Explanation of all PySpark RDD, DataFrame and SQL examples present on this project are available at Apache PySpark Tutorial, All these examples are coded in Python language and tested in our development environment.

Table of Contents (Spark Examples in Python)

===
What is SparkSession

SparkSession was introduced in version 2.0,
It is an entry point to underlying PySpark functionality in order to
programmatically create PySpark RDD,DataFrame.
It’s object spark is default available in pyspark-shell
and it can be created programmatically using SparkSession.

Spark Session:

```
pyspark.import sparksession
sparksession.builder
```

SparkSession also includes all the APIs available in different contexts:

SparkContext
SQLContext
StreamingContext
HiveContext.

How many SparkSessions can you create in a PySpark application?

```
sparksession()
SparkSession.builder()
SparkSession.newSession()

# Usage of spark object in PySpark shell
>>>spark.version
3.1.2

# Create SparkSession from builder?

import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[1]") \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()
```

Note: SparkSession object spark is by default available in the PySpark shell.

```
# Create new SparkSession
spark2 = SparkSession.newSession
print(spark2)

# Get Existing SparkSession
spark3 = SparkSession.builder.getOrCreate
print(spark3)

# Usage of config()
spark = SparkSession.builder \
      .master("local[1]") \
      .appName("SparkByExamples.com") \
      .config("spark.some.config.option", "config-value") \
      .getOrCreate()

# Enabling Hive to use in Spark
spark = SparkSession.builder \
      .master("local[1]") \
      .appName("SparkByExamples.com") \
      .config("spark.sql.warehouse.dir", "<path>/spark-warehouse") \
      .enableHiveSupport() \
      .getOrCreate()

# Set Config
spark.conf.set("spark.executor.memory", "5g")

# Get a Spark Config
partitions = spark.conf.get("spark.sql.shuffle.partitions")
print(partitions)

# Create DataFrame
df = spark.createDataFrame([("Scala", 25000), ("Spark", 35000), ("PHP", 21000)])
df.show()

# Output
#+-----+-----+
#|   _1|   _2|
#+-----+-----+
#|Scala|25000|
#|Spark|35000|
#|  PHP|21000|
#+-----+-----+

# Spark SQL
df.createOrReplaceTempView("sample_table") or we used createGlobalTempView()
df2 = spark.sql("SELECT _1,_2 FROM sample_table")
df2.show()

# Create Hive table & query it.
spark.table("sample_table").write.saveAsTable("sample_hive_table")
df3 = spark.sql("SELECT _1,_2 FROM sample_hive_table")
df3.show()
```

Working with Catalogs

To get the catalog metadata, PySpark Session exposes catalog variable.
spark.catalog.listDatabases()
spark.catalog.listTables()
returns the DataSet.

```
# Get metadata from the Catalog
# List databases
dbs = spark.catalog.listDatabases()
print(dbs)

# Output
#[Database(name='default', description='default database',
#locationUri='file:/Users/admin/.spyder-py3/spark-warehouse')]

# List Tables
tbls = spark.catalog.listTables()
print(tbls)

#Output
#[Table(name='sample_hive_table', database='default',
description=None, table, isTemporary=True)]
```

SparkSession Commonly Used Methods:

version()

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

getActiveSession() – returns an active Spark session.

read()
readStream()
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

What is PySpark Accumulator?

Accumulators are write-only and initialize once variables where only tasks that are running on workers are allowed
to update and updates from the workers get propagated automatically to the driver program.
Accumulator variable using the value property.

How to create Accumulator variable in PySpark?

sparkContext.accumulator() > define accumulator variables.

add() > add/update a value in accumulator

value property on the accumulator variable is used to retrieve the value from the accumulator.
We can create Accumulators in PySpark for primitive types int and float.

PySpark accumulator examples:

```
import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName("accumulator").getOrCreate()

accum=spark.sparkContext.accumulator(0)
rdd=spark.sparkContext.parallelize([1,2,3,4,5])
rdd.foreach(lambda x:accum.add(x))
print(accum.value)

accuSum=spark.sparkContext.accumulator(0)
def countFun(x):
    global accuSum
    accuSum+=x
rdd.foreach(countFun)
print(accuSum.value)

accumCount=spark.sparkContext.accumulator(0)
rdd2=spark.sparkContext.parallelize([1,2,3,4,5])
rdd2.foreach(lambda x:accumCount.add(1))
print(accumCount.value)
```

Complete Example of PySpark RDD repartition and coalesce:

```
# Complete Example
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com') \
        .master("local[5]").getOrCreate()

df = spark.range(0,20)
print(df.rdd.getNumPartitions())

spark.conf.set("spark.sql.shuffle.partitions", "500")

rdd = spark.sparkContext.parallelize(range(0,20))
print("From local[5]"+str(rdd.getNumPartitions()))

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

"""rddFromFile = spark.sparkContext.textFile("src/main/resources/test.txt",10)
print("TextFile : "+str(rddFromFile.getNumPartitions())) """

rdd1.saveAsTextFile("c://tmp/partition2")
```

```
rdd2 = rdd1.repartition(4)
print("Repartition size : "+str(rdd2.getNumPartitions()))
rdd2.saveAsTextFile("c://tmp/re-partition2")

rdd3 = rdd1.coalesce(4)
print("Repartition size : "+str(rdd3.getNumPartitions()))
rdd3.saveAsTextFile("c:/tmp/coalesce2")

# DataFrame example
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com') \
        .master("local[5]").getOrCreate()

df=spark.range(0,20)
print(df.rdd.getNumPartitions())

df.write.mode("overwrite").csv("c:/tmp/partition.csv")

# DataFrame repartition
df2 = df.repartition(6)
print(df2.rdd.getNumPartitions())

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

# DataFrame coalesce
df3 = df.coalesce(2)
print(df3.rdd.getNumPartitions())

# Output:
Partition 1 : 0 1 2 3 8 9 10 11
Partition 2 : 4 5 6 7 12 13 14 15 16 17 18 19
-------------------------------------------------------------------------
spark.sql.shuffle.partitions

Union,groupBy and joins etc.

# Default shuffle partition count

df4 = df.groupBy("id").count()
print(df4.rdd.getNumPartitions())
-----------------------------------------------------------------
Broad cast Variables:


import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

states = {"NY":"New York", "CA":"California", "FL":"Florida"}
broadcastStates = spark.sparkContext.broadcast(states)

data = [("James","Smith","USA","CA"),
    ("Michael","Rose","USA","NY"),
    ("Robert","Williams","USA","CA"),
    ("Maria","Jones","USA","FL")
```

```
  ]

columns = ["firstname","lastname","country","state"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

def state_convert(code):
    return broadcastStates.value[code]

result = df.rdd.map(lambda x:
(x[0],x[1],x[2],state_convert(x[3]))).toDF(columns)
result.show(truncate=False)
```

Empty data parallelize etc:

```
emptyRDD = sparkContext.emptyRDD()
emptyRDD2 = rdd=sparkContext.parallelize([])

print("is Empty RDD : "+str(emptyRDD2.isEmpty()))
```

What is RDD (Resilient Distributed Dataset)?

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.

map,filter,persist,groupByKey,join()

Note: RDD’s can have a name and unique identifier (id)

PySpark RDD Benefits:

>In-Memory Processing
>immutable
>fault tolerant
>Lazy Evolution
>Partitioning

Create RDD:

```
from pyspark.sql import SparkSession
spark:SparkSession = SparkSession.builder()
      .master("local[1]")
      .appName("SparkByExamples.com")
      .getOrCreate()
```

master() – If you are running it on the cluster you need to use your master name or
mesos depends on your cluster setup

appName() – Used to set your application name.

getOrCreate() – This returns a SparkSession object if already exists,
and creates a new one if not exist.

```
python
1                          1,2  partition 1
2---->   RDD CREATION -->
3                          3,4  partition 2
4
```

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

we mostly create RDD by using external storage systems like HDFS, S3, HBase e.t.c

sparkContext.txtfile:

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

sparkContext.Wholetextfile:

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

sparkContext.empty RDD:

```
# Creates empty RDD with no partition
rdd = spark.sparkContext.emptyRDD
# rddString = spark.sparkContext.emptyRDD[String]


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

RDD Parallelize:

parallelize() or
textFile() or
wholeTextFiles()
methods of SparkContxt to initiate RDD.

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

```
print("initial partition count:"+str(rdd.getNumPartitions()))
#Outputs: initial partition count:2
```

Set parallelize manually:

```
sparkContext.parallelize([1,2,3,4,56,7,8,9,12,3], 10)
```

Repartition and Coalesce:

```
reparRdd = rdd.repartition(4)
print("re-partition count:"+str(reparRdd.getNumPartitions()))

#Outputs: "re-partition count:4
```

Note: repartition() or coalesce() methods also returns a new RDD.

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 RDD values.

flatMap:

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

map:

```
rdd3 = rdd2.map(lambda x: (x,1))
```

reduceByKey:

```
rdd4 = rdd3.reduceByKey(lambda a,b: a+b)
```

sortByKey:

```
rdd5 = rdd4.map(lambda x: (x[1],x[0])).sortByKey()
#Print rdd5 result to console
print(rdd5.collect())
```

filter:

```
rdd4 = rdd3.filter(lambda x : 'an' in x[1])
print(rdd4.collect())
```

RDD Actions with example:

count() – Returns the number of records in an RDD

```
# Action - count
print("Count : "+str(rdd6.count()))
```

first() – Returns the first record.

```
# Action - first
firstRec = rdd6.first()
print("First Record : "+str(firstRec[0]) + ","+ firstRec[1])
```

max() – Returns max record.

```
# Action - max
datMax = rdd6.max()
print("Max Record : "+str(datMax[0]) + ","+ datMax[1])
```

reduce() – Reduces the records to single, we can use this to count or sum.

```
# Action - reduce
totalWordCount = rdd6.reduce(lambda a,b: (a[0]+b[0],a[1]))
print("dataReduce Record : "+str(totalWordCount[0]))
```

take() – Returns the record specified as an argument.

```
# Action - take
data3 = rdd6.take(3)
for f in data3:
    print("data3 Key:"+ str(f[0]) +", Value:"+f[1])
```

collect() – Returns all data from RDD as an array. Be careful when you use this action when you are working with huge RDD with millions and billions of data as you may run out of memory on the driver.

```
# Action - collect
data = rdd6.collect()
for f in data:
    print("Key:"+ str(f[0]) +", Value:"+f[1])
```

saveAsTextFile() – Using saveAsTestFile action, we can write the RDD to a text file.

```
rdd6.saveAsTextFile("/tmp/wordCount")
```

Types of RDD:

PairRDDFunctions or PairRDD – Pair RDD is a key-value pair This is mostly used

RDD type

ShuffledRDD
DoubleRDD
SequenceFileRDD
HadoopRDD
ParallelCollectionRDD

ShuffledRDD:

repartition()
coalesce()
groupByKey()
reduceByKey()
cogroup() and join() but not countByKey().

RDD Persistant:
cache() and persist() etc.

Advantages of Persisting RDD:

Cost efficient – PySpark computations are very expensive hence reusing the computations are used to save cost.
Time efficient – Reusing the repeated computations saves lots of time.
Execution time – Saves execution time of the job which allows us to perform more jobs on the same cluster.

```
cachedRdd = rdd.cache().
```

RDD Persist method:

PySpark persist() method is used to store the RDD to one of the storage levels
MEMORY_ONLY,
MEMORY_AND_DISK,
MEMORY_ONLY_SER,
MEMORY_AND_DISK_SER,
DISK_ONLY,
MEMORY_ONLY_2,
MEMORY_AND_DISK_2

```
import pyspark
dfPersist = rdd.persist(pyspark.StorageLevel.MEMORY_ONLY)
dfPersist.show(false)
```

Unpersist:remove the data using by itselfs.

```
rddPersist2 = rddPersist.unpersist()
```

==
Table of Contents (Spark Examples in Python):

UDF:USER DEFINED Function:

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, udf
from pyspark.sql.types import StringType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
```

```
columns = ["Seqno","Name"]
data = [("1", "john jones"),
    ("2", "tracey smith"),
    ("3", "amy sanders")]

df = spark.createDataFrame(data=data,schema=columns)

df.show(truncate=False)

def convertCase(str):
    resStr=""
    arr = str.split(" ")
    for x in arr:
       resStr= resStr + x[0:1].upper() + x[1:len(x)] + " "
    return resStr

""" Converting function to UDF """
convertUDF = udf(lambda z: convertCase(z))

df.select(col("Seqno"), \
    convertUDF(col("Name")).alias("Name") ) \
.show(truncate=False)

def upperCase(str):
    return str.upper()

upperCaseUDF = udf(lambda z:upperCase(z),StringType())

df.withColumn("Cureated Name", upperCaseUDF(col("Name"))) \
.show(truncate=False)

""" Using UDF on SQL """
spark.udf.register("convertUDF", convertCase,StringType())
df.createOrReplaceTempView("NAME_TABLE")
spark.sql("select Seqno, convertUDF(Name) as Name from NAME_TABLE") \
     .show(truncate=False)

spark.sql("select Seqno, convertUDF(Name) as Name from NAME_TABLE " + \
          "where Name is not null and convertUDF(Name) like '%John%'") \
     .show(truncate=False)

""" null check """

columns = ["Seqno","Name"]
data = [("1", "john jones"),
    ("2", "tracey smith"),
    ("3", "amy sanders"),
    ('4',None)]

df2 = spark.createDataFrame(data=data,schema=columns)
df2.show(truncate=False)
df2.createOrReplaceTempView("NAME_TABLE2")

spark.udf.register("_nullsafeUDF", lambda str: convertCase(str) if not str is
None else "" , StringType())

spark.sql("select _nullsafeUDF(Name) from NAME_TABLE2") \
     .show(truncate=False)

spark.sql("select Seqno, _nullsafeUDF(Name) as Name from NAME_TABLE2 " + \
          " where Name is not null and _nullsafeUDF(Name) like '%John%'") \
     .show(truncate=False)

--------------------------------------------------------------------------------
```

```
       FlatMap Examples:
       -----------------
       from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = ["Project Gutenberg’s",
        "Alice’s Adventures in Wonderland",
        "Project Gutenberg’s",
        "Adventures in Wonderland",
        "Project Gutenberg’s"]
rdd=spark.sparkContext.parallelize(data)
for element in rdd.collect():
    print(element)

#Flatmap
rdd2=rdd.flatMap(lambda x: x.split(" "))
for element in rdd2.collect():
    print(element)
================================================================================
=======
Map() examples:
---------------

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = ["Project",
"Gutenberg’s",
"Alice’s",
"Adventures",
"in",
"Wonderland",
"Project",
"Gutenberg’s",
"Adventures",
"in",
"Wonderland",
"Project",
"Gutenberg’s"]

rdd=spark.sparkContext.parallelize(data)

rdd2=rdd.map(lambda x: (x,1))
for element in rdd2.collect():
    print(element)

data = [('James','Smith','M',30),
  ('Anna','Rose','F',41),
  ('Robert','Williams','M',62),
]

columns = ["firstname","lastname","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()

rdd2=df.rdd.map(lambda x:
    (x[0]+","+x[1],x[2],x[3]*2)
    )
df2=rdd2.toDF(["name","gender","new_salary"]   )
df2.show()

#Referring Column Names
rdd2=df.rdd.map(lambda x:
    (x["firstname"]+","+x["lastname"],x["gender"],x["salary"]*2)
    )

#Referring Column Names
rdd2=df.rdd.map(lambda x:
    (x.firstname+","+x.lastname,x.gender,x.salary*2)
    )

def func1(x):
    firstName=x.firstname
    lastName=x.lastname
    name=firstName+","+lastName
    gender=x.gender.lower()
    salary=x.salary*2
    return (name,gender,salary)

rdd2=df.rdd.map(lambda x: func1(x))
==========================================================================================
==============
PySpark Aggregate examples:
---------------------------

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import approx_count_distinct,collect_list
from pyspark.sql.functions import collect_set,sum,avg,max,countDistinct,count
from pyspark.sql.functions import first, last, kurtosis, min, mean, skewness
from pyspark.sql.functions import stddev, stddev_samp, stddev_pop, sumDistinct
from pyspark.sql.functions import variance,var_samp,  var_pop

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = [("James", "Sales", 3000),
    ("Michael", "Sales", 4600),
    ("Robert", "Sales", 4100),
    ("Maria", "Finance", 3000),
    ("James", "Sales", 3000),
    ("Scott", "Finance", 3300),
    ("Jen", "Finance", 3900),
    ("Jeff", "Marketing", 3000),
    ("Kumar", "Marketing", 2000),
    ("Saif", "Sales", 4100)
  ]
schema = ["employee_name", "department", "salary"]

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

print("approx_count_distinct: " + \
      str(df.select(approx_count_distinct("salary")).collect()[0][0]))

print("avg: " + str(df.select(avg("salary")).collect()[0][0]))

df.select(collect_list("salary")).show(truncate=False)

df.select(collect_set("salary")).show(truncate=False)

df2 = df.select(countDistinct("department", "salary"))
df2.show(truncate=False)
print("Distinct Count of Department & Salary: "+str(df2.collect()[0][0]))

print("count: "+str(df.select(count("salary")).collect()[0]))
df.select(first("salary")).show(truncate=False)
df.select(last("salary")).show(truncate=False)
df.select(kurtosis("salary")).show(truncate=False)
df.select(max("salary")).show(truncate=False)
df.select(min("salary")).show(truncate=False)
df.select(mean("salary")).show(truncate=False)
df.select(skewness("salary")).show(truncate=False)
df.select(stddev("salary"), stddev_samp("salary"), \
    stddev_pop("salary")).show(truncate=False)
df.select(sum("salary")).show(truncate=False)
df.select(sumDistinct("salary")).show(truncate=False)
df.select(variance("salary"),var_samp("salary"),var_pop("salary")) \
  .show(truncate=False)
================================================================================
======
```

