1. **import** findspark
2. findspark.init()
3. **import** pyspark # only run after findspark.init()
4. from pyspark.sql **import** SparkSession
5. spark = SparkSession.builder.getOrCreate()
6. df = spark.sql('''select 'spark' as hello ''')
7. df.show()

**Code Explanation:**

In the above code, we have imported the **findspark** module and called **findspark.init()** constructor; then, we imported the SparkSession module to create spark session.

***from pyspark.sql import SparkSession***

A spark session can be used to create the Dataset and DataFrame API. A SparkSession can also be used to create DataFrame, register DataFrame as a table, execute SQL over tables, cache table, and read parquet file.

***class builder***

It is a builder of Spark Session.

***getOrCreate()***

It is used to get an existing **SparkSession,** or if there is no existing one, create a new one based on the options set in the builder.

Few other Methods

Few methods of PySpark SQL are following:

***1. appName(name)***

It is used to set the name of the application, which will be displayed in the Spark web UI. The parameter **name** accepts the name of the parameter.

***2. config(key=None, value = None, conf = None)***

It is used to set a config option. Options set using this method are automatically propagated to both **SparkConf** and **SparkSession**'s configuration.

1. from pyspark.conf **import** SparkConfSparkSession.builder.config(conf=SparkConf())

**Parameters:**

* **key-** A key name string of a configuration property.
* **value-** It represents the value of a configuration property.
* **conf -** An instance of SparkConf.

**3. master(master)**

It sets the spark master url to connect to, such as "local" to run locally, "local[4]" to run locally with 4 cores.

**Parameters:**

* **master:**a url for spark master.

**4. SparkSession.catalog**

It is an interface that the user may create, drop, alter, or query the underlying database, tables, functions, etc.

**5. SparkSession.conf**

It is runtime configuration interface for spark. This is the interface through that the user can get and set all Spark and Hadoop configurations that are relevant to Spark SQL.

## class pyspark.sql.DataFrame

It is a distributed collection of data grouped into named columns. A DataFrame is similar as the relational table in Spark SQL, can be created using various function in SQLContext.

1. student = sqlContext.read.csv("...")

After creation of dataframe, we can manipulate it using the several domain-specific-languages (DSL) which are pre-defined functions of DataFrame. Consider the following example.

1. # To create DataFrame using SQLContext
2. student = sqlContext.read.parquet("...")
3. department = sqlContext.read.parquet("...")
4. student.filter(marks > 55).join(department, student.student\_Id == department.id) \
5. .groupBy(student.name, "gender").({"name": "student\_Id", "mark": "department"})

Let's consider the following example:

**Querying Using Spark SQL**

In the following code, first, we create a DataFrame and execute the SQL queries to retrieve the data. Consider the following code:

1. from pyspark.sql **import** \*
2. #Create DataFrame
3. songdf = spark.read.csv(r"C:\Users\DEVANSH SHARMA\top50.csv", inferSchema = True, header = True)
4. #Perform SQL queries
5. songdf.select("Genre").show()
6. songdf.filter(songdf["Genre"]=="pop").show()

**Output:**

+----------------+

| Genre|

+----------------+

| canadian pop|

| reggaeton flow|

| dance pop|

| pop|

| dfw rap|

| pop|

| trap music|

| pop|

| country rap|

| electropop|

| reggaeton|

| dance pop|

| pop|

| panamanian pop|

|canadian hip hop|

| dance pop|

| latin|

| dfw rap|

|canadian hip hop|

| escape room|

+----------------+

only showing top 20 rows

+---+--------------------+-------------+-----+----------------+------+------------+--------------+--------+--------+-------+--------------+------------+----------+

|\_c0| Track.Name| Artist.Name|Genre|Beats.Per.Minute|Energy|Danceability|Loudness..dB..|Liveness|Valence.|Length.|Acousticness..|Speechiness.|Popularity|

+---+--------------------+-------------+-----+----------------+------+------------+--------------+--------+--------+-------+--------------+------------+----------+

| 4|Beautiful People ...| Ed Sheeran| pop| 93| 65| 64| -8| 8| 55| 198| 12| 19| 86|

| 6|I Don't Care (wit...| Ed Sheeran| pop| 102| 68| 80| -5| 9| 84| 220| 9| 4| 84|

| 8| How Do You Sleep?| Sam Smith| pop| 111| 68| 48| -5| 8| 35| 202| 15| 9| 90|

| 13| Someone You Loved|Lewis Capaldi| pop| 110| 41| 50| -6| 11| 45| 182| 75| 3| 88|

| 38|Antisocial (with ...| Ed Sheeran| pop| 152| 82| 72| -5| 36| 91| 162| 13| 5| 87|

| 44| Talk| Khalid| pop| 136| 40| 90| -9| 6| 35| 198| 5| 13| 84|

| 50|Cross Me (feat. C...| Ed Sheeran| pop| 95| 79| 75| -6| 7| 61| 206| 21| 12| 82|

+---+--------------------+-------------+-----+----------------+------+------------+--------------+--------+--------+-------+--------------+------------+----------+

**Using groupBy() function**

The groupBy() function collects the similar category data.

