Big Data

PySpark

Instructor: Trong-Hop Do

April 24th 2021

S3Lab
Smart Software System Laboratory

“Big data is at the foundation of all the megatrends that are happening today, from social to mobile to cloud to gaming.”

– Chris Lynch, Vertica Systems

Install Spark on Windows

Install Java 8 or Later

- To install Apache Spark on windows, you would need Java 8 or later version hence download the Java version from Oracle and install it on your system.

- https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html

Windows x64	166.79 MB	jdk-8u271-windows-x64.exe

Apache Spark Installation on Windows

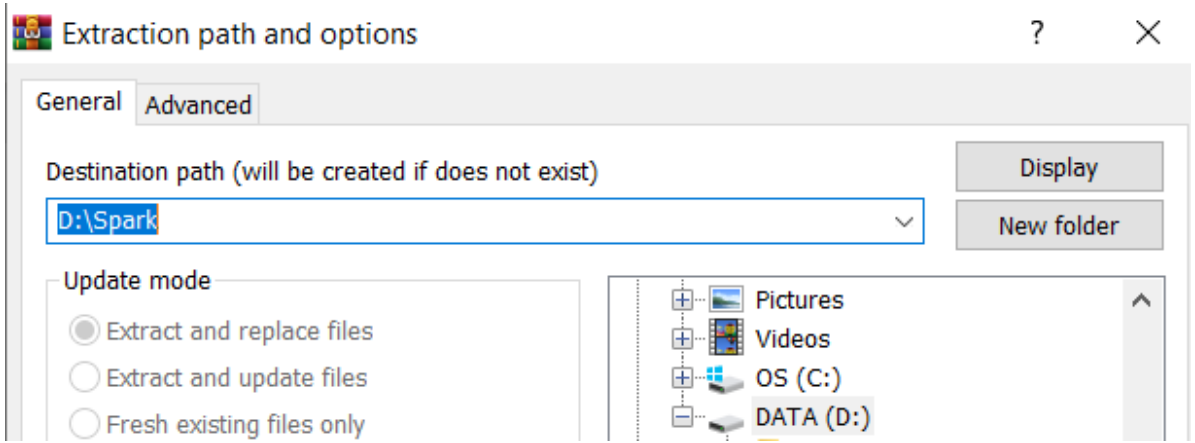
- Download Apache spark
- https://spark.apache.org/downloads.html

Download Apache Spark™

1. Choose a Spark release: 3.0.1 (Sep 02 2020)
2. Choose a package type: Pre-built for Apache Hadoop 2.7
3. Download Spark: spark-3.0.1-bin-hadoop2.7.tgz
4. Verify this release using the 3.0.1 signatures, checksums and project release KEYS.

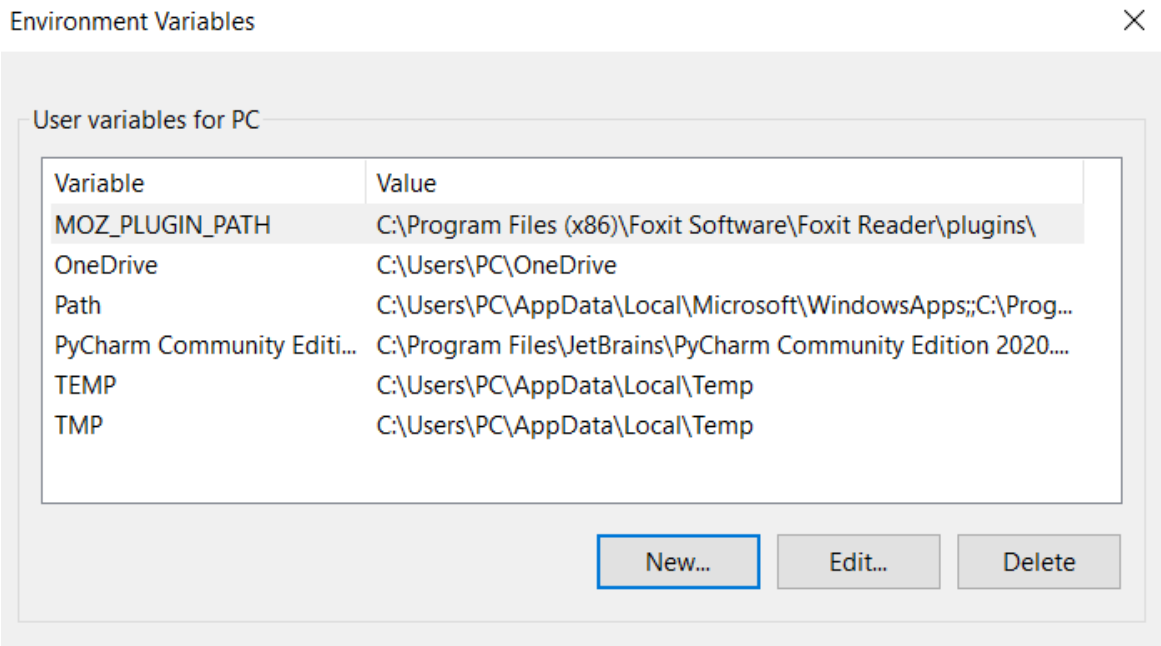
Apache Spark Installation on Windows

- Extract the zip file to any folder

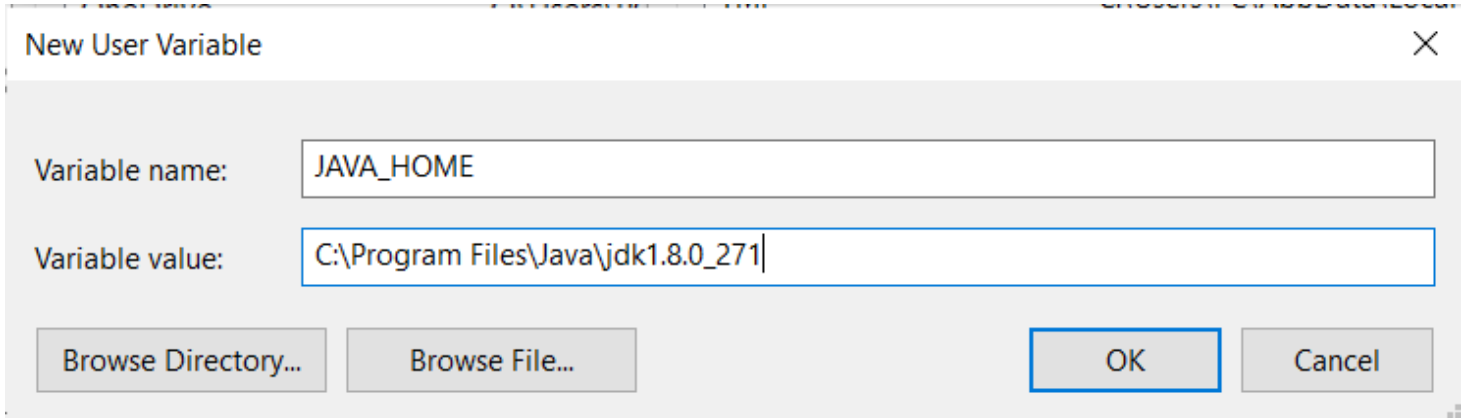


Environment Variables Setting

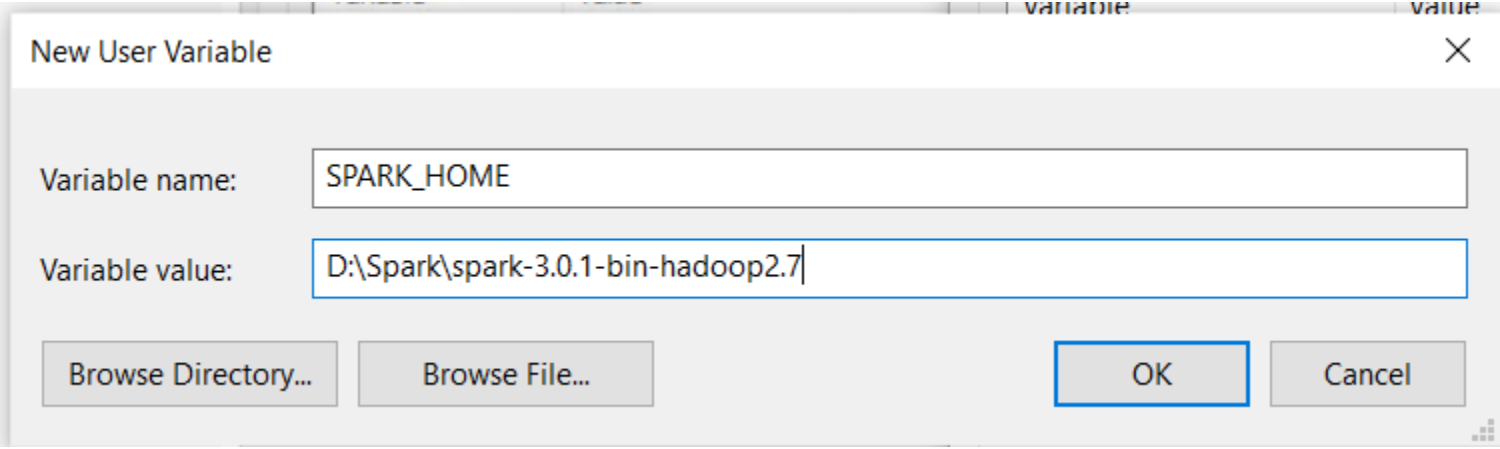
- Open System Environment Variables window and select Environment Variables.