Source Code of Window Functions Example:

```
import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = (("James", "Sales", 3000), \
    ("Michael", "Sales", 4600),  \
    ("Robert", "Sales", 4100),   \
    ("Maria", "Finance", 3000),  \
    ("James", "Sales", 3000),    \
    ("Scott", "Finance", 3300),  \
    ("Jen", "Finance", 3900),    \
    ("Jeff", "Marketing", 3000), \
    ("Kumar", "Marketing", 2000),\
    ("Saif", "Sales", 4100) \
  )

columns= ["employee_name", "department", "salary"]

df = spark.createDataFrame(data = simpleData, schema = columns)

df.printSchema()
df.show(truncate=False)

from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
windowSpec  = Window.partitionBy("department").orderBy("salary")

df.withColumn("row_number",row_number().over(windowSpec)) \
    .show(truncate=False)

from pyspark.sql.functions import rank
df.withColumn("rank",rank().over(windowSpec)) \
    .show()

from pyspark.sql.functions import dense_rank
df.withColumn("dense_rank",dense_rank().over(windowSpec)) \
    .show()

from pyspark.sql.functions import percent_rank
df.withColumn("percent_rank",percent_rank().over(windowSpec)) \
    .show()

from pyspark.sql.functions import ntile
df.withColumn("ntile",ntile(2).over(windowSpec)) \
    .show()

from pyspark.sql.functions import cume_dist
```

```
df.withColumn("cume_dist",cume_dist().over(windowSpec)) \
   .show()

from pyspark.sql.functions import lag
df.withColumn("lag",lag("salary",2).over(windowSpec)) \
      .show()

from pyspark.sql.functions import lead
df.withColumn("lead",lead("salary",2).over(windowSpec)) \
    .show()

windowSpecAgg  = Window.partitionBy("department")
from pyspark.sql.functions import col,avg,sum,min,max,row_number
df.withColumn("row",row_number().over(windowSpec)) \
  .withColumn("avg", avg(col("salary")).over(windowSpecAgg)) \
  .withColumn("sum", sum(col("salary")).over(windowSpecAgg)) \
  .withColumn("min", min(col("salary")).over(windowSpecAgg)) \
  .withColumn("max", max(col("salary")).over(windowSpecAgg)) \
  .where(col("row")==1).select("department","avg","sum","min","max") \
  .show()
  ============================================================================
   PySpark Read CSV Complete Example

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
from pyspark.sql.types import ArrayType, DoubleType, BooleanType
from pyspark.sql.functions import col,array_contains

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

df = spark.read.csv("/tmp/resources/zipcodes.csv")

df.printSchema()

df2 = spark.read.option("header",True) \
     .csv("/tmp/resources/zipcodes.csv")
df2.printSchema()

df3 = spark.read.options(header='True', delimiter=',') \
  .csv("/tmp/resources/zipcodes.csv")
df3.printSchema()

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(/tmp/resources/zipcodes.csv")
df_with_schema.printSchema()

df2.write.option("header",True) \
 .csv("/tmp/spark_output/zipcodes123")
```

==
======
 Complete Example of PySpark read and write Parquet file

```
import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName("parquetFile").getOrCreate()
data =[("James ","","Smith","36636","M",3000),
              ("Michael ","Rose","","40288","M",4000),
              ("Robert ","","Williams","42114","M",4000),
              ("Maria ","Anne","Jones","39192","F",4000),
              ("Jen","Mary","Brown","","F",-1)]
columns=["firstname","middlename","lastname","dob","gender","salary"]
df=spark.createDataFrame(data,columns)
df.write.mode("overwrite").parquet("/tmp/output/people.parquet")
parDF1=spark.read.parquet("/tmp/output/people.parquet")
parDF1.createOrReplaceTempView("parquetTable")
parDF1.printSchema()
parDF1.show(truncate=False)

parkSQL = spark.sql("select * from ParquetTable where salary >= 4000 ")
parkSQL.show(truncate=False)

spark.sql("CREATE TEMPORARY VIEW PERSON USING parquet OPTIONS (path
\"/tmp/output/people.parquet\")")
spark.sql("SELECT * FROM PERSON").show()

df.write.partitionBy("gender","salary").mode("overwrite").parquet("/tmp/output/
people2.parquet")

parDF2=spark.read.parquet("/tmp/output/people2.parquet/gender=M")
parDF2.show(truncate=False)

spark.sql("CREATE TEMPORARY VIEW PERSON2 USING parquet OPTIONS (path
\"/tmp/output/people2.parquet/gender=F\")")
spark.sql("SELECT * FROM PERSON2" ).show()
```
==
================
Real time programming examples:
----
convert-column-python-list.py:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[1]") \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()

data = [("James","Smith","USA","CA"),("Michael","Rose","USA","NY"), \
    ("Robert","Williams","USA","CA"),("Maria","Jones","USA","FL") \
  ]
columns=["firstname","lastname","country","state"]
df=spark.createDataFrame(data=data,schema=columns)
df.show()
print(df.collect())
```

```
states1=df.rdd.map(lambda x: x[3]).collect()
print(states1)
#['CA', 'NY', 'CA', 'FL']
from collections import OrderedDict
res = list(OrderedDict.fromkeys(states1))
print(res)
#['CA', 'NY', 'FL']


#Example 2
states2=df.rdd.map(lambda x: x.state).collect()
print(states2)
#['CA', 'NY', 'CA', 'FL']

states3=df.select(df.state).collect()
print(states3)
#[Row(state='CA'), Row(state='NY'), Row(state='CA'), Row(state='FL')]

states4=df.select(df.state).rdd.flatMap(lambda x: x).collect()
print(states4)
#['CA', 'NY', 'CA', 'FL']

states5=df.select(df.state).toPandas()['state']
states6=list(states5)
print(states6)
#['CA', 'NY', 'CA', 'FL']

pandDF=df.select(df.state,df.firstname).toPandas()
print(list(pandDF['state']))
print(list(pandDF['firstname']))
----------------------------------------------------------------------
CurreentDate.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.functions import to_timestamp, current_timestamp
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,
LongType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

schema = StructType([
            StructField("seq", StringType(), True)])

dates = ['1']

df = spark.createDataFrame(list('1'), schema=schema)

df.show()
--------------------------------------------------------------------------------
--
Pandas pyspark dataframe:

import pandas as pd
data = [['Scott', 50], ['Jeff', 45], ['Thomas', 54],['Ann',34]]

# Create the pandas DataFrame
pandasDF = pd.DataFrame(data, columns = ['Name', 'Age'])

# print dataframe.
print(pandasDF)
```

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master("local[1]") \
    .appName("SparkByExamples.com") \
    .getOrCreate()

sparkDF=spark.createDataFrame(pandasDF)
sparkDF.printSchema()
sparkDF.show()

#sparkDF=spark.createDataFrame(pandasDF.astype(str))
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
mySchema = StructType([ StructField("First Name", StringType(), True)\
                       ,StructField("Age", IntegerType(), True)])

sparkDF2 = spark.createDataFrame(pandasDF,schema=mySchema)
sparkDF2.printSchema()
sparkDF2.show()


spark.conf.set("spark.sql.execution.arrow.enabled","true")
spark.conf.set("spark.sql.execution.arrow.pyspark.fallback.enabled","true")

pandasDF2=sparkDF2.select("*").toPandas
print(pandasDF2)


test=spark.conf.get("spark.sql.execution.arrow.enabled")
print(test)

test123=spark.conf.get("spark.sql.execution.arrow.pyspark.fallback.enabled")
print(test123)
--------------------------------------------------------------------------------
----
pyspark add_month date:

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

from pyspark.sql.functions import col,expr
data=[("2019-01-23",1),("2019-06-24",2),("2019-09-20",3)]
spark.createDataFrame(data).toDF("date","increment") \
    .select(col("date"),col("increment"), \
      expr("add_months(to_date(date,'yyyy-MM-dd'),cast(increment as
int))").alias("inc_date")) \
    .show()
-------------------------------------------------------------
pyspark add new column:

from pyspark.sql import SparkSession

spark = SparkSession.builder \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()

data = [('James','Smith','M',3000),
  ('Anna','Rose','F',4100),
  ('Robert','Williams','M',6200),
]

columns = ["firstname","lastname","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()
```

```
if 'salary1' not in df.columns:
    print("aa")

# Add new constanct column
from pyspark.sql.functions import lit
df.withColumn("bonus_percent", lit(0.3)) \
  .show()

#Add column from existing column
df.withColumn("bonus_amount", df.salary*0.3) \
  .show()

#Add column by concatinating existing columns
from pyspark.sql.functions import concat_ws
df.withColumn("name", concat_ws(",","firstname",'lastname')) \
  .show()

#Add current date
from pyspark.sql.functions import current_date
df.withColumn("current_date", current_date()) \
  .show()


from pyspark.sql.functions import when
df.withColumn("grade", \
   when((df.salary < 4000), lit("A")) \
     .when((df.salary >= 4000) & (df.salary <= 5000), lit("B")) \
     .otherwise(lit("C")) \
  ).show()

# Add column using select
df.select("firstname","salary", lit(0.3).alias("bonus")).show()
df.select("firstname","salary", lit(df.salary *
0.3).alias("bonus_amount")).show()
df.select("firstname","salary", current_date().alias("today_date")).show()

#Add columns using SQL
df.createOrReplaceTempView("PER")
spark.sql("select firstname,salary, '0.3' as bonus from PER").show()
spark.sql("select firstname,salary, salary * 0.3 as bonus_amount from
PER").show()
spark.sql("select firstname,salary, current_date() as today_date from
PER").show()
spark.sql("select firstname,salary, " +
          "case salary when salary < 4000 then 'A' "+
          "else 'B' END as grade from PER").show()
--------------------------------------------------------------------------------
------------
pyspark-aggregate.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import approx_count_distinct,collect_list
from pyspark.sql.functions import collect_set,sum,avg,max,countDistinct,count
from pyspark.sql.functions import first, last, kurtosis, min, mean, skewness
from pyspark.sql.functions import stddev, stddev_samp, stddev_pop, sumDistinct
from pyspark.sql.functions import variance,var_samp,  var_pop

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = [("James", "Sales", 3000),
    ("Michael", "Sales", 4600),
    ("Robert", "Sales", 4100),
    ("Maria", "Finance", 3000),
    ("James", "Sales", 3000),
    ("Scott", "Finance", 3300),
    ("Jen", "Finance", 3900),
    ("Jeff", "Marketing", 3000),
    ("Kumar", "Marketing", 2000),
    ("Saif", "Sales", 4100)
  ]
schema = ["employee_name", "department", "salary"]


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

print("approx_count_distinct: " + \
      str(df.select(approx_count_distinct("salary")).collect()[0][0]))

print("avg: " + str(df.select(avg("salary")).collect()[0][0]))

df.select(collect_list("salary")).show(truncate=False)

df.select(collect_set("salary")).show(truncate=False)

df2 = df.select(countDistinct("department", "salary"))
df2.show(truncate=False)
print("Distinct Count of Department &amp; Salary: "+str(df2.collect()[0][0]))

print("count: "+str(df.select(count("salary")).collect()[0]))
df.select(first("salary")).show(truncate=False)
df.select(last("salary")).show(truncate=False)
df.select(kurtosis("salary")).show(truncate=False)
df.select(max("salary")).show(truncate=False)
df.select(min("salary")).show(truncate=False)
df.select(mean("salary")).show(truncate=False)
df.select(skewness("salary")).show(truncate=False)
df.select(stddev("salary"), stddev_samp("salary"), \
    stddev_pop("salary")).show(truncate=False)
df.select(sum("salary")).show(truncate=False)
df.select(sumDistinct("salary")).show(truncate=False)
df.select(variance("salary"),var_samp("salary"),var_pop("salary")) \
  .show(truncate=False)
--------------------------------------------------------------------------------
--------
pyspark-array_string:

import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[1]") \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()

columns = ["name","languagesAtSchool","currentState"]
data = [("James,,Smith",["Java","Scala","C++"],"CA"), \
    ("Michael,Rose,",["Spark","Java","C++"],"NJ"), \
    ("Robert,,Williams",["CSharp","VB"],"NV")]

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

from pyspark.sql.functions import col, concat_ws
df2 = df.withColumn("languagesAtSchool",
   concat_ws(",",col("languagesAtSchool")))
df2.printSchema()
df2.show(truncate=False)


df.createOrReplaceTempView("ARRAY_STRING")
spark.sql("select name, concat_ws(',',languagesAtSchool) as
languagesAtSchool,currentState from ARRAY_STRING")
.show(truncate=False)
--------------------------------------------------------------------------------
-----------------------------------
```

pyspark type.py:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StringType, ArrayType,StructType,StructField
spark = SparkSession.builder \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()


arrayCol = ArrayType(StringType(),False)

data = [
 ("James,,Smith",["Java","Scala","C++"],["Spark","Java"],"OH","CA"),
 ("Michael,Rose,",["Spark","Java","C++"],["Spark","Java"],"NY","NJ"),
 ("Robert,,Williams",["CSharp","VB"],["Spark","Python"],"UT","NV")
]

schema = StructType([
    StructField("name",StringType(),True),
    StructField("languagesAtSchool",ArrayType(StringType()),True),
    StructField("languagesAtWork",ArrayType(StringType()),True),
    StructField("currentState", StringType(), True),
    StructField("previousState", StringType(), True)
  ])

