Spark Sessioin at the first place when working with PySpark.

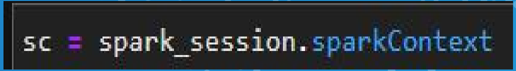


```
[84]: spark_session = (
          SparkSession
          .builder
          .master("local[*]")
          .appName('Spark Intro')
          .getOrCreate()
      )
```



◉ Spark Context

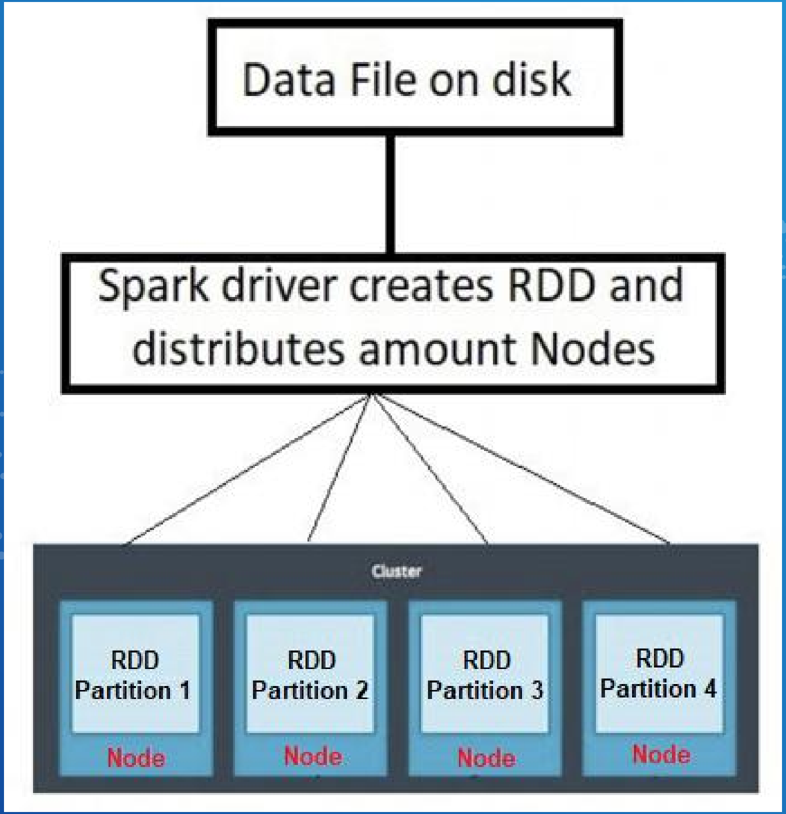
SparkContext is the client of the Spark execution environment and acts as the master of the Spark application. SparkContext sets up internal services and establishes a connection to the Spark execution environment.



```
sc = spark_session.sparkContext
```

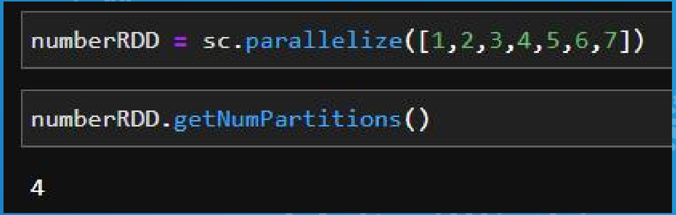
We can create RDDs, accumulators and variables, access Spark services and run jobs (until SparkContext stops) after SparkContext creation. Only one SparkContext can be active per JVM. We have to stop() the active SparkContext before creating a new one.

Resilient Distributed Dataset (RDD)



- **Resilient: Ability to withstand failures. The dataset will not corrupt or fail easily.**
- **Distributed: Spanning across multiple machines.**
- **Datasets: Collection of partitioned data e.g, Arrays, Tables, Tuples etc.**

By default, Spark will divide data into 4 partitions.

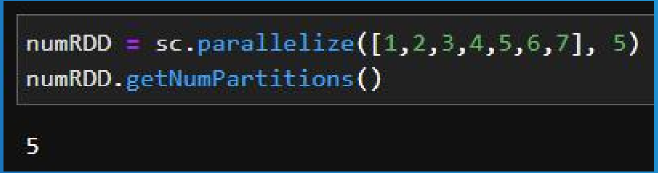


```
numberRDD = sc.parallelize([1,2,3,4,5,6,7])
```

```
numberRDD.getNumPartitions()
```

```
4
```

However, We can change the number of partitions by adding parameters while creating the RDD.



```
numRDD = sc.parallelize([1,2,3,4,5,6,7], 5)
numRDD.getNumPartitions()
```

```
5
```

Spark is more expensive than pandas, so be wise and don't overkill in designing. When dividing partitions, it is not always necessarily to be a lot, because too many partitions will have a draw back as well.

