PySPark Introduction

what is pyspark?

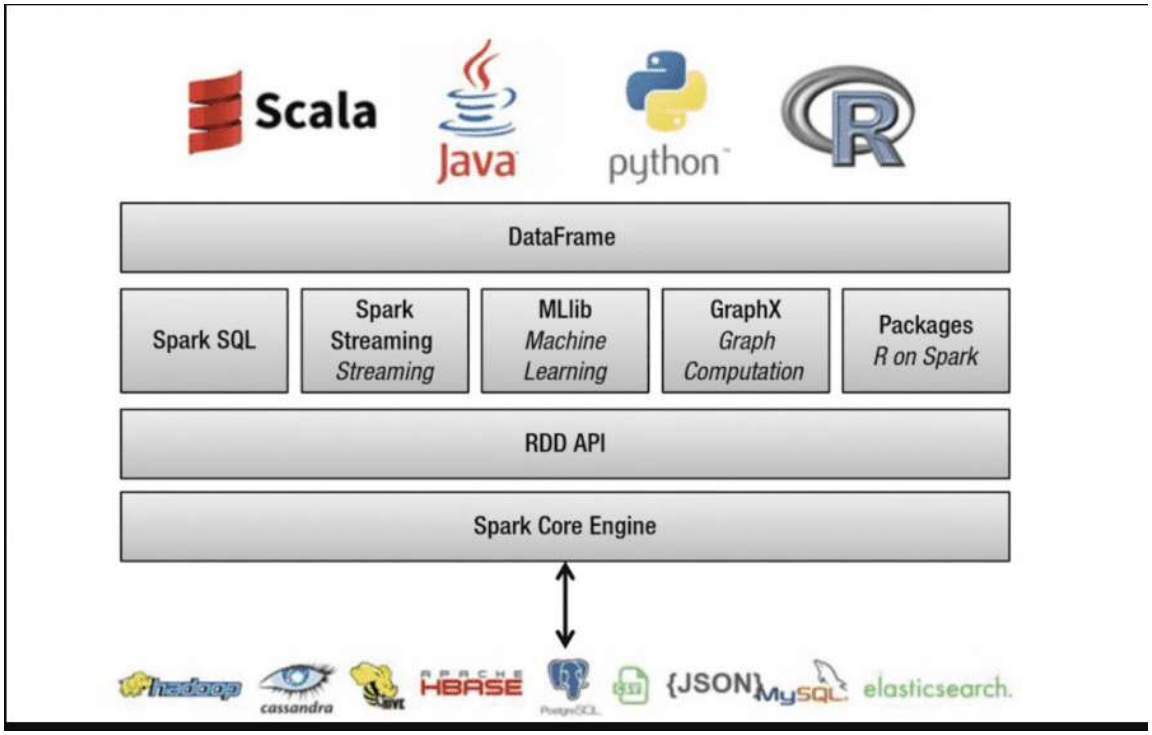
- PySpark is a Spark library written in Python to run Python applications using Apache Spark capabilities, using PySpark we can run applications parallelly on the distributed cluster (multiple nodes).
- PySpark is a Python API for Apache Spark. Apache Spark is an analytical processing engine for large scale powerful distributed data processing and machine learning applications.

PySpark Features

What are Pyspark features

- In-memory computation
- Distributed processing using parallelize
- Can be used with many cluster managers (Spark, Yarn, Mesos e.t.c)
- Fault-tolerant
- Immutable
- Lazy evaluation
- Cache & persistence
- Inbuild-optimization when using DataFrames
- Supports ANSI SQL

PySpark Architecture



Cluster Managers

What are the different clusters that can be integrated into pyspark ?

- Standalone Cluster: This is a local cluster and need to setup manually.
- Apache Mesos: Mesons is a Cluster manager that can also run Hadoop MapReduce and PySpark applications
- Hadoop YARN: This is mostly used, cluster manager which integrated in a hadoop eco system
- Kubernets: An open-source system for automating deployment, scaling, and management of containerized applications.

PySpark Modules

- PySpark.RDD
- Pyspark.sql
- Pyspark.streaming
- Pyspark.mlib
- Pyspark.ml
- Pyspark.GraphFrames
- Pyspark.resource

PySpark Installation

- Follow the below link to install PySpark

1. Spark Session

Sparksession has Two Modes

- Client mode
- Cluster mode

Session includes all the APIs available in different contexts –

- SparkContext
- SQLContext
- StreamingContext
- HiveContext.

Note:

- Before spark-2.0 version there is a spark context
- From spark-2.0 Sparksession is introduced and developed
- Above all contexts are integrated in the sparksession
- It is not recommended to create the spark context explicitly becuase spark session itself already have all the above contexts
- Remember, stop the spark context always afer using it because you can create only one spark context for one JVM.

In [ ]:
```
# CLient Mode
# Create SparkSession from builder
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]") \
                    .appName('My Application') \
                    .getOrCreate()
```

In [ ]:
```
# Create new SparkSession using newSession method
spark2 = SparkSession.newSession
print(spark2)
```

```
<function SparkSession.newSession at 0x000002787FC41A80>
```

In [ ]:
```
# Get Existing SparkSession
spark3 = SparkSession.builder.getOrCreate
print(spark3)
```

```
<bound method SparkSession.Builder.getOrCreate of <pyspark.sql.session.SparkSessio
n.Builder object at 0x000002787FDC7250>>
```

Core Spark Session

pyspark.sql.SparkSession

- The entry point to programming Spark with the Dataset and DataFrame API.
- A SparkSession can be used to create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files. To create a SparkSession, use the

following builder pattern.

SparkSession(sparkContext[, jsparkSession, ...])

- spark = (SparkSession.builder

```
  .master("local")
  .appName("Word Count")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()
)
```

- spark = (SparkSession.builder

```
  .remote("sc://localhost")
  .getActive
  .config("spark.some.config.option", "some-value")
  .getOrCreate()
) '''
```

Methods

`createDataFrame`(data[, schema, ...])	Creates a `DataFrame` from an `RDD`, a list, a `pandas.DataFrame` or a `numpy.ndarray`.
`getActiveSession`()	Returns the active `SparkSession` for the current thread, returned by the builder
`newSession`()	Returns a new `SparkSession` as new session, that has separate SQLConf, registered temporary views and UDFs, but shared `SparkContext` and table cache.
`range`(start[, end, step, numPartitions])	Create a `DataFrame` with single `pyspark.sql.types.LongType` column named `id`, containing elements in a range from `start` to `end` (exclusive) with step value `step`.
`sql`(sqlQuery[, args])	Returns a `DataFrame` representing the result of the given query.
`stop`()	Stop the underlying `SparkContext`.
`table`(tableName)	Returns the specified table as a `DataFrame`.

Attributes

`builder`	
`catalog`	Interface through which the user may create, drop, alter or query underlying databases, tables, functions, etc.
`conf`	Runtime configuration interface for Spark.
`read`	Returns a `DataFrameReader` that can be used to read data in as a `DataFrame`.
`readStream`	Returns a `DataStreamReader` that can be used to read data streams as a streaming `DataFrame`.
`sparkContext`	Returns the underlying `SparkContext`.
`streams`	Returns a `StreamingQueryManager` that allows managing all the `StreamingQuery` instances active on *this* context.
`udf`	Returns a `UDFRegistration` for UDF registration.
`version`	The version of Spark on which this application is running.

pyspark.sql.sparkSession.builder.appName(name)

Sets a name for the application, which will be shown in the Spark web UI

- syntax: SparkSession.builder.appName("My app")

pyspark.sql.SparkSession.builder.config

Sets a config option. Options set using this method are automatically propagated to both SparkConf and SparkSession's own configuration.

- from pyspark.conf import SparkConf
- SparkSession.builder.config(conf=SparkConf())
- SparkSession.builder.config("spark.some.config.option", "some-value")
- syntax: SparkSession.builder.config(map={"spark.some.config.number": 123, "spark.some.config.float": 0.123})

pyspark.sql.SparkSession.builder.enableHiveSupport

Enables Hive support, including connectivity to a persistent Hive metastore, support for Hive SerDes, and Hive user-defined functions.

- syntax: SparkSession.builder.enableHiveSupport()

pyspark.sql.SparkSession.builder.getOrCreate

- Gets an existing SparkSession or, if there is no existing one, creates a new one based on the options set in this builder.

This method first checks whether there is a valid global default SparkSession, and if yes, return that one. If no valid global default SparkSession exists, the method creates a new SparkSession and assigns the newly created SparkSession as the global default.

- syntax: s1 = SparkSession.builder.config("k1", "v1").getOrCreate()

The configuration of the SparkSession can be changed afterwards

- s1.conf.set("k1", "v1_new")
- s1.conf.get("k1") == "v1_new"

In case an existing SparkSession is returned, the config options specified in this builder will be applied to the existing SparkSession.