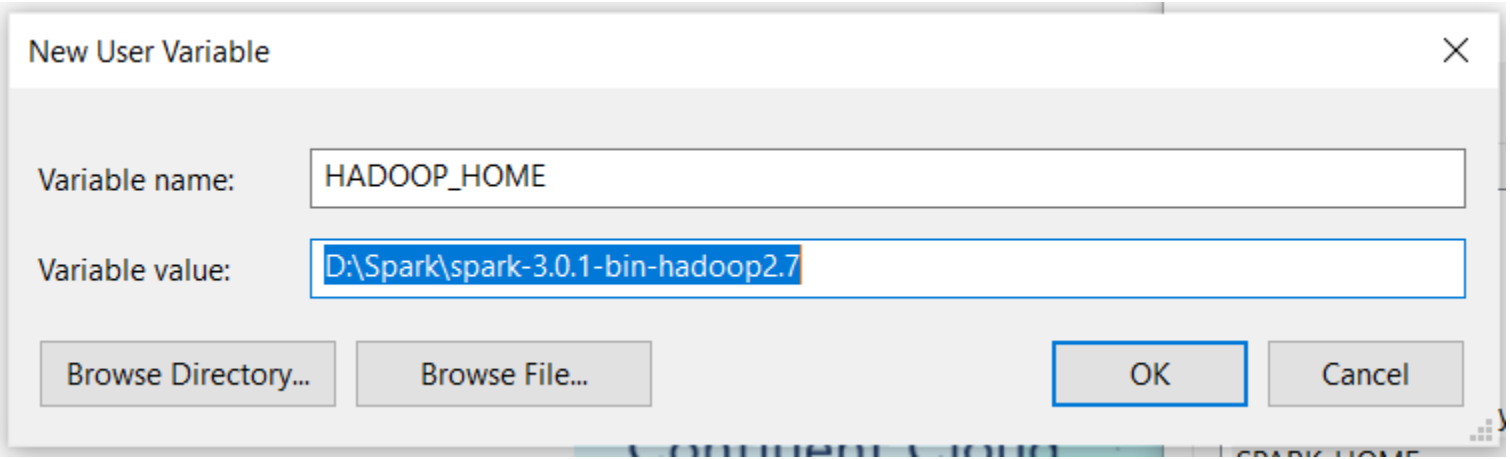
Environment Variables Setting



Apache Spark Installation on Windows

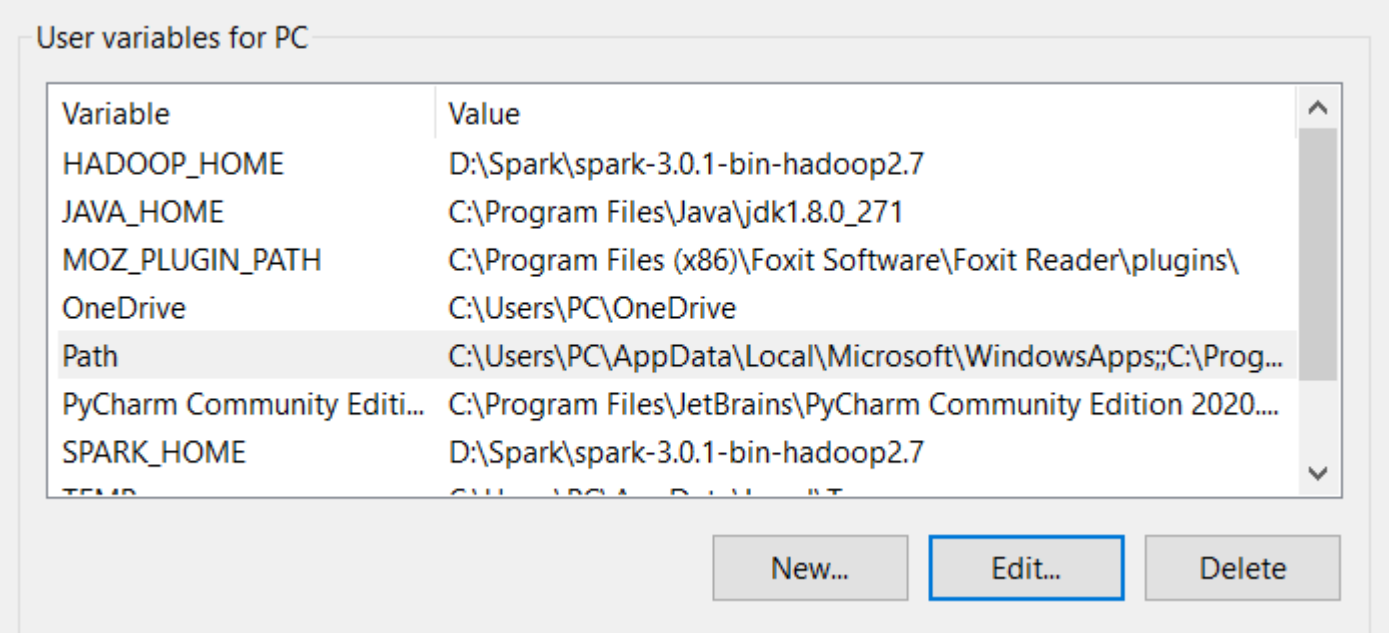


Apache Spark Installation on Windows



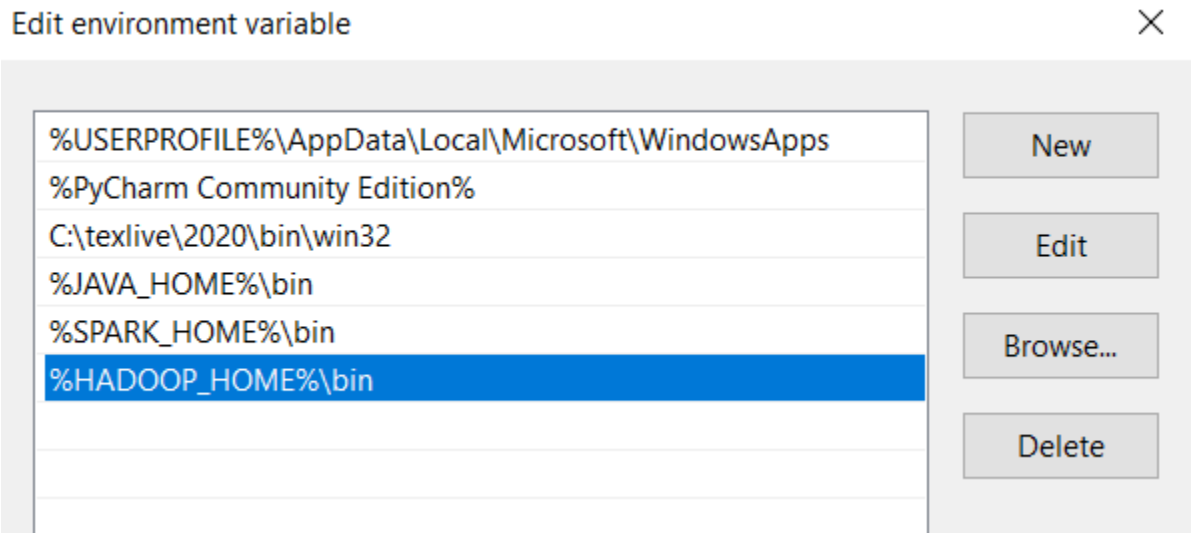
Apache Spark Installation on Windows

- Now Edit the PATH variable



Apache Spark Installation on Windows

- Add Spark, Java, and Hadoop bin location by selecting New option.



Test apache Spark shell

```
C:\Users\PC>spark-shell
20/12/07 16:50:25 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
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).
Spark context Web UI available at http://192.168.56.1:4040
Spark context available as 'sc' (master = local[*], app id = local-1607334630777).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_271)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 20/12/07 16:50:45 WARN ProcfsMetricsGetter: Exception when trying to compute pagesize, as a result reporting of P
rocessTree metrics is stopped


scala>
```

Run PySpark on PySpark Shell

Type **pyspark** on command prompt

```
C:\Users\PC>pyspark
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
21/05/09 15:58:47 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
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).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Python version 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021 17:27:52)
SparkSession available as 'spark'.
>>> 21/05/09 15:59:03 WARN ProcfsMetricsGetter: Exception when trying to compute pagesize, as a result reporting of Proc
essTree metrics is stopped

>>>
```

Run PySpark on Jupyter lab

- Open Anaconda prompt and type **“python -m pip install findspark”.** This package is necessary to run spark from Jupyter notebook.

```
(base) D:\Spark\spark-3.0.1-bin-hadoop2.7>python -m pip install findspark
Collecting findspark
  Downloading findspark-1.4.2-py2.py3-none-any.whl (4.2 kB)
Installing collected packages: findspark
Successfully installed findspark-1.4.2
```

Run PySpark on Jupyter lab

- Open jupyter notebook
- New -> Python 3

In [1]:
```
import findspark

findspark.init()
```

In [2]:
```
import pyspark

from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

df = spark.sql("select 'spark' as hello ")

df.show()
```

```
+-----+
|hello|
+-----+
|spark|
+-----+
```

Big Data Analytics with PySpark SQL

What is PySpark

PySpark is a Spark library written in Python to run Python application using Apache Spark capabilities, using PySpark we can run applications parallelly on the distributed cluster (multiple nodes).

In other words, PySpark is a Python API for Apache Spark. Apache Spark is an analytical processing engine for large scale powerful distributed data processing and machine learning applications.

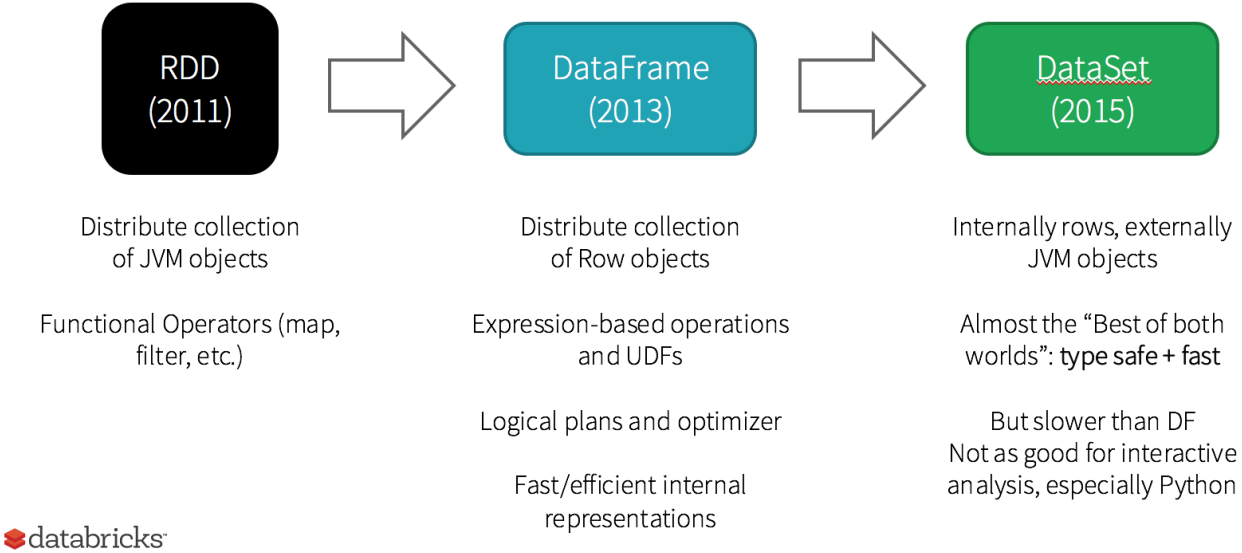
source: https://databricks.com/

PySpark Modules and Packages

- PySpark RDD (pyspark.RDD)
- PySpark DataFrame and SQL (pyspark.sql)
- PySpark Streaming (pyspark.streaming)
- PySpark MLib (pyspark.ml, pyspark.mllib)
- PySpark GraphFrames (GraphFrames)
- PySpark Resource (pyspark.resource) It's new in PySpark 3.0

RDD vs DataFrame vs DataSet

History of Spark APIs



In version 2.0, DataSet and DataFrame APIs are unified to provide a single API for developers. A DataFrame is a specific Dataset[T], where T=Row type, so DataFrame shares the same methods as Dataset.

RDD vs DataFrame vs DataSet

Feature	RDD	DataFrame	DataSet
Immutable	Yes	Yes	Yes
Fault tolerant	Yes	Yes	Yes
Type-safe	Yes	No	Yes
Schema	No	Yes	Yes
Execution optimization	No	Yes	Yes
Level	Low	High	High

What is SparkSession?