Partition Quantity	Draw Back
Too many partitions	Process will be slow due to many shuffling
Too less partitions	Process will be slow due to less parallel computation

Therefore, it is necessary to select the optimal number of partitions and cores. This can be done by trial and error and by considering:

- How many resources are available and can be used
- What is the amount or size of the data
- What operations will be performed, for example a complex join operation

✦ Create RDD

```
numRDD = sc.parallelize([1,2,3,4,5])
helloRDD = sc.parallelize("Hello world")
```

✦ Generate RDD from txt

```
myRDD = sc.textFile("Dataset/purplecow.txt")
```

✦ Generate RDD from json

```
userRDD = sc.wholeTextFiles("Dataset/json")
```

✦ Generate RDD from csv

```
cityRDD = sc.textFile("Dataset/cities1.csv")
```

◉ Pair RDD

* The actual data set is usually a key/value pair.
* Each row is a key and maps to one or more values.
* The key is the identifier, and the value is data.

✦ Create Pair RDD

```
my_tuple = [('Sam', 23), ('Mary', 34), ('Peter', 25)]
pairRDD_tuple = sc.parallelize(my_tuple)
```

```
airports = [("US", "JFK"),("UK", "LHR"),("FR", "CDG"),("US", "SFO")]
airportsRDD = sc.parallelize(airports)
```

PySpark DataFrame

* PySpark DataFrame is an immutable distributed data set.

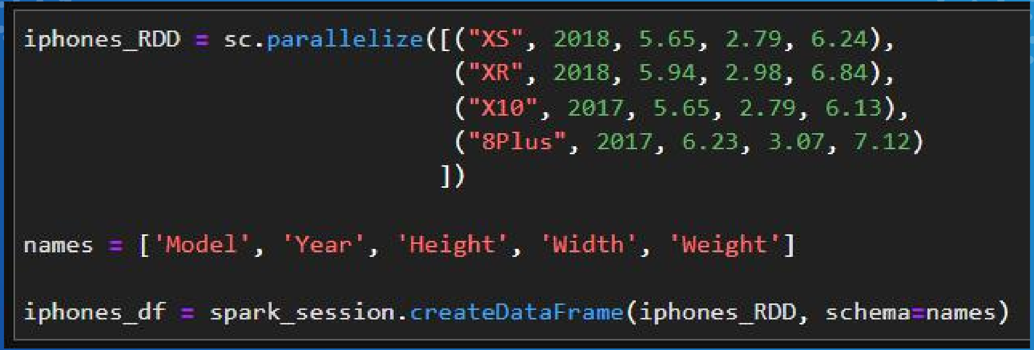
* PySpark SQL is a Spark library for structured data. It can provide a lot of information about data structures and computations.

* Designed to process both structured data (e.g. relational databases) and semi-structured data (e.g. JSON)

* The Dataframe API is available in Python, R, Scala, and Java.

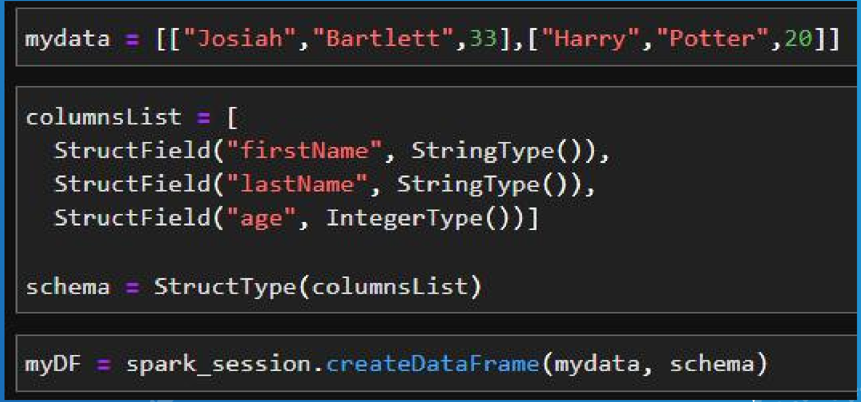
* DataFrames in PySpark support SQL queries (SELECT * from table) or expression methods (df.select())

✦ Create a DataFrame from RDD (Example-1)



```
iphones_RDD = sc.parallelize([("XS", 2018, 5.65, 2.79, 6.24),
                              ("XR", 2018, 5.94, 2.98, 6.84),
                              ("X10", 2017, 5.65, 2.79, 6.13),
                              ("8Plus", 2017, 6.23, 3.07, 7.12)
                             ])

names = ['Model', 'Year', 'Height', 'Width', 'Weight']

iphones_df = spark_session.createDataFrame(iphones_RDD, schema=names)
```

✦ Create a DataFrame from RDD (Example-2)