- s2 = SparkSession.builder.config("k2", "v2").getOrCreate()
- s1.conf.get("k1") == s2.conf.get("k1") == "v1_new"
- s1.conf.get("k2") == s2.conf.get("k2") == "v2"

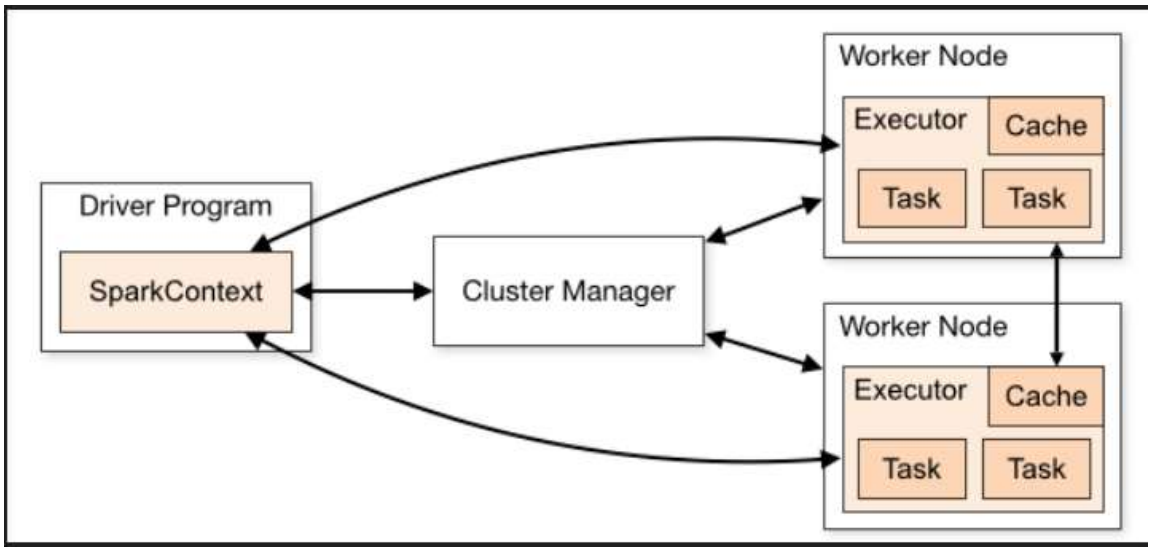
pyspark.sql.SparkSession.builder.master

Sets the Spark master URL to connect to, such as “local” to run locally, “local[4]” to run locally with 4 cores, or “spark://master:7077” to run on a Spark standalone cluster.

- syntax: SparkSession.builder.master("local")

2. Spark Context

In PySpark, initializing a SparkContext can be done in a few different ways.



1. Using SparkSession (Recommended):

- Starting from Spark 2.0, the recommended way to initialize Spark is by using the SparkSession, which encapsulates both the SparkContext and SQLContext.

In [ ]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

# Use spark to create RDDs, DataFrames, etc.

# Stop the SparkSession when done
spark.stop()
```

2. Creating SparkContext Directly (Legacy):

- If you're working with an older version of Spark or have specific requirements, you can create a SparkContext directly.

In [ ]:
```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName("My Application").setMaster("local[*]")
sc = SparkContext(conf=conf)

# Use sc to create RDDs and perform operations

# Stop the SparkContext when done
sc.stop()
```

3. Creating SparkContext with Additional Configuration:

- If you need to set additional configuration options for your SparkContext, you can do so by passing them in the SparkConf

In [ ]:
```
from pyspark import SparkContext, SparkConf

conf = SparkConf() \
    .setAppName("My Application") \
    .setMaster("local[*]") \
    .set("spark.some.config.option", "config-value")

sc = SparkContext(conf=conf)

# Use sc to create RDDs and perform operations

# Stop the SparkContext when done
sc.stop()
```

Note

- you can create only one SparkContext per JVM, in order to create another first you need to stop the existing one using stop() method.

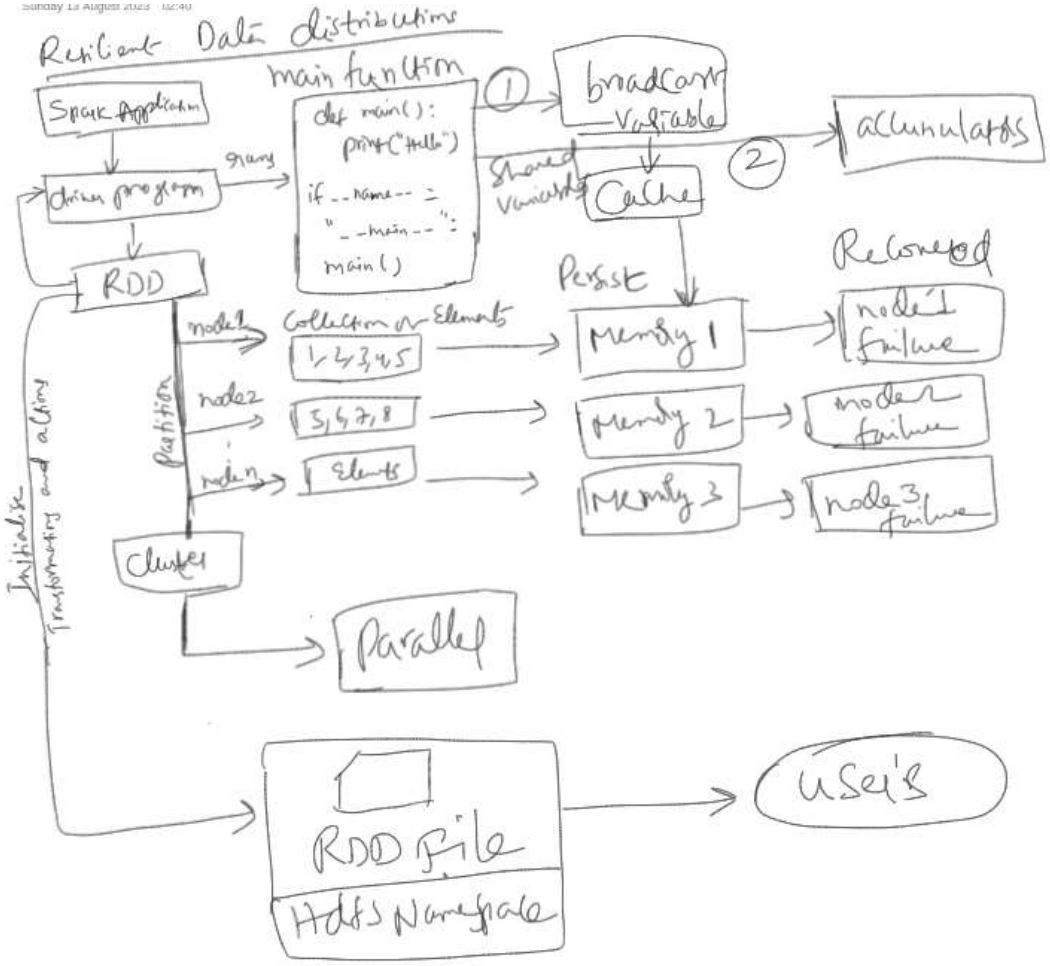
SparkContext Commonly Used Methods

- accumulator(value[, accum_param]) – It creates an pyspark accumulator variable with initial specified value. Only a driver can access accumulator variables.
- broadcast(value) – read-only PySpark broadcast variable. This will be broadcast to the entire cluster. You can broadcast a variable to a PySpark cluster only once.
- emptyRDD() – Creates an empty RDD
- getOrCreate() – Creates or returns a SparkContext
- hadoopFile() – Returns an RDD of a Hadoop file
- newAPIHadoopFile() – Creates an RDD for a Hadoop file with a new API InputFormat.
- sequenceFile() – Get an RDD for a Hadoop SequenceFile with given key and value types.
- setLogLevel() – Change log level to debug, info, warn, fatal, and error
- textFile() – Reads a text file from HDFS, local or any Hadoop supported file systems and returns an RDD
- union() – Union two RDDs
- wholeTextFiles() – Reads a text file in the folder from HDFS, local or any Hadoop supported file systems and returns an RDD of Tuple2. The first element of the tuple consists file name and the second element consists context of the text file.

3.RDD (Resilient Distributed Dataset)

What is a RDD?

- RDD (Resilient Distributed Dataset) is a fundamental building block of PySpark which is fault-tolerant, immutable distributed collections of objects. Immutable meaning once you create an RDD you cannot change it. Each record in RDD is divided into logical partitions, which can be computed on different nodes of the cluster.
- In other words, RDDs are a collection of objects similar to list in Python, with the difference being RDD is computed on several processes scattered across multiple physical servers also called nodes in a cluster while a Python collection lives and process in just one process.