- Since Spark 2.0 SparkSession has become an entry point to PySpark to work with RDD, DataFrame. Prior to 2.0, SparkContext used to be an entry point.

- Spark Session also includes all the APIs available in different contexts –
  - Spark Context,
  - SQL Context,
  - Streaming Context,
  - Hive Context.

SparkSession in PySpark shell

- Be default PySpark shell provides “**spark**” object; which is an instance of SparkSession class. We can directly use this object where required in spark-shell.

```
>>> spark.version
'3.0.1'
>>> spark.createDataFrame([("Java","20000"),("Python","10000"),("Scala","5000")]).show()
+------+-----+
|    _1|   _2|
+------+-----+
|  Java|20000|
|Python|10000|
| Scala| 5000|
+------+-----+
```

Create SparkSession in Jupyter lab

[1]:
```
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("VeryFirstSparkExample").getOrCreate()
```

[2]:
```
spark.version
```

[2]:
```
'3.0.1'
```

[3]:
```
spark.createDataFrame([("Java","20000"),("Python","10000"),("Scala","5000")]).show()
```

```
+------+-----+
|    _1|   _2|
+------+-----+
|  Java|20000|
|Python|10000|
| Scala| 5000|
+------+-----+
```

SparkSession Commonly Used Methods

version – Returns Spark version where your application is running, probably the Spark version you cluster is configured with.

createDataFrame() – This creates a DataFrame from a collection and an RDD

getActiveSession() – returns an active Spark session.

read() – Returns an instance of DataFrameReader class, this is used to read records from csv, parquet, avro and more file formats into DataFrame.

readStream() – Returns an instance of DataStreamReader class, this is used to read streaming data. that can be used to read streaming data into DataFrame.

sparkContext() – Returns a SparkContext.

sql – Returns a DataFrame after executing the SQL mentioned.

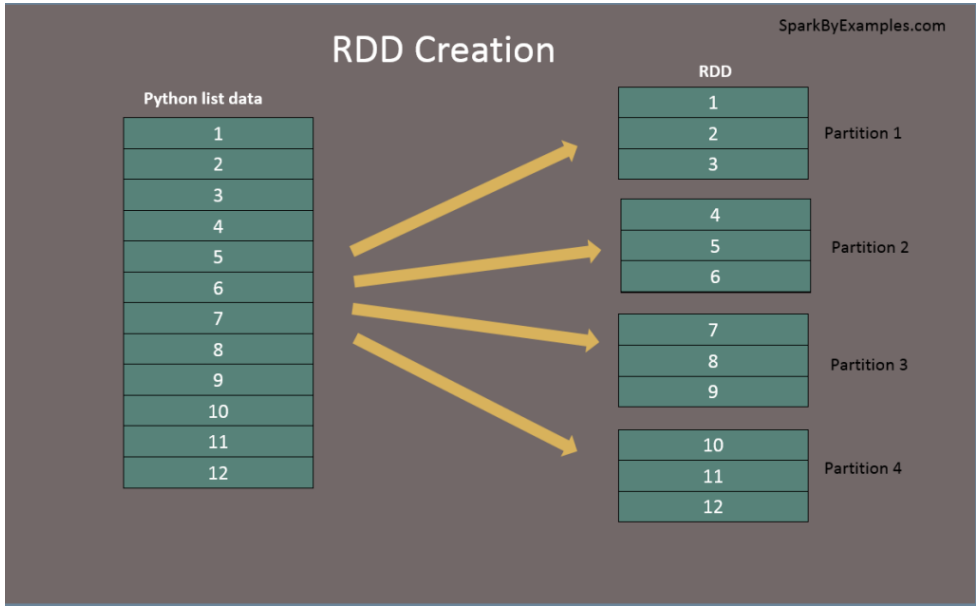
sqlContext() – Returns SQLContext.

stop() – Stop the current SparkContext.

table() – Returns a DataFrame of a table or view.

udf() – Creates a PySpark UDF to use it on DataFrame, Dataset, and SQL.

Create RDD using sparkContext.parallelize()



By using parallelize() function of SparkContext (sparkContext.parallelize()) you can create an RDD. This function loads the existing collection from your driver program into parallelizing RDD. This is a basic method to create RDD and used when you already have data in memory that either loaded from a file or from a database. and it required all data to be present on the driver program prior to creating RDD.

Create RDD using sparkContext.parallelize()

```
[4]: #Create RDD from parallelize
     data = [1,2,3,4,5,6,7,8,9,10,11,12]
     rdd=spark.sparkContext.parallelize(data)
```

```
[6]: rdd.collect()
```

```
[6]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Create RDD using sparkContext.textFile()

```
[7]: #Create RDD from external Data source
     rdd2 = spark.sparkContext.textFile("/path/textFile.txt")
```

```
[8]: #Reads entire file into a RDD as single record.
     rdd3 = spark.sparkContext.wholeTextFiles("/path/textFile.txt")
```

PySpark RDD Operations

- **RDD transformations** – Transformations are lazy operations, instead of updating an RDD, these operations return another RDD.
- **RDD actions** – operations that trigger computation and return non-RDD values.
- Transformations on PySpark RDD returns another RDD and transformations are lazy meaning they don't execute until you call an action on RDD. Some transformations on RDD's are flatMap(), map(), reduceByKey(), filter(), sortByKey() and return new RDD instead of updating the current.

RDD transformation: flatMap

flatMap – flatMap() transformation flattens the RDD after applying the function and returns a new RDD. On the below example, first, it splits each record by space in an RDD and finally flattens it. Resulting RDD consists of a single word on each record.

```
[9]: #Create RDD from external Data source
     transRDD = spark.sparkContext.textFile("trans.txt")

[10]: transRDD.collect()

[10]: ['00000000,06-26-2011,4000001,040.33,Exercise & Fitness,Cardio Machine Accessories,Clarksville,Tennessee,credit',
       '00000001,05-26-2011,4000002,198.44,Exercise & Fitness,Weightlifting Gloves,Long Beach,California,credit',

[11]: transRDD.flatMap(lambda x: x.split(",")).collect()

[11]: ['00000000',
       '06-26-2011',
       '4000001',
       '040.33',
       'Exercise & Fitness',
       'Cardio Machine Accessories',
       'Clarksville',
```

RDD transformation: map

map – map() transformation is used the apply any complex operations like adding a column, updating a column e.t.c, the output of map transformations would always have the same number of records as input.

```
[11]: transRDD.flatMap(lambda x: x.split(",")).collect()
```

```
[11]: ['00000000',
       '06-26-2011',
       '4000001',
       '040.33',
       'Exercise & Fitness',
       'Cardio Machine Accessories',
       'Clarksville',
       'Tennessee',
       'credit',
       '00000001',
       '05-26-2011',
```

```
[14]: transRDD.map(lambda x: x.split(",")).collect()
```

```
[14]: [['00000000',
        '06-26-2011',
        '4000001',
        '040.33',
        'Exercise & Fitness',
        'Cardio Machine Accessories',
        'Clarksville',
        'Tennessee',
        'credit'],
       ['00000001',
        '05-26-2011',
```

```
[16]: transRDD.flatMap(lambda x: x.split(",")).count()
```

```
[16]: 540
```

```
[15]: transRDD.map(lambda x: x.split(",")).count()
```

```
[15]: 60
```

RDD transformation: map

[22]:
```
#Show customer ID and amount of each transaction
transRDD.map(lambda x: x.split(",")).map(lambda x: (x[2],x[3])).collect()
```

[22]:
```
[('4000001', '040.33'),
 ('4000002', '198.44'),
 ('4000002', '005.58'),
 ('4000003', '198.19'),
 ('4000002', '098.81'),
 ('4000004', '193.63'),
 ('4000005', '027.89'),
 ('4000006', '096.01'),
 ('4000006', '010.44'),
 ('4000006', '152.46'),
 ('4000007', '180.28'),
 ('4000009', '121.39'),
```

RDD transformation: reduceByKey

reduceByKey – reduceByKey() merges the values for each key with the function specified. In our example, it reduces the word string by applying the sum function on value. The result of our RDD contains unique words and their count.

```
[27]: #Show ID and total cost of all transctions of each customer
transRDD.map(lambda x: x.split(",")).map(lambda x: (x[2],x[3])).reduceByKey(lambda amount1,amount2: float(amount1)+float(amount2) ).collect()
```

```
[27]: [('4000004', 337.06),
 ('4000007', 699.55),
 ('4000008', 859.42),
 ('4000001', 651.0500000000001),
 ('4000002', 706.97),
 ('4000003', 527.5899999999999),
 ('4000005', 325.15),
 ('4000006', 539.3800000000001),
 ('4000009', 457.83),
 ('4000010', 447.09000000000003)]
```

RDD transformation: sortByKey

```
[32]: #Show ID and total cost of all transctions of each customer, sorted by customer ID
transRDD.map(lambda x: x.split(",")).map(lambda x: (x[2],x[3])).reduceByKey(lambda amount1,amount2: float(amount1)+float(amount2) ).sortByKey().collect()
```

```
[32]: [('4000001', 651.0500000000001),
 ('4000002', 706.97),
 ('4000003', 527.5899999999999),
 ('4000004', 337.06),
 ('4000005', 325.15),
 ('4000006', 539.3800000000001),
 ('4000007', 699.55),
 ('4000008', 859.42),
 ('4000009', 457.83),
 ('4000010', 447.09000000000003)]
```

RDD transformation: filter

```
[40]: #Show customer IDs and games of all transaction where games include 'Sport'
transRDD.map(lambda x: x.split(",")).map(lambda x: (x[2],x[4])).filter(lambda x: 'Sport' in x[1]).collect()
```

```
[40]: [('4000002', 'Team Sports'),
       ('4000006', 'Winter Sports'),
       ('4000010', 'Team Sports'),
       ('4000001', 'Combat Sports'),
       ('4000008', 'Water Sports'),
       ('4000008', 'Team Sports'),
       ('4000008', 'Water Sports'),
       ('4000005', 'Air Sports'),
       ('4000009', 'Water Sports'),
       ('4000003', 'Water Sports'),
       ('4000009', 'Combat Sports'),
       ('4000008', 'Team Sports'),
       ('4000001', 'Water Sports'),
       ('4000008', 'Team Sports'),
       ('4000008', 'Team Sports'),
       ('4000007', 'Team Sports'),
       ('4000005', 'Team Sports'),
       ('4000004', 'Water Sports'),
```

RDD functions

https://spark.apache.org/docs/latest/api/python/reference/pyspark.html

Exercises

- Show the ID and all game types played by customers who play “Water Sports”.

Hint: use **reduceByKey()** to concatenate the game types of each customer IDs and then apply **filter()**. To remove duplicate game types for each ID, use **distinct()** function