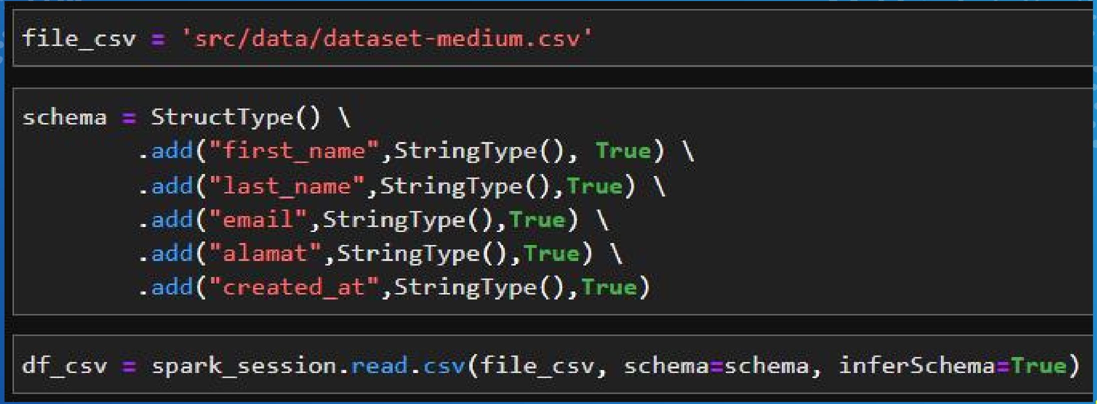


```
mydata = [["Josiah","Bartlett",33],["Harry","Potter",20]]
```

```
columnsList = [
  StructField("firstName", StringType()),
  StructField("lastName", StringType()),
  StructField("age", IntegerType())]

schema = StructType(columnsList)
```

```
myDF = spark_session.createDataFrame(mydata, schema)
```

✦ Create a DataFrame from Read File (Example-1)

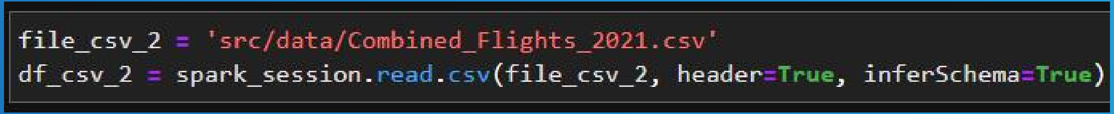


```
file_csv = 'src/data/dataset-medium.csv'
```

```
schema = StructType() \
        .add("first_name",StringType(), True) \
        .add("last_name",StringType(),True) \
        .add("email",StringType(),True) \
        .add("alamat",StringType(),True) \
        .add("created_at",StringType(),True)
```

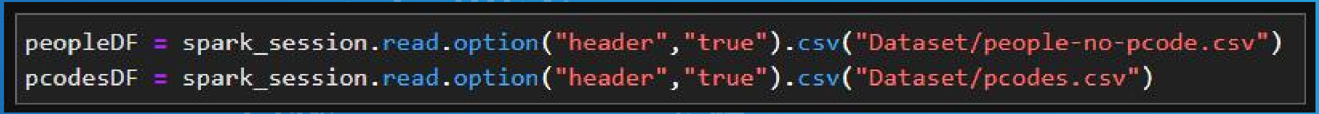
```
df_csv = spark_session.read.csv(file_csv, schema=schema, inferSchema=True)
```

✦ Create a DataFrame from Read File (Example-2)



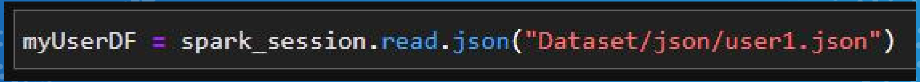
```
file_csv_2 = 'src/data/Combined_Flights_2021.csv'
df_csv_2 = spark_session.read.csv(file_csv_2, header=True, inferSchema=True)
```

✦ Create a DataFrame from Read File (Example-3)



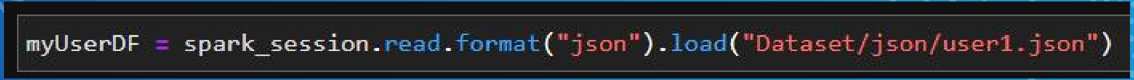
```
peopleDF = spark_session.read.option("header","true").csv("Dataset/people-no-pcode.csv")
pcodesDF = spark_session.read.option("header","true").csv("Dataset/pcodes.csv")
```

✦ Create a DataFrame from Read File (Example-4)



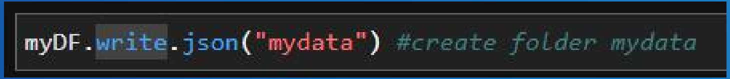
```
myUserDF = spark_session.read.json("Dataset/json/user1.json")
```

OR

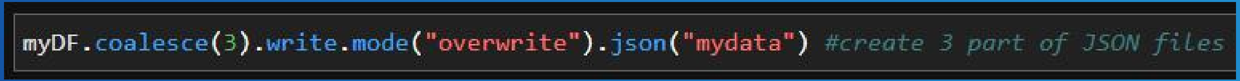


```
myUserDF = spark_session.read.format("json").load("Dataset/json/user1.json")
```