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

from pyspark.sql.functions import explode
df.select(df.name,explode(df.languagesAtSchool)).show()

from pyspark.sql.functions import split
df.select(split(df.name,",").alias("nameAsArray")).show()

from pyspark.sql.functions import array
df.select(df.name,array(df.currentState,df.previousState).alias("States")).show(
)

from pyspark.sql.functions import array_contains
df.select(df.name,array_contains(df.languagesAtSchool,"Java")
    .alias("array_contains")).show()
--------------------------------------------------------------------------------
-------
```

Broad cast dataframe:

```
import pyspark
from pyspark.sql import SparkSession


spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

states = {"NY":"New York", "CA":"California", "FL":"Florida"}
```

```
broadcastStates = spark.sparkContext.broadcast(states)

data = [("James","Smith","USA","CA"),
    ("Michael","Rose","USA","NY"),
    ("Robert","Williams","USA","CA"),
    ("Maria","Jones","USA","FL")
  ]

columns = ["firstname","lastname","country","state"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

def state_convert(code):
    return broadcastStates.value[code]

result = df.rdd.map(lambda x:
(x[0],x[1],x[2],state_convert(x[3]))).toDF(columns)
result.show(truncate=False)

# Broadcast variable on filter

filteDf= df.where((df['state'].isin(broadcastStates.value)))
--------------------------------------------------------------------------------
--------
pyspark cast columns.py:

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = [("James",34,"2006-01-01","true","M",3000.60),
    ("Michael",33,"1980-01-10","true","F",3300.80),
    ("Robert",37,"06-01-1992","false","M",5000.50)
  ]

columns = ["firstname","age","jobStartDate","isGraduated","gender","salary"]
df = spark.createDataFrame(data = simpleData, schema = columns)
df.printSchema()
df.show(truncate=False)

from pyspark.sql.functions import col
from pyspark.sql.types import StringType,BooleanType,DateType
df2 = df.withColumn("age",col("age").cast(StringType())) \
    .withColumn("isGraduated",col("isGraduated").cast(BooleanType())) \
    .withColumn("jobStartDate",col("jobStartDate").cast(DateType()))
df2.printSchema()

df3 = df2.selectExpr("cast(age as int) age",
    "cast(isGraduated as string) isGraduated",
    "cast(jobStartDate as string) jobStartDate")
df3.printSchema()
df3.show(truncate=False)

df3.createOrReplaceTempView("CastExample")
df4 = spark.sql("SELECT STRING(age),BOOLEAN(isGraduated),DATE(jobStartDate) from
CastExample")
df4.printSchema()
df4.show(truncate=False)
----------------------------------------------------------------------------
pyspark chanfe-doubletype.py:

from pyspark.sql import SparkSession
from pyspark.sql.types import DoubleType, IntegerType
# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()

simpleData = [("James","34","true","M","3000.6089"),
    ("Michael","33","true","F","3300.8067"),
    ("Robert","37","false","M","5000.5034")
  ]

columns = ["firstname","age","isGraduated","gender","salary"]
df = spark.createDataFrame(data = simpleData, schema = columns)
df.printSchema()
df.show(truncate=False)

from pyspark.sql.functions import col,round,expr
df.withColumn("salary",df.salary.cast('double')).printSchema()
df.withColumn("salary",df.salary.cast(DoublerType())).printSchema()
df.withColumn("salary",col("salary").cast('double')).printSchema()

#df.withColumn("salary",round(df.salary.cast(DoubleType()),2)).show(truncate=Fal
se).printSchema()
df.selectExpr("firstname","isGraduated","cast(salary as double)
salary").printSchema()

df.createOrReplaceTempView("CastExample")
spark.sql("SELECT firstname,isGraduated,DOUBLE(salary) as salary from
CastExample").printSchema()


#df.select("firstname",expr(df.age),"isGraduated",col("salary").cast('float').al
ias("salary")).show()
--------------------------------------------------------------------------------
---------------------
collect.py:

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

dept = [("Finance",10), \
    ("Marketing",20), \
    ("Sales",30), \
    ("IT",40) \
  ]
deptColumns = ["dept_name","dept_id"]
deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
deptDF.printSchema()
deptDF.show(truncate=False)

dataCollect = deptDF.collect()

print(dataCollect)

dataCollect2 = deptDF.select("dept_name").collect()
print(dataCollect2)

for row in dataCollect:
    print(row['dept_name'] + "," +str(row['dept_id']))
--------------------------------------------------------------------------------
----
column function.py:
```

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data=[("James","Bond","100",None),
      ("Ann","Varsa","200",'F'),
      ("Tom Cruise","XXX","400",''),
      ("Tom Brand",None,"400",'M')]
columns=["fname","lname","id","gender"]
df=spark.createDataFrame(data,columns)

#alias
from pyspark.sql.functions import expr
df.select(df.fname.alias("first_name"), \
          df.lname.alias("last_name"), \
          expr(" fname ||','|| lname").alias("fullName") \
   ).show()

#asc, desc
df.sort(df.fname.asc()).show()
df.sort(df.fname.desc()).show()

#cast
df.select(df.fname,df.id.cast("int")).printSchema()

#between
df.filter(df.id.between(100,300)).show()

#contains
df.filter(df.fname.contains("Cruise")).show()

#startswith, endswith()
df.filter(df.fname.startswith("T")).show()
df.filter(df.fname.endswith("Cruise")).show()

#eqNullSafe

#isNull & isNotNull
df.filter(df.lname.isNull()).show()
df.filter(df.lname.isNotNull()).show()

#like , rlike
df.select(df.fname,df.lname,df.id) \
  .filter(df.fname.like("%om"))

#over

#substr
df.select(df.fname.substr(1,2).alias("substr")).show()

#when & otherwise
from pyspark.sql.functions import when
df.select(df.fname,df.lname,when(df.gender=="M","Male") \
              .when(df.gender=="F","Female") \
              .when(df.gender==None ,"") \
              .otherwise(df.gender).alias("new_gender") \
    ).show()

#isin
li=["100","200"]
df.select(df.fname,df.lname,df.id) \
  .filter(df.id.isin(li)) \
  .show()
```

```
from pyspark.sql.types import
StructType,StructField,StringType,ArrayType,MapType
data=[(("James","Bond"),["Java","C#"],{'hair':'black','eye':'brown'}),
      (("Ann","Varsa"),[".NET","Python"],{'hair':'brown','eye':'black'}),
      (("Tom Cruise",""),["Python","Scala"],{'hair':'red','eye':'grey'}),
      (("Tom Brand",None),["Perl","Ruby"],{'hair':'black','eye':'blue'})]

schema = StructType([
        StructField('name', StructType([
            StructField('fname', StringType(), True),
            StructField('lname', StringType(), True)])),
        StructField('languages', ArrayType(StringType()),True),
        StructField('properties', MapType(StringType(),StringType()),True)
     ])
df=spark.createDataFrame(data,schema)
df.printSchema()
#getItem()
df.select(df.languages.getItem(1)).show()

df.select(df.properties.getItem("hair")).show()

#getField from Struct or Map
df.select(df.properties.getField("hair")).show()

df.select(df.name.getField("fname")).show()

#dropFields
#from pyspark.sql.functions import col
#df.withColumn("name1",col("name").dropFields(["fname"])).show()

#withField
#from pyspark.sql.functions import lit
#df.withColumn("name",df.name.withField("fname",lit("AA"))).show()

#from pyspark.sql import Row
#from pyspark.sql.functions import lit
#df = spark.createDataFrame([Row(a=Row(b=1, c=2))])
#df.withColumn('a', df['a'].withField('b', lit(3))).select('a.b').show()

#from pyspark.sql import Row
#from pyspark.sql.functions import col, lit
#df = spark.createDataFrame([
#Row(a=Row(b=1, c=2, d=3, e=Row(f=4, g=5, h=6)))])
#df.withColumn('a', df['a'].dropFields('b')).show()
--------------------------------------------------------------------------------
---------
column operations.py:

from pyspark.sql import SparkSession,Row
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data=[("James",23),("Ann",40)]
df=spark.createDataFrame(data).toDF("name.fname","gender")
df.printSchema()
df.show()

from pyspark.sql.functions import col
df.select(col("`name.fname`")).show()
df.select(df["`name.fname`"]).show()
df.withColumn("new_col",col("`name.fname`").substr(1,2)).show()
df.filter(col("`name.fname`").startswith("J")).show()
new_cols=(column.replace('.', '_') for column in df.columns)
df2 = df.toDF(*new_cols)
df2.show()
```

```
# Using DataFrame object
df.select(df.gender).show()
df.select(df["gender"]).show()
#Accessing column name with dot (with backticks)
df.select(df["`name.fname`"]).show()

#Using SQL col() function
from pyspark.sql.functions import col
df.select(col("gender")).show()
#Accessing column name with dot (with backticks)
df.select(col("`name.fname`")).show()

#Access struct column
data=[Row(name="James",prop=Row(hair="black",eye="blue")),
      Row(name="Ann",prop=Row(hair="grey",eye="black"))]
df=spark.createDataFrame(data)
df.printSchema()

df.select(df.prop.hair).show()
df.select(df["prop.hair"]).show()
df.select(col("prop.hair")).show()
df.select(col("prop.*")).show()

# Column operators
data=[(100,2,1),(200,3,4),(300,4,4)]
df=spark.createDataFrame(data).toDF("col1","col2","col3")
df.select(df.col1 + df.col2).show()
df.select(df.col1 - df.col2).show()
df.select(df.col1 * df.col2).show()
df.select(df.col1 / df.col2).show()
df.select(df.col1 % df.col2).show()

df.select(df.col2 > df.col3).show()
df.select(df.col2 < df.col3).show()
df.select(df.col2 == df.col3).show()
--------------------------------------------------------------------------------
----------
convert map to columns.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

dataDictionary = [
        ('James',{'hair':'black','eye':'brown'}),
        ('Michael',{'hair':'brown','eye':None}),
        ('Robert',{'hair':'red','eye':'black'}),
        ('Washington',{'hair':'grey','eye':'grey'}),
        ('Jefferson',{'hair':'brown','eye':''})
        ]

df = spark.createDataFrame(data=dataDictionary, schema = ['name','properties'])
df.printSchema()
df.show(truncate=False)

df3=df.rdd.map(lambda x: \
    (x.name,x.properties["hair"],x.properties["eye"])) \
    .toDF(["name","hair","eye"])
df3.printSchema()
df3.show()

df.withColumn("hair",df.properties.getItem("hair")) \
  .withColumn("eye",df.properties.getItem("eye")) \
  .drop("properties") \
  .show()

df.withColumn("hair",df.properties["hair"]) \
  .withColumn("eye",df.properties["eye"]) \
  .drop("properties") \
  .show()

# Functions
from pyspark.sql.functions import explode,map_keys,col
keysDF = df.select(explode(map_keys(df.properties))).distinct()
keysList = keysDF.rdd.map(lambda x:x[0]).collect()
keyCols = list(map(lambda x: col("properties").getItem(x).alias(str(x)),
keysList))
df.select(df.name, *keyCols).show()
--------------------------------------------------------------------------------
----------
columns to map.py:

from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType, IntegerType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
data = [ ("36636","Finance",3000,"USA"),
    ("40288","Finance",5000,"IND"),
    ("42114","Sales",3900,"USA"),
    ("39192","Marketing",2500,"CAN"),
    ("34534","Sales",6500,"USA") ]
schema = StructType([
     StructField('id', StringType(), True),
     StructField('dept', StringType(), True),
     StructField('salary', IntegerType(), True),
     StructField('location', StringType(), True)
     ])

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

#Convert scolumns to Map
from pyspark.sql.functions import col,lit,create_map
df = df.withColumn("propertiesMap",create_map(
        lit("salary"),col("salary"),
        lit("location"),col("location")
        )).drop("salary","location")
df.printSchema()
df.show(truncate=False)
--------------------------------------------------------------------------------
-
count distinct.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder \
         .appName('SparkByExamples.com') \
         .getOrCreate()

data = [("James", "Sales", 3000),
    ("Michael", "Sales", 4600),
    ("Robert", "Sales", 4100),
    ("Maria", "Finance", 3000),
    ("James", "Sales", 3000),
    ("Scott", "Finance", 3300),
    ("Jen", "Finance", 3900),
    ("Jeff", "Marketing", 3000),
    ("Kumar", "Marketing", 2000),
    ("Saif", "Sales", 4100)
  ]
columns = ["Name","Dept","Salary"]
df = spark.createDataFrame(data=data,schema=columns)
df.distinct().show()
print("Distinct Count: " + str(df.distinct().count()))

# Using countDistrinct()
from pyspark.sql.functions import countDistinct
df2=df.select(countDistinct("Dept","Salary"))
df2.show()

print("Distinct Count of Department &amp; Salary: "+ str(df2.collect()[0][0]))

df.createOrReplaceTempView("PERSON")
spark.sql("select distinct(count(*)) from PERSON").show()
----------------------------------------------------------------------------
create dataFrame Dictionary.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

dataDictionary = [
        ('James',{'hair':'black','eye':'brown'}),
        ('Michael',{'hair':'brown','eye':None}),
        ('Robert',{'hair':'red','eye':'black'}),
        ('Washington',{'hair':'grey','eye':'grey'}),
        ('Jefferson',{'hair':'brown','eye':''})
        ]

df = spark.createDataFrame(data=dataDictionary, schema = ['name','properties'])
df.printSchema()
df.show(truncate=False)

# Using StructType schema
from pyspark.sql.types import StructField, StructType, StringType,
MapType,IntegerType
schema = StructType([
    StructField('name', StringType(), True),
    StructField('properties', MapType(StringType(),StringType()),True)
])
df2 = spark.createDataFrame(data=dataDictionary, schema = schema)
df2.printSchema()
df2.show(truncate=False)

df3=df.rdd.map(lambda x: \
    (x.name,x.properties["hair"],x.properties["eye"])) \
    .toDF(["name","hair","eye"])
df3.printSchema()
df3.show()

df.withColumn("hair",df.properties.getItem("hair")) \
  .withColumn("eye",df.properties.getItem("eye")) \
  .drop("properties") \
  .show()

df.withColumn("hair",df.properties["hair"]) \
  .withColumn("eye",df.properties["eye"]) \
  .drop("properties") \
  .show()

# Functions
from pyspark.sql.functions import explode,map_keys,col
```

```
keysDF = df.select(explode(map_keys(df.properties))).distinct()
keysList = keysDF.rdd.map(lambda x:x[0]).collect()
keyCols = list(map(lambda x: col("properties").getItem(x).alias(str(x)),
keysList))
df.select(df.name, *keyCols).show()
--------------------------------------------------------------------------------
-
ceate dataframe:

import pyspark
from pyspark.sql import SparkSession, Row
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
from pyspark.sql.functions import *

columns = ["language","users_count"]
data = [("Java", "20000"), ("Python", "100000"), ("Scala", "3000")]

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
rdd = spark.sparkContext.parallelize(data)

dfFromRDD1 = rdd.toDF()
dfFromRDD1.printSchema()

dfFromRDD1 = rdd.toDF(columns)
dfFromRDD1.printSchema()

dfFromRDD2 = spark.createDataFrame(rdd).toDF(*columns)
dfFromRDD2.printSchema()

dfFromData2 = spark.createDataFrame(data).toDF(*columns)
dfFromData2.printSchema()

rowData = map(lambda x: Row(*x), data)
dfFromData3 = spark.createDataFrame(rowData,columns)
dfFromData3.printSchema()
--------------------------------------------------------------------
create list.py:

import pyspark
from pyspark.sql import SparkSession, Row
from pyspark.sql.types import StructType,StructField, StringType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

#Using List
dept = [("Finance",10),
        ("Marketing",20),
        ("Sales",30),
        ("IT",40)
      ]

deptColumns = ["dept_name","dept_id"]
deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
deptDF.printSchema()
deptDF.show(truncate=False)

deptSchema = StructType([
    StructField('firstname', StringType(), True),
    StructField('middlename', StringType(), True),
    StructField('lastname', StringType(), True)
])

deptDF1 = spark.createDataFrame(data=dept, schema = deptSchema)
deptDF1.printSchema()
```

```
deptDF1.show(truncate=False)

# Using list of Row type
dept2 = [Row("Finance",10),
        Row("Marketing",20),
        Row("Sales",30),
        Row("IT",40)
      ]

deptDF2 = spark.createDataFrame(data=dept, schema = deptColumns)
deptDF2.printSchema()
deptDF2.show(truncate=False)

# Convert list to RDD
rdd = spark.sparkContext.parallelize(dept)
---------------------------------------------------------------------------
create datetimestamp.py:
from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
               .appName('SparkByExamples.com') \
               .getOrCreate()
data=[["1"]]
df=spark.createDataFrame(data,["id"])

from pyspark.sql.functions import *

#current_date() & current_timestamp()
df.withColumn("current_date",current_date()) \
  .withColumn("current_timestamp",current_timestamp()) \
  .show(truncate=False)

#SQL
spark.sql("select current_date(), current_timestamp()") \
     .show(truncate=False)

# Date & Timestamp into custom format
df.withColumn("date_format",date_format(current_date(),"MM-dd-yyyy")) \
  .withColumn("to_timestamp",to_timestamp(current_timestamp(),"MM-dd-yyyy HH mm
ss SSS")) \
  .show(truncate=False)

#SQL
spark.sql("select date_format(current_date(),'MM-dd-yyyy') as date_format ," + \
          "to_timestamp(current_timestamp(),'MM-dd-yyyy HH mm ss SSS') as
to_timestamp") \
     .show(truncate=False)
--------------------------------------------------------------------------------
-----------
create dataframe flatmap.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

columns = ["name","languagesAtSchool","currentState"]
data = [("James,,Smith",["Java","Scala","C++"],"CA"), \
    ("Michael,Rose,",["Spark","Java","C++"],"NJ"), \
    ("Robert,,Williams",["CSharp","VB"],"NV")]

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