```
[('4000008', 'Water Sports;Team Sports;Games;Outdoor Play Equipment;Outdoor Recreation'),
 ('4000004', 'Indoor Games;Water Sports;Outdoor Recreation'),
 ('4000003', 'Gymnastics;Outdoor Recreation;Water Sports'),
 ('4000006', 'Jumping;Outdoor Play Equipment;Winter Sports;Water Sports'),
 ('4000001', 'Combat Sports;Outdoor Recreation;Gymnastics;Exercise & Fitness;Water Sports;Winter Sports'),
 ('4000009', 'Gymnastics;Combat Sports;Outdoor Play Equipment;Indoor Games;Water Sports'),
 ('4000002', 'Outdoor Recreation;Exercise & Fitness;Team Sports;Water Sports')]
```

- Other exercises

1. Show IDs and number of transactions of each customer
2. Show IDs and number of transactions of each customer, sorted by customer ID
3. Show IDs and total cost of transactions of each customer, sorted by total cost
4. Show ID, number of transactions, and total cost for each customer, sorted by customer ID
5. Show name, number of transactions, and total cost for each customer, sorted by totall cost
6. Show ID, name, game types played by each customer
7. Show ID, name, game types of all players who play 5 or more game types
8. Show name of all distinct players of each game types
9. Show all game types which don’t have player under 40
10. Show min, max, average age of players of all game types

Create DataFrame from RDD

SPARKSESSION	RDD	DATAFRAME
createDataFrame(rdd)	toDF()	toDF(*cols)
createDataFrame(dataList)	toDF(*cols)	
createDataFrame(rowData,columns)		
createDataFrame(dataList,schema)		

Create DataFrame from RDD

Using toDF() function

[54]:
```
columns = ["language","users_count"]
data = [("Java", "20000"), ("Python", "100000"), ("Scala", "3000")]
rdd = spark.sparkContext.parallelize(data)
```

[55]:
```
dfFromRDD1 = rdd.toDF()
dfFromRDD1.printSchema()
```

```
root
 |-- _1: string (nullable = true)
 |-- _2: string (nullable = true)
```

[56]:
```
dfFromRDD1 = rdd.toDF(columns)
dfFromRDD1.printSchema()
```

```
root
 |-- language: string (nullable = true)
 |-- users_count: string (nullable = true)
```

Create DataFrame from RDD

Using createDataFrame() from SparkSession

- Calling createDataFrame() from SparkSession is another way to create PySpark DataFrame manually, it takes a list object as an argument. and chain with toDF() to specify names to the columns.

[57]:
```
dfFromRDD2 = spark.createDataFrame(rdd).toDF(*columns)
dfFromRDD2.printSchema()
```

```
root
 |-- language: string (nullable = true)
 |-- users_count: string (nullable = true)
```

Create DataFrame from List Collection

Using createDataFrame() from SparkSession

```
[67]: data = [("Java", "20000"), ("Python", "100000"), ("Scala", "3000")]
dfFromData2 = spark.createDataFrame(data).toDF(*columns)
dfFromData2.show()
```

```
+--------+-----------+
|language|users_count|
+--------+-----------+
|    Java|      20000|
|  Python|     100000|
|   Scala|       3000|
+--------+-----------+
```

Create DataFrame from List Collection

Using createDataFrame() with the Row type

createDataFrame() has another signature in PySpark which takes the collection of Row type and schema for column names as arguments. To use this first we need to convert our “data” object from the list to list of Row.

[72]:
```
from pyspark.sql import Row
rowData = map(lambda x: Row(*x), data)
dfFromData3 = spark.createDataFrame(rowData,columns)
dfFromData3.show()
```

```
+--------+-----------+
|language|users_count|
+--------+-----------+
|    Java|      20000|
|  Python|     100000|
|   Scala|       3000|
+--------+-----------+
```

Create DataFrame from List Collection

Create DataFrame with schema

If you wanted to specify the column names along with their data types, you should create the StructType schema first and then assign this while creating a DataFrame.

```
[78]: from pyspark.sql.types import StructType,StructField, StringType, IntegerType
data2 = [("James","","Smith","36636","M",3000),
         ("Maria","Anne","Jones","39192","F",4000),
         ("Jen","Mary","Brown","","F",-1)
  ]

schema = StructType([ \
    StructField("firstname",StringType(),True), \
    StructField("middlename",StringType(),True), \
    StructField("lastname",StringType(),True), \
    StructField("id", StringType(), True), \
    StructField("gender", StringType(), True), \
    StructField("salary", IntegerType(), True) \
  ])

df = spark.createDataFrame(data=data2,schema=schema)
df.printSchema()
df.show()
```

```
root
 |-- firstname: string (nullable = true)
 |-- middlename: string (nullable = true)
 |-- lastname: string (nullable = true)
 |-- id: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: integer (nullable = true)

+---------+----------+--------+-----+------+------+
|firstname|middlename|lastname|   id|gender|salary|
+---------+----------+--------+-----+------+------+
|    James|          |   Smith|36636|     M|  3000|
|    Maria|      Anne|   Jones|39192|     F|  4000|
|      Jen|      Mary|   Brown|     |     F|    -1|
+---------+----------+--------+-----+------+------+
```

Create DataFrame from Data sources

Creating DataFrame from CSV

```
[2]: df = spark.read.csv("zipcodes.csv")
     df.printSchema()
     df.show()
```

```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
 |-- _c7: string (nullable = true)
 |-- _c8: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- _c10: string (nullable = true)
 |-- _c11: string (nullable = true)
 |-- _c12: string (nullable = true)
 |-- _c13: string (nullable = true)
 |-- _c14: string (nullable = true)
 |-- _c15: string (nullable = true)
 |-- _c16: string (nullable = true)
 |-- _c17: string (nullable = true)
 |-- _c18: string (nullable = true)
 |-- _c19: string (nullable = true)
```

Create DataFrame from Data sources

Creating DataFrame from CSV

- Using fully qualified data source name, you can alternatively do the following.

```
[7]: df = spark.read.format("csv").load("zipcodes.csv")
     df.show()
```

```
+------------+-------+-----------+-------------------+-----+--------------+-----+-------+-----+-----+-----+-----------+-------+-
-------------------+--------------------+-------------+---------------+-------------------+----------+-------------+
|         _c0|    _c1|        _c2|                _c3|  _c4|           _c5|  _c6|    _c7|  _c8|  _c9| _c10|       _c11|   _c12|
_c13|                _c14|         _c15|           _c16|               _c17|      _c18|         _c19|
+------------+-------+-----------+-------------------+-----+--------------+-----+-------+-----+-----+-----+-----------+-------+-
-------------------+--------------------+-------------+---------------+-------------------+----------+-------------+
|RecordNumber|Zipcode|ZipCodeType|               City|State|  LocationType|  Lat|   Long|Xaxis|Yaxis|Zaxis|WorldRegion|Country|
LocationText|            Location|Decommisioned|TaxReturnsFiled|EstimatedPopulation|TotalWages|        Notes|
|           1|    704|   STANDARD|        PARC PARQUE|   PR|NOT ACCEPTABLE|17.96| -66.22| 0.38|-0.87|  0.3|         NA|     US|
Parc Parque, PR|NA-US-PR-PARC PARQUE|        FALSE|           null|               null|      null|         null|
|           2|    704|   STANDARD|PASEO COSTA DEL SUR|   PR|NOT ACCEPTABLE|17.96| -66.22| 0.38|-0.87|  0.3|         NA|     US|P
```

Create DataFrame from Data sources

Creating DataFrame from CSV - Using Header Record For Column Names

[11]:
```
df2 = spark.read.option("header",True).csv("zipcodes.csv")
df2.printSchema()
```

```
root
 |-- RecordNumber: string (nullable = true)
 |-- Zipcode: string (nullable = true)
 |-- ZipCodeType: string (nullable = true)
 |-- City: string (nullable = true)
 |-- State: string (nullable = true)
 |-- LocationType: string (nullable = true)
 |-- Lat: string (nullable = true)
 |-- Long: string (nullable = true)
 |-- Xaxis: string (nullable = true)
 |-- Yaxis: string (nullable = true)
 |-- Zaxis: string (nullable = true)
 |-- WorldRegion: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- LocationText: string (nullable = true)
 |-- Location: string (nullable = true)
 |-- Decommisioned: string (nullable = true)
 |-- TaxReturnsFiled: string (nullable = true)
 |-- EstimatedPopulation: string (nullable = true)
 |-- TotalWages: string (nullable = true)
 |-- Notes: string (nullable = true)
```

Create DataFrame from Data sources

Creating DataFrame from CSV - Read Multiple CSV Files

- df = spark.read.csv("path1,path2,path3")

Create DataFrame from Data sources

Creating DataFrame from CSV - Read all CSV Files in a Directory

- df = spark.read.csv("Folder path")

Create DataFrame from Data sources

Creating DataFrame from CSV - Options While Reading CSV File

- **delimiter** option is used to specify the column delimiter of the CSV file. By default, it is comma (,) character, but can be set to any character like pipe(|), tab (\t), space using this option.

```
df3 = spark.read.options(delimiter=',').csv("zipcodes.csv")
df3.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
 |-- _c7: string (nullable = true)
 |-- _c8: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- _c10: string (nullable = true)
 |-- _c11: string (nullable = true)
 |-- _c12: string (nullable = true)
 |-- _c13: string (nullable = true)
 |-- _c14: string (nullable = true)
 |-- _c15: string (nullable = true)
 |-- _c16: string (nullable = true)
 |-- _c17: string (nullable = true)
 |-- _c18: string (nullable = true)
 |-- _c19: string (nullable = true)
```

Create DataFrame from Data sources

Creating DataFrame from CSV - Options While Reading CSV File

- **inferSchema**: The default value set to this option is False when setting to true it automatically infers column types based on the data. Note that, it requires reading the data one more time to infer the schema.

```
[21]: df4 = spark.read.options(inferSchema='True',delimiter=',').csv("zipcodes.csv")
      df4.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
 |-- _c7: string (nullable = true)
 |-- _c8: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- _c10: string (nullable = true)
 |-- _c11: string (nullable = true)
 |-- _c12: string (nullable = true)
 |-- _c13: string (nullable = true)
 |-- _c14: string (nullable = true)
 |-- _c15: string (nullable = true)
 |-- _c16: string (nullable = true)
 |-- _c17: string (nullable = true)
 |-- _c18: string (nullable = true)
 |-- _c19: string (nullable = true)
```

Why're all String?

```
[28]: df3 = spark.read.options(inferSchema='True', delimiter=',').csv("zipcodesNoHeader.csv")
      df3.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- _c1: integer (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: double (nullable = true)
 |-- _c7: double (nullable = true)
 |-- _c8: double (nullable = true)
 |-- _c9: double (nullable = true)
 |-- _c10: double (nullable = true)
 |-- _c11: string (nullable = true)
 |-- _c12: string (nullable = true)
 |-- _c13: string (nullable = true)
 |-- _c14: string (nullable = true)
 |-- _c15: boolean (nullable = true)
 |-- _c16: integer (nullable = true)
 |-- _c17: integer (nullable = true)
 |-- _c18: integer (nullable = true)
 |-- _c19: string (nullable = true)
```

Create DataFrame from Data sources

Creating DataFrame from CSV - Options While Reading CSV File

- **header:** This option is used to read the first line of the CSV file as column names. By default the value of this option is False , and all column types are assumed to be a string.