✦ Write a DataFrame into File



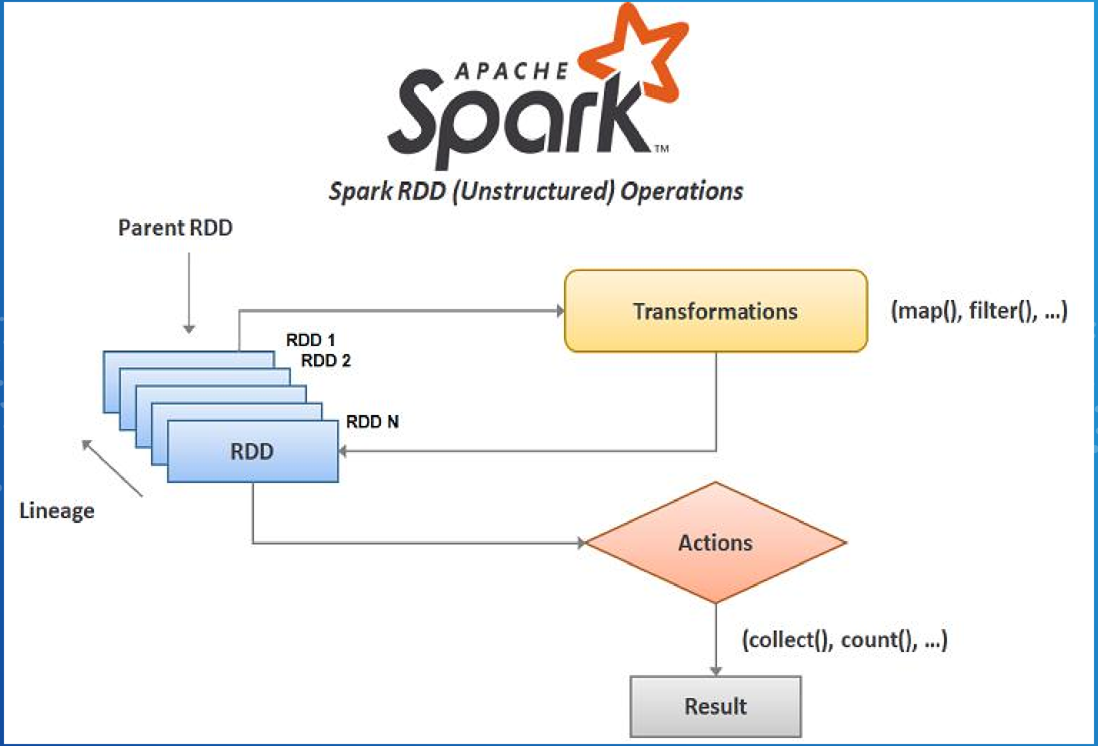
```
myDF.write.json("mydata") #create folder mydata
```



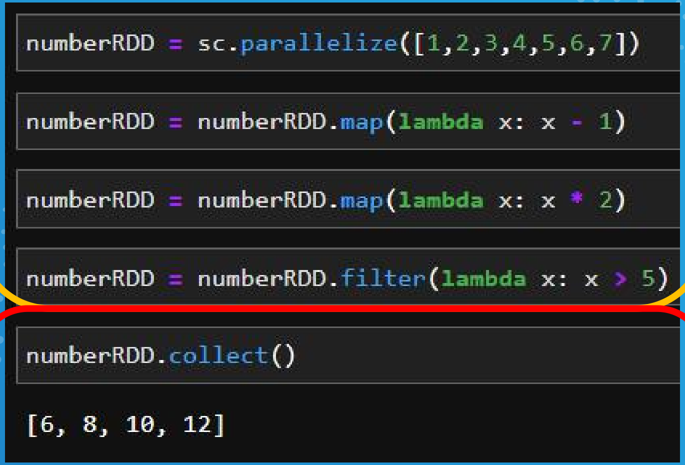
```
myDF.coalesce(3).write.mode("overwrite").json("mydata") #create 3 part of JSON files
```

PySpark Operations

Two processes in PySpark operations are **Transformations** to create new RDDs and **Actions** to perform computation on the RDDs.



Transformations



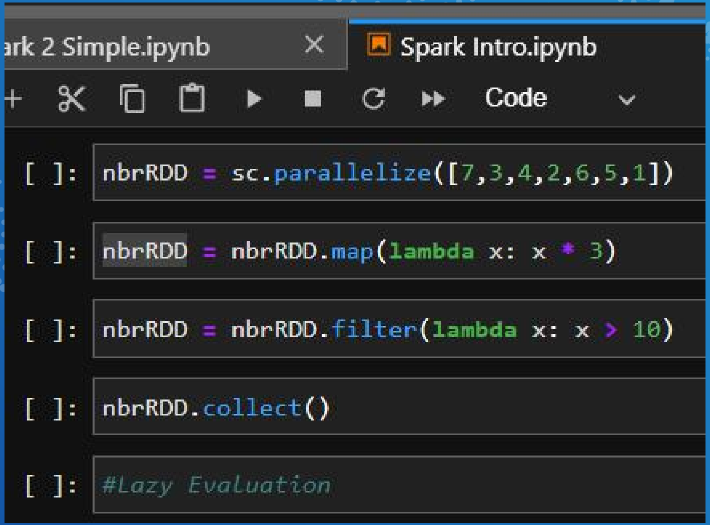
```
numberRDD = sc.parallelize([1,2,3,4,5,6,7])
numberRDD = numberRDD.map(lambda x: x - 1)
numberRDD = numberRDD.map(lambda x: x * 2)
numberRDD = numberRDD.filter(lambda x: x > 5)
numberRDD.collect()
[6, 8, 10, 12]
```