```
#Flatmap
--------------------------------------------------------------------------------
---------
DataFrame Repartition.py:

import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com') \
        .master("local[5]").getOrCreate()

df=spark.range(0,20)
print(df.rdd.getNumPartitions())

df.write.mode("overwrite").csv("c:/tmp/partition.csv")

df2 = df.repartition(6)
print(df2.rdd.getNumPartitions())

df3 = df.coalesce(2)
print(df3.rdd.getNumPartitions())

df4 = df.groupBy("id").count()
print(df4.rdd.getNumPartitions())
-----------------------------------------------------------------
dataframe.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField, StringType,IntegerType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

print(spark)
--------------------------------------------------------------------------------
----
Date string.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
               .appName('SparkByExamples.com') \
               .getOrCreate()

from pyspark.sql.functions import *

df=spark.createDataFrame([["1"]],["id"])
df.select(current_date().alias("current_date"), \
      date_format(current_date(),"yyyy MM dd").alias("yyyy MM dd"), \
      date_format(current_timestamp(),"MM/dd/yyyy hh:mm").alias("MM/dd/yyyy"), \
      date_format(current_timestamp(),"yyyy MMM dd").alias("yyyy MMMM dd"), \
      date_format(current_timestamp(),"yyyy MMMM dd E").alias("yyyy MMMM dd
E") \
   ).show()

#SQL

spark.sql("select current_date() as current_date, "+
      "date_format(current_timestamp(),'yyyy MM dd') as yyyy_MM_dd, "+
      "date_format(current_timestamp(),'MM/dd/yyyy hh:mm') as MM_dd_yyyy, "+
      "date_format(current_timestamp(),'yyyy MMM dd') as yyyy_MMMM_dd, "+
      "date_format(current_timestamp(),'yyyy MMMM dd E') as
yyyy_MMMM_dd_E").show()
```

```
--------------------------------------------------------------------------------
------
Date timestamp functions.py:

from pyspark.sql import SparkSession
# Create SparkSession
spark = SparkSession.builder \
               .appName('SparkByExamples.com') \
               .getOrCreate()
data=[["1","2020-02-01"],["2","2019-03-01"],["3","2021-03-01"]]
df=spark.createDataFrame(data,["id","input"])
df.show()

from pyspark.sql.functions import *

#current_date()
df.select(current_date().alias("current_date")
  ).show(1)

#date_format()
df.select(col("input"),
    date_format(col("input"), "MM-dd-yyyy").alias("date_format")
  ).show()

#to_date()
df.select(col("input"),
    to_date(col("input"), "yyy-MM-dd").alias("to_date")
  ).show()

#datediff()
df.select(col("input"),
    datediff(current_date(),col("input")).alias("datediff")
  ).show()

#months_between()
df.select(col("input"),
    months_between(current_date(),col("input")).alias("months_between")
  ).show()

#trunc()
df.select(col("input"),
    trunc(col("input"),"Month").alias("Month_Trunc"),
    trunc(col("input"),"Year").alias("Month_Year"),
    trunc(col("input"),"Month").alias("Month_Trunc")
   ).show()

#add_months() , date_add(), date_sub()

df.select(col("input"),
    add_months(col("input"),3).alias("add_months"),
    add_months(col("input"),-3).alias("sub_months"),
    date_add(col("input"),4).alias("date_add"),
    date_sub(col("input"),4).alias("date_sub")
  ).show()

#

df.select(col("input"),
     year(col("input")).alias("year"),
     month(col("input")).alias("month"),
     next_day(col("input"),"Sunday").alias("next_day"),
     weekofyear(col("input")).alias("weekofyear")
  ).show()
```

```
df.select(col("input"),
     dayofweek(col("input")).alias("dayofweek"),
     dayofmonth(col("input")).alias("dayofmonth"),
     dayofyear(col("input")).alias("dayofyear"),
  ).show()

data=[["1","02-01-2020 11 01 19 06"],["2","03-01-2019 12 01 19 406"],["3","03-
01-2021 12 01 19 406"]]
df2=spark.createDataFrame(data,["id","input"])
df2.show(truncate=False)

#current_timestamp()
df2.select(current_timestamp().alias("current_timestamp")
  ).show(1,truncate=False)

#to_timestamp()
df2.select(col("input"),
    to_timestamp(col("input"), "MM-dd-yyyy HH mm ss SSS").alias("to_timestamp")
  ).show(truncate=False)


#hour, minute,second
data=[["1","2020-02-01 11:01:19.06"],["2","2019-03-01 12:01:19.406"],["3","2021-
03-01 12:01:19.406"]]
df3=spark.createDataFrame(data,["id","input"])

df3.select(col("input"),
    hour(col("input")).alias("hour"),
    minute(col("input")).alias("minute"),
    second(col("input")).alias("second")
  ).show(truncate=False)
--------------------------------------------------------------------------------
------
Date diffrence.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()
data = [("1","2019-07-01"),("2","2019-06-24"),("3","2019-08-24")]

df=spark.createDataFrame(data=data,schema=["id","date"])

from pyspark.sql.functions import *

df.select(
      col("date"),
      current_date().alias("current_date"),
      datediff(current_date(),col("date")).alias("datediff")
    ).show()

df.withColumn("datesDiff", datediff(current_date(),col("date"))) \
  .withColumn("montsDiff", months_between(current_date(),col("date"))) \
  .withColumn("montsDiff_round",round(months_between(current_date(),col("date"))
,2)) \
  .withColumn("yearsDiff",months_between(current_date(),col("date"))/lit(12)) \
  .withColumn("yearsDiff_round",round(months_between(current_date(),col("date"))
/lit(12),2)) \
  .show()

data2 = [("1","07-01-2019"),("2","06-24-2019"),("3","08-24-2019")]
df2=spark.createDataFrame(data=data2,schema=["id","date"])
```

```
df2.select(
    to_date(col("date"),"MM-dd-yyyy").alias("date"),
    current_date().alias("endDate")
    )

#SQL

spark.sql("select round(months_between('2019-07-01',current_date())/12,2) as 
years_diff").show()
--------------------------------------------------------------------------------
----------
Distinct.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import expr
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = [("James", "Sales", 3000), \
    ("Michael", "Sales", 4600), \
    ("Robert", "Sales", 4100), \
    ("Maria", "Finance", 3000), \
    ("James", "Sales", 3000), \
    ("Scott", "Finance", 3300), \
    ("Jen", "Finance", 3900), \
    ("Jeff", "Marketing", 3000), \
    ("Kumar", "Marketing", 2000), \
    ("Saif", "Sales", 4100) \
  ]
columns= ["employee_name", "department", "salary"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

distinctDF = df.distinct()
print("Distinct count: "+str(distinctDF.count()))
distinctDF.show(truncate=False)

df2 = df.dropDuplicates()
print("Distinct count: "+str(df2.count()))
df2.show(truncate=False)

dropDisDF = df.dropDuplicates(["department","salary"])
print("Distinct count of department salary : "+str(dropDisDF.count()))
dropDisDF.show(truncate=False)
-----------------------------------------------------------------------------
drop columns.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
simpleData = (("James","","Smith","36636","NewYork",3100), \
    ("Michael","Rose","","40288","California",4300), \
    ("Robert","","Williams","42114","Florida",1400), \
    ("Maria","Anne","Jones","39192","Florida",5500), \
    ("Jen","Mary","Brown","34561","NewYork",3000) \
  )
columns= ["firstname","middlename","lastname","id","location","salary"]

df = spark.createDataFrame(data = simpleData, schema = columns)

df.printSchema()
df.show(truncate=False)

df.drop("firstname") \
  .printSchema()

df.drop(col("firstname")) \
  .printSchema()

df.drop(df.firstname) \
  .printSchema()

df.drop("firstname","middlename","lastname") \
    .printSchema()

cols = ("firstname","middlename","lastname")

df.drop(*cols) \
   .printSchema()
--------------------------------------------------------------------------------
----
drop null.py:

from pyspark.sql import SparkSession

spark: SparkSession = SparkSession.builder \
    .master("local[1]") \
    .appName("SparkByExamples.com") \
    .getOrCreate()

filePath="resources/small_zipcode.csv"
df = spark.read.options(header='true', inferSchema='true') \
          .csv(filePath)

df.printSchema()
df.show(truncate=False)

df.na.drop().show(truncate=False)

df.na.drop(how="any").show(truncate=False)

df.na.drop(subset=["population","type"]) \
   .show(truncate=False)

df.dropna().show(truncate=False)
--------------------------------------------------------------------------------
----
   Empty data frame.py:

   import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

schema = StructType([
  StructField('firstname', StringType(), True),
  StructField('middlename', StringType(), True),
  StructField('lastname', StringType(), True)
  ])
df = spark.createDataFrame(spark.sparkContext.emptyRDD(),schema)
df.printSchema()

df1 = spark.sparkContext.parallelize([]).toDF(schema)
df1.printSchema()
```

```
df2 = spark.createDataFrame([], schema)
df2.printSchema()

df3 = spark.emptyDataFrame()
df3.printSchema()
--------------------------------------------------------------------------------
----------
Explode array map.py:

import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('pyspark-by-examples').getOrCreate()

arrayData = [
        ('James',['Java','Scala'],{'hair':'black','eye':'brown'}),
        ('Michael',['Spark','Java',None],{'hair':'brown','eye':None}),
        ('Robert',['CSharp',''],{'hair':'red','eye':''}),
        ('Washington',None,None),
        ('Jefferson',['1','2'],{})
        ]
df = spark.createDataFrame(data=arrayData, schema =
['name','knownLanguages','properties'])
df.printSchema()
df.show()

from pyspark.sql.functions import explode
df2 = df.select(df.name,explode(df.knownLanguages))
df2.printSchema()
df2.show()

from pyspark.sql.functions import explode
df3 = df.select(df.name,explode(df.properties))
df3.printSchema()
df3.show()

from pyspark.sql.functions import explode_outer
""" with array """
df.select(df.name,explode_outer(df.knownLanguages)).show()
""" with map """
df.select(df.name,explode_outer(df.properties)).show()


from pyspark.sql.functions import posexplode
""" with array """
df.select(df.name,posexplode(df.knownLanguages)).show()
""" with map """
df.select(df.name,posexplode(df.properties)).show()

from pyspark.sql.functions import posexplode_outer
""" with array """
df.select(df.name,posexplode_outer(df.knownLanguages)).show()

""" with map """
df.select(df.name,posexplode_outer(df.properties)).show()

--------------------------------------------------------------------------------
-------
Exploded nested array.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, flatten
```