RDD Benefits

- In-Memory Processing PySpark loads the data from disk and process in memory and keeps the data in memory, this is the main difference between PySpark and Mapreduce (I/O intensive). In between the transformations, we can also cache/persists the RDD in memory to reuse the previous computations.
- Immutability PySpark RDD's are immutable in nature meaning, once RDDs are created you cannot modify. When we apply transformations on RDD, PySpark creates a new RDD and maintains the RDD Lineage.
- Fault Tolerance PySpark operates on fault-tolerant data stores on HDFS, S3 e.t.c hence any RDD operation fails, it automatically reloads the data from other partitions. Also, When PySpark applications running on a cluster, PySpark task failures are automatically recovered for a certain number of times (as per the configuration) and finish the application seamlessly.
- Lazy Evolution PySpark does not evaluate the RDD transformations as they appear/encountered by Driver instead it keeps the all transformations as it encounters(DAG) and evaluates the all transformation when it sees the first RDD action.
- Partitioning When you create RDD from a data, It by default partitions the elements in a RDD. By default it partitions to the number of cores available.

what types of data structures can rdd does process ?

- RDD (Resilient Distributed Dataset) in Spark is designed to store and manage any type of data. It is a fundamental abstraction that can hold a wide range of data structures, including:

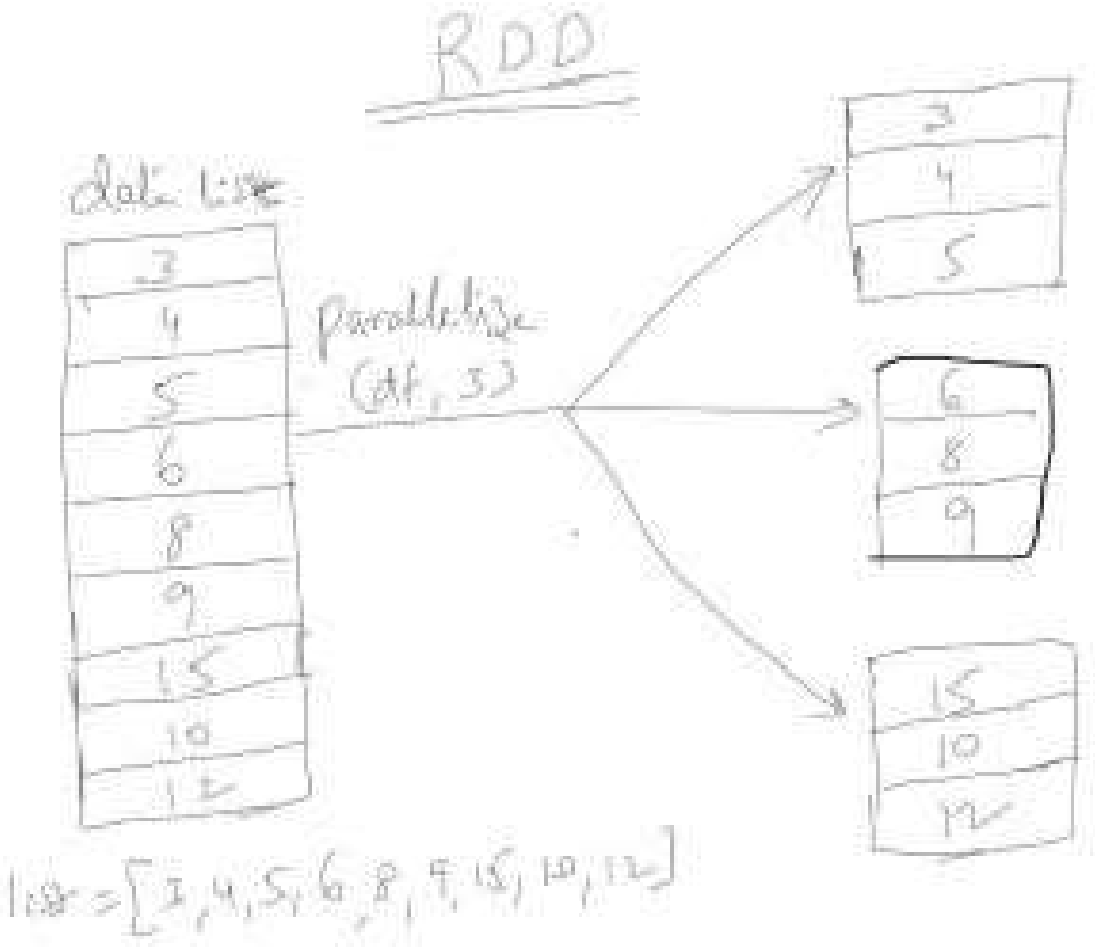
1. **Basic Types**: RDDs can store simple types such as integers, floats, strings, and booleans.
2. **Tuples**: RDDs can store tuples, which are ordered collections of elements. Tuples can contain a mix of different data types.
3. **Lists and Arrays**: RDDs can store lists or arrays of elements. Each element in the RDD can be a list or array of arbitrary length.
4. **Dictionaries/Maps**: RDDs can store dictionaries or maps, which are key-value pairs. Each element in the RDD can be a dictionary with key-value pairs.
5. **Custom Objects**: RDDs can store instances of custom classes or objects that you define. However, the objects need to be serializable since RDDs are distributed across the cluster.
6. **Structured Data**: RDDs can store structured data, such as rows or records with named fields.
7. **Nested Structures**: RDDs can store complex nested structures, where elements can be combinations of lists, tuples, dictionaries, or custom objects.

Keep in mind that while RDDs can store a variety of data structures, the actual data type and structure of the RDD is determined by how you create and transform it. The operations you perform on the RDD, such as `map`, `filter`, and `reduce`, influence the structure and type of the resulting RDD.

It's also important to note that while RDDs offer a lot of flexibility in terms of data types, they don't provide built-in optimizations for complex data types as compared to higher-level abstractions like DataFrames or Datasets in Spark. These higher-level abstractions provide schema information and optimizations that make working with structured and semi-structured data more efficient.

Creating RDD's

- RDD's are created primarily in two different ways
  - parallelizing collection (creating a new collection from an existing collection)
  - referencing a dataset from an external storage system (HDFS, S3 and many more).



First Method: Parallelized Collections

Create RDD using sparkContext.parallelize()

In [ ]:
```
# Firstly, you need to create a spark session which contains sparkContext