Action

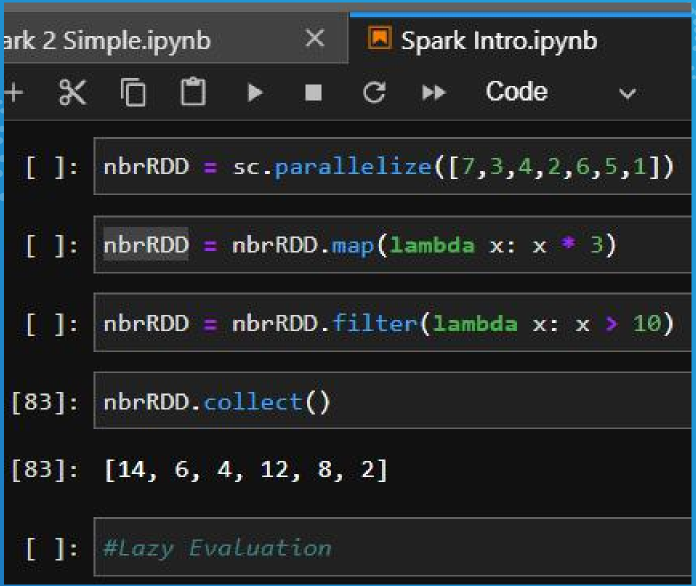
✦ Transformations

Data Structures in Spark are immutable. It means data structure cannot be changed once it is created. Hence, to "transform" a DataFrame, we need to provide instructions to Spark what modifications we want to have. These instructions are called transformations.

Transformations are Lazy evaluation, meaning execution won't start until the action is triggered.

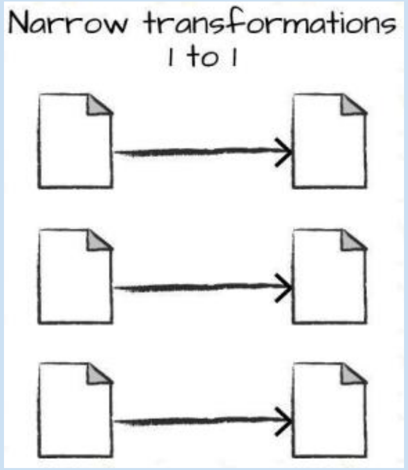
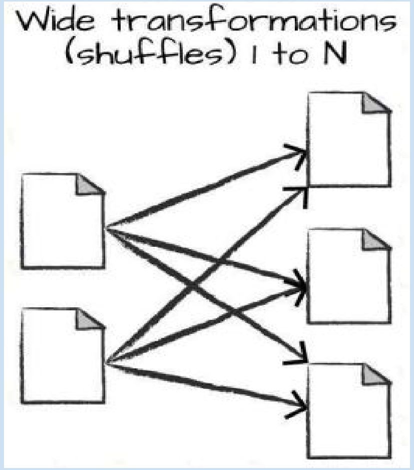


```
[ ]: nbrRDD = sc.parallelize([7,3,4,2,6,5,1])

[ ]: nbrRDD = nbrRDD.map(lambda x: x * 3)

[ ]: nbrRDD = nbrRDD.filter(lambda x: x > 10)

[ ]: nbrRDD.collect()

[ ]: #Lazy Evaluation
```



```
[ ]: nbrRDD = sc.parallelize([7,3,4,2,6,5,1])

[ ]: nbrRDD = nbrRDD.map(lambda x: x * 3)

[ ]: nbrRDD = nbrRDD.filter(lambda x: x > 10)

[83]: nbrRDD.collect()

[83]: [14, 6, 4, 12, 8, 2]

[ ]: #Lazy Evaluation
```

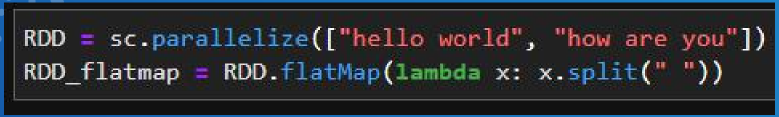
Two types of transformation are Narrow Dependencies and Wide Dependencies. check the comparison below.

Narrow Dependencies	Wide Dependencies
Each input partition produces only one output partition.	Input partition contributes to multiple output partition.
The operation performed is called pipelining, all are applied in memory.	Also known as shuffle. Can write to disk.
	

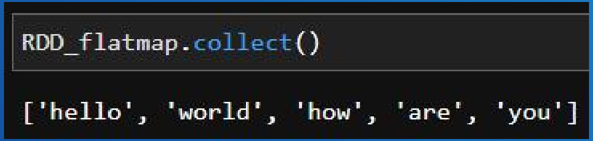
Transformation Operations on RDDs

Transformation	Description
map()	applies a function to all elements in RDD
filter()	returns a new RDD with only elements that match certain conditions
flatMap()	returns multiple values for each element in the original RDD
union()	combines two DataFrames and returns the new DataFrame with all rows from two Dataframes regardless of duplicate data

Example of flatMap()



```
RDD = sc.parallelize(["hello world", "how are you"])
RDD_flatmap = RDD.flatMap(lambda x: x.split(" "))
```