```
spark = SparkSession.builder.appName('pyspark-by-examples').getOrCreate()

arrayArrayData = [
  ("James",[["Java","Scala","C++"],["Spark","Java"]]),
  ("Michael",[["Spark","Java","C++"],["Spark","Java"]]),
  ("Robert",[["CSharp","VB"],["Spark","Python"]])
]

df = spark.createDataFrame(data=arrayArrayData, schema = ['name','subjects'])
df.printSchema()
df.show(truncate=False)

""" """
df.select(df.name,explode(df.subjects)).show(truncate=False)

""" creates a single array from an array of arrays. """
df.select(df.name,flatten(df.subjects)).show(truncate=False)
--------------------------------------------------------------------------------
---
Expr.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

from pyspark.sql.functions import expr
#Concatenate columns
data=[("James","Bond"),("Scott","Varsa")]
df=spark.createDataFrame(data).toDF("col1","col2")
df.withColumn("Name",expr(" col1 ||','|| col2")).show()

#Using CASE WHEN sql expression
data = [("James","M"),("Michael","F"),("Jen","")]
columns = ["name","gender"]
df = spark.createDataFrame(data = data, schema = columns)
df2 = df.withColumn("gender", expr("CASE WHEN gender = 'M' THEN 'Male' " +
           "WHEN gender = 'F' THEN 'Female' ELSE 'unknown' END"))
df2.show()

#Add months from a value of another column
data=[("2019-01-23",1),("2019-06-24",2),("2019-09-20",3)]
df=spark.createDataFrame(data).toDF("date","increment")
df.select(df.date,df.increment,
     expr("add_months(date,increment)")
  .alias("inc_date")).show()

# Providing alias using 'as'
df.select(df.date,df.increment,
     expr("""add_months(date,increment) as inc_date""")
  ).show()

# Add
df.select(df.date,df.increment,
     expr("increment + 5 as new_increment")
  ).show()

df.select("increment",expr("cast(increment as string) as str_increment")) \
  .printSchema()
#Use expr()  to filter the rows
data=[(100,2),(200,3000),(500,500)]
df=spark.createDataFrame(data).toDF("col1","col2")
df.filter(expr("col1 == col2")).show()
--------------------------------------------------------------------------------
--------
```

Filter null.py:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
spark: SparkSession = SparkSession.builder \
    .master("local[1]") \
    .appName("SparkByExamples.com") \
    .getOrCreate()

data = [
    ("James",None,"M"),
    ("Anna","NY","F"),
    ("Julia",None,None)
]

columns = ["name","state","gender"]
df =spark.createDataFrame(data,columns)

df.printSchema()
df.show()

df.filter("state is NULL").show()
df.filter(df.state.isNull()).show()
df.filter(col("state").isNull()).show()

df.filter("state IS NULL AND gender IS NULL").show()
df.filter(df.state.isNull() & df.gender.isNull()).show()

df.filter("state is not NULL").show()
df.filter("NOT state is NULL").show()
df.filter(df.state.isNotNull()).show()
df.filter(col("state").isNotNull()).show()
df.na.drop(subset=["state"]).show()

df.createOrReplaceTempView("DATA")
spark.sql("SELECT * FROM DATA where STATE IS NULL").show()
spark.sql("SELECT * FROM DATA where STATE IS NULL AND GENDER IS NULL").show()
spark.sql("SELECT * FROM DATA where STATE IS NOT NULL").show()
```


Groupby sort.py:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col,sum,avg,max

spark = SparkSession.builder \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()

simpleData = [("James","Sales","NY",90000,34,10000),
    ("Michael","Sales","NV",86000,56,20000),
    ("Robert","Sales","CA",81000,30,23000),
    ("Maria","Finance","CA",90000,24,23000),
    ("Raman","Finance","DE",99000,40,24000),
    ("Scott","Finance","NY",83000,36,19000),
    ("Jen","Finance","NY",79000,53,15000),
    ("Jeff","Marketing","NV",80000,25,18000),
    ("Kumar","Marketing","NJ",91000,50,21000)
  ]

schema = ["employee_name","department","state","salary","age","bonus"]
df = spark.createDataFrame(data=simpleData, schema = schema)
df.printSchema()
df.show(truncate=False)
df.groupBy("state").sum("salary").show()

dfGroup=df.groupBy("state") \
          .agg(sum("salary").alias("sum_salary"))

dfGroup.show(truncate=False)

dfFilter=dfGroup.filter(dfGroup.sum_salary > 100000)
dfFilter.show()

from pyspark.sql.functions import asc
dfFilter.sort("sum_salary").show()

from pyspark.sql.functions import desc
dfFilter.sort(desc("sum_salary")).show()

df.groupBy("state") \
  .agg(sum("salary").alias("sum_salary")) \
  .filter(col("sum_salary") > 100000)  \
  .sort(desc("sum_salary")) \
  .show()

df.createOrReplaceTempView("EMP")
spark.sql("select state, sum(salary) as sum_salary from EMP " +
          "group by state having sum_salary > 100000 " +
          "order by sum_salary desc").show()

df.groupBy("state") \
  .sum("salary") \
  .withColumnRenamed("sum(salary)", "sum_salary") \
  .show()

df.groupBy("state") \
  .sum("salary") \
  .select(col("state"),col("sum(salary)").alias("sum_salary")) \
  .show()
--------------------------------------------------------------------------------
---------
groupBy.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col,sum,avg,max

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = [("James","Sales","NY",90000,34,10000),
    ("Michael","Sales","NY",86000,56,20000),
    ("Robert","Sales","CA",81000,30,23000),
    ("Maria","Finance","CA",90000,24,23000),
    ("Raman","Finance","CA",99000,40,24000),
    ("Scott","Finance","NY",83000,36,19000),
    ("Jen","Finance","NY",79000,53,15000),
    ("Jeff","Marketing","CA",80000,25,18000),
    ("Kumar","Marketing","NY",91000,50,21000)
  ]

schema = ["employee_name","department","state","salary","age","bonus"]
df = spark.createDataFrame(data=simpleData, schema = schema)
df.printSchema()
df.show(truncate=False)

df.groupBy("department").sum("salary").show(truncate=False)
df.groupBy("department").count().show(truncate=False)


df.groupBy("department","state") \
    .sum("salary","bonus") \
   .show(truncate=False)

df.groupBy("department") \
    .agg(sum("salary").alias("sum_salary"), \
         avg("salary").alias("avg_salary"), \
         sum("bonus").alias("sum_bonus"), \
         max("bonus").alias("max_bonus") \
     ) \
    .show(truncate=False)

df.groupBy("department") \
    .agg(sum("salary").alias("sum_salary"), \
      avg("salary").alias("avg_salary"), \
      sum("bonus").alias("sum_bonus"), \
      max("bonus").alias("max_bonus")) \
    .where(col("sum_bonus") >= 50000) \
    .show(truncate=False)
--------------------------------------------------------------------------------
--
join the two dataframe.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()
#EMP DataFrame
empData = [(1,"Smith",10), (2,"Rose",20),
    (3,"Williams",10), (4,"Jones",30)
  ]
empColumns = ["emp_id","name","emp_dept_id"]
empDF = spark.createDataFrame(empData,empColumns)
empDF.show()

#DEPT DataFrame
deptData = [("Finance",10), ("Marketing",20),
    ("Sales",30),("IT",40)
  ]
deptColumns = ["dept_name","dept_id"]
deptDF=spark.createDataFrame(deptData,deptColumns)
deptDF.show()

#Address DataFrame
addData=[(1,"1523 Main St","SFO","CA"),
    (2,"3453 Orange St","SFO","NY"),
    (3,"34 Warner St","Jersey","NJ"),
    (4,"221 Cavalier St","Newark","DE"),
    (5,"789 Walnut St","Sandiago","CA")
  ]
addColumns = ["emp_id","addline1","city","state"]
addDF = spark.createDataFrame(addData,addColumns)
addDF.show()

#Join two DataFrames
empDF.join(addDF,empDF["emp_id"] == addDF["emp_id"]).show()

#Drop duplicate column
```

```
empDF.join(addDF,["emp_id"]).show()

#Join Multiple DataFrames
empDF.join(addDF,["emp_id"]) \
     .join(deptDF,empDF["emp_dept_id"] == deptDF["dept_id"]) \
     .show()

#Using Where for Join Condition
empDF.join(deptDF).where(empDF["emp_dept_id"] == deptDF["dept_id"]) \
    .join(addDF).where(empDF["emp_id"] == addDF["emp_id"]) \
    .show()

#SQL
empDF.createOrReplaceTempView("EMP")
deptDF.createOrReplaceTempView("DEPT")
addDF.createOrReplaceTempView("ADD")

spark.sql("select * from EMP e, DEPT d, ADD a " + \
    "where e.emp_dept_id == d.dept_id and e.emp_id == a.emp_id") \
    .show()

#
df1 = spark.createDataFrame(
    [(1, "A"), (2, "B"), (3, "C")],
    ["A1", "A2"])

df2 = spark.createDataFrame(
    [(1, "F"), (2, "B")],
    ["B1", "B2"])

df = df1.join(df2, (df1.A1 == df2.B1) & (df1.A2 == df2.B2))
df.show()
--------------------------------------------------------------------------------
---------
join.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

emp = [(1,"Smith",-1,"2018","10","M",3000), \
    (2,"Rose",1,"2010","20","M",4000), \
    (3,"Williams",1,"2010","10","M",1000), \
    (4,"Jones",2,"2005","10","F",2000), \
    (5,"Brown",2,"2010","40","",-1), \
      (6,"Brown",2,"2010","50","",-1) \
  ]
empColumns = ["emp_id","name","superior_emp_id","year_joined", \
       "emp_dept_id","gender","salary"]

empDF = spark.createDataFrame(data=emp, schema = empColumns)
empDF.printSchema()
empDF.show(truncate=False)


dept = [("Finance",10), \
    ("Marketing",20), \
    ("Sales",30), \
    ("IT",40) \
  ]
deptColumns = ["dept_name","dept_id"]
deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
```

```
deptDF.printSchema()
deptDF.show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"inner") \
     .show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"outer") \
    .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"full") \
    .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"fullouter") \
    .show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"left") \
    .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftouter") \
   .show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"right") \
   .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"rightouter") \
   .show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftsemi") \
   .show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftanti") \
   .show(truncate=False)

empDF.alias("emp1").join(empDF.alias("emp2"), \
    col("emp1.superior_emp_id") == col("emp2.emp_id"),"inner") \
    .select(col("emp1.emp_id"),col("emp1.name"), \
      col("emp2.emp_id").alias("superior_emp_id"), \
      col("emp2.name").alias("superior_emp_name")) \
   .show(truncate=False)

empDF.createOrReplaceTempView("EMP")
deptDF.createOrReplaceTempView("DEPT")

joinDF = spark.sql("select * from EMP e, DEPT d where e.emp_dept_id ==
d.dept_id") \
  .show(truncate=False)

joinDF2 = spark.sql("select * from EMP e INNER JOIN DEPT d ON e.emp_dept_id ==
d.dept_id") \
  .show(truncate=False)
--------------------------------------------------------------------------------
---
leftanti join.py:

import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('machinelearninggeeks.com').getOrCreate()

emp = [(1,"Smith",-1,"2018","10","M",3000), \
    (2,"Rose",1,"2010","20","M",4000), \
    (3,"Williams",1,"2010","10","M",1000), \
    (4,"Jones",2,"2005","10","F",2000), \
    (5,"Brown",2,"2010","40","",-1), \
      (6,"Brown",2,"2010","50","",-1) \
  ]
empColumns = ["emp_id","name","superior_emp_id","year_joined", \
       "emp_dept_id","gender","salary"]
```

```
empDF = spark.createDataFrame(data=emp, schema = empColumns)
empDF.printSchema()
empDF.show(truncate=False)

dept = [("Finance",10), \
    ("Marketing",20), \
    ("Sales",30), \
    ("IT",40) \
  ]
deptColumns = ["dept_name","dept_id"]
deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
deptDF.printSchema()
deptDF.show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==
deptDF.dept_id,"left").show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==
deptDF.dept_id,"leftouter").show(truncate=False)

empDF.createOrReplaceTempView("EMP")
deptDF.createOrReplaceTempView("DEPT")

joinDF2 = spark.sql("SELECT e.* FROM EMP e LEFT ANTI JOIN DEPT d ON
e.emp_dept_id == d.dept_id") \
  .show(truncate=False)
--------------------------------------------------------------------------------
--------
  lit.py:

  import pyspark
from pyspark.sql import SparkSession


spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
data = [("111",50000),("222",60000),("333",40000)]
columns= ["EmpId","Salary"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

from pyspark.sql.functions import col,lit
df2 = df.select(col("EmpId"),col("Salary"),lit("1").alias("lit_value1"))
df2.show(truncate=False)


from pyspark.sql.functions import when
df3 = df2.withColumn("lit_value2", when(col("Salary") >=40000 & col("Salary") <=
50000,lit("100")).otherwise(lit("200")))
df3.show(truncate=False)
--------------------------------------------------------------------------------
----------
loop.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()

data = [('James','Smith','M',30),
  ('Anna','Rose','F',41),
  ('Robert','Williams','M',62),
]

columns = ["firstname","lastname","gender","salary"]
```

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

from pyspark.sql.functions import concat_ws,col,lit
df.select(concat_ws(",",df.firstname,df.lastname).alias("name"), \
          df.gender,lit(df.salary*2).alias("new_salary")).show()

print(df.collect())
rdd=df.rdd.map(lambda x:
    (x[0]+","+x[1],x[2],x[3]*2)
    )
df2=rdd.toDF(["name","gender","new_salary"]   )
df2.show()


#Referring Column Names
rdd2=df.rdd.map(lambda x:
    (x["firstname"]+","+x["lastname"],x["gender"],x["salary"]*2)
    )


#Referring Column Names
rdd2=df.rdd.map(lambda x:
    (x.firstname+","+x.lastname,x.gender,x.salary*2)
    )


def func1(x):
    firstName=x.firstname
    lastName=x.lastName
    name=firstName+","+lastName
    gender=x.gender.lower()
    salary=x.salary*2
    return (name,gender,salary)

rdd2=df.rdd.map(lambda x: func1(x))

#Foeeach example
def f(x): print(x)
df.rdd.foreach(f)

df.rdd.foreach(lambda x:
    print("Data ==>"+x["firstname"]+","+x["lastname"]+","+x["gender"]
+","+str(x["salary"]*2))
    )

#Iterate collected data
dataCollect = df.collect()
for row in dataCollect:
    print(row['firstname'] + "," +row['lastname'])

#Convert to Pandas and Iterate

dataCollect=df.rdd.toLocalIterator()
for row in dataCollect:
    print(row['firstname'] + "," +row['lastname'])

import pandas as pd
pandasDF = df.toPandas()
for index, row in pandasDF.iterrows():
    print(row['firstname'], row['gender'])
--------------------------------------------------------------------------------
-------
map partition.py:
```

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
data = [('James','Smith','M',3000),
  ('Anna','Rose','F',4100),
  ('Robert','Williams','M',6200),
]

columns = ["firstname","lastname","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()