from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").appName("My Application").getOrCrea

# This is the basic method in creating RDD.
# When you alreaady have data in memory that is either loaded from a file or a data
# Initialising RDD usoing Parallelize()

data = [1,2,3,4,5,6,7,8]

rdd = spark.sparkContext.parallelize(data)

print(rdd.collect())

### Creating an empy RDD

rdd1 = spark.sparkContext.emptyRDD

print(rdd1)

#Create empty RDD with partition
rdd2 = spark.sparkContext.parallelize([],10) #This creates 10 partitions

print(rdd2.collect())
```

```
spark.stop()
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
<bound method SparkContext.emptyRDD of <SparkContext master=local[*] appName=My Ap
plication>>
[]
```

Second Method: External Datasets

In [ ]:

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

data = pd.read_csv("../input/car_data.csv")

print(data.columns)

df = data[["Selling_Price","Car_Name"]]

df1 = df.values.tolist()

rdd_df = spark.sparkContext.parallelize(df1)

print(rdd_df.collect())

spark.stop()
```

```
Index(['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Kms_Driven',
       'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner'],
      dtype='object')
[[3.35, 'ritz'], [4.75, 'sx4'], [7.25, 'ciaz'], [2.85, 'wagon r'], [4.6, 'swift'],
[9.25, 'vitara brezza'], [6.75, 'ciaz'], [6.5, 's cross'], [8.75, 'ciaz'], [7.45,
'ciaz'], [2.85, 'alto 800'], [6.85, 'ciaz'], [7.5, 'ciaz'], [6.1, 'ertiga'], [2.2
5, 'dzire'], [7.75, 'ertiga'], [7.25, 'ertiga'], [7.75, 'ertiga'], [3.25, 'wagon
r'], [2.65, 'sx4'], [2.85, 'alto k10'], [4.9, 'ignis'], [4.4, 'sx4'], [2.5, 'alto
k10'], [2.9, 'wagon r'], [3.0, 'swift'], [4.15, 'swift'], [6.0, 'swift'], [1.95,
'alto k10'], [7.45, 'ciaz'], [3.1, 'ritz'], [2.35, 'ritz'], [4.95, 'swift'], [6.0,
'ertiga'], [5.5, 'dzire'], [2.95, 'sx4'], [4.65, 'dzire'], [0.35, '800'], [3.0, 'a
lto k10'], [2.25, 'sx4'], [5.85, 'baleno'], [2.55, 'alto k10'], [1.95, 'sx4'], [5.
5, 'dzire'], [1.25, 'omni'], [7.5, 'ciaz'], [2.65, 'ritz'], [1.05, 'wagon r'], [5.
8, 'ertiga'], [7.75, 'ciaz'], [14.9, 'fortuner'], [23.0, 'fortuner'], [18.0, 'inno
va'], [16.0, 'fortuner'], [2.75, 'innova'], [3.6, 'corolla altis'], [4.5, 'etios c
ross'], [4.75, 'corolla altis'], [4.1, 'etios g'], [19.99, 'fortuner'], [6.95, 'co
rolla altis'], [4.5, 'etios cross'], [18.75, 'fortuner'], [23.5, 'fortuner'], [33.
0, 'fortuner'], [4.75, 'etios liva'], [19.75, 'innova'], [9.25, 'fortuner'], [4.3
5, 'corolla altis'], [14.25, 'corolla altis'], [3.95, 'etios liva'], [4.5, 'coroll
a altis'], [7.45, 'corolla altis'], [2.65, 'etios liva'], [4.9, 'etios cross'],
[3.95, 'etios g'], [5.5, 'corolla altis'], [1.5, 'corolla'], [5.25, 'corolla alti
s'], [14.5, 'fortuner'], [14.73, 'corolla altis'], [4.75, 'etios gd'], [23.0, 'inn
ova'], [12.5, 'innova'], [3.49, 'innova'], [2.5, 'camry'], [35.0, 'land cruiser'],
[5.9, 'corolla altis'], [3.45, 'etios liva'], [4.75, 'etios g'], [3.8, 'corolla al
tis'], [11.25, 'innova'], [3.51, 'innova'], [23.0, 'fortuner'], [4.0, 'corolla alt
is'], [5.85, 'corolla altis'], [20.75, 'innova'], [17.0, 'corolla altis'], [7.05,
'corolla altis'], [9.65, 'fortuner'], [1.75, 'Royal Enfield Thunder 500'], [1.7,
'UM Renegade Mojave'], [1.65, 'KTM RC200'], [1.45, 'Bajaj Dominar 400'], [1.35, 'R
oyal Enfield Classic 350'], [1.35, 'KTM RC390'], [1.35, 'Hyosung GT250R'], [1.25,
'Royal Enfield Thunder 350'], [1.2, 'Royal Enfield Thunder 350'], [1.2, 'Royal Enf
ield Classic 350'], [1.2, 'KTM RC200'], [1.15, 'Royal Enfield Thunder 350'], [1.1
5, 'KTM 390 Duke '], [1.15, 'Mahindra Mojo XT300'], [1.15, 'Royal Enfield Classic
350'], [1.11, 'Royal Enfield Classic 350'], [1.1, 'Royal Enfield Classic 350'],
[1.1, 'Royal Enfield Thunder 500'], [1.1, 'Royal Enfield Classic 350'], [1.05, 'Ro
yal Enfield Thunder 500'], [1.05, 'Bajaj Pulsar RS200'], [1.05, 'Royal Enfield Thu
nder 350'], [1.05, 'Royal Enfield Bullet 350'], [1.0, 'Royal Enfield Classic 35
0'], [0.95, 'Royal Enfield Classic 500'], [0.9, 'Royal Enfield Classic 500'], [0.
9, 'Bajaj Avenger 220'], [0.75, 'Bajaj Avenger 150'], [0.8, 'Honda CB Hornet 160
R'], [0.78, 'Yamaha FZ S V 2.0'], [0.75, 'Honda CB Hornet 160R'], [0.75, 'Yamaha F
Z 16'], [0.75, 'Bajaj Avenger 220'], [0.72, 'Bajaj Avenger 220'], [0.65, 'TVS Apac
he RTR 160'], [0.65, 'Bajaj Pulsar 150'], [0.65, 'Honda CBR 150'], [0.65, 'Hero Ex
treme'], [0.6, 'Honda CB Hornet 160R'], [0.6, 'Bajaj Avenger 220 dtsi'], [0.6, 'Ho
nda CBR 150'], [0.6, 'Bajaj Avenger 150 street'], [0.6, 'Yamaha FZ  v 2.0'], [0.6,
'Yamaha FZ  v 2.0'], [0.6, 'Bajaj Pulsar  NS 200'], [0.6, 'TVS Apache RTR 160'],
[0.55, 'Hero Extreme'], [0.55, 'Yamaha FZ S V 2.0'], [0.52, 'Bajaj Pulsar 220 F'],
[0.51, 'Bajaj Pulsar 220 F'], [0.5, 'TVS Apache RTR 180'], [0.5, 'Hero Passion X p
ro'], [0.5, 'Bajaj Pulsar NS 200'], [0.5, 'Bajaj Pulsar NS 200'], [0.5, 'Yamaha Fa
zer '], [0.48, 'Honda Activa 4G'], [0.48, 'TVS Sport '], [0.48, 'Yamaha FZ S V 2.
0'], [0.48, 'Honda Dream Yuga '], [0.45, 'Honda Activa 4G'], [0.45, 'Bajaj Avenger
Street 220'], [0.45, 'TVS Apache RTR 180'], [0.45, 'Bajaj Pulsar NS 200'], [0.45,
'Bajaj Avenger 220 dtsi'], [0.45, 'Hero Splender iSmart'], [0.45, 'Activa 3g'],
[0.45, 'Hero Passion Pro'], [0.42, 'TVS Apache RTR 160'], [0.42, 'Honda CB Trigge
r'], [0.4, 'Hero Splender iSmart'], [0.4, 'Yamaha FZ S '], [0.4, 'Hero Passion Pr
o'], [0.4, 'Bajaj Pulsar 135 LS'], [0.4, 'Activa 4g'], [0.38, 'Honda CB Unicorn'],
[0.38, 'Hero Honda CBZ extreme'], [0.35, 'Honda Karizma'], [0.35, 'Honda Activa 12
5'], [0.35, 'TVS Jupyter'], [0.31, 'Honda Karizma'], [0.3, 'Hero Honda Passion Pr
o'], [0.3, 'Hero Splender Plus'], [0.3, 'Honda CB Shine'], [0.27, 'Bajaj Discover
100'], [0.25, 'Bajaj Pulsar 150'], [0.25, 'Suzuki Access 125'], [0.25, 'TVS Weg
o'], [0.25, 'Honda CB twister'], [0.25, 'Hero Glamour'], [0.2, 'Hero Super Splendo
r'], [0.2, 'Bajaj Pulsar 150'], [0.2, 'Bajaj Discover 125'], [0.2, 'Hero Hunk'],
[0.2, 'Hero  Ignitor Disc'], [0.2, 'Hero  CBZ Xtreme'], [0.18, 'Bajaj  ct 100'],
[0.17, 'Activa 3g'], [0.16, 'Honda CB twister'], [0.15, 'Bajaj Discover 125'], [0.
12, 'Honda CB Shine'], [0.1, 'Bajaj Pulsar 150'], [3.25, 'i20'], [4.4, 'grand i1
0'], [2.95, 'i10'], [2.75, 'eon'], [5.25, 'grand i10'], [5.75, 'xcent'], [5.15, 'g
```

```
rand i10'], [7.9, 'i20'], [4.85, 'grand i10'], [3.1, 'i10'], [11.75, 'elantra'],
[11.25, 'creta'], [2.9, 'i20'], [5.25, 'grand i10'], [4.5, 'verna'], [2.9, 'eon'],
[3.15, 'eon'], [6.45, 'verna'], [4.5, 'verna'], [3.5, 'eon'], [4.5, 'i20'], [6.0,
'i20'], [8.25, 'verna'], [5.11, 'verna'], [2.7, 'i10'], [5.25, 'grand i10'], [2.5
5, 'i10'], [4.95, 'verna'], [3.1, 'i20'], [6.15, 'verna'], [9.25, 'verna'], [11.4
5, 'elantra'], [3.9, 'grand i10'], [5.5, 'grand i10'], [9.1, 'verna'], [3.1, 'eo
n'], [11.25, 'creta'], [4.8, 'verna'], [2.0, 'eon'], [5.35, 'verna'], [4.75, 'xcen
t'], [4.4, 'xcent'], [6.25, 'i20'], [5.95, 'verna'], [5.2, 'verna'], [3.75, 'i2
0'], [5.95, 'verna'], [4.0, 'i10'], [5.25, 'i20'], [12.9, 'creta'], [5.0, 'city'],
[5.4, 'brio'], [7.2, 'city'], [5.25, 'city'], [3.0, 'brio'], [10.25, 'city'], [8.
5, 'city'], [8.4, 'city'], [3.9, 'amaze'], [9.15, 'city'], [5.5, 'brio'], [4.0, 'a
maze'], [6.6, 'jazz'], [4.0, 'amaze'], [6.5, 'jazz'], [3.65, 'amaze'], [8.35, 'cit
y'], [4.8, 'brio'], [6.7, 'city'], [4.1, 'city'], [3.0, 'city'], [7.5, 'city'],
[2.25, 'jazz'], [5.3, 'brio'], [10.9, 'city'], [8.65, 'city'], [9.7, 'city'], [6.
0, 'jazz'], [6.25, 'city'], [5.25, 'brio'], [2.1, 'city'], [8.25, 'city'], [8.99,
'city'], [3.5, 'brio'], [7.4, 'jazz'], [5.65, 'jazz'], [5.75, 'amaze'], [8.4, 'cit
y'], [10.11, 'city'], [4.5, 'amaze'], [5.4, 'brio'], [6.4, 'jazz'], [3.25, 'cit
y'], [3.75, 'amaze'], [8.55, 'city'], [9.5, 'city'], [4.0, 'brio'], [3.35, 'cit
y'], [11.5, 'city'], [5.3, 'brio']]
```