```
[22]: df3 = spark.read.options(header='True', inferSchema='True', delimiter=',').csv("zipcodes.csv")
      df3.printSchema()
```

```
root
 |-- RecordNumber: integer (nullable = true)
 |-- Zipcode: integer (nullable = true)
 |-- ZipCodeType: string (nullable = true)
 |-- City: string (nullable = true)
 |-- State: string (nullable = true)
 |-- LocationType: string (nullable = true)
 |-- Lat: double (nullable = true)
 |-- Long: double (nullable = true)
 |-- Xaxis: double (nullable = true)
 |-- Yaxis: double (nullable = true)
 |-- Zaxis: double (nullable = true)
 |-- WorldRegion: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- LocationText: string (nullable = true)
 |-- Location: string (nullable = true)
 |-- Decommisioned: boolean (nullable = true)
 |-- TaxReturnsFiled: integer (nullable = true)
 |-- EstimatedPopulation: integer (nullable = true)
 |-- TotalWages: integer (nullable = true)
 |-- Notes: string (nullable = true)
```

Create DataFrame from Data sources

Creating DataFrame from CSV – user specified custom schema

- We can specify schema by using the **schema** option belonging to read.csv()

  s = spark.read.schema(**user_schema**)

- Where **user_schema** is a
  - pyspark.sql.types.StructType object

  or

  - DDL-formatted string

Create DataFrame from Data sources

Creating DataFrame from CSV - StructType custom schema

```
from pyspark.sql.types import *
schema = StructType() \
    .add("RecordNumber",IntegerType(),True) \
    .add("Zipcode",IntegerType(),True) \
    .add("ZipCodeType",StringType(),True) \
    .add("City",StringType(),True) \
    .add("State",StringType(),True) \
    .add("LocationType",StringType(),True) \
    .add("Lat",DoubleType(),True) \
    .add("Long",DoubleType(),True) \
    .add("Xaxis",IntegerType(),True) \
    .add("Yaxis",DoubleType(),True) \
    .add("Zaxis",DoubleType(),True) \
    .add("WorldRegion",StringType(),True) \
    .add("Country",StringType(),True) \
    .add("LocationText",StringType(),True) \
    .add("Location",StringType(),True) \
    .add("Decommisioned",BooleanType(),True) \
    .add("TaxReturnsFiled",StringType(),True) \
    .add("EstimatedPopulation",IntegerType(),True) \
    .add("TotalWages",IntegerType(),True) \
    .add("Notes",StringType(),True)

df_with_schema = spark.read.format("csv").option("header", True).schema(schema).load("zipcodes.csv")
df_with_schema.printSchema()
```

```
root
 |-- RecordNumber: integer (nullable = true)
 |-- Zipcode: integer (nullable = true)
 |-- ZipCodeType: string (nullable = true)
 |-- City: string (nullable = true)
 |-- State: string (nullable = true)
 |-- LocationType: string (nullable = true)
 |-- Lat: double (nullable = true)
 |-- Long: double (nullable = true)
 |-- Xaxis: integer (nullable = true)
 |-- Yaxis: double (nullable = true)
 |-- Zaxis: double (nullable = true)
 |-- WorldRegion: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- LocationText: string (nullable = true)
 |-- Location: string (nullable = true)
 |-- Decommisioned: boolean (nullable = true)
 |-- TaxReturnsFiled: string (nullable = true)
 |-- EstimatedPopulation: integer (nullable = true)
 |-- TotalWages: integer (nullable = true)
 |-- Notes: string (nullable = true)
```

Create DataFrame from Data sources

Creating DataFrame from CSV – DLL formatted string custom schema

df = spark.read.options(delimiter=',').schema('trans_id INT, date STRING, cust_ID INT, amount DOUBLE, game STRING, equipment STRING, city STRING, state STRING, mode STRING').csv("trans.txt")

df.printSchema()

df.show()

```
root
 |-- trans_id: integer (nullable = true)
 |-- date: string (nullable = true)
 |-- cust_ID: integer (nullable = true)
 |-- amount: double (nullable = true)
 |-- game: string (nullable = true)
 |-- equipment: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- mode: string (nullable = true)

+--------+----------+-------+------+--------------------+--------------------+--------------+--------------+------+
|trans_id|      date|cust_ID|amount|                game|           equipment|          city|         state|  mode|
+--------+----------+-------+------+--------------------+--------------------+--------------+--------------+------+
|       0|06-26-2011|4000001| 40.33|  Exercise & Fitness|Cardio Machine Ac...|   Clarksville|     Tennessee|credit|
|       1|05-26-2011|4000002|198.44|  Exercise & Fitness|Weightlifting Gloves|    Long Beach|    California|credit|
|       2|06-01-2011|4000002|  5.58|  Exercise & Fitness|Weightlifting Mac...|       Anaheim|    California|credit|
|       3|06-05-2011|4000003|198.19|          Gymnastics|    Gymnastics Rings|     Milwaukee|     Wisconsin|credit|
|       4|12-17-2011|4000002| 98.81|         Team Sports|        Field Hockey|   Nashville  |     Tennessee|credit|
|       5|02-14-2011|4000004|193.63|  Outdoor Recreation|Camping & Backpac...|       Chicago|      Illinois|credit|
|       6|10-28-2011|4000005| 27.89|             Puzzles|      Jigsaw Puzzles|    Charleston|South Carolina|credit|
|       7|07-14-2011|4000006| 96.01|Outdoor Play Equi...|           Sandboxes|      Columbus|          Ohio|credit|
```

Create DataFrame from Data sources

Creating DataFrame from CSV - Write PySpark DataFrame to CSV file-

- Use the write() method of the PySpark DataFrameWriter object to write PySpark DataFrame to a CSV file.

```
df.write.option("header",True).csv(“newzipcodes")
```

- While writing a CSV file you can use several options. for example, header to output the DataFrame column names as header record and delimiter to specify the delimiter on the CSV output file.

```
df2.write.options(header='True', delimiter=',').csv(“newzipcodes")
```

Create DataFrame from Data sources

Creating DataFrame from CSV - Write PySpark DataFrame to CSV file-

Saving modes

PySpark DataFrameWriter also has a method mode() to specify saving mode.

`overwrite` – mode is used to overwrite the existing file.

`append` – To add the data to the existing file.

`ignore` – Ignores write operation when the file already exists.

`error` – This is a default option when the file already exists, it returns an error.

```
df2.write.mode('overwrite').csv("newzipcodes")
#you can also use this
df2.write.format("csv").mode('overwrite').save("newzipcodes")
```

Create DataFrame from Data sources

Creating DataFrame from text file

You can use .text()

```
[41]: df = spark.read.text("zipcodes.txt")
      df.printSchema()
      df.collect()
```

```
root
 |-- value: string (nullable = true)
```

```
[41]: [Row(value='RecordNumber\tZipcode\tZipCodeType\tCity\tState\tLocationType'),
       Row(value='1\t704\tSTANDARD\tPARC PARQUE\tPR\tNOT ACCEPTABLE'),
       Row(value='2\t704\tSTANDARD\tPASEO COSTA DEL SUR\tPR\tNOT ACCEPTABLE'),
       Row(value='10\t709\tSTANDARD\tBDA SAN LUIS\tPR\tNOT ACCEPTABLE'),
       Row(value='61391\t76166\tUNIQUE\tCINGULAR WIRELESS\tTX\tNOT ACCEPTABLE'),
       Row(value='61392\t76177\tSTANDARD\tFORT WORTH\tTX\tPRIMARY'),
       Row(value='61393\t76177\tSTANDARD\tFT WORTH\tTX\tACCEPTABLE'),
       Row(value='4\t704\tSTANDARD\tURB EUGENE RICE\tPR\tNOT ACCEPTABLE'),
       Row(value='39827\t85209\tSTANDARD\tMESA\tAZ\tPRIMARY'),
       Row(value='39828\t85210\tSTANDARD\tMESA\tAZ\tPRIMARY'),
       Row(value='49345\t32046\tSTANDARD\tHILLIARD\tFL\tPRIMARY'),
       Row(value='49346\t34445\tPO BOX\tHOLDER\tFL\tPRIMARY'),
       Row(value='49347\t32564\tSTANDARD\tHOLT\tFL\tPRIMARY'),
       Row(value='49348\t34487\tPO BOX\tHOMOSASSA\tFL\tPRIMARY'),
       Row(value='10\t708\tSTANDARD\tBDA SAN LUIS\tPR\tNOT ACCEPTABLE'),
       Row(value='3\t704\tSTANDARD\tSECT LANAUSSE\tPR\tNOT ACCEPTABLE'),
       Row(value='54354\t36275\tPO BOX\tSPRING GARDEN\tAL\tPRIMARY'),
       Row(value='54355\t35146\tSTANDARD\tSPRINGVILLE\tAL\tPRIMARY'),
       Row(value='54356\t35585\tSTANDARD\tSPRUCE PINE\tAL\tPRIMARY'),
       Row(value='76511\t27007\tSTANDARD\tASH HILL\tNC\tNOT ACCEPTABLE'),
       Row(value='76512\t27203\tSTANDARD\tASHEBORO\tNC\tPRIMARY'),
       Row(value='76513\t27204\tPO BOX\tASHEBORO\tNC\tPRIMARY')]
```

But .csv() is still much better

```
[48]: df = spark.read.options(header='True',inferSchema='True', delimiter='\t').csv("zipcodes.txt")
      df.printSchema()
```

```
root
 |-- RecordNumber: integer (nullable = true)
 |-- Zipcode: integer (nullable = true)
 |-- ZipCodeType: string (nullable = true)
 |-- City: string (nullable = true)
 |-- State: string (nullable = true)
 |-- LocationType: string (nullable = true)
```

PySpark dataframe function

Select Columns From DataFrame

transDF = spark.read.options(delimiter=',').schema('trans_id INT, date STRING, cust_id INT, amount DOUBLE, game STRING, equipment STRING, city STRING, state STRING, mode STRING').csv("trans.txt")

```
transDF.printSchema()
transDF.show()
#you have several way to select columns
transDF.select('cust_id','amount').show()
transDF.select(transDF.cust_id,transDF.amount).show()
transDF.select(transDF['cust_id'],transDF['amount']).show()
#select from a list
twocolumns = ['cust_id','amount']
transDF.select(twocolumns).show()
#select all column
transDF.select([col for col in transDF.columns]).show()
transDF.select('*').show()
```

PySpark dataframe function

PySpark withColumn()

- PySpark **withColumn()** is a transformation function of DataFrame which is used to change the value, convert the datatype of an existing column, create a new column, and many more.

- You can use withColumn() to
  - **Change DataType using PySpark withColumn()**
  - **Update The Value of an Existing Column**
  - **Create a Column from an Existing**
  - **Add a New Column using withColumn()**
  - **Rename Column Name**

PySpark dataframe function

withColumn() - Change DataType

```
[13]: from pyspark.sql.functions import col
      transDF.withColumn('trans_id',col('trans_id').cast('String')).printSchema()
```