#Example 1 mapPartitions()
def reformat(partitionData):
    for row in partitionData:
        yield [row.firstname+","+row.lastname,row.salary*10/100]
df.rdd.mapPartitions(reformat).toDF().show()

#Example 2 mapPartitions()
def reformat2(partitionData):
  updatedData = []
  for row in partitionData:
    name=row.firstname+","+row.lastname
    bonus=row.salary*10/100
    updatedData.append([name,bonus])
  return iter(updatedData)

df2=df.rdd.mapPartitions(reformat2).toDF("name","bonus")
df2.show()
------------------------------------------------------------------------
maptype-dataframe-column.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

dataDictionary = [
        ('James',{'hair':'black','eye':'brown'}),
        ('Michael',{'hair':'brown','eye':None}),
        ('Robert',{'hair':'red','eye':'black'}),
        ('Washington',{'hair':'grey','eye':'grey'}),
        ('Jefferson',{'hair':'brown','eye':''})
        ]

# Using StructType schema
from pyspark.sql.types import StructField, StructType, StringType, MapType
schema = StructType([
    StructField('name', StringType(), True),
    StructField('properties', MapType(StringType(),StringType()),True)
])
df = spark.createDataFrame(data=dataDictionary, schema = schema)
df.printSchema()
df.show(truncate=False)

df3=df.rdd.map(lambda x: \
    (x.name,x.properties["hair"],x.properties["eye"])) \
    .toDF(["name","hair","eye"])
df3.printSchema()
df3.show()

df.withColumn("hair",df.properties.getItem("hair")) \
  .withColumn("eye",df.properties.getItem("eye")) \
  .drop("properties") \
  .show()
```

```
df.withColumn("hair",df.properties["hair"]) \
  .withColumn("eye",df.properties["eye"]) \
  .drop("properties") \
  .show()

from pyspark.sql.functions import explode
df.select(df.name,explode(df.properties)).show()

from pyspark.sql.functions import map_keys
df.select(df.name,map_keys(df.properties)).show()

from pyspark.sql.functions import map_values
df.select(df.name,map_values(df.properties)).show()

#from pyspark.sql.functions import explode,map_keys
#keysDF = df.select(explode(map_keys(df.properties))).distinct()
#keysList = keysDF.rdd.map(lambda x:x[0]).collect()
#print(keysList)
----------------------------------------------------------------------------
order by group by.py:

from pyspark.sql import SparkSession
from pyspark.sql.functions import col,sum,avg,max

spark = SparkSession.builder \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()

simpleData = [("James","Sales","NY",90000,34,10000),
    ("Michael","Sales","NV",86000,56,20000),
    ("Robert","Sales","CA",81000,30,23000),
    ("Maria","Finance","CA",90000,24,23000),
    ("Raman","Finance","DE",99000,40,24000),
    ("Scott","Finance","NY",83000,36,19000),
    ("Jen","Finance","NY",79000,53,15000),
    ("Jeff","Marketing","NV",80000,25,18000),
    ("Kumar","Marketing","NJ",91000,50,21000)
  ]

schema = ["employee_name","department","state","salary","age","bonus"]
df = spark.createDataFrame(data=simpleData, schema = schema)
df.printSchema()
df.show(truncate=False)

dfSort=df.sort(df.state,df.salary).groupBy(df.state).agg(sum(df.salary))
dfSort.show()
----------------------------------------------------------------------------
order by.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, asc,desc

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = [("James","Sales","NY",90000,34,10000), \
    ("Michael","Sales","NY",86000,56,20000), \
    ("Robert","Sales","CA",81000,30,23000), \
    ("Maria","Finance","CA",90000,24,23000), \
    ("Raman","Finance","CA",99000,40,24000), \
    ("Scott","Finance","NY",83000,36,19000), \
    ("Jen","Finance","NY",79000,53,15000), \
    ("Jeff","Marketing","CA",80000,25,18000), \
    ("Kumar","Marketing","NY",91000,50,21000) \
  ]
columns= ["employee_name","department","state","salary","age","bonus"]

df = spark.createDataFrame(data = simpleData, schema = columns)

df.printSchema()
df.show(truncate=False)

df.sort("department","state").show(truncate=False)
df.sort(col("department"),col("state")).show(truncate=False)

df.orderBy("department","state").show(truncate=False)
df.orderBy(col("department"),col("state")).show(truncate=False)

df.sort(df.department.asc(),df.state.asc()).show(truncate=False)
df.sort(col("department").asc(),col("state").asc()).show(truncate=False)
df.orderBy(col("department").asc(),col("state").asc()).show(truncate=False)

df.sort(df.department.asc(),df.state.desc()).show(truncate=False)
df.sort(col("department").asc(),col("state").desc()).show(truncate=False)
df.orderBy(col("department").asc(),col("state").desc()).show(truncate=False)


df.createOrReplaceTempView("EMP")
df.select("employee_name",asc("department"),desc("state"),"salary","age","bonus"
).show(truncate=False)

spark.sql("select employee_name,department,state,salary,age,bonus from EMP ORDER
BY department asc").show(truncate=False)
--------------------------------------------------------------------------------
-------------
parallelize.py:

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
rdd=spark.sparkContext.parallelize([1,2,3,4,5])

rddCollect = rdd.collect()
print("Number of Partitions: "+str(rdd.getNumPartitions()))
print("Action: First element: "+str(rdd.first()))
print(rddCollect)

emptyRDD = spark.sparkContext.emptyRDD()
emptyRDD2 = rdd=spark.sparkContext.parallelize([])

print(""+str(emptyRDD2.isEmpty()))

--------------------------------------------------------------------------------
Partition by.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()

df=spark.read.option("header",True) \
        .csv("C:/apps/sparkbyexamples/src/pyspark-examples/resources/simple-
zipcodes.csv")
```

```
df.show()
print(df.rdd.getNumPartitions())

df.write.option("header",True) \
        .partitionBy("state") \
        .mode("overwrite") \
        .csv("c:/tmp/zipcodes-state")

df.write.option("header",True) \
        .partitionBy("state","city") \
        .mode("overwrite") \
        .csv("c:/tmp/zipcodes-state-city")


df=df.repartition(2)

print(df.rdd.getNumPartitions())

df.write.option("header",True) \
        .partitionBy("state") \
        .mode("overwrite") \
        .csv("c:/tmp/zipcodes-state-more")

dfPartition=spark.read.option("header",True)\
                 .csv("c:/tmp/zipcodes-state")
dfPartition.printSchema()

dfSinglePart=spark.read.option("header",True) \
                  .csv("c:/tmp/zipcodes-state/state=AL/city=SPRINGVILLE")
dfSinglePart.printSchema()
dfSinglePart.show()

parqDF = spark.read.option("header",True) \
                  .csv("c:/tmp/zipcodes-state")
parqDF.createOrReplaceTempView("ZIPCODE")
spark.sql("select * from ZIPCODE  where state='AL' and city = 'SPRINGVILLE'") \
    .show()

df.write.option("header",True) \
        .option("maxRecordsPerFile", 2) \
        .partitionBy("state") \
        .mode("overwrite") \
        .csv("/tmp/zipcodes-state-maxrecords")
--------------------------------------------------------------------------------
----
pivot by.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import expr
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = [("Banana",1000,"USA"), ("Carrots",1500,"USA"), ("Beans",1600,"USA"), \
      ("Orange",2000,"USA"),("Orange",2000,"USA"),("Banana",400,"China"), \
      ("Carrots",1200,"China"),("Beans",1500,"China"),("Orange",4000,"China"), \
      ("Banana",2000,"Canada"),("Carrots",2000,"Canada"),
("Beans",2000,"Mexico")]

columns= ["Product","Amount","Country"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

pivotDF = df.groupBy("Product").pivot("Country").sum("Amount")
```

```
pivotDF.printSchema()
pivotDF.show(truncate=False)

pivotDF = df.groupBy("Product","Country") \
      .sum("Amount") \
      .groupBy("Product") \
      .pivot("Country") \
      .sum("sum(Amount)")
pivotDF.printSchema()
pivotDF.show(truncate=False)


""" unpivot """
""" unpivot """
unpivotExpr = "stack(3, 'Canada', Canada, 'China', China, 'Mexico', Mexico) as
(Country,Total)"
unPivotDF = pivotDF.select("Product", expr(unpivotExpr)) \
    .where("Total is not null")
unPivotDF.show(truncate=False)
--------------------------------------------------------------------------------
--------
print conten.py:

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

dept = [("Finance",10), \
    ("Marketing",20), \
    ("Sales",30), \
    ("IT",40) \
  ]

rdd=spark.sparkContext.parallelize(dept)
print(rdd)
dataColl=rdd.collect()

for row in dataColl:
    print(row[0] + "," +str(row[1]))
"""
deptColumns = ["dept_name","dept_id"]
deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
deptDF.printSchema()
deptDF.show(truncate=False)
dataCollect = deptDF.collect()
print(dataCollect)
dataCollect2 = deptDF.select("dept_name").collect()
print(dataCollect2)
for row in dataCollect:
    print(row['dept_name'] + "," +str(row['dept_id']))
--------------------------------------------------------------------------------
--
python dataframe.py:

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = [("James","","Smith","36636","M",60000),
        ("Michael","Rose","","40288","M",70000),
        ("Robert","","Williams","42114","",400000),
        ("Maria","Anne","Jones","39192","F",500000),
        ("Jen","Mary","Brown","","F",0)]
```

```
columns = ["first_name","middle_name","last_name","dob","gender","salary"]
pysparkDF = spark.createDataFrame(data = data, schema = columns)
pysparkDF.printSchema()
pysparkDF.show(truncate=False)

pandasDF = pysparkDF.toPandas()
print(pandasDF)

# Nested structure elements
from pyspark.sql.types import StructType, StructField, StringType,IntegerType
dataStruct = [(("James","","Smith"),"36636","M","3000"), \
      (("Michael","Rose",""),"40288","M","4000"), \
      (("Robert","","Williams"),"42114","M","4000"), \
      (("Maria","Anne","Jones"),"39192","F","4000"), \
      (("Jen","Mary","Brown"),"","F","-1") \
]

schemaStruct = StructType([
        StructField('name', StructType([
             StructField('firstname', StringType(), True),
             StructField('middlename', StringType(), True),
             StructField('lastname', StringType(), True)
             ])),
          StructField('dob', StringType(), True),
         StructField('gender', StringType(), True),
         StructField('salary', StringType(), True)
         ])


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

pandasDF2 = df.toPandas()
print(pandasDF2)
```
----
range partition by:

```
from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()


data = [(1,10),(2,20),(3,10),(4,20),(5,10),
    (6,30),(7,50),(8,50),(9,50),(10,30),
    (11,10),(12,10),(13,40),(14,40),(15,40),
    (16,40),(17,50),(18,10),(19,40),(20,40)
  ]

df=spark.createDataFrame(data,["id","value"])

df.repartition(3,"value").explain(True)
df.repartition("value") \
  .write.option("header",True) \
  .mode("overwrite") \
  .csv("c:/tmp/range-partition")

df.repartitionByRange("value").explain(True)
df.repartitionByRange(3,"value").explain(True)
df.repartitionByRange(3,"value") \
  .write.option("header",True) \
  .mode("overwrite") \
  .csv("c:/tmp/range-partition-count")

--------------------------------------------------------------------------------
------
  RDD Action.py:

  from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
data=[("Z", 1),("A", 20),("B", 30),("C", 40),("B", 30),("B", 60)]
inputRDD = spark.sparkContext.parallelize(data)

listRdd = spark.sparkContext.parallelize([1,2,3,4,5,3,2])

#aggregate
seqOp = (lambda x, y: x + y)
combOp = (lambda x, y: x + y)
agg=listRdd.aggregate(0, seqOp, combOp)
print(agg) # output 20

#aggregate 2
seqOp2 = (lambda x, y: (x[0] + y, x[1] + 1))
combOp2 = (lambda x, y: (x[0] + y[0], x[1] + y[1]))
agg2=listRdd.aggregate((0, 0), seqOp2, combOp2)
print(agg2) # output (20,7)

agg2=listRdd.treeAggregate(0,seqOp, combOp)
print(agg2) # output 20

#fold
from operator import add
foldRes=listRdd.fold(0, add)
print(foldRes) # output 20

#reduce
redRes=listRdd.reduce(add)
print(redRes) # output 20

#treeReduce. This is similar to reduce
add = lambda x, y: x + y
redRes=listRdd.treeReduce(add)
print(redRes) # output 20

#Collect
data = listRdd.collect()
print(data)

#count, countApprox, countApproxDistinct
print("Count : "+str(listRdd.count()))
#Output: Count : 20
print("countApprox : "+str(listRdd.countApprox(1200)))
#Output: countApprox : (final: [7.000, 7.000])
print("countApproxDistinct : "+str(listRdd.countApproxDistinct()))
#Output: countApproxDistinct : 5
print("countApproxDistinct : "+str(inputRDD.countApproxDistinct()))
#Output: countApproxDistinct : 5

#countByValue, countByValueApprox
print("countByValue :  "+str(listRdd.countByValue()))
```

```
#first
print("first :  "+str(listRdd.first()))
#Output: first :  1
print("first :  "+str(inputRDD.first()))
#Output: first :  (Z,1)

#top
print("top : "+str(listRdd.top(2)))
#Output: take : 5,4
print("top : "+str(inputRDD.top(2)))
#Output: take : (Z,1),(C,40)

#min
print("min :  "+str(listRdd.min()))
#Output: min :  1
print("min :  "+str(inputRDD.min()))
#Output: min :  (A,20)

#max
print("max :  "+str(listRdd.max()))
#Output: max :  5
print("max :  "+str(inputRDD.max()))
#Output: max :  (Z,1)

#take, takeOrdered, takeSample
print("take : "+str(listRdd.take(2)))
#Output: take : 1,2
print("takeOrdered : "+ str(listRdd.takeOrdered(2)))
#Output: takeOrdered : 1,2
print("take : "+str(listRdd.takeSample()))
--------------------------------------------------------------------------------
-----
RDD Broad cast:

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

states = {"NY":"New York", "CA":"California", "FL":"Florida"}
broadcastStates = spark.sparkContext.broadcast(states)

data = [("James","Smith","USA","CA"),
    ("Michael","Rose","USA","NY"),
    ("Robert","Williams","USA","CA"),
    ("Maria","Jones","USA","FL")
  ]

rdd = spark.sparkContext.parallelize(data)

def state_convert(code):
    return broadcastStates.value[code]

result = rdd.map(lambda x: (x[0],x[1],x[2],state_convert(x[3]))).collect()
print(result)
--------------------------------------------------------------------------------
-----
Rdd flatmap.py:
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = ["Project Gutenberg’s",
        "Alice’s Adventures in Wonderland",
        "Project Gutenberg’s",
        "Adventures in Wonderland",
        "Project Gutenberg’s"]
rdd=spark.sparkContext.parallelize(data)

for element in rdd.collect():
    print(element)