In [ ]:

```python
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

# data = pd.read_csv("../input/car_data.csv")

data = spark.read.csv("../input/car_data.csv", header=True, inferSchema=True)

data.show()

spark.stop()
```

```
+-------------+----+-------------+-------------+----------+---------+-----------+-
-----------+-----+
|     Car_Name|Year|Selling_Price|Present_Price|Kms_Driven|Fuel_Type|Seller_Type|T
ransmission|Owner|
+-------------+----+-------------+-------------+----------+---------+-----------+-
-----------+-----+
|         ritz|2014|         3.35|         5.59|     27000|   Petrol|     Dealer|
Manual|    0|
|          sx4|2013|         4.75|         9.54|     43000|   Diesel|     Dealer|
Manual|    0|
|         ciaz|2017|         7.25|         9.85|      6900|   Petrol|     Dealer|
Manual|    0|
|      wagon r|2011|         2.85|         4.15|      5200|   Petrol|     Dealer|
Manual|    0|
|        swift|2014|          4.6|         6.87|     42450|   Diesel|     Dealer|
Manual|    0|
|vitara brezza|2018|         9.25|         9.83|      2071|   Diesel|     Dealer|
Manual|    0|
|         ciaz|2015|         6.75|         8.12|     18796|   Petrol|     Dealer|
Manual|    0|
|      s cross|2015|          6.5|         8.61|     33429|   Diesel|     Dealer|
Manual|    0|
|         ciaz|2016|         8.75|         8.89|     20273|   Diesel|     Dealer|
Manual|    0|
|         ciaz|2015|         7.45|         8.92|     42367|   Diesel|     Dealer|
Manual|    0|
|     alto 800|2017|         2.85|          3.6|      2135|   Petrol|     Dealer|
Manual|    0|
|         ciaz|2015|         6.85|        10.38|     51000|   Diesel|     Dealer|
Manual|    0|
|         ciaz|2015|          7.5|         9.94|     15000|   Petrol|     Dealer|
Automatic|    0|
|       ertiga|2015|          6.1|         7.71|     26000|   Petrol|     Dealer|
Manual|    0|
|        dzire|2009|         2.25|         7.21|     77427|   Petrol|     Dealer|
Manual|    0|
|       ertiga|2016|         7.75|        10.79|     43000|   Diesel|     Dealer|
Manual|    0|
|       ertiga|2015|         7.25|        10.79|     41678|   Diesel|     Dealer|
Manual|    0|
|       ertiga|2016|         7.75|        10.79|     43000|   Diesel|     Dealer|
Manual|    0|
|      wagon r|2015|         3.25|         5.09|     35500|      CNG|     Dealer|
Manual|    0|
|          sx4|2010|         2.65|         7.98|     41442|   Petrol|     Dealer|
Manual|    0|
+-------------+----+-------------+-------------+----------+---------+-----------+-
-----------+-----+
only showing top 20 rows
```

Conversions of data columns from one structure to another

In [ ]:

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
```

```
    .master("local[*]") \
    .getOrCreate()

data = pd.read_csv("../input/car_data.csv")

df = data[["Car_Name", "Selling_Price"]]

my_list = df.values.tolist()

# print(my_list[0:5])

rdd = spark.sparkContext.parallelize(my_list)

print("list elements:", rdd.collect())

rdd_df = rdd.collect()

# Converting to a dictionary
data_dict = dict(rdd_df)

print("dictionaries:", data_dict)

spark.stop()
```

```
list elements: [['ritz', 3.35], ['sx4', 4.75], ['ciaz', 7.25], ['wagon r', 2.85],
['swift', 4.6], ['vitara brezza', 9.25], ['ciaz', 6.75], ['s cross', 6.5], ['cia
z', 8.75], ['ciaz', 7.45], ['alto 800', 2.85], ['ciaz', 6.85], ['ciaz', 7.5], ['er
tiga', 6.1], ['dzire', 2.25], ['ertiga', 7.75], ['ertiga', 7.25], ['ertiga', 7.7
5], ['wagon r', 3.25], ['sx4', 2.65], ['alto k10', 2.85], ['ignis', 4.9], ['sx4',
4.4], ['alto k10', 2.5], ['wagon r', 2.9], ['swift', 3.0], ['swift', 4.15], ['swif
t', 6.0], ['alto k10', 1.95], ['ciaz', 7.45], ['ritz', 3.1], ['ritz', 2.35], ['swi
ft', 4.95], ['ertiga', 6.0], ['dzire', 5.5], ['sx4', 2.95], ['dzire', 4.65], ['80
0', 0.35], ['alto k10', 3.0], ['sx4', 2.25], ['baleno', 5.85], ['alto k10', 2.55],
['sx4', 1.95], ['dzire', 5.5], ['omni', 1.25], ['ciaz', 7.5], ['ritz', 2.65], ['wa
gon r', 1.05], ['ertiga', 5.8], ['ciaz', 7.75], ['fortuner', 14.9], ['fortuner', 2
3.0], ['innova', 18.0], ['fortuner', 16.0], ['innova', 2.75], ['corolla altis', 3.
6], ['etios cross', 4.5], ['corolla altis', 4.75], ['etios g', 4.1], ['fortuner',
19.99], ['corolla altis', 6.95], ['etios cross', 4.5], ['fortuner', 18.75], ['fort
uner', 23.5], ['fortuner', 33.0], ['etios liva', 4.75], ['innova', 19.75], ['fortu
ner', 9.25], ['corolla altis', 4.35], ['corolla altis', 14.25], ['etios liva', 3.9
5], ['corolla altis', 4.5], ['corolla altis', 7.45], ['etios liva', 2.65], ['etios
cross', 4.9], ['etios g', 3.95], ['corolla altis', 5.5], ['corolla', 1.5], ['corol
la altis', 5.25], ['fortuner', 14.5], ['corolla altis', 14.73], ['etios gd', 4.7
5], ['innova', 23.0], ['innova', 12.5], ['innova', 3.49], ['camry', 2.5], ['land c
ruiser', 35.0], ['corolla altis', 5.9], ['etios liva', 3.45], ['etios g', 4.75],
['corolla altis', 3.8], ['innova', 11.25], ['innova', 3.51], ['fortuner', 23.0],
['corolla altis', 4.0], ['corolla altis', 5.85], ['innova', 20.75], ['corolla alti
s', 17.0], ['corolla altis', 7.05], ['fortuner', 9.65], ['Royal Enfield Thunder 50
0', 1.75], ['UM Renegade Mojave', 1.7], ['KTM RC200', 1.65], ['Bajaj Dominar 400',
1.45], ['Royal Enfield Classic 350', 1.35], ['KTM RC390', 1.35], ['Hyosung GT250
R', 1.35], ['Royal Enfield Thunder 350', 1.25], ['Royal Enfield Thunder 350', 1.
2], ['Royal Enfield Classic 350', 1.2], ['KTM RC200', 1.2], ['Royal Enfield Thunde
r 350', 1.15], ['KTM 390 Duke ', 1.15], ['Mahindra Mojo XT300', 1.15], ['Royal Enf
ield Classic 350', 1.15], ['Royal Enfield Classic 350', 1.11], ['Royal Enfield Cla
ssic 350', 1.1], ['Royal Enfield Thunder 500', 1.1], ['Royal Enfield Classic 350',
1.1], ['Royal Enfield Thunder 500', 1.05], ['Bajaj Pulsar RS200', 1.05], ['Royal E
nfield Thunder 350', 1.05], ['Royal Enfield Bullet 350', 1.05], ['Royal Enfield Cl
assic 350', 1.0], ['Royal Enfield Classic 500', 0.95], ['Royal Enfield Classic 50
0', 0.9], ['Bajaj Avenger 220', 0.9], ['Bajaj Avenger 150', 0.75], ['Honda CB Horn
et 160R', 0.8], ['Yamaha FZ S V 2.0', 0.78], ['Honda CB Hornet 160R', 0.75], ['Yam
aha FZ 16', 0.75], ['Bajaj Avenger 220', 0.75], ['Bajaj Avenger 220', 0.72], ['TVS
Apache RTR 160', 0.65], ['Bajaj Pulsar 150', 0.65], ['Honda CBR 150', 0.65], ['Her
o Extreme', 0.65], ['Honda CB Hornet 160R', 0.6], ['Bajaj Avenger 220 dtsi', 0.6],
['Honda CBR 150', 0.6], ['Bajaj Avenger 150 street', 0.6], ['Yamaha FZ  v 2.0', 0.
6], ['Yamaha FZ  v 2.0', 0.6], ['Bajaj Pulsar  NS 200', 0.6], ['TVS Apache RTR 16
0', 0.6], ['Hero Extreme', 0.55], ['Yamaha FZ S V 2.0', 0.55], ['Bajaj Pulsar 220
F', 0.52], ['Bajaj Pulsar 220 F', 0.51], ['TVS Apache RTR 180', 0.5], ['Hero Passi
on X pro', 0.5], ['Bajaj Pulsar NS 200', 0.5], ['Bajaj Pulsar NS 200', 0.5], ['Yam
aha Fazer ', 0.5], ['Honda Activa 4G', 0.48], ['TVS Sport ', 0.48], ['Yamaha FZ S
V 2.0', 0.48], ['Honda Dream Yuga ', 0.48], ['Honda Activa 4G', 0.45], ['Bajaj Ave
nger Street 220', 0.45], ['TVS Apache RTR 180', 0.45], ['Bajaj Pulsar NS 200', 0.4
5], ['Bajaj Avenger 220 dtsi', 0.45], ['Hero Splender iSmart', 0.45], ['Activa 3
g', 0.45], ['Hero Passion Pro', 0.45], ['TVS Apache RTR 160', 0.42], ['Honda CB Tr
igger', 0.42], ['Hero Splender iSmart', 0.4], ['Yamaha FZ S ', 0.4], ['Hero Passio
n Pro', 0.4], ['Bajaj Pulsar 135 LS', 0.4], ['Activa 4g', 0.4], ['Honda CB Unicor
n', 0.38], ['Hero Honda CBZ extreme', 0.38], ['Honda Karizma', 0.35], ['Honda Acti
va 125', 0.35], ['TVS Jupyter', 0.35], ['Honda Karizma', 0.31], ['Hero Honda Passi
on Pro', 0.3], ['Hero Splender Plus', 0.3], ['Honda CB Shine', 0.3], ['Bajaj Disco
ver 100', 0.27], ['Bajaj Pulsar 150', 0.25], ['Suzuki Access 125', 0.25], ['TVS We
go', 0.25], ['Honda CB twister', 0.25], ['Hero Glamour', 0.25], ['Hero Super Splen
dor', 0.2], ['Bajaj Pulsar 150', 0.2], ['Bajaj Discover 125', 0.2], ['Hero Hunk',
0.2], ['Hero  Ignitor Disc', 0.2], ['Hero  CBZ Xtreme', 0.2], ['Bajaj  ct 100', 0.
18], ['Activa 3g', 0.17], ['Honda CB twister', 0.16], ['Bajaj Discover 125', 0.1
5], ['Honda CB Shine', 0.12], ['Bajaj Pulsar 150', 0.1], ['i20', 3.25], ['grand i1
0', 4.4], ['i10', 2.95], ['eon', 2.75], ['grand i10', 5.25], ['xcent', 5.75], ['gr
and i10', 5.15], ['i20', 7.9], ['grand i10', 4.85], ['i10', 3.1], ['elantra', 11.7
5], ['creta', 11.25], ['i20', 2.9], ['grand i10', 5.25], ['verna', 4.5], ['eon',
2.9], ['eon', 3.15], ['verna', 6.45], ['verna', 4.5], ['eon', 3.5], ['i20', 4.5],
```