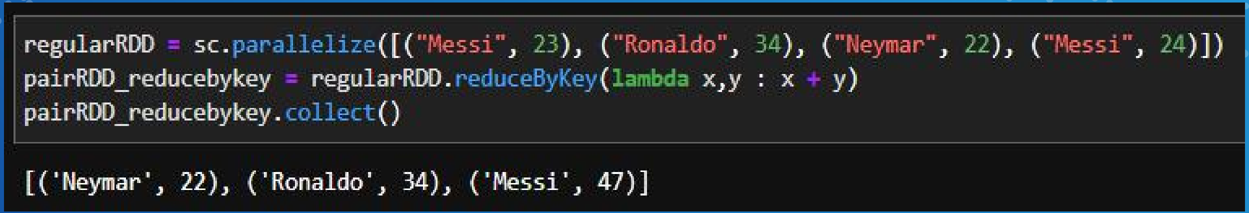
```
RDD_flatmap.collect()

['hello', 'world', 'how', 'are', 'you']
```

DigitalSkola

Transformation Operations on Pair RDDs

Transformation	Description
reduceByKey()	combines values with the same key. It performs parallel operations for each key in the dataset.
sortByKey()	command operations pair RDD with key. It returns the RDDs sorted by key in ascending or descending order.
groupByKey()	groups all values with the same key in an RDD pair
join()	a transformation that joins two RDD pairs based on their key

Example of reduceByKey()

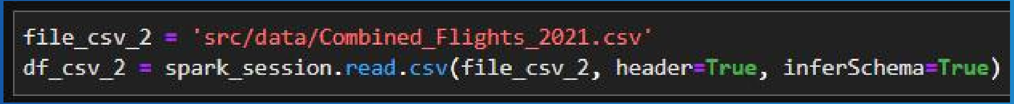


```
regularRDD = sc.parallelize([("Messi", 23), ("Ronaldo", 34), ("Neymar", 22), ("Messi", 24)])
pairRDD_reducebykey = regularRDD.reduceByKey(lambda x,y : x + y)
pairRDD_reducebykey.collect()

[('Neymar', 22), ('Ronaldo', 34), ('Messi', 47)]
```

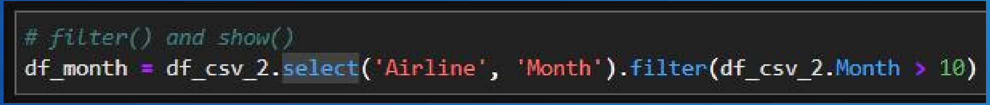
Transformation Operations on DataFrame

Transformation	Description
select()	returns a new DataFrame with the desired columns as specified in the inputs.
filter()	returns a new dataset formed by selecting those elements of the source on which the function returns true.
groupby()	to collect the identical data into groups on DataFrame/Dataset and perform aggregate functions on the grouped data.
orderby()	a spark sorting function used to sort the DataFrame
dropDuplicates()	returns a new DataFrame with duplicate rows removed, optionally only considering certain columns.
withColumnRenamed()	to rename one column or multiple DataFrame column names.

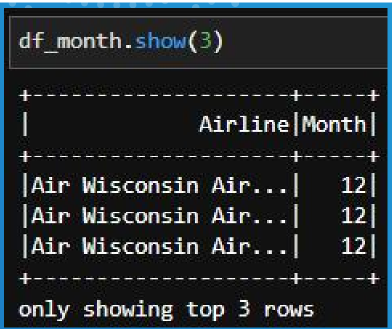
Example of select() and filter()



```
file_csv_2 = 'src/data/Combined_Flights_2021.csv'
df_csv_2 = spark_session.read.csv(file_csv_2, header=True, inferSchema=True)
```



```
# filter() and show()
df_month = df_csv_2.select('Airline', 'Month').filter(df_csv_2.Month > 10)
```



```
df_month.show(3)

+--------------------+-----+
|             Airline|Month|
+--------------------+-----+
|Air Wisconsin Air...|   12|
|Air Wisconsin Air...|   12|
|Air Wisconsin Air...|   12|
+--------------------+-----+
only showing top 3 rows
```

DigitalSkola

✦ Actions

▸ Action Operations on RDDs

This operation returns a value after we run the computation on RDD

Action	Description
collect()	returns all dataset elements as array
take(N)	returns an array with the first N elements of the dataset
first()	prints the first element of RDD
count()	returns the number of elements in RDD
getNumPartitions()	return the number of partitions of RDD

▸ Action Operations on Pair RDDs

Action	Description
countByKey()	count the number of elements for each key
collectAsMap()	returns key-value pairs in RDD as dictionary
getNumPartitions()	return the number of partitions of RDD

DigitalSkola

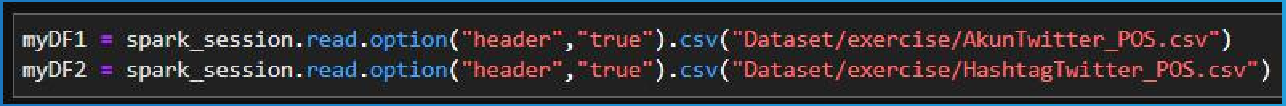
Action Operations on DataFrame

Action	Description
printSchema()	to print or display the schema of the DataFrame in the tree format along with column name and data type
head()	returns the first row
show()	print/show top 20 rows in a tabular form
count()	to get the number of rows
columns	to get the all columns of a DataFrame
describe()	to calculate the summary statistics of columns present in the DataFrame
rdd.getNumPartitions()	returns the number of partitions of DataFrame