```
root
 |-- trans_id: string (nullable = true)
 |-- date: string (nullable = true)
 |-- cust_id: integer (nullable = true)
 |-- amount: double (nullable = true)
 |-- game: string (nullable = true)
 |-- equipment: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- mode: string (nullable = true)
```

`pyspark.sql.functions.col(col)`

Returns a `Column` based on the given column name.'

`Column.cast(dataType)`

Convert the column into type `dataType`.

PySpark dataframe function

withColumn() - Update The Value of an Existing Column

```
[15]: from pyspark.sql.functions import col
transDF.withColumn('amount',col('amount')*2).show()
```

trans_id	date	cust_id	amount	game	equipment	city	state	mode
0	06-26-2011	4000001	80.66	Exercise & Fitness	Cardio Machine Ac...	Clarksville	Tennessee	credit
1	05-26-2011	4000002	396.88	Exercise & Fitness	Weightlifting Gloves	Long Beach	California	credit
2	06-01-2011	4000002	11.16	Exercise & Fitness	Weightlifting Mac...	Anaheim	California	credit
3	06-05-2011	4000003	396.38	Gymnastics	Gymnastics Rings	Milwaukee	Wisconsin	credit
4	12-17-2011	4000002	197.62	Team Sports	Field Hockey	Nashville	Tennessee	credit
5	02-14-2011	4000004	387.26	Outdoor Recreation	Camping & Backpac...	Chicago	Illinois	credit
6	10-28-2011	4000005	55.78	Puzzles	Jigsaw Puzzles	Charleston	South Carolina	credit
7	07-14-2011	4000006	192.02	Outdoor Play Equi...	Sandboxes	Columbus	Ohio	credit
8	01-17-2011	4000006	20.88	Winter Sports	Snowmobiling	Des Moines	Iowa	credit
9	05-17-2011	4000006	304.92	Jumping	Bungee Jumping	St. Petersburg	Florida	credit
10	05-29-2011	4000007	360.56	Outdoor Recreation	Archery	Reno	Nevada	credit
11	06-18-2011	4000009	242.78	Outdoor Play Equi...	Swing Sets	Columbus	Ohio	credit
12	02-08-2011	4000009	83.04	Indoor Games	Bowling	San Francisco	California	credit
13	03-13-2011	4000010	215.6	Team Sports	Field Hockey	Honolulu	Hawaii	credit
14	02-25-2011	4000010	73.62	Gymnastics	Vaulting Horses	Los Angeles	California	credit
15	10-20-2011	4000001	275.28	Combat Sports	Fencing	Honolulu	Hawaii	credit
16	05-28-2011	4000010	71.12	Exercise & Fitness	Free Weight Bars	Columbia	South Carolina	credit
17	10-18-2011	4000008	151.1	Water Sports	Scuba Diving & Sn...	Omaha	Nebraska	credit
18	11-18-2011	4000008	177.3	Team Sports	Baseball	Salt Lake City	Utah	credit
19	08-28-2011	4000008	103.62	Water Sports	Life Jackets	Newark	New Jersey	credit

PySpark dataframe function

withColumn() - Create a Column from an Existing

```
[10]: from pyspark.sql.functions import col
transDF.withColumn("new amount",col("amount")*2).show()
```

```
+--------+----------+-------+------+--------------------+--------------------+--------------+--------------+------+----------+
|trans_id|      date|cust_id|amount|                game|           equipment|          city|         state|  mode|new amount|
+--------+----------+-------+------+--------------------+--------------------+--------------+--------------+------+----------+
|       0|06-26-2011|4000001| 40.33|  Exercise & Fitness|Cardio Machine Ac...|   Clarksville|     Tennessee|credit|     80.66|
|       1|05-26-2011|4000002|198.44|  Exercise & Fitness|Weightlifting Gloves|    Long Beach|    California|credit|    396.88|
|       2|06-01-2011|4000002|  5.58|  Exercise & Fitness|Weightlifting Mac...|       Anaheim|    California|credit|     11.16|
|       3|06-05-2011|4000003|198.19|          Gymnastics|    Gymnastics Rings|     Milwaukee|     Wisconsin|credit|    396.38|
|       4|12-17-2011|4000002| 98.81|         Team Sports|        Field Hockey|   Nashville  |     Tennessee|credit|    197.62|
|       5|02-14-2011|4000004|193.63|  Outdoor Recreation|Camping & Backpac...|       Chicago|      Illinois|credit|    387.26|
|       6|10-28-2011|4000005| 27.89|             Puzzles|      Jigsaw Puzzles|    Charleston|South Carolina|credit|     55.78|
|       7|07-14-2011|4000006| 96.01|Outdoor Play Equi...|           Sandboxes|      Columbus|          Ohio|credit|    192.02|
|       8|01-17-2011|4000006| 10.44|       Winter Sports|        Snowmobiling|    Des Moines|          Iowa|credit|     20.88|
|       9|05-17-2011|4000006|152.46|             Jumping|      Bungee Jumping|St. Petersburg|       Florida|credit|    304.92|
|      10|05-29-2011|4000007|180.28|  Outdoor Recreation|             Archery|          Reno|        Nevada|credit|    360.56|
|      11|06-18-2011|4000009|121.39|Outdoor Play Equi...|          Swing Sets|      Columbus|          Ohio|credit|    242.78|
|      12|02-08-2011|4000009| 41.52|        Indoor Games|             Bowling| San Francisco|    California|credit|     83.04|
|      13|03-13-2011|4000010| 107.8|         Team Sports|        Field Hockey|    Honolulu  |        Hawaii|credit|     215.6|
|      14|02-25-2011|4000010| 36.81|          Gymnastics|     Vaulting Horses|   Los Angeles|    California|credit|     73.62|
|      15|10-20-2011|4000001|137.64|       Combat Sports|             Fencing|    Honolulu  |        Hawaii|credit|    275.28|
|      16|05-28-2011|4000010| 35.56|  Exercise & Fitness|    Free Weight Bars|      Columbia|South Carolina|credit|     71.12|
|      17|10-18-2011|4000008| 75.55|        Water Sports|Scuba Diving & Sn...|         Omaha|      Nebraska|credit|     151.1|
|      18|11-18-2011|4000008| 88.65|         Team Sports|            Baseball|Salt Lake City|          Utah|credit|     177.3|
|      19|08-28-2011|4000008| 51.81|        Water Sports|        Life Jackets|        Newark|    New Jersey|credit|    103.62|
+--------+----------+-------+------+--------------------+--------------------+--------------+--------------+------+----------+
```

PySpark dataframe function

withColumn() - Add a New Column

`pyspark.sql.functions.lit(col)`

Creates a `Column` of literal value.

```
[19]: from pyspark.sql.functions import lit
      transDF.withColumn("Country", lit("USA")).show()
```

trans_id	date	cust_id	amount	game	equipment	city	state	mode	Country
0	06-26-2011	4000001	40.33	Exercise & Fitness	Cardio Machine Ac...	Clarksville	Tennessee	credit	USA
1	05-26-2011	4000002	198.44	Exercise & Fitness	Weightlifting Gloves	Long Beach	California	credit	USA
2	06-01-2011	4000002	5.58	Exercise & Fitness	Weightlifting Mac...	Anaheim	California	credit	USA
3	06-05-2011	4000003	198.19	Gymnastics	Gymnastics Rings	Milwaukee	Wisconsin	credit	USA
4	12-17-2011	4000002	98.81	Team Sports	Field Hockey	Nashville	Tennessee	credit	USA
5	02-14-2011	4000004	193.63	Outdoor Recreation	Camping & Backpac...	Chicago	Illinois	credit	USA
6	10-28-2011	4000005	27.89	Puzzles	Jigsaw Puzzles	Charleston	South Carolina	credit	USA
7	07-14-2011	4000006	96.01	Outdoor Play Equi...	Sandboxes	Columbus	Ohio	credit	USA
8	01-17-2011	4000006	10.44	Winter Sports	Snowmobiling	Des Moines	Iowa	credit	USA
9	05-17-2011	4000006	152.46	Jumping	Bungee Jumping	St. Petersburg	Florida	credit	USA
10	05-29-2011	4000007	180.28	Outdoor Recreation	Archery	Reno	Nevada	credit	USA
11	06-18-2011	4000009	121.39	Outdoor Play Equi...	Swing Sets	Columbus	Ohio	credit	USA
12	02-08-2011	4000009	41.52	Indoor Games	Bowling	San Francisco	California	credit	USA
13	03-13-2011	4000010	107.8	Team Sports	Field Hockey	Honolulu	Hawaii	credit	USA
14	02-25-2011	4000010	36.81	Gymnastics	Vaulting Horses	Los Angeles	California	credit	USA
15	10-20-2011	4000001	137.64	Combat Sports	Fencing	Honolulu	Hawaii	credit	USA
16	05-28-2011	4000010	35.56	Exercise & Fitness	Free Weight Bars	Columbia	South Carolina	credit	USA
17	10-18-2011	4000008	75.55	Water Sports	Scuba Diving & Sn...	Omaha	Nebraska	credit	USA
18	11-18-2011	4000008	88.65	Team Sports	Baseball	Salt Lake City	Utah	credit	USA
19	08-28-2011	4000008	51.81	Water Sports	Life Jackets	Newark	New Jersey	credit	USA

only showing top 20 rows

PySpark dataframe function

withColumn() - Rename Column Name

```
[21]: from pyspark.sql.functions import lit
transDF.withColumnRenamed('amount', 'cost').show()
```

```
+--------+----------+-------+------+--------------------+--------------------+--------------+--------------+------+
|trans_id|      date|cust_id|  cost|                game|           equipment|          city|         state|  mode|
+--------+----------+-------+------+--------------------+--------------------+--------------+--------------+------+
|       0|06-26-2011|4000001| 40.33|  Exercise & Fitness|Cardio Machine Ac...|   Clarksville|     Tennessee|credit|
|       1|05-26-2011|4000002|198.44|  Exercise & Fitness|Weightlifting Gloves|    Long Beach|    California|credit|
|       2|06-01-2011|4000002|  5.58|  Exercise & Fitness|Weightlifting Mac...|       Anaheim|    California|credit|
|       3|06-05-2011|4000003|198.19|          Gymnastics|    Gymnastics Rings|     Milwaukee|     Wisconsin|credit|
|       4|12-17-2011|4000002| 98.81|         Team Sports|        Field Hockey|   Nashville  |     Tennessee|credit|
|       5|02-14-2011|4000004|193.63|  Outdoor Recreation|Camping & Backpac...|       Chicago|      Illinois|credit|
|       6|10-28-2011|4000005| 27.89|             Puzzles|      Jigsaw Puzzles|    Charleston|South Carolina|credit|
```

PySpark dataframe function

Where Filter Function | Multiple Conditions