```
['i20', 6.0], ['verna', 8.25], ['verna', 5.11], ['i10', 2.7], ['grand i10', 5.25],
['i10', 2.55], ['verna', 4.95], ['i20', 3.1], ['verna', 6.15], ['verna', 9.25],
['elantra', 11.45], ['grand i10', 3.9], ['grand i10', 5.5], ['verna', 9.1], ['eo
n', 3.1], ['creta', 11.25], ['verna', 4.8], ['eon', 2.0], ['verna', 5.35], ['xcen
t', 4.75], ['xcent', 4.4], ['i20', 6.25], ['verna', 5.95], ['verna', 5.2], ['i20',
3.75], ['verna', 5.95], ['i10', 4.0], ['i20', 5.25], ['creta', 12.9], ['city', 5.
0], ['brio', 5.4], ['city', 7.2], ['city', 5.25], ['brio', 3.0], ['city', 10.25],
['city', 8.5], ['city', 8.4], ['amaze', 3.9], ['city', 9.15], ['brio', 5.5], ['ama
ze', 4.0], ['jazz', 6.6], ['amaze', 4.0], ['jazz', 6.5], ['amaze', 3.65], ['city',
8.35], ['brio', 4.8], ['city', 6.7], ['city', 4.1], ['city', 3.0], ['city', 7.5],
['jazz', 2.25], ['brio', 5.3], ['city', 10.9], ['city', 8.65], ['city', 9.7], ['ja
zz', 6.0], ['city', 6.25], ['brio', 5.25], ['city', 2.1], ['city', 8.25], ['city',
8.99], ['brio', 3.5], ['jazz', 7.4], ['jazz', 5.65], ['amaze', 5.75], ['city', 8.
4], ['city', 10.11], ['amaze', 4.5], ['brio', 5.4], ['jazz', 6.4], ['city', 3.25],
['amaze', 3.75], ['city', 8.55], ['city', 9.5], ['brio', 4.0], ['city', 3.35], ['c
ity', 11.5], ['brio', 5.3]]
dictionaries: {'ritz': 2.65, 'sx4': 1.95, 'ciaz': 7.75, 'wagon r': 1.05, 'swift':
4.95, 'vitara brezza': 9.25, 's cross': 6.5, 'alto 800': 2.85, 'ertiga': 5.8, 'dzi
re': 5.5, 'alto k10': 2.55, 'ignis': 4.9, '800': 0.35, 'baleno': 5.85, 'omni': 1.2
5, 'fortuner': 9.65, 'innova': 20.75, 'corolla altis': 7.05, 'etios cross': 4.9,
'etios g': 4.75, 'etios liva': 3.45, 'corolla': 1.5, 'etios gd': 4.75, 'camry': 2.
5, 'land cruiser': 35.0, 'Royal Enfield Thunder 500': 1.05, 'UM Renegade Mojave':
1.7, 'KTM RC200': 1.2, 'Bajaj Dominar 400': 1.45, 'Royal Enfield Classic 350': 1.
0, 'KTM RC390': 1.35, 'Hyosung GT250R': 1.35, 'Royal Enfield Thunder 350': 1.05,
'KTM 390 Duke ': 1.15, 'Mahindra Mojo XT300': 1.15, 'Bajaj Pulsar RS200': 1.05, 'R
oyal Enfield Bullet 350': 1.05, 'Royal Enfield Classic 500': 0.9, 'Bajaj Avenger 2
20': 0.72, 'Bajaj Avenger 150': 0.75, 'Honda CB Hornet 160R': 0.6, 'Yamaha FZ S V
2.0': 0.48, 'Yamaha FZ 16': 0.75, 'TVS Apache RTR 160': 0.42, 'Bajaj Pulsar 150':
0.1, 'Honda CBR 150': 0.6, 'Hero Extreme': 0.55, 'Bajaj Avenger 220 dtsi': 0.45,
'Bajaj Avenger 150 street': 0.6, 'Yamaha FZ  v 2.0': 0.6, 'Bajaj Pulsar  NS 200':
0.6, 'Bajaj Pulsar 220 F': 0.51, 'TVS Apache RTR 180': 0.45, 'Hero Passion X pro':
0.5, 'Bajaj Pulsar NS 200': 0.45, 'Yamaha Fazer ': 0.5, 'Honda Activa 4G': 0.45,
'TVS Sport ': 0.48, 'Honda Dream Yuga ': 0.48, 'Bajaj Avenger Street 220': 0.45,
'Hero Splender iSmart': 0.4, 'Activa 3g': 0.17, 'Hero Passion Pro': 0.4, 'Honda CB
Trigger': 0.42, 'Yamaha FZ S ': 0.4, 'Bajaj Pulsar 135 LS': 0.4, 'Activa 4g': 0.4,
'Honda CB Unicorn': 0.38, 'Hero Honda CBZ extreme': 0.38, 'Honda Karizma': 0.31,
'Honda Activa 125': 0.35, 'TVS Jupyter': 0.35, 'Hero Honda Passion Pro': 0.3, 'Her
o Splender Plus': 0.3, 'Honda CB Shine': 0.12, 'Bajaj Discover 100': 0.27, 'Suzuki
Access 125': 0.25, 'TVS Wego': 0.25, 'Honda CB twister': 0.16, 'Hero Glamour': 0.2
5, 'Hero Super Splendor': 0.2, 'Bajaj Discover 125': 0.15, 'Hero Hunk': 0.2, 'Hero
Ignitor Disc': 0.2, 'Hero  CBZ Xtreme': 0.2, 'Bajaj  ct 100': 0.18, 'i20': 5.25,
'grand i10': 5.5, 'i10': 4.0, 'eon': 2.0, 'xcent': 4.4, 'elantra': 11.45, 'creta':
12.9, 'verna': 5.95, 'city': 11.5, 'brio': 5.3, 'amaze': 3.75, 'jazz': 6.4}
```

In [ ]:

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

data = pd.read_csv("../input/car_data.csv")

df = data[["Car_Name", "Selling_Price"]]

my_list = df.values.tolist()

print(my_list[0:5])

rdd = spark.sparkContext.parallelize(my_list)
```

```
rdd_df = rdd.collect()

#Converting to a set
rdd_set = set(df)

print(rdd_set)

spark.stop()
```

```
[['ritz', 3.35], ['sx4', 4.75], ['ciaz', 7.25], ['wagon r', 2.85], ['swift', 4.6]]
{'Selling_Price', 'Car_Name'}
```

Getting partitions used by resource

- ### getNumPartitions() This a RDD function which returns a number of partitions our dataset split into.