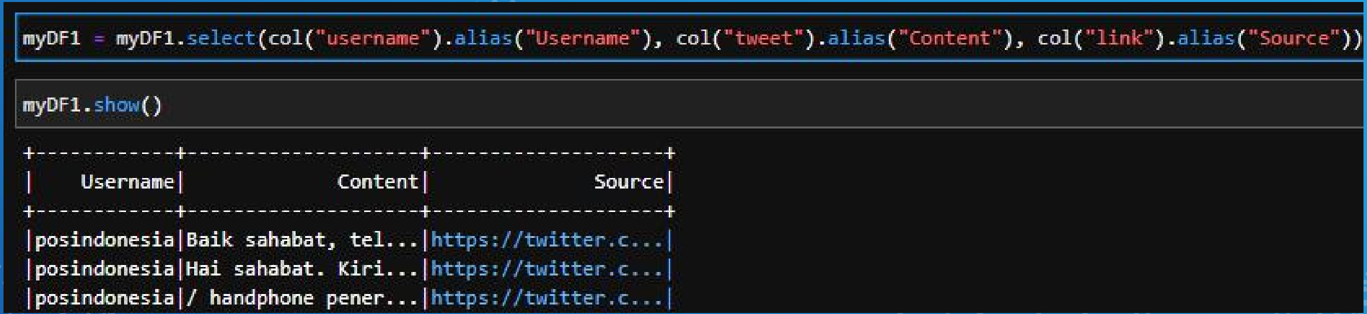
DigitalSkola

Project Homework

Load and combine the dataframe of the 3 data: AkunTwitter_POS.csv, HashtagTwitter_POS.csv, and Instagram_POS.json.



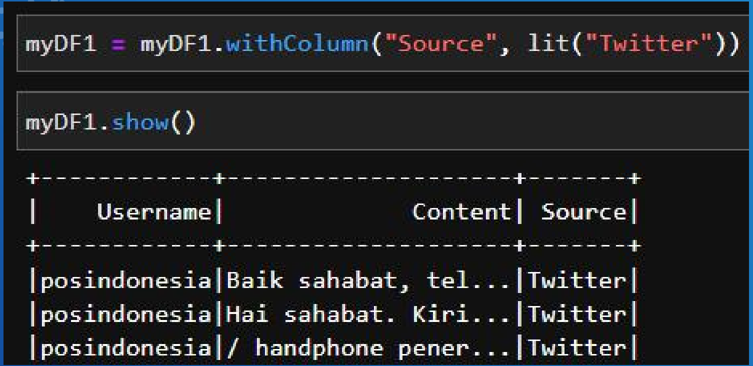
```
myDF1 = spark_session.read.option("header","true").csv("Dataset/exercise/AkunTwitter_POS.csv")
myDF2 = spark_session.read.option("header","true").csv("Dataset/exercise/HashtagTwitter_POS.csv")
```



```
myDF1 = myDF1.select(col("username").alias("Username"), col("tweet").alias("Content"), col("link").alias("Source"))
```

```
myDF1.show()
```

```
+------------+--------------------+--------------------+
|    Username|             Content|              Source|
+------------+--------------------+--------------------+
|posindonesia|Baik sahabat, tel...|https://twitter.c...|
|posindonesia|Hai sahabat. Kiri...|https://twitter.c...|
|posindonesia|/ handphone pener...|https://twitter.c...|
```

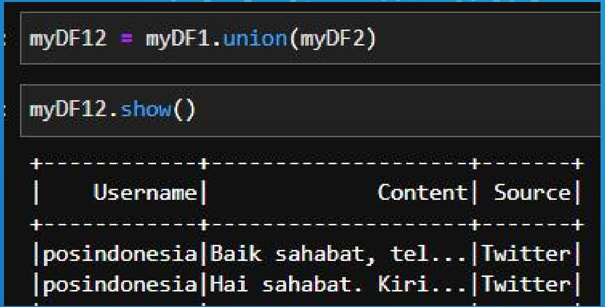


```
myDF1 = myDF1.withColumn("Source", lit("Twitter"))
```

```
myDF1.show()
```

```
+------------+--------------------+-------+
|    Username|             Content| Source|
+------------+--------------------+-------+
|posindonesia|Baik sahabat, tel...|Twitter|
|posindonesia|Hai sahabat. Kiri...|Twitter|
|posindonesia|/ handphone pener...|Twitter|
```

Combine data from the two csv files

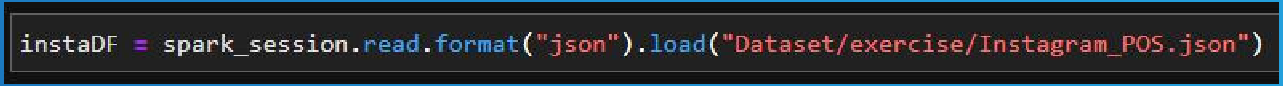


```
myDF12 = myDF1.union(myDF2)
```

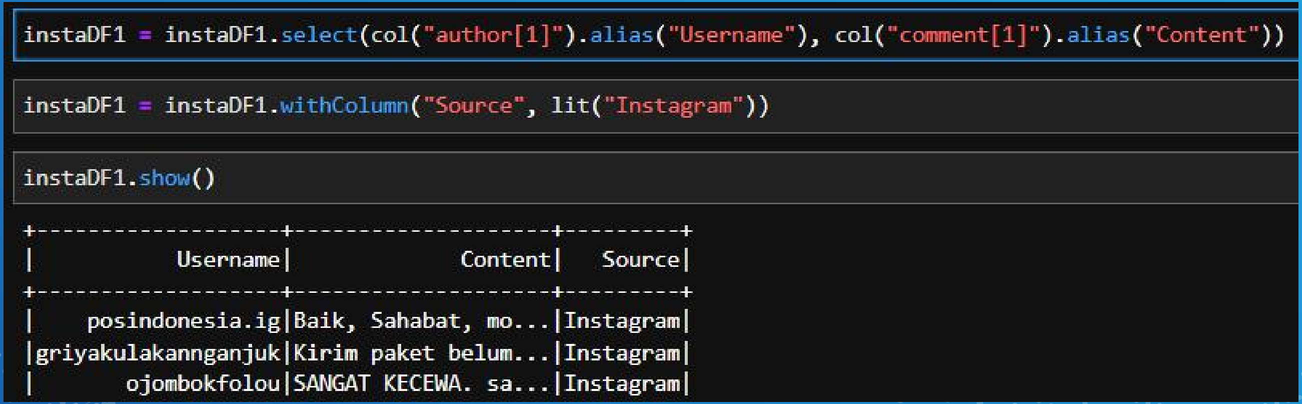
```
myDF12.show()
```

```
+------------+--------------------+-------+
|    Username|             Content| Source|
+------------+--------------------+-------+
|posindonesia|Baik sahabat, tel...|Twitter|
|posindonesia|Hai sahabat. Kiri...|Twitter|
```

From Instagram_POS.json :



```
instaDF = spark_session.read.format("json").load("Dataset/exercise/Instagram_POS.json")
```



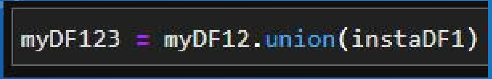
```
instaDF1 = instaDF1.select(col("author[1]").alias("Username"), col("comment[1]").alias("Content"))
```

```
instaDF1 = instaDF1.withColumn("Source", lit("Instagram"))
```

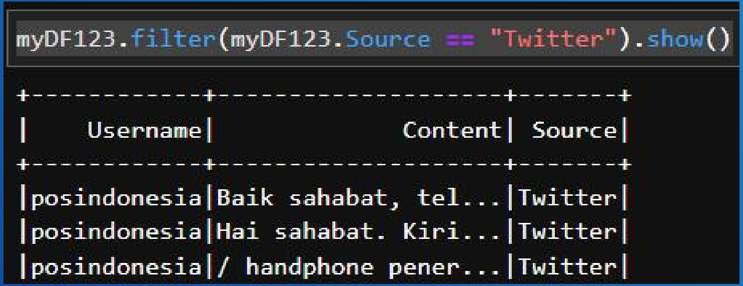
```
instaDF1.show()
```

```
+------------------+------------------+---------+
|          Username|           Content|   Source|
+------------------+------------------+---------+
|    posindonesia.ig|Baik, Sahabat, mo...|Instagram|
|griyakulakannganjuk|Kirim paket belum...|Instagram|
|       ojombokfolou|SANGAT KECEWA. sa...|Instagram|
```

Combine data from all csv and json files into single DataFrame called "myDF123".

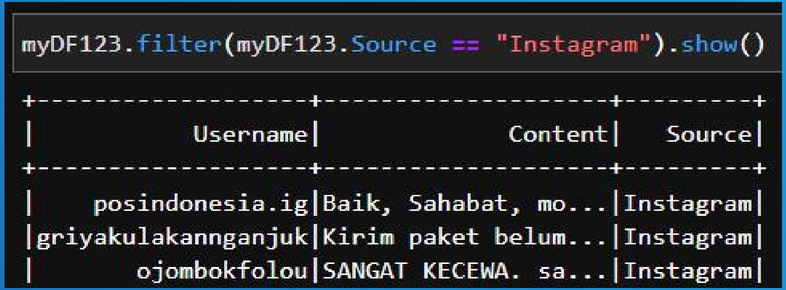


```
myDF123 = myDF12.union(instaDF1)
```



```
myDF123.filter(myDF123.Source == "Twitter").show()
```

```
+------------+--------------------+-------+
|    Username|             Content| Source|
+------------+--------------------+-------+
|posindonesia|Baik sahabat, tel...|Twitter|
|posindonesia|Hai sahabat. Kiri...|Twitter|
|posindonesia|/ handphone pener...|Twitter|
```



```
myDF123.filter(myDF123.Source == "Instagram").show()
```

```
+-------------------+--------------------+---------+
|           Username|             Content|   Source|
+-------------------+--------------------+---------+
|    posindonesia.ig|Baik, Sahabat, mo...|Instagram|
|griyakulakannganjuk|Kirim paket belum...|Instagram|
|       ojombokfolou|SANGAT KECEWA. sa...|Instagram|
```

DigitalSkola

Thank You

End of Presentation