1. songdf.groupBy("Genre").count().show()

**Output:**

+----------------+-----+

| Genre|count|

+----------------+-----+

| boy band| 1|

| electropop| 2|

| pop| 7|

| brostep| 2|

| big room| 1|

| pop house| 1|

| australian pop| 1|

| edm| 3|

| r&b en espanol| 1|

| dance pop| 8|

| reggaeton| 2|

| canadian pop| 2|

| trap music| 1|

| escape room| 1|

| reggaeton flow| 2|

| panamanian pop| 2|

| atl hip hop| 1|

| country rap| 2|

|canadian hip hop| 3|

| dfw rap| 2|

+----------------+-----+

**repartition(numpartitions, \*cols)**

The **repartition()** returns a new DataFrame which is a partitioning expression. This function accepts two parameter **numpartitions** and **\*col.** The **numpartitions** parameter specifies the target number of columns.

1. song\_spotify.repartition(10).rdd.getNumPartitions()
2. data = song\_spotify.union(song\_spotify).repartition("Energy")
3. data.show(5)

**Output:**

+---+--------------------+-----------+-------+----------------+------+------------+--------------+--------+--------+-------+--------------+------------+----------+

|\_c0| Track.Name|Artist.Name| Genre|Beats.Per.Minute|Energy|Danceability|Loudness..dB..|Liveness|Valence.|Length.|Acousticness..|Speechiness.|Popularity|

+---+--------------------+-----------+-------+----------------+------+------------+--------------+--------+--------+-------+--------------+------------+----------+

| 4|Beautiful People ...| Ed Sheeran| pop| 93| 65| 64| -8| 8| 55| 198| 12| 19| 86|

| 5|Goodbyes (Feat. Y...|Post Malone|dfw rap| 150| 65| 58| -4| 11| 18| 175| 45| 7| 94|

| 17| LA CANCI?N| J Balvin| latin| 176| 65| 75| -6| 11| 43| 243| 15| 32| 90|

| 4|Beautiful People ...| Ed Sheeran| pop| 93| 65| 64| -8| 8| 55| 198| 12| 19| 86|

| 5|Goodbyes (Feat. Y...|Post Malone|dfw rap| 150| 65| 58| -4| 11| 18| 175| 45| 7| 94|

+---+--------------------+-----------+-------+----------------+------+------------+--------------+--------+--------+-------+--------------+------------+----------+

only showing top 5 rows

* **Using parallelize()**

1. from pyspark.sql **import** SparkSession
2. spark = SparkSession \
3. .builder \
4. .appName("PySpark create using parallelize() function RDD example") \
5. .config("spark.some.config.option", "some-value") \
6. .getOrCreate()
7. df = spark.sparkContext.parallelize([(12, 20, 35, 'a b c'),
8. (41, 58, 64, 'd e f'),
9. (70, 85, 90, 'g h i')]).toDF(['col1', 'col2', 'col3','col4'])
10. df.show()

**Output:**

+----+----+----+-----+

|col1|col2|col3| col4|

+----+----+----+-----+

| 12| 20| 35|a b c|

| 41| 58| 64|d e f|

| 70| 85| 90|g h i|

+----+----+----+-----+

* **Using createDataFrame() Function**

1. from pyspark.sql **import** SparkSession
2. spark = SparkSession \
3. .builder \
4. .appName("PySpark Create RDD example") \
5. .config("spark.some.config.option", "some-value") \
6. .getOrCreate()
7. Student = spark.createDataFrame([
8. ('009001', 'Anuj', '70%', 'B.tech(cs)'),
9. ('009002', 'Sachin', '80%', 'B.tech(cs)'),
10. ('008005', 'Yogesh', '94%', 'MCA'),
11. ('007014', 'Ananya', '98%', 'MCA')],
12. ['Roll\_Num', 'Name', 'Percentage','Department']
13. )
14. Student.show()

**Output:**

The above code will give the following RDD data.

+--------+------+----------+----------+

|Roll\_Num| Name|Percentage|Department|

+--------+------+----------+----------+

| 009001| Anuj| 70%|B.tech(cs)|

| 009002|Sachin| 80%|B.tech(cs)|

| 008005|Yogesh| 94%| MCA|

| 007014|Ananya| 98%| MCA|

+--------+------+----------+----------+

* **Using read and load functions**

Here we read dataset from .csv file using the **read()** function.