you can set the partitions manually inside the parallelize() method

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark sessiona
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

data = pd.read_csv("../input/car_data.csv")

df = data["Selling_Price"]

start_time = time.time()

rdd_df = spark.sparkContext.parallelize(df)

print("initial partition count:"+str(rdd_df.getNumPartitions()))

print(rdd_df.collect())

end_time = time.time()

execution_time = end_time - start_time
print(f"Execution time: {execution_time} seconds")

spark.stop()
```

```
initial partition count:16
[3.35, 4.75, 7.25, 2.85, 4.6, 9.25, 6.75, 6.5, 8.75, 7.45, 2.85, 6.85, 7.5, 6.1,
2.25, 7.75, 7.25, 7.75, 3.25, 2.65, 2.85, 4.9, 4.4, 2.5, 2.9, 3.0, 4.15, 6.0, 1.9
5, 7.45, 3.1, 2.35, 4.95, 6.0, 5.5, 2.95, 4.65, 0.35, 3.0, 2.25, 5.85, 2.55, 1.95,
5.5, 1.25, 7.5, 2.65, 1.05, 5.8, 7.75, 14.9, 23.0, 18.0, 16.0, 2.75, 3.6, 4.5, 4.7
5, 4.1, 19.99, 6.95, 4.5, 18.75, 23.5, 33.0, 4.75, 19.75, 9.25, 4.35, 14.25, 3.95,
4.5, 7.45, 2.65, 4.9, 3.95, 5.5, 1.5, 5.25, 14.5, 14.73, 4.75, 23.0, 12.5, 3.49,
2.5, 35.0, 5.9, 3.45, 4.75, 3.8, 11.25, 3.51, 23.0, 4.0, 5.85, 20.75, 17.0, 7.05,
9.65, 1.75, 1.7, 1.65, 1.45, 1.35, 1.35, 1.35, 1.25, 1.2, 1.2, 1.2, 1.15, 1.15, 1.
15, 1.15, 1.11, 1.1, 1.1, 1.1, 1.05, 1.05, 1.05, 1.05, 1.0, 0.95, 0.9, 0.9, 0.75,
0.8, 0.78, 0.75, 0.75, 0.75, 0.72, 0.65, 0.65, 0.65, 0.65, 0.6, 0.6, 0.6, 0.6, 0.
6, 0.6, 0.6, 0.6, 0.55, 0.55, 0.52, 0.51, 0.5, 0.5, 0.5, 0.5, 0.5, 0.48, 0.48, 0.4
8, 0.48, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.42, 0.42, 0.4, 0.4, 0.
4, 0.4, 0.4, 0.38, 0.38, 0.35, 0.35, 0.35, 0.31, 0.3, 0.3, 0.3, 0.27, 0.25, 0.25,
0.25, 0.25, 0.25, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.18, 0.17, 0.16, 0.15, 0.12, 0.1,
3.25, 4.4, 2.95, 2.75, 5.25, 5.75, 5.15, 7.9, 4.85, 3.1, 11.75, 11.25, 2.9, 5.25,
4.5, 2.9, 3.15, 6.45, 4.5, 3.5, 4.5, 6.0, 8.25, 5.11, 2.7, 5.25, 2.55, 4.95, 3.1,
6.15, 9.25, 11.45, 3.9, 5.5, 9.1, 3.1, 11.25, 4.8, 2.0, 5.35, 4.75, 4.4, 6.25, 5.9
5, 5.2, 3.75, 5.95, 4.0, 5.25, 12.9, 5.0, 5.4, 7.2, 5.25, 3.0, 10.25, 8.5, 8.4, 3.
9, 9.15, 5.5, 4.0, 6.6, 4.0, 6.5, 3.65, 8.35, 4.8, 6.7, 4.1, 3.0, 7.5, 2.25, 5.3,
10.9, 8.65, 9.7, 6.0, 6.25, 5.25, 2.1, 8.25, 8.99, 3.5, 7.4, 5.65, 5.75, 8.4, 10.1
1, 4.5, 5.4, 6.4, 3.25, 3.75, 8.55, 9.5, 4.0, 3.35, 11.5, 5.3]
Execution time: 0.040078163146972656 seconds
```

In [ ]:

```python
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

data = pd.read_csv("../input/car_data.csv")

df = data["Selling_Price"]

start_time = time.time()

rdd_gnp = spark.sparkContext.parallelize(df,20)

print("partition count is:"+str(rdd_gnp.getNumPartitions()))

print(rdd_gnp.collect())

end_time = time.time()

execution_time = end_time - start_time

print(f"Execution time: {execution_time} seconds")

spark.stop()
```

```
partition count is:20
[3.35, 4.75, 7.25, 2.85, 4.6, 9.25, 6.75, 6.5, 8.75, 7.45, 2.85, 6.85, 7.5, 6.1,
2.25, 7.75, 7.25, 7.75, 3.25, 2.65, 2.85, 4.9, 4.4, 2.5, 2.9, 3.0, 4.15, 6.0, 1.9
5, 7.45, 3.1, 2.35, 4.95, 6.0, 5.5, 2.95, 4.65, 0.35, 3.0, 2.25, 5.85, 2.55, 1.95,
5.5, 1.25, 7.5, 2.65, 1.05, 5.8, 7.75, 14.9, 23.0, 18.0, 16.0, 2.75, 3.6, 4.5, 4.7
5, 4.1, 19.99, 6.95, 4.5, 18.75, 23.5, 33.0, 4.75, 19.75, 9.25, 4.35, 14.25, 3.95,
4.5, 7.45, 2.65, 4.9, 3.95, 5.5, 1.5, 5.25, 14.5, 14.73, 4.75, 23.0, 12.5, 3.49,
2.5, 35.0, 5.9, 3.45, 4.75, 3.8, 11.25, 3.51, 23.0, 4.0, 5.85, 20.75, 17.0, 7.05,
9.65, 1.75, 1.7, 1.65, 1.45, 1.35, 1.35, 1.35, 1.25, 1.2, 1.2, 1.2, 1.15, 1.15, 1.
15, 1.15, 1.11, 1.1, 1.1, 1.1, 1.05, 1.05, 1.05, 1.05, 1.0, 0.95, 0.9, 0.9, 0.75,
0.8, 0.78, 0.75, 0.75, 0.75, 0.72, 0.65, 0.65, 0.65, 0.65, 0.6, 0.6, 0.6, 0.6, 0.
6, 0.6, 0.6, 0.6, 0.55, 0.55, 0.52, 0.51, 0.5, 0.5, 0.5, 0.5, 0.5, 0.48, 0.48, 0.4
8, 0.48, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.42, 0.42, 0.4, 0.4, 0.
4, 0.4, 0.4, 0.38, 0.38, 0.35, 0.35, 0.35, 0.31, 0.3, 0.3, 0.3, 0.27, 0.25, 0.25,
0.25, 0.25, 0.25, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.18, 0.17, 0.16, 0.15, 0.12, 0.1,
3.25, 4.4, 2.95, 2.75, 5.25, 5.75, 5.15, 7.9, 4.85, 3.1, 11.75, 11.25, 2.9, 5.25,
4.5, 2.9, 3.15, 6.45, 4.5, 3.5, 4.5, 6.0, 8.25, 5.11, 2.7, 5.25, 2.55, 4.95, 3.1,
6.15, 9.25, 11.45, 3.9, 5.5, 9.1, 3.1, 11.25, 4.8, 2.0, 5.35, 4.75, 4.4, 6.25, 5.9
5, 5.2, 3.75, 5.95, 4.0, 5.25, 12.9, 5.0, 5.4, 7.2, 5.25, 3.0, 10.25, 8.5, 8.4, 3.
9, 9.15, 5.5, 4.0, 6.6, 4.0, 6.5, 3.65, 8.35, 4.8, 6.7, 4.1, 3.0, 7.5, 2.25, 5.3,
10.9, 8.65, 9.7, 6.0, 6.25, 5.25, 2.1, 8.25, 8.99, 3.5, 7.4, 5.65, 5.75, 8.4, 10.1
1, 4.5, 5.4, 6.4, 3.25, 3.75, 8.55, 9.5, 4.0, 3.35, 11.5, 5.3]
Execution time: 0.039556264877319336 seconds
```

In [ ]:

```python
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

data = pd.read_csv("../input/car_data.csv")

df = data["Selling_Price"]

start_time = time.time()

rdd_gnp = spark.sparkContext.parallelize(df,5)

print("partition count is:"+str(rdd_gnp.getNumPartitions()))

print(rdd_gnp.collect())

end_time = time.time()

execution_time = end_time - start_time

print(f"Execution time: {execution_time} seconds")