```
from pyspark.sql.types import StructType,StructField
from pyspark.sql.types import StringType, IntegerType, ArrayType
data = [
    (("James","","Smith"),["Java","Scala","C++"],"OH","M"),
    (("Anna","Rose",""),["Spark","Java","C++"],"NY","F"),
    (("Julia","","Williams"),["CSharp","VB"],"OH","F"),
    (("Maria","Anne","Jones"),["CSharp","VB"],"NY","M"),
    (("Jen","Mary","Brown"),["CSharp","VB"],"NY","M"),
    (("Mike","Mary","Williams"),["Python","VB"],"OH","M")
 ]

schema = StructType([
     StructField('name', StructType([
        StructField('firstname', StringType(), True),
        StructField('middlename', StringType(), True),
         StructField('lastname', StringType(), True)
     ])),
     StructField('languages', ArrayType(StringType()), True),
     StructField('state', StringType(), True),
     StructField('gender', StringType(), True)
 ])

df = spark.createDataFrame(data = data, schema = schema)
df.printSchema()
df.show(truncate=False)
```

PySpark dataframe function

Where Filter Function | Multiple Conditions

```
# Using equals condition
df.filter(df.state == "OH").show(truncate=False)

# not equals condition
df.filter(df.state != "OH").show(truncate=False)
df.filter(~(df.state == "OH")).show(truncate=False)

from pyspark.sql.functions import col
df.filter(col("state") == "OH").show(truncate=False)
```

PySpark dataframe function

Where Filter Function | Multiple Conditions

```
[*]: #Using SQL Expression
     df.filter("gender == 'M'").show()
     #For not equal
     df.filter("gender != 'M'").show()
     df.filter("gender <> 'M'").show()
```

PySpark dataframe function

Where Filter Function | Multiple Conditions

```
[29]: #Filter multiple condition
      df.filter( (df.state  == "OH") & (df.gender  == "M") ).show(truncate=False)
```

```
+----------------------+------------------+-----+------+
|name                  |languages         |state|gender|
+----------------------+------------------+-----+------+
|[James, , Smith]      |[Java, Scala, C++]|OH   |M     |
|[Mike, Mary, Williams]|[Python, VB]      |OH   |M     |
+----------------------+------------------+-----+------+
```

PySpark dataframe function

Where Filter Function | Multiple Conditions

```
[30]: #Filter IS IN List values
li=["OH","CA","DE"]
df.filter(df.state.isin(li)).show()

# Filter NOT IS IN List values
#These show all records with NY (NY is not part of the list)
df.filter(~df.state.isin(li)).show()
df.filter(df.state.isin(li)==False).show()
```

PySpark dataframe function

Where Filter Function | Multiple Conditions

- You can also filter DataFrame rows by using startswith(), endswith() and contains() methods of Column class.

[31]:
```
# Using startswith
df.filter(df.state.startswith("N")).show()

#using endswith
df.filter(df.state.endswith("H")).show()

#contains
df.filter(df.state.contains("H")).show()
```

```
+--------------------+------------------+-----+------+
|                name|         languages|state|gender|
+--------------------+------------------+-----+------+
|      [Anna, Rose, ]|[Spark, Java, C++]|   NY|     F|
|[Maria, Anne, Jones]|      [CSharp, VB]|   NY|     M|
|  [Jen, Mary, Brown]|      [CSharp, VB]|   NY|     M|
+--------------------+------------------+-----+------+

+--------------------+------------------+-----+------+
|                name|         languages|state|gender|
+--------------------+------------------+-----+------+
|    [James, , Smith]|[Java, Scala, C++]|   OH|     M|
| [Julia, , Williams]|      [CSharp, VB]|   OH|     F|
|[Mike, Mary, Will...|      [Python, VB]|   OH|     M|
+--------------------+------------------+-----+------+
```

PySpark dataframe function

Where Filter Function | Multiple Conditions

```
[32]: data2 = [(2,"Michael Rose"),(3,"Robert Williams"),
            (4,"Rames Rose"),(5,"Rames rose")
        ]
      df2 = spark.createDataFrame(data = data2, schema = ["id","name"])

      # like - SQL LIKE pattern
      df2.filter(df2.name.like("%rose%")).show()


      # rlike - SQL RLIKE pattern (LIKE with Regex)
      #This check case insensitive
      df2.filter(df2.name.rlike("(?i)^*rose$")).show()
```

```
+---+----------+
| id|      name|
+---+----------+
|  5|Rames rose|
+---+----------+

+---+------------+
| id|        name|
+---+------------+
|  2|Michael Rose|
|  4|  Rames Rose|
|  5|  Rames rose|
+---+------------+
```

PySpark dataframe function

Where Filter Function | Multiple Conditions

Filter on an Array column

```
[33]: from pyspark.sql.functions import array_contains
      df.filter(array_contains(df.languages,"Java")).show(truncate=False)
```

```
+----------------+------------------+-----+------+
|name            |languages         |state|gender|
+----------------+------------------+-----+------+
|[James, , Smith]|[Java, Scala, C++]|OH   |M     |
|[Anna, Rose, ]  |[Spark, Java, C++]|NY   |F     |
+----------------+------------------+-----+------+
```

PySpark dataframe function

Where Filter Function | Multiple Conditions

Filtering on Nested Struct columns

[34]:
```
#Struct condition
df.filter(df.name.lastname == "Williams").show(truncate=False)
```

```
+----------------------+------------+-----+------+
|name                  |languages   |state|gender|
+----------------------+------------+-----+------+
|[Julia, , Williams]   |[CSharp, VB]|OH   |F     |
|[Mike, Mary, Williams]|[Python, VB]|OH   |M     |
+----------------------+------------+-----+------+
```

PySpark dataframe function

Where Filter Function | Multiple Conditions

How about Where()?

pyspark.sql.DataFrame.where¶

`DataFrame.where(condition)`

`where()` is an alias for `filter()`.

PySpark dataframe function

Get Distinct Rows (By Comparing All Columns)

```
[45]: transDF.select('cust_id','game').count()

[45]: 60

[46]: transDF.select('cust_id','game').distinct().count()

[46]: 43
```

PySpark dataframe function

Distinct of Selected Multiple Columns

```
[47]: dropDisDF = transDF.dropDuplicates(['cust_id','game'])
      print("Distinct count of customer ID & game : "+str(dropDisDF.count()))
      dropDisDF.show(truncate=False)
```

```
Distinct count of customer ID & game : 43
+--------+----------+-------+------+----------------------+-------------------+-------------+--------------+------+
|trans_id|date      |cust_id|amount|game                  |equipment          |city         |state         |mode  |
+--------+----------+-------+------+----------------------+-------------------+-------------+--------------+------+
|13      |03-13-2011|4000010|107.8 |Team Sports           |Field Hockey       |Honolulu     |Hawaii        |credit|
|48      |09-27-2011|4000007|157.94|Exercise & Fitness    |Exercise Bands     |Philadelphia |Pennsylvania  |credit|
|20      |06-29-2011|4000005|41.55 |Exercise & Fitness    |Weightlifting Belts|New Orleans  |Louisiana     |credit|
|33      |06-15-2011|4000008|154.15|Outdoor Recreation    |Lawn Games         |Nashville    |Tennessee     |credit|
|49      |07-12-2011|4000010|144.59|Jumping               |Jumping Stilts     |Cambridge    |Massachusetts |credit|
|46      |05-27-2011|4000001|52.29 |Gymnastics            |Vaulting Horses    |Cleveland    |Ohio          |credit|
|55      |12-16-2011|4000006|106.11|Water Sports          |Swimming           |New York     |New York      |credit|
|3       |06-05-2011|4000003|198.19|Gymnastics            |Gymnastics Rings   |Milwaukee    |Wisconsin     |credit|
|6       |10-28-2011|4000005|27.89 |Puzzles               |Jigsaw Puzzles     |Charleston   |South Carolina|credit|
|12      |02-08-2011|4000009|41.52 |Indoor Games          |Bowling            |San Francisco|California    |credit|
|10      |05-29-2011|4000007|180.28|Outdoor Recreation    |Archery            |Reno         |Nevada        |credit|
|47      |10-23-2011|4000008|100.1 |Outdoor Play Equipment|Swing Sets         |Everett      |Washington    |credit|
|24      |06-10-2011|4000003|151.2 |Water Sports          |Surfing            |Plano        |Texas         |credit|
|52      |02-04-2011|4000005|44.82 |Outdoor Play Equipment|Lawn Water Slides  |Hampton      |Virginia      |cash  |
|15      |10-20-2011|4000001|137.64|Combat Sports         |Fencing            |Honolulu     |Hawaii        |credit|
|43      |04-22-2011|4000004|32.34 |Water Sports          |Water Polo         |Las Vegas    |Nevada        |cash  |
|16      |05-28-2011|4000010|35.56 |Exercise & Fitness    |Free Weight Bars   |Columbia     |South Carolina|credit|
|14      |02-25-2011|4000010|36.81 |Gymnastics            |Vaulting Horses    |Los Angeles  |California    |credit|
|22      |10-10-2011|4000009|19.64 |Water Sports          |Kitesurfing        |Saint Paul   |Minnesota     |credit|
|7       |07-14-2011|4000006|96.01 |Outdoor Play Equipment|Sandboxes          |Columbus     |Ohio          |credit|
+--------+----------+-------+------+----------------------+-------------------+-------------+--------------+------+
```

PySpark dataframe function

https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql.html

PySpark SLQ tutorial 1

[3]:
```
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql import functions as f

spark = SparkSession.builder.appName("PySparkTutorial").getOrCreate()
```

PySpark SLQ tutorial 1

To be able to perform comparison on the timestamp, we need to convert its data type from string to timestamp type. The third transformation will modify the values in the timestamp column using the values from this very column. We use the function to_timestamp to convert a string to timestamp data type.

[5]:
```
df = (
    spark.read.csv(
        path="ratings_small.csv",
        sep=",",
        header=True,
        quote='"',
        schema="userId INT, movieId INT, rating DOUBLE, timestamp INT",
    )
    #.withColumnRenamed("timestamp", "timestamp_unix")
    #.withColumn("timestamp", f.from_unixtime("timestamp_unix"))
    #.withColumn("timestamp", f.to_timestamp("timestamp"))
)

df.show(5)
df.printSchema()
```

```
+------+-------+------+--------------+-------------------+
|userId|movieId|rating|timestamp_unix|          timestamp|
+------+-------+------+--------------+-------------------+
|     1|      1|   4.0|     964982703|2000-07-31 01:45:03|
|     1|      3|   4.0|     964981247|2000-07-31 01:20:47|
|     1|      6|   4.0|     964982224|2000-07-31 01:37:04|
|     1|     47|   5.0|     964983815|2000-07-31 02:03:35|
|     1|     50|   5.0|     964982931|2000-07-31 01:48:51|
+------+-------+------+--------------+-------------------+
```

PySpark SLQ tutorial 1

You can create the new column "timestamp" using the column "timestamp_unix" and covert it to timestamp type in a single command.

[7]:
```
df = (
    spark.read.csv(
        path="D:/UIT/HK2 2020-2021/Big data analysis/Data/ml-latest-small/ratings.csv",
        sep=",",
        header=True,
        quote='"',
        schema="userId INT, movieId INT, rating DOUBLE, timestamp INT",
    )
    .withColumnRenamed("timestamp", "timestamp_unix")
    .withColumn("timestamp", f.to_timestamp(f.from_unixtime("timestamp_unix")))
)
```

[6]:
```
df.show(5)
df.printSchema()
```

```
+------+-------+------+--------------+-------------------+
|userId|movieId|rating|timestamp_unix|          timestamp|
+------+-------+------+--------------+-------------------+
|     1|      1|   4.0|     964982703|2000-07-31 01:45:03|
|     1|      3|   4.0|     964981247|2000-07-31 01:20:47|
|     1|      6|   4.0|     964982224|2000-07-31 01:37:04|
|     1|     47|   5.0|     964983815|2000-07-31 02:03:35|
|     1|     50|   5.0|     964982931|2000-07-31 01:48:51|
+------+-------+------+--------------+-------------------+
```

PySpark SLQ tutorial 1

```python
#count the number of review of each user, sorted by userId
df.groupBy("userId").count().sort("userId").show()
```

```
+------+-----+
|userId|count|
+------+-----+
|     1|  232|
|     2|   29|
|     3|   39|
|     4|  216|
|     5|   44|
|     6|  314|
|     7|  152|
|     8|   47|
|     9|   46|
|    10|  140|
|    11|   64|
|    12|   32|
|    13|   31|
|    14|   48|
|    15|  135|
|    16|   98|
|    17|  105|
|    18|  502|
|    19|  703|
|    20|  242|
+------+-----+
only showing top 20 rows
```

PySpark SLQ tutorial 1

```
#show min rating of each user
df.groupBy("userId").agg(f.min("rating").alias("min_rating")).sort("userId").show()
```

```
+------+----------+
|userId|min_rating|
+------+----------+
|     1|       1.0|
|     2|       2.0|
|     3|       0.5|
|     4|       1.0|
|     5|       1.0|
|     6|       1.0|
|     7|       0.5|
|     8|       1.0|
|     9|       1.0|
|    10|       0.5|
|    11|       1.0|
|    12|       3.0|
|    13|       1.0|
|    14|       1.0|
|    15|       1.0|
|    16|       1.5|
|    17|       3.0|
|    18|       0.5|
|    19|       1.0|
|    20|       0.5|
+------+----------+
only showing top 20 rows
```

PySpark SLQ tutorial 1

[31]:
```
#lets try to drop a column
#it's ok to add some collumns which don't exist

df.drop("timestamp_unix", "foobar").show(5)
```

```
+------+-------+------+-------------------+
|userId|movieId|rating|          timestamp|
+------+-------+------+-------------------+
|     1|      1|   4.0|2000-07-30 18:45:03|
|     1|      3|   4.0|2000-07-30 18:20:47|
|     1|      6|   4.0|2000-07-30 18:37:04|
|     1|     47|   5.0|2000-07-30 19:03:35|
|     1|     50|   5.0|2000-07-30 18:48:51|
+------+-------+------+-------------------+
only showing top 5 rows
```

PySpark SLQ tutorial 2

```python
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql import functions as f

spark = SparkSession.builder.appName("PySparkTutorial").getOrCreate()
```

```python
movies = (
    spark.read.csv(
        path="movies_small.csv",
        sep=",",
        header=True,
        quote='"',
        schema="movieId INT, title STRING, genres STRING",
    )
)
movies.show(5, truncate=False)
movies.printSchema()
```

PySpark SLQ tutorial 2

```python
movies.show(5, truncate=False)
movies.printSchema()
```

```
+-------+---------------------------------+-----------------------------------------+
|movieId|title                            |genres                                   |
+-------+---------------------------------+-----------------------------------------+
|1      |Toy Story (1995)                 |Adventure|Animation|Children|Comedy|Fantasy|
|2      |Jumanji (1995)                   |Adventure|Children|Fantasy                |
|3      |Grumpier Old Men (1995)          |Comedy|Romance                           |
|4      |Waiting to Exhale (1995)         |Comedy|Drama|Romance                     |
|5      |Father of the Bride Part II (1995)|Comedy                                  |
+-------+---------------------------------+-----------------------------------------+
only showing top 5 rows

root
 |-- movieId: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- genres: string (nullable = true)
```

PySpark SLQ tutorial 2

```
movies.where(f.col("genres") == "Action").show(5, False)
movies.where("genres == 'Action'").show(5, False)
```

```
+-------+-------------------------------------------------------------+------+
|movieId|title                                                        |genres|
+-------+-------------------------------------------------------------+------+
|9      |Sudden Death (1995)                                          |Action|
|71     |Fair Game (1995)                                             |Action|
|204    |Under Siege 2: Dark Territory (1995)                         |Action|
|251    |Hunted, The (1995)                                           |Action|
|667    |Bloodsport 2 (a.k.a. Bloodsport II: The Next Kumite) (1996)|Action|
+-------+-------------------------------------------------------------+------+
only showing top 5 rows
```

PySpark SLQ tutorial 2

```
#convert genres string to genres array and store to new column
movies.withColumn("genres_array",f.split(f.col("genres"),"\|")).show(5,False)
```

```
+-------+---------------------------------+-----------------------------------------+-----------------------------------------------+
|movieId|title                            |genres                                   |genres array                                   |
+-------+---------------------------------+-----------------------------------------+-----------------------------------------------+
|1      |Toy Story (1995)                 |Adventure|Animation|Children|Comedy|Fantasy|[Adventure, Animation, Children, Comedy, Fantasy]|
|2      |Jumanji (1995)                   |Adventure|Children|Fantasy                |[Adventure, Children, Fantasy]                  |
|3      |Grumpier Old Men (1995)          |Comedy|Romance                           |[Comedy, Romance]                              |
|4      |Waiting to Exhale (1995)         |Comedy|Drama|Romance                     |[Comedy, Drama, Romance]                       |
|5      |Father of the Bride Part II (1995)|Comedy                                  |[Comedy]                                       |
+-------+---------------------------------+-----------------------------------------+-----------------------------------------------+
only showing top 5 rows
```

PySpark SLQ tutorial 2

```
#use explode function to get a new row for each element in the genres_array
movies.withColumn("genres_array", f.split("genres", "\|")).withColumn("genre", f.explode("genres_array")).show(15,False)
```

```
+-------+---------------------------------+-----------------------------------------+-----------------------------------------------+---------+
|movieId|title                            |genres                                   |genres_array                                   |genre    |
+-------+---------------------------------+-----------------------------------------+-----------------------------------------------+---------+
|1      |Toy Story (1995)                 |Adventure|Animation|Children|Comedy|Fantasy|[Adventure, Animation, Children, Comedy, Fantasy]|Adventure|
|1      |Toy Story (1995)                 |Adventure|Animation|Children|Comedy|Fantasy|[Adventure, Animation, Children, Comedy, Fantasy]|Animation|
|1      |Toy Story (1995)                 |Adventure|Animation|Children|Comedy|Fantasy|[Adventure, Animation, Children, Comedy, Fantasy]|Children |
|1      |Toy Story (1995)                 |Adventure|Animation|Children|Comedy|Fantasy|[Adventure, Animation, Children, Comedy, Fantasy]|Comedy   |
|1      |Toy Story (1995)                 |Adventure|Animation|Children|Comedy|Fantasy|[Adventure, Animation, Children, Comedy, Fantasy]|Fantasy  |
|2      |Jumanji (1995)                   |Adventure|Children|Fantasy                |[Adventure, Children, Fantasy]                  |Adventure|
|2      |Jumanji (1995)                   |Adventure|Children|Fantasy                |[Adventure, Children, Fantasy]                  |Children |
|2      |Jumanji (1995)                   |Adventure|Children|Fantasy                |[Adventure, Children, Fantasy]                  |Fantasy  |
|3      |Grumpier Old Men (1995)          |Comedy|Romance                           |[Comedy, Romance]                              |Comedy   |
|3      |Grumpier Old Men (1995)          |Comedy|Romance                           |[Comedy, Romance]                              |Romance  |
|4      |Waiting to Exhale (1995)         |Comedy|Drama|Romance                     |[Comedy, Drama, Romance]                       |Comedy   |
|4      |Waiting to Exhale (1995)         |Comedy|Drama|Romance                     |[Comedy, Drama, Romance]                       |Drama    |
|4      |Waiting to Exhale (1995)         |Comedy|Drama|Romance                     |[Comedy, Drama, Romance]                       |Romance  |
|5      |Father of the Bride Part II (1995)|Comedy                                   |[Comedy]                                       |Comedy   |
|6      |Heat (1995)                      |Action|Crime|Thriller                     |[Action, Crime, Thriller]                      |Action   |
+-------+---------------------------------+-----------------------------------------+-----------------------------------------------+---------+
only showing top 15 rows
```

PySpark SLQ tutorial 2

```
#show final listed genres of each movie
movies.withColumn("genres_array", f.split("genres", "\|")).withColumn("last_genre",f.element_at("genres_array",-1)).show(15, False)
```

```
+-------+---------------------------------+-----------------------------------------+-----------------------------------------------+----------+
|movieId|title                            |genres                                   |genres_array                                   |last_genre|
+-------+---------------------------------+-----------------------------------------+-----------------------------------------------+----------+
|1      |Toy Story (1995)                 |Adventure|Animation|Children|Comedy|Fantasy|[Adventure, Animation, Children, Comedy, Fantasy]|Fantasy   |
|2      |Jumanji (1995)                   |Adventure|Children|Fantasy                |[Adventure, Children, Fantasy]                  |Fantasy   |
|3      |Grumpier Old Men (1995)          |Comedy|Romance                           |[Comedy, Romance]                              |Romance   |
|4      |Waiting to Exhale (1995)         |Comedy|Drama|Romance                     |[Comedy, Drama, Romance]                       |Romance   |
|5      |Father of the Bride Part II (1995)|Comedy                                   |[Comedy]                                       |Comedy    |
|6      |Heat (1995)                      |Action|Crime|Thriller                     |[Action, Crime, Thriller]                      |Thriller  |
|7      |Sabrina (1995)                   |Comedy|Romance                           |[Comedy, Romance]                              |Romance   |
|8      |Tom and Huck (1995)              |Adventure|Children                        |[Adventure, Children]                          |Children  |
|9      |Sudden Death (1995)              |Action                                   |[Action]                                       |Action    |
|10     |GoldenEye (1995)                 |Action|Adventure|Thriller                 |[Action, Adventure, Thriller]                  |Thriller  |
|11     |American President, The (1995)   |Comedy|Drama|Romance                     |[Comedy, Drama, Romance]                       |Romance   |
|12     |Dracula: Dead and Loving It (1995)|Comedy|Horror                            |[Comedy, Horror]                               |Horror    |
|13     |Balto (1995)                     |Adventure|Animation|Children              |[Adventure, Animation, Children]               |Children  |
|14     |Nixon (1995)                     |Drama                                    |[Drama]                                        |Drama     |
|15     |Cutthroat Island (1995)          |Action|Adventure|Romance                   |[Action, Adventure, Romance]                   |Romance   |
+-------+---------------------------------+-----------------------------------------+-----------------------------------------------+----------+
only showing top 15 rows
```

Q & A

Cảm ơn đã theo dõi

Chúng tôi hy vọng cùng nhau đi đến thành công.