1. ## set up SparkSession
2. from pyspark.sql **import** SparkSession
3. spark = SparkSession \
4. .builder \
5. .appName("PySpark create RDD example") \
6. .config("spark.some.config.option", "some-value") \
7. .getOrCreate()
8. df = spark.read.format('com.databricks.spark.csv').\
9. options(header='true', \
10. inferschema='true').\
11. df = spark.read.format('com.databricks.spark.csv').\
12. options(header='true', \
13. inferschema='true').\
14. load(r"C:\Users\DEVANSH SHARMA\top50.csv",
15. header=True)
16. df.show(5)
17. df.printSchema()

**Output:**

+---+--------------------+-------------+--------------+----------------+------+------------+--------------+--------+--------+-------+--------------+------------+----------+

|\_c0| Track.Name| Artist.Name| Genre|Beats.Per.Minute|Energy|Danceability|Loudness..dB..|Liveness|Valence.|Length.|Acousticness..|Speechiness.|Popularity|

+---+--------------------+-------------+--------------+----------------+------+------------+--------------+--------+--------+-------+--------------+------------+----------+

| 1| Se?orita| Shawn Mendes| canadian pop| 117| 55| 76| -6| 8| 75| 191| 4| 3| 79|

| 2| China| Anuel AA|reggaeton flow| 105| 81| 79| -4| 8| 61| 302| 8| 9| 92|

| 3|boyfriend (with S...|Ariana Grande| dance pop| 190| 80| 40| -4| 16| 70| 186| 12| 46| 85|

| 4|Beautiful People ...| Ed Sheeran| pop| 93| 65| 64| -8| 8| 55| 198| 12| 19| 86|

| 5|Goodbyes (Feat. Y...| Post Malone| dfw rap| 150| 65| 58| -4| 11| 18| 175| 45| 7| 94|

+---+--------------------+-------------+--------------+----------------+------+------------+--------------+--------+--------+-------+--------------+------------+----------+

only showing top 5 rows

root

|-- \_c0: integer (nullable = true)

|-- Track.Name: string (nullable = true)

|-- Artist.Name: string (nullable = true)

|-- Genre: string (nullable = true)

|-- Beats.Per.Minute: integer (nullable = true)

|-- Energy: integer (nullable = true)

|-- Danceability: integer (nullable = true)

|-- Loudness..dB..: integer (nullable = true)

|-- Liveness: integer (nullable = true)

|-- Valence.: integer (nullable = true)

|-- Length.: integer (nullable = true)

|-- Acousticness..: integer (nullable = true)

|-- Speechiness.: integer (nullable = true)

|-- Popularity: integer (nullable = true)

RDD Operations in PySpark

The RDD supports two types of operations:

1. Transformations

Transformations are the process which are used to create a new RDD. It follows the principle of Lazy Evaluations (the execution will not start until an action is triggered**).** Few of transformations are given below:

* map
* flatMap
* filter
* distinct
* reduceByKey
* mapPartitions
* sortBy

2. Actions

Actions are the processes which are applied on an RDD to initiate Apache Spark to apply calculation and pass the result back to driver. Few actions are following:

* collect
* collectAsMap
* reduce
* countByKey/countByValue
* take
* first

1. from pyspark **import** SparkContext
2. words = sc.parallelize (
3. ["scala",
4. "java",
5. "hadoop",
6. "spark",
7. "akka",
8. "spark vs hadoop",
9. "pyspark",
10. "pyspark and spark"]
11. )
12. words\_filter = words.filter(lambda x: 'spark' in x)
13. filtered = words\_filter.collect()
14. print("Filtered RDD : %s" % (filtered))

**Output:**

Filtered RDD : ['spark', 'spark vs hadoop', 'pyspark', 'pyspark and spark']

## DataFrame from RDD

PySpark provides two methods to convert a RDD to DF. These methods are given following:

* **toDF()**

When we create RDD by parallelize function, we should identify the same row element in DataFrame and wrap those element by the parentheses. The **row()** can accept the **\*\*kwargs** argument.

1. from pyspark.sql.types **import** Row
2. from pyspark.sql **import** SparkSession
3. #here we are going to create a function
4. def f(x):
5. d = {}
6. **for** i in range(len(x)):
7. d[str(i)] = x[i]
8. **return** d
9. #Now populate that
10. df = rdd.map(lambda x: Row(\*\*f(x))).toDF()

* **createDataFrame(rdd, schema)**

We can also convert the RDD to DataFrame by using createDataFrame. Consider the following example:

1. from pyspark.sql.types **import** StructType
2. from pyspark.sql.types **import** StructField
3. from pyspark.sql.types **import** StringType
4. schema = StructType([StructField(str(i), StringType(), True) **for** i in range(32)])
5. df = sqlContext.createDataFrame(rdd, schema)