spark.stop()
```

```
partition count is:5
[3.35, 4.75, 7.25, 2.85, 4.6, 9.25, 6.75, 6.5, 8.75, 7.45, 2.85, 6.85, 7.5, 6.1,
2.25, 7.75, 7.25, 7.75, 3.25, 2.65, 2.85, 4.9, 4.4, 2.5, 2.9, 3.0, 4.15, 6.0, 1.9
5, 7.45, 3.1, 2.35, 4.95, 6.0, 5.5, 2.95, 4.65, 0.35, 3.0, 2.25, 5.85, 2.55, 1.95,
5.5, 1.25, 7.5, 2.65, 1.05, 5.8, 7.75, 14.9, 23.0, 18.0, 16.0, 2.75, 3.6, 4.5, 4.7
5, 4.1, 19.99, 6.95, 4.5, 18.75, 23.5, 33.0, 4.75, 19.75, 9.25, 4.35, 14.25, 3.95,
4.5, 7.45, 2.65, 4.9, 3.95, 5.5, 1.5, 5.25, 14.5, 14.73, 4.75, 23.0, 12.5, 3.49,
2.5, 35.0, 5.9, 3.45, 4.75, 3.8, 11.25, 3.51, 23.0, 4.0, 5.85, 20.75, 17.0, 7.05,
9.65, 1.75, 1.7, 1.65, 1.45, 1.35, 1.35, 1.35, 1.25, 1.2, 1.2, 1.2, 1.15, 1.15, 1.
15, 1.15, 1.11, 1.1, 1.1, 1.1, 1.05, 1.05, 1.05, 1.05, 1.0, 0.95, 0.9, 0.9, 0.75,
0.8, 0.78, 0.75, 0.75, 0.75, 0.72, 0.65, 0.65, 0.65, 0.65, 0.6, 0.6, 0.6, 0.6, 0.
6, 0.6, 0.6, 0.6, 0.55, 0.55, 0.52, 0.51, 0.5, 0.5, 0.5, 0.5, 0.5, 0.48, 0.48, 0.4
8, 0.48, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.45, 0.42, 0.42, 0.4, 0.4, 0.
4, 0.4, 0.4, 0.38, 0.38, 0.35, 0.35, 0.35, 0.31, 0.3, 0.3, 0.3, 0.27, 0.25, 0.25,
0.25, 0.25, 0.25, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.18, 0.17, 0.16, 0.15, 0.12, 0.1,
3.25, 4.4, 2.95, 2.75, 5.25, 5.75, 5.15, 7.9, 4.85, 3.1, 11.75, 11.25, 2.9, 5.25,
4.5, 2.9, 3.15, 6.45, 4.5, 3.5, 4.5, 6.0, 8.25, 5.11, 2.7, 5.25, 2.55, 4.95, 3.1,
6.15, 9.25, 11.45, 3.9, 5.5, 9.1, 3.1, 11.25, 4.8, 2.0, 5.35, 4.75, 4.4, 6.25, 5.9
5, 5.2, 3.75, 5.95, 4.0, 5.25, 12.9, 5.0, 5.4, 7.2, 5.25, 3.0, 10.25, 8.5, 8.4, 3.
9, 9.15, 5.5, 4.0, 6.6, 4.0, 6.5, 3.65, 8.35, 4.8, 6.7, 4.1, 3.0, 7.5, 2.25, 5.3,
10.9, 8.65, 9.7, 6.0, 6.25, 5.25, 2.1, 8.25, 8.99, 3.5, 7.4, 5.65, 5.75, 8.4, 10.1
1, 4.5, 5.4, 6.4, 3.25, 3.75, 8.55, 9.5, 4.0, 3.35, 11.5, 5.3]
Execution time: 0.04695725440979004 seconds
```

RDD Repartition() vs Coalesce()

In [ ]:

```python
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

# Create spark session with local cores
rdd = spark.sparkContext.parallelize(range(0,1000))
print("From local cores : "+str(rdd.getNumPartitions()))

# Use parallelize with 6 partitions
rdd1 = spark.sparkContext.parallelize(range(0,25), 6)
print("parallelize : "+str(rdd1.getNumPartitions()))

spark.stop()
```

```
From local cores : 16
parallelize : 6
```

- Partition 1 : 0 1 2
- Partition 2 : 3 4 5
- Partition 3 : 6 7 8 9
- Partition 4 : 10 11 12
- Partition 5 : 13 14 15
- Partition 6 : 16 17 18 19

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

data = pd.read_csv("../input/car_data.csv")

rddFromFile = spark.sparkContext.parallelize(data, 30)
print("Partitions : "+str(rddFromFile.getNumPartitions()))

spark.stop()
```

```
Partitions : 30
```

RDD repartition()

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

data = pd.read_csv("../input/car_data.csv")

rddFromFile = spark.sparkContext.parallelize(data, 10)
print("TextFile : "+str(rddFromFile.getNumPartitions()))

rdd_rePartition = rddFromFile.repartition(6)
print("TextFile : "+str(rdd_rePartition.getNumPartitions()))

rdd_rePartition1 = rddFromFile.repartition(20)
print("TextFile : "+str(rdd_rePartition1.getNumPartitions()))

spark.stop()
```

```
TextFile : 10
TextFile : 6
TextFile : 20
```

RDD coalesce()

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
```

```
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

data = pd.read_csv("../input/car_data.csv")

rdd = spark.sparkContext.parallelize(data)

print("initial count:", rdd.getNumPartitions())

rddFromFile = spark.sparkContext.parallelize(data, 10)
print("TextFile : "+str(rddFromFile.getNumPartitions()))

rdd_coalesce = rddFromFile.coalesce(4)
print("TextFile : "+str(rdd_coalesce.getNumPartitions()))

rdd_coalesce1 = rddFromFile.coalesce(11)
print("TextFile : "+str(rdd_coalesce1.getNumPartitions()))

spark.stop()
```

```
initial count: 16
TextFile : 10
TextFile : 4
TextFile : 10
```

In [ ]:

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()

data = spark.read.csv("../input/car_data.csv", header=True, inferSchema=True)

print("Initial Count:", data.rdd.getNumPartitions())

rdd = data.rdd.repartition(6)

print("After Partioning : "+str(rdd.getNumPartitions()))

data.show()

spark.stop()
```

```
Initial Count: 1
After Partioning : 6
+-------------+----+------------+------------+---------+---------+-----------+-
-----------+-----+
|     Car_Name|Year|Selling_Price|Present_Price|Kms_Driven|Fuel_Type|Seller_Type|T
ransmission|Owner|
+-------------+----+------------+------------+---------+---------+-----------+-
-----------+-----+
|         ritz|2014|         3.35|         5.59|     27000|   Petrol|     Dealer|
Manual|    0|
|          sx4|2013|         4.75|         9.54|     43000|   Diesel|     Dealer|
Manual|    0|
|         ciaz|2017|         7.25|         9.85|      6900|   Petrol|     Dealer|
Manual|    0|
|      wagon r|2011|         2.85|         4.15|      5200|   Petrol|     Dealer|
Manual|    0|
|        swift|2014|          4.6|         6.87|     42450|   Diesel|     Dealer|
Manual|    0|
|vitara brezza|2018|         9.25|         9.83|      2071|   Diesel|     Dealer|
Manual|    0|
|         ciaz|2015|         6.75|         8.12|     18796|   Petrol|     Dealer|
Manual|    0|
|      s cross|2015|          6.5|         8.61|     33429|   Diesel|     Dealer|
Manual|    0|
|         ciaz|2016|         8.75|         8.89|     20273|   Diesel|     Dealer|
Manual|    0|
|         ciaz|2015|         7.45|         8.92|     42367|   Diesel|     Dealer|
Manual|    0|
|     alto 800|2017|         2.85|          3.6|      2135|   Petrol|     Dealer|
Manual|    0|
|         ciaz|2015|         6.85|        10.38|     51000|   Diesel|     Dealer|
Manual|    0|
|         ciaz|2015|          7.5|         9.94|     15000|   Petrol|     Dealer|
Automatic|    0|
|       ertiga|2015|          6.1|         7.71|     26000|   Petrol|     Dealer|
Manual|    0|
|        dzire|2009|         2.25|         7.21|     77427|   Petrol|     Dealer|
Manual|    0|
|       ertiga|2016|         7.75|        10.79|     43000|   Diesel|     Dealer|
Manual|    0|
|       ertiga|2015|         7.25|        10.79|     41678|   Diesel|     Dealer|
Manual|    0|
|       ertiga|2016|         7.75|        10.79|     43000|   Diesel|     Dealer|
Manual|    0|
|      wagon r|2015|         3.25|         5.09|     35500|      CNG|     Dealer|
Manual|    0|
|          sx4|2010|         2.65|         7.98|     41442|   Petrol|     Dealer|
Manual|    0|
+-------------+----+------------+------------+---------+---------+-----------+-
-----------+-----+
only showing top 20 rows
```

RDD Operations

- RDD has two internal 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 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.

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.
- In Apache Spark's Resilient Distributed Dataset (RDD) API, the flatMap() transformation is used to apply a function to each element of the RDD and then flatten the resulting sequence of sequences into a single RDD. This is particularly useful when you want to transform each element into multiple elements and then merge those elements into a single RDD, essentially "flattening" the structure.

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()


rdd = spark.sparkContext.textFile("../input/test.txt")

rdd2 = rdd.flatMap(lambda x: x.split(" "))

spark.stop()
```

map

- map() transformation is used to 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.

- Key points
- Both map() & flatMap() returns Dataset (DataFrame=Dataset[Row]).
- Both these transformations are narrow meaning they do not result in Spark Data Shuffle.
- flatMap() results in redundant data on some columns.
- One of the use cases of flatMap() is to flatten column which contains arrays, list, or any nested collection(one cell with one value).
- map() always return the same size/records as in input DataFrame whereas flatMap() returns many records for each record (one-many).

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()


rdd = spark.sparkContext.textFile("../input/test.txt")

rdd3 = rdd.map(lambda x: (x,1))

spark.stop()
```

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.

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import time

# Create a Spark session
spark = SparkSession.builder \
    .appName("My Application") \
    .master("local[*]") \
    .getOrCreate()


rdd = spark.sparkContext.textFile("../input/test.txt")

rdd4 = rdd.reduceByKey(lambda a,b: a+b)

spark.stop()
```

sortByKey

- sortByKey() transformation is used to sort RDD elements on key. In our example, first, we convert RDD[(String,Int]) to RDD[(Int, String]) using map transformation and apply sortByKey which ideally does sort on an integer value. And finally, foreach with println statements returns all words in RDD and their count as key-value pair

In [ ]:
```
rdd = spark.sparkContext.textFile("../input/test.txt")
```

filter

- filter() transformation is used to filter the records in an RDD. In our example we are filtering all words starts with “a”

In [ ]:
```

```

In [ ]:

In [ ]:

In [ ]: