06/2/24

Spark Session :-

* entry point to create PySpark RDD, Dataframe.
* created using SparkSession.builder

1) builder() ⇒ builder() pattern method.
2) appName() - used to set application name
3) getOrCreate() ⇒ returns a SparkSession obj if already exists, and creates a new one if not exist
4) master()- If you are running it on cluster you need to master name as an argument to master()

* SparkSession object spark is by default available in Pyspark shell.

PySpark RDD operations

* RDD → core data structure of PySpark
  - → low-level obj
  - → highly efficient in performing distributed tasks

Two types → 1) Transformations
→ 2) Actions

Transformations :-

* takes an RDD as i/p & produces another RDD as o/p.
* If transformation is applied to RDD, it returns new RDD, original RDD is same (immutable)
* After applying transformations it creates DAG and gives result for actions => Lazy Evaluation process

Actions

* applied on RDD to produce single value.
* produces non RDD value. (thus laziness is removed)

Transformations → applied on RDD
→ gives another RDD

Actions → performed on RDD
→ gives non-RDD value.

Actions

1) .collect() => returns a list of all elements of RDD

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
```

3) .first() action

* first element of RDD

```
collect_rdd = sc.parallelize([1,2,3,6,8,9])
print(collect_rdd.count())
```

2) .count() => returns no. of elements of RDD

rdd =>

```
count_rdd = sc.parallelize([1,2,3,4])
print(count_rdd.count())
```

3) .first() => returns the first element of RDD

* used when we want to verify exact data is loaded in our RDD as per requirements

Ex:- For natural nums, we can check if first element is 1.

```
first_rdd = sc.parallelize([1,2,3,4])
print(first_rdd.first())
```

4) .take() => .take(n) => returns n number of elements from RDD. (n = first)

```
take_rdd = sc.parallelize([1,2,3,4,5])
print(take_rdd.take(3))   // 1 2 3 (first 3 ele)
```

5) .reduce() => this operation uses an anonymous function or lambda.

Ex:- add all elements of RDD.

```
reduce_rdd = sc.parallelize([1,3,4,9])
print(reduce_rdd.reduce(lambda x,y: x+y))
```

6) saveAsTextFile() ⇒ serve resultant RDD as a text file.

```
save_rdd = sc.parallelize([1,2,3,4,5,6])
save_rdd.saveAsTextFile("file.txt")
```

⇓ generates a directory

Transformations

1) .map() transformation ⇒ maps a value to the elements of RDD.

* .map() " takes an anonymous fn and applies this fn to all elements of RDD.

Ex:-
```
my_rdd = sc.parallelize([1,2,3,4])
print(my_rdd.map(lambda x: x+10).collect())
```

2) .filter() ⇒ filters elements from RDD.

* .filter() takes anonymous fn to mention condition

Ex:- ① Even nos
```
filter_rdd = sc.parallelize([1,2,4,7,8])
print(filter_rdd.filter(lambda x: x%2==0).collect())
```

Starting with R

②
```
filter_rdd2 = sc.parallelize(["Rahul","Ram","Sam"])
print(filter_rdd2.filter(lambda x: x.startsWith("R")).collect())
```

3) .union() ⇒ Combines two RDDs and returns union of input of two RDDs

Ex:- union_inp = sc.parallelize([2,4,5,6,7,8,9])

union_rdd1 = union_inp.filter(lambda x: x%2==0)

" _rdd2 = " " : " (" x: x%3==0)

print(union_rdd1.union(union_rdd2).collect())

4) .flatMap() ⇒ same as .map() but return separate values for each element from original RDD.

Ex:- flatmap_rdd = sc.parallelize(["Hey there", "This is PySpark"])

flatmap_rdd.flatMap(lambda x: x.split(" ")).collect())

Dataframes :-

created by two ways

(i) from existing RDD

(ii) from external file sources csv, txt, json

(i) Creating dataframe from existing RDD

~~from~~ (i) create RDD using parallelize()

(ii) convert RDD into dataframe

```
from pyspark import SparkContext
from pyspark.sql import SparkSession
sc = SparkContext.getOrCreate()
spark = SparkSession.builder.appName("DF from RDD")
    .getOrCreate()
```

```
rdd = sc.parallelize([('C', 85, 76, 87, 91),
                      ('B', 70, 77, 90, 88),
                      ('A', 12, 14, 90, 100)], 4)
```
no. of Partitions

```
print(type(rdd))    #RDD
sub = ['Division', 'Eng', 'math', 'chem', 'phy']
marks_df = spark.createDataFrame(rdd, schema = sub)
print(type(marks_df))    #.dataframe
marks_df.printSchema()
marks_df.show()
```

(ii) from External File

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Df from External file').getOrCreate()
```

(a) from csv

```
df = spark.read.csv('path', sep=',', inferSchema=True, header=True)
df.show()
print(type(df))
print df.printSchema()
```

(b) from text file

```
df = spark.read.text("path")
df.show()
```

(c) from JSON file

```
df = spark.read.json("path", multiLine=True)
df.show()
```

(iii) additional methods

using pandas.

```
df1 = spark.read.csv('path')
df2 = df1.toPandas()
df2
```

Reading multiple files

```
files = ['path1', 'path2']
df = spark.read.csv(files, sep=',', inferSchema=True,
                    header=True)
df.show()
```

Renaming columns in dataframe

1) withColumnRenamed()

DataFrame.withColumnRenamed(existing, new)

Ex:- df.withColumnRenamed("DOB", "Date Of Birth").show()

Renaming multiple columns

df.withColumnRenamed("Gender", "Sex").withC e
("Salary", "Amt").
show()

2) selectExpr()

DataFrame.selectExpr(expr) → SQL expression

df.selectExpr("Name as name", "DOB", "Gender",
"Salary").show()

3) select()

DataFrame.select(cols) → list of colmn names

from pyspark.sql.functions import col

```
data = df.select(col("Name"), col("Age"), col("Gender"),
                 col("Salary").alias("Amount"))
data.show()
```

④ toDF()

toDF(*col) ⇒ returns new df with new specified column names.

Ex:- Data_List = ["Emp Name", "Date of Birth", "M/F", "Amt"]

```
new_df = df.toDF(*Data_list)
new_df.show()
```