#Flatmap
rdd2=rdd.flatMap(lambda x: x.split(" "))
for element in rdd2.collect():
    print(element)
---------------------------------------------------------------------------
rdd map.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = ["Project",
"Gutenberg’s",
"Alice’s",
"Adventures",
"in",
"Wonderland",
"Project",
"Gutenberg’s",
"Adventures",
"in",
"Wonderland",
"Project",
"Gutenberg’s"]

rdd=spark.sparkContext.parallelize(data)

rdd2=rdd.map(lambda x: (x,1))
for element in rdd2.collect():
    print(element)

data = [('James','Smith','M',30),
  ('Anna','Rose','F',41),
  ('Robert','Williams','M',62),
]

columns = ["firstname","lastname","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()

rdd2=df.rdd.map(lambda x:
    (x[0]+","+x[1],x[2],x[3]*2)
    )
df2=rdd2.toDF(["name","gender","new_salary"]   )
df2.show()


#Referring Column Names
rdd2=df.rdd.map(lambda x:
    (x["firstname"]+","+x["lastname"],x["gender"],x["salary"]*2)
    )


#Referring Column Names
rdd2=df.rdd.map(lambda x:
    (x.firstname+","+x.lastname,x.gender,x.salary*2)
    )
```

```
def func1(x):
    firstName=x.firstname
    lastName=x.lastname
    name=firstName+","+lastName
    gender=x.gender.lower()
    salary=x.salary*2
    return (name,gender,salary)

rdd2=df.rdd.map(lambda x: func1(x)).toDF().show()
rdd2=df.rdd.map(func1).toDF().show()
```

----
rdd reduced by key.py:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = [('Project', 1),
('Gutenberg’s', 1),
('Alice’s', 1),
('Adventures', 1),
('in', 1),
('Wonderland', 1),
('Project', 1),
('Gutenberg’s', 1),
('Adventures', 1),
('in', 1),
('Wonderland', 1),
('Project', 1),
('Gutenberg’s', 1)]

rdd=spark.sparkContext.parallelize(data)

rdd2=rdd.reduceByKey(lambda a,b: a+b)
for element in rdd2.collect():
    print(element)
```


Rdd to dataframe.py:

```
import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

dept = [("Finance",10),
        ("Marketing",20),
        ("Sales",30),
        ("IT",40)
      ]
rdd = spark.sparkContext.parallelize(dept)

df = rdd.toDF()
df.printSchema()
df.show(truncate=False)

deptColumns = ["dept_name","dept_id"]
df2 = rdd.toDF(deptColumns)
df2.printSchema()
df2.show(truncate=False)

deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
deptDF.printSchema()
deptDF.show(truncate=False)


from pyspark.sql.types import StructType,StructField, StringType
deptSchema = StructType([
    StructField('dept_name', StringType(), True),
    StructField('dept_id', StringType(), True)
])

deptDF1 = spark.createDataFrame(data=dept, schema = deptSchema)
deptDF1.printSchema()
deptDF1.show(truncate=False)
--------------------------------------------------------------------------------
-------
rdd word count.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
data = ["Project Gutenberg’s",
        "Alice’s Adventures in Wonderland",
        "Project Gutenberg’s",
        "Adventures in Wonderland",
        "Project Gutenberg’s"]
rdd=spark.sparkContext.parallelize(data)

for element in rdd.collect():
    print(element)

#Flatmap
rdd2=rdd.flatMap(lambda x: x.split(" "))
for element in rdd2.collect():
    print(element)
#map
rdd3=rdd2.map(lambda x: (x,1))
for element in rdd3.collect():
    print(element)
#reduceByKey
rdd4=rdd3.reduceByKey(lambda a,b: a+b)
for element in rdd4.collect():
    print(element)
#map
rdd5 = rdd4.map(lambda x: (x[1],x[0])).sortByKey()
for element in rdd5.collect():
    print(element)
#filter
rdd6 = rdd5.filter(lambda x : 'a' in x[1])
for element in rdd6.collect():
    print(element)

from pyspark.sql.functions import col,expr
data=[("2019-01-23",1),("2019-06-24",2),("2019-09-20",3)]
spark.createDataFrame(data).toDF("date","increment") \
    .select(col("date"),col("increment"), \
      expr("add_months(to_date(date,'yyyy-MM-dd'),cast(increment as
int))").alias("inc_date")) \
    .show()
--------------------------------------------------------------------------------
-----------
rdd.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
data = ["Project Gutenberg’s",
        "Alice’s Adventures in Wonderland",
        "Project Gutenberg’s",
        "Adventures in Wonderland",
        "Project Gutenberg’s"]
rdd=spark.sparkContext.parallelize(data)

for element in rdd.collect():
    print(element)

#Flatmap
rdd2=rdd.flatMap(lambda x: x.split(" "))
for element in rdd2.collect():
    print(element)
#map
rdd3=rdd2.map(lambda x: (x,1))
for element in rdd3.collect():
    print(element)
#reduceByKey
rdd4=rdd3.reduceByKey(lambda a,b: a+b)
for element in rdd4.collect():
    print(element)
#map
rdd5 = rdd4.map(lambda x: (x[1],x[0])).sortByKey()
for element in rdd5.collect():
    print(element)
#filter
rdd6 = rdd5.filter(lambda x : 'a' in x[1])
for element in rdd6.collect():
    print(element)

from pyspark.sql.functions import col,expr
data=[("2019-01-23",1),("2019-06-24",2),("2019-09-20",3)]
spark.createDataFrame(data).toDF("date","increment") \
    .select(col("date"),col("increment"), \
      expr("add_months(to_date(date,'yyyy-MM-dd'),cast(increment as
int))").alias("inc_date")) \
    .show()
--------------------------------------------------------------------------------
-------
read.csv.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
from pyspark.sql.types import ArrayType, DoubleType, BooleanType
from pyspark.sql.functions import col,array_contains

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

df = spark.read.csv("C:/apps/sparkbyexamples/src/pyspark-examples/resources/
zipcodes.csv")

df.printSchema()

df2 = spark.read.option("header",True) \
     .csv("C:/apps/sparkbyexamples/src/pyspark-examples/resources/zipcodes.csv")
df2.printSchema()



df3 = spark.read.options(header='True', delimiter=',') \
  .csv("C:/apps/sparkbyexamples/src/pyspark-examples/resources/zipcodes.csv")
df3.printSchema()
```

```
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("C:/apps/sparkbyexamples/src/pyspark-examples/resources/zipcodes.csv")
df_with_schema.printSchema()

df2.write.option("header",True) \
 .csv("/tmp/spark_output/zipcodes123")
--------------------------------------------------------------------------------
----------
Readjson.py:

from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType,
IntegerType,BooleanType,DoubleType
spark = SparkSession.builder \
    .master("local[1]") \
    .appName("SparkByExamples.com") \
    .getOrCreate()

# Read JSON file into dataframe
df = spark.read.json("resources/zipcodes.json")
df.printSchema()
df.show()

# Read multiline json file
multiline_df = spark.read.option("multiline","true") \
      .json("resources/multiline-zipcode.json")
multiline_df.show()

#Read multiple files
df2 = spark.read.json(
    ['resources/zipcode2.json','resources/zipcode1.json'])
df2.show()

#Read All JSON files from a directory
df3 = spark.read.json("resources/*.json")
df3.show()

# Define custom schema
schema = StructType([
      StructField("RecordNumber",IntegerType(),True),
      StructField("Zipcode",IntegerType(),True),
      StructField("ZipCodeType",StringType(),True),
      StructField("City",StringType(),True),
      StructField("State",StringType(),True),
      StructField("LocationType",StringType(),True),
      StructField("Lat",DoubleType(),True),
      StructField("Long",DoubleType(),True),
      StructField("Xaxis",IntegerType(),True),
      StructField("Yaxis",DoubleType(),True),
      StructField("Zaxis",DoubleType(),True),
      StructField("WorldRegion",StringType(),True),
      StructField("Country",StringType(),True),
      StructField("LocationText",StringType(),True),
      StructField("Location",StringType(),True),
      StructField("Decommisioned",BooleanType(),True),
      StructField("TaxReturnsFiled",StringType(),True),
      StructField("EstimatedPopulation",IntegerType(),True),
      StructField("TotalWages",IntegerType(),True),
      StructField("Notes",StringType(),True)
  ])

df_with_schema = spark.read.schema(schema) \
        .json("resources/zipcodes.json")
df_with_schema.printSchema()
df_with_schema.show()

# Create a table from Parquet File
spark.sql("CREATE OR REPLACE TEMPORARY VIEW zipcode3 USING json OPTIONS" +
      " (path 'resources/zipcodes.json')")
spark.sql("select * from zipcode3").show()

# PySpark write Parquet File
df2.write.mode('Overwrite').json("/tmp/spark_output/zipcodes.json")
--------------------------------------------------------------------------------
-------
read column.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
from pyspark.sql.functions import *


spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()


dataDF = [(('James','','Smith'),'1991-04-01','M',3000),
  (('Michael','Rose',''),'2000-05-19','M',4000),
  (('Robert','','Williams'),'1978-09-05','M',4000),
  (('Maria','Anne','Jones'),'1967-12-01','F',4000),
  (('Jen','Mary','Brown'),'1980-02-17','F',-1)
]

schema = StructType([
        StructField('name', StructType([
             StructField('firstname', StringType(), True),
             StructField('middlename', StringType(), True),
             StructField('lastname', StringType(), True)
             ])),
         StructField('dob', StringType(), True),
         StructField('gender', StringType(), True),
         StructField('salary', IntegerType(), True)
         ])
```

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

''' Example 1 '''
df.withColumnRenamed("dob","DateOfBirth").printSchema()
''' Example 2 '''
df2 = df.withColumnRenamed("dob","DateOfBirth") \
    .withColumnRenamed("salary","salary_amount")
df2.printSchema()

''' Example 3 '''
schema2 = StructType([
    StructField("fname",StringType()),
    StructField("middlename",StringType()),
    StructField("lname",StringType())])

df.select(col("name").cast(schema2),
  col("dob"),
  col("gender"),
  col("salary")) \
    .printSchema()

''' Example 4 '''
df.select(col("name.firstname").alias("fname"),
  col("name.middlename").alias("mname"),
  col("name.lastname").alias("lname"),
  col("dob"),col("gender"),col("salary")) \
  .printSchema()

''' Example 5 '''
df4 = df.withColumn("fname",col("name.firstname")) \
      .withColumn("mname",col("name.middlename")) \
      .withColumn("lname",col("name.lastname")) \
      .drop("name")
df4.printSchema()
--------------------------------------------------------------------------------
--------
Replace null.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .master("local[1]") \
    .appName("SparkByExamples.com") \
    .getOrCreate()

filePath="resources/small_zipcode.csv"
df = spark.read.options(header='true', inferSchema='true') \
          .csv(filePath)

df.printSchema()
df.show(truncate=False)


df.fillna(value=0).show()
df.fillna(value=0,subset=["population"]).show()
df.na.fill(value=0).show()
df.na.fill(value=0,subset=["population"]).show()


df.fillna(value="").show()
df.na.fill(value="").show()

df.fillna("unknown",["city"]) \
    .fillna("",["type"]).show()

df.fillna({"city": "unknown", "type": ""}) \
    .show()

df.na.fill("unknown",["city"]) \
    .na.fill("",["type"]).show()

df.na.fill({"city": "unknown", "type": ""}) \
    .show()
--------------------------------------------------------------------------------
-------
repartition2.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()

df=spark.read.option("header",True) \
        .csv("C:/apps/sparkbyexamples/src/pyspark-examples/resources/simple-
zipcodes.csv")

newDF=df.repartition(3)
print(newDF.rdd.getNumPartitions())

newDF.write.option("header",True).mode("overwrite") \
        .csv("/tmp/zipcodes-state")

df2=df.repartition(3,"state")
df2.write.option("header",True).mode("overwrite") \
   .csv("/tmp/zipcodes-state-3states")

df3=df.repartition("state")
df3.write.option("header",True).mode("overwrite") \
   .csv("/tmp/zipcodes-state-allstates")
--------------------------------------------------------------------------------
------
Repartition.py:

import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com') \
        .master("local[5]").getOrCreate()

df = spark.range(0,20)
print(df.rdd.getNumPartitions())

spark.conf.set("spark.sql.shuffle.partitions", "500")

rdd = spark.sparkContext.parallelize((0,20))
print("From local[5]"+str(rdd.getNumPartitions()))

rdd1 = spark.sparkContext.parallelize((0,25), 6)
print("parallelize : "+str(rdd1.getNumPartitions()))

"""rddFromFile = spark.sparkContext.textFile("src/main/resources/test.txt",10)
print("TextFile : "+str(rddFromFile.getNumPartitions())) """

rdd1.saveAsTextFile("c://tmp/partition2")

rdd2 = rdd1.repartition(4)
```

```
print("Repartition size : "+str(rdd2.getNumPartitions()))
rdd2.saveAsTextFile("c://tmp/re-partition2")

rdd3 = rdd1.coalesce(4)
print("Repartition size : "+str(rdd3.getNumPartitions()))
rdd3.saveAsTextFile("c:/tmp/coalesce2")
--------------------------------------------------------------------------------
--------
row.py:

from pyspark.sql import SparkSession, Row


row=Row("James",40)
print(row[0] +","+str(row[1]))
row2=Row(name="Alice", age=11)
print(row2.name)

Person = Row("name", "age")
p1=Person("James", 40)
p2=Person("Alice", 35)
print(p1.name +","+p2.name)

#PySpark Example
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

rdd2 = spark.sparkContext.parallelize([],10)

data = [Row(name="James,,Smith",lang=["Java","Scala","C++"],state="CA"),
    Row(name="Michael,Rose,",lang=["Spark","Java","C++"],state="NJ"),
    Row(name="Robert,,Williams",lang=["CSharp","VB"],state="NV")]

#RDD Example 1
rdd=spark.sparkContext.parallelize(data)
collData=rdd.collect()
print(collData)
for row in collData:
    print(row.name + "," +str(row.lang))

# RDD Example 2
Person=Row("name","lang","state")
data = [Person("James,,Smith",["Java","Scala","C++"],"CA"),
    Person("Michael,Rose,",["Spark","Java","C++"],"NJ"),
    Person("Robert,,Williams",["CSharp","VB"],"NV")]
rdd=spark.sparkContext.parallelize(data)
collData=rdd.collect()
print(collData)
for person in collData:
    print(person.name + "," +str(person.lang))

#DataFrame Example 1
columns = ["name","languagesAtSchool","currentState"]
df=spark.createDataFrame(data)
df.printSchema()
df.show()

collData=df.collect()
print(collData)
for row in collData:
    print(row.name + "," +str(row.lang))

#DataFrame Example 2
data = [("James,,Smith",["Java","Scala","C++"],"CA"),
("Michael,Rose,",["Spark","Java","C++"],"NJ"),
("Robert,,Williams",["CSharp","VB"],"NV")]
columns = ["name","languagesAtSchool","currentState"]
df=spark.createDataFrame(data).toDF(*columns)
df.printSchema()
for row in df.collect():
    print(row.name)
---------------------------------------------------------------------------
sampling.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .master("local[1]") \
    .appName("SparkByExamples.com") \
    .getOrCreate()

df=spark.range(100)

'''sample() '''
print(df.sample(0.06).collect())
print(df.sample(0.1,123).collect())
print(df.sample(0.1,123).collect())
print(df.sample(0.1,456).collect())
print("withReplacement Examples")
print(df.sample(True,0.3,123).collect())
print(df.sample(0.3,123).collect())

'''sampleBy() '''
print("sampleBy Examples")
df2=df.select((df.id % 3).alias("key"))
print(df2.sampleBy("key", {0: 0.1, 1: 0.2},0).collect())


print("RDD Examples")
'''RDD'''
rdd = spark.sparkContext.range(0,100)
print(rdd.sample(False,0.1,0).collect())
print(rdd.sample(True,0.3,123).collect())

''' RDD takeSample() '''
print(rdd.takeSample(False,10,0))
print(rdd.takeSample(True,30,123))
--------------------------------------------------------------------------------
-------
select columns.py:

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = [("James","Smith","USA","CA"),
    ("Michael","Rose","USA","NY"),
    ("Robert","Williams","USA","CA"),
    ("Maria","Jones","USA","FL")
  ]

columns = ["firstname","lastname","country","state"]
df = spark.createDataFrame(data = data, schema = columns)
df.show(truncate=False)


df.select("firstname","lastname").show()

#Using Dataframe object name
```

```
df.select(df.firstname,df.lastname).show()
df.select(df["firstname"],df["lastname"]).show()

# Using col function
from pyspark.sql.functions import col
df.select(col("firstname").alias("fname"),col("lastname")).show()

# Show all columns
df.select("*").show()
df.select([col for col in df.columns]).show()
df.select(*columns).show()

df.select(df.columns[:3]).show(3)
df.select(df.columns[2:4]).show(3)

df.select(df.colRegex("`^.*name*`")).show()

data = [
        (("James",None,"Smith"),"OH","M"),
        (("Anna","Rose",""),"NY","F"),
        (("Julia","","Williams"),"OH","F"),
        (("Maria","Anne","Jones"),"NY","M"),
        (("Jen","Mary","Brown"),"NY","M"),
        (("Mike","Mary","Williams"),"OH","M")
        ]

from pyspark.sql.types import StructType,StructField, StringType
schema = StructType([
    StructField('name', StructType([
         StructField('firstname', StringType(), True),
         StructField('middlename', StringType(), True),
         StructField('lastname', StringType(), True)
         ])),
     StructField('state', StringType(), True),
     StructField('gender', StringType(), True)
     ])


df2 = spark.createDataFrame(data = data, schema = schema)
df2.printSchema()
df2.show(truncate=False) # shows all columns
df2.select("name").show(truncate=False)
df2.select("name.firstname","name.lastname").show(truncate=False)
df2.select("name.*").show(truncate=False)
--------------------------------------------------------------------------------
-----------
same dataframe.py:

from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master("local[1]") \
    .appName("SparkByExamples.com") \
    .getOrCreate()
data = [('Scott', 50), ('Jeff', 45), ('Thomas', 54),('Ann',34)]
sparkDF=spark.createDataFrame(data,["name","age"])
sparkDF.printSchema()
sparkDF.show()

print((sparkDF.count(), len(sparkDF.columns)))

def sparkShape(dataFrame):
    return (dataFrame.count(), len(dataFrame.columns))
import pyspark
```

```
pyspark.sql.dataframe.DataFrame.shape = sparkShape
print(sparkDF.shape())

import pandas as pd
pandasDF=sparkDF.toPandas()
print(pandasDF.shape)
--------------------------------------------------------------------------------
--
shwo top row n.py:

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = [("James",34),("Ann",34),
    ("Michael",33),("Scott",53),
    ("Robert",37),("Chad",27)
  ]

columns = ["firstname","age",]
df = spark.createDataFrame(data = simpleData, schema = columns)


df.show()
#Returns the first ``num`` rows as a :class:`list` of :class:`Row`.
# Internally calls limit and collect
#Action, Return Array[T]
print(df.take(2))

#Returns the last ``num`` rows as a :class:`list` of :class:`Row`.
#Running tail requires moving data into the application's driver process, and
doing so with
#a very large ``num`` can crash the driver process with OutOfMemoryError.
#Return Array[T]
print(df.tail(2))


"""Returns the first ``n`` rows.
.. note:: This method should only be used if the resulting array is expected
    to be small, as all the data is loaded into the driver's memory.
:param n: int, default 1. Number of rows to return.
:return: If n is greater than 1, return a list of :class:`Row`.
    If n is 1, return a single Row."""
#Return Array[T]
print(df.head(2))


#Returns the first row, same as df.head(1)
print(df.first())

#Returns all the records as a list of :class:`Row`.
#Action, Return Array[T]
print(df.collect())
#"Limits the result count to the number specified.
#Returns a new Dataset by taking the first n rows.
pandasDF=df.limit(3).toPandas()
print(pandasDF)
--------------------------------------------------------------------------------
---------
spark session.py:

import pyspark
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.master("local[1]") \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()

print("First SparkContext:");
print("APP Name :"+spark.sparkContext.appName);
print("Master :"+spark.sparkContext.master);

sparkSession2 = SparkSession.builder \
      .master("local[1]") \
      .appName("SparkByExample-test") \
      .getOrCreate();

print("Second SparkContext:")
print("APP Name :"+sparkSession2.sparkContext.appName);
print("Master :"+sparkSession2.sparkContext.master);


sparkSession3 = SparkSession.newSession

print("Second SparkContext:")
print("APP Name :"+sparkSession3.sparkContext.appName);
print("Master :"+sparkSession3.sparkContext.master);
--------------------------------------------------------------------------------
--
split function.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import split, col
spark=SparkSession.builder.appName("sparkbyexamples").getOrCreate()

data=data = [('James','','Smith','1991-04-01'),
  ('Michael','Rose','','2000-05-19'),
  ('Robert','','Williams','1978-09-05'),
  ('Maria','Anne','Jones','1967-12-01'),
  ('Jen','Mary','Brown','1980-02-17')
]

columns=["firstname","middlename","lastname","dob"]
df=spark.createDataFrame(data,columns)
df.printSchema()
df.show(truncate=False)
df1 = df.withColumn('year', split(df['dob'], '-').getItem(0)) \
       .withColumn('month', split(df['dob'], '-').getItem(1)) \
       .withColumn('day', split(df['dob'], '-').getItem(2))
df1.printSchema()
df1.show(truncate=False)

 # Alternatively we can do like below
split_col = pyspark.sql.functions.split(df['dob'], '-')
df2 = df.withColumn('year', split_col.getItem(0)) \
       .withColumn('month', split_col.getItem(1)) \
       .withColumn('day', split_col.getItem(2))
df2.show(truncate=False)

# Using split() function of Column class
split_col = pyspark.sql.functions.split(df['dob'], '-')
df3 = df.select("firstname","middlename","lastname","dob",
split_col.getItem(0).alias('year'),split_col.getItem(1).alias('month'),split_col
.getItem(2).alias('day'))
df3.show(truncate=False)

"""
```

```
df4=spark.createDataFrame([("20-13-2012-monday",)], ['date',])
df4.select(split(df4.date,'^([\d]+-[\d]+-[\d])').alias('date'),
    regexp_replace(split(df4.date,'^([\d]+-[\d]+-[\d]
+)').getItem(1),'-','').alias('day')).show()
    """
df4 = spark.createDataFrame([('oneAtwoBthree',)], ['str',])
df4.select(split(df4.str, '[AB]').alias('str')).show()

df4.select(split(df4.str, '[AB]',2).alias('str')).show()
df4.select(split(df4.str, '[AB]',1).alias('str')).show()
--------------------------------------------------------------------------------
-------
sql case when.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
data = [("James","M",60000), ("Michael","M",70000),
        ("Robert",None,400000), ("Maria","F",500000),
        ("Jen","",None)]

columns = ["name","gender","salary"]
df = spark.createDataFrame(data = data, schema = columns)
df.show()

#Using When otherwise
from pyspark.sql.functions import when,col
df2 = df.withColumn("new_gender", when(df.gender == "M","Male")
                                 .when(df.gender == "F","Female")
                                 .when(df.gender.isNull() ,"")
                                 .otherwise(df.gender))
df2.show()
df2 = df.withColumn("new_gender", when(df.gender == "M","Male")
                                 .when(df.gender == "F","Female")
                                 .when(df.gender.isNull() ,"")
                                 .otherwise(df.gender))

df2=df.select(col("*"),when(df.gender == "M","Male")
                  .when(df.gender == "F","Female")
                  .when(df.gender.isNull() ,"")
                  .otherwise(df.gender).alias("new_gender"))
df2.show()
# Using SQL Case When
from pyspark.sql.functions import expr
df3 = df.withColumn("new_gender", expr("CASE WHEN gender = 'M' THEN 'Male' " +
           "WHEN gender = 'F' THEN 'Female' WHEN gender IS NULL THEN ''" +
          "ELSE gender END"))
df3.show()

df4 = df.select(col("*"), expr("CASE WHEN gender = 'M' THEN 'Male' " +
           "WHEN gender = 'F' THEN 'Female' WHEN gender IS NULL THEN ''" +
           "ELSE gender END").alias("new_gender"))

df.createOrReplaceTempView("EMP")
spark.sql("select name, CASE WHEN gender = 'M' THEN 'Male' " +
               "WHEN gender = 'F' THEN 'Female' WHEN gender IS NULL THEN ''" +
              "ELSE gender END as new_gender from EMP").show()
---------------------------------------------------------------------------
string date.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
               .appName('SparkByExamples.com') \
               .getOrCreate()

from pyspark.sql.functions import *

df=spark.createDataFrame([["02-03-2013"],["05-06-2023"]],["input"])
df.select(col("input"),to_date(col("input"),"MM-dd-yyyy").alias("date")) \
  .show()

#SQL
spark.sql("select to_date('02-03-2013','MM-dd-yyyy') date").show()
--------------------------------------------------------------------------------
------
string timestamp.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()

from pyspark.sql.functions import *

df=spark.createDataFrame(
        data = [ ("1","2019-06-24 12:01:19.000")],
        schema=["id","input_timestamp"])
df.printSchema()

#Timestamp String to DateType
df.withColumn("timestamp",to_timestamp("input_timestamp")) \
  .show(truncate=False)

# Using Cast to convert TimestampType to DateType
df.withColumn('timestamp', \
         to_timestamp('input_timestamp').cast('string')) \
  .show(truncate=False)


df.select(to_timestamp(lit('06-24-2019 12:01:19.000'),'MM-dd-yyyy
HH:mm:ss.SSSS')) \
  .show(truncate=False)


#SQL string to TimestampType
spark.sql("select to_timestamp('2019-06-24 12:01:19.000') as timestamp")
#SQL CAST timestamp string to TimestampType
spark.sql("select timestamp('2019-06-24 12:01:19.000') as timestamp")
#SQL Custom string to TimestampType
spark.sql("select to_timestamp('06-24-2019 12:01:19.000','MM-dd-yyyy
HH:mm:ss.SSSS') as timestamp")
--------------------------------------------------------------------------------
------------------
string array.py:

from pyspark.sql import SparkSession
spark = SparkSession.builder \
         .appName('SparkByExamples.com') \
         .getOrCreate()

data = [("James, A, Smith","2018","M",3000),
            ("Michael, Rose, Jones","2010","M",4000),
            ("Robert,K,Williams","2010","M",4000),
            ("Maria,Anne,Jones","2005","F",4000),
            ("Jen,Mary,Brown","2010","",-1)
            ]

columns=["name","dob_year","gender","salary"]
df=spark.createDataFrame(data,columns)
df.printSchema()
df.show(truncate=False)

from pyspark.sql.functions import split, col
df2 = df.select(split(col("name"),",").alias("NameArray")) \
    .drop("name")
df2.printSchema()
df2.show()

df.createOrReplaceTempView("PERSON")
spark.sql("select SPLIT(name,',') as NameArray from PERSON") \
    .show()
--------------------------------------------------------------------------------
-------
struct to map.py:

from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType, IntegerType
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
data = [ ("36636","Finance",(3000,"USA")),
    ("40288","Finance",(5000,"IND")),
    ("42114","Sales",(3900,"USA")),
    ("39192","Marketing",(2500,"CAN")),
    ("34534","Sales",(6500,"USA")) ]
schema = StructType([
     StructField('id', StringType(), True),
     StructField('dept', StringType(), True),
     StructField('properties', StructType([
         StructField('salary', IntegerType(), True),
         StructField('location', StringType(), True)
         ]))
     ])

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


#Convert struct type to Map
from pyspark.sql.functions import col,lit,create_map
df = df.withColumn("propertiesMap",create_map(
        lit("salary"),col("properties.salary"),
        lit("location"),col("properties.location")
        )).drop("properties")
df.printSchema()
df.show(truncate=False)
-------------------------------------------------------------------------------
struct type.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType,
IntegerType,ArrayType,MapType
from pyspark.sql.functions import col,struct,when

spark = SparkSession.builder.master("local[1]") \
                    .appName('SparkByExamples.com') \
                    .getOrCreate()

data = [("James","","Smith","36636","M",3000),
    ("Michael","Rose","","40288","M",4000),
    ("Robert","","Williams","42114","M",4000),
    ("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=data,schema=schema)
df.printSchema()
df.show(truncate=False)

structureData = [
    (("James","","Smith"),"36636","M",3100),
    (("Michael","Rose",""),"40288","M",4300),
    (("Robert","","Williams"),"42114","M",1400),
    (("Maria","Anne","Jones"),"39192","F",5500),
    (("Jen","Mary","Brown"),"","F",-1)
  ]
structureSchema = StructType([
        StructField('name', 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)
         ])

df2 = spark.createDataFrame(data=structureData,schema=structureSchema)
df2.printSchema()
df2.show(truncate=False)


updatedDF = df2.withColumn("OtherInfo",
    struct(col("id").alias("identifier"),
    col("gender").alias("gender"),
    col("salary").alias("salary"),
    when(col("salary").cast(IntegerType()) < 2000,"Low")
      .when(col("salary").cast(IntegerType()) < 4000,"Medium")
      .otherwise("High").alias("Salary_Grade")
  )).drop("id","gender","salary")

updatedDF.printSchema()
updatedDF.show(truncate=False)


""" Array & Map"""


arrayStructureSchema = StructType([
    StructField('name', StructType([
       StructField('firstname', StringType(), True),
       StructField('middlename', StringType(), True),
       StructField('lastname', StringType(), True)
       ])),
       StructField('hobbies', ArrayType(StringType()), True),
       StructField('properties', MapType(StringType(),StringType()), True)
    ])
--------------------------------------------------------------------------------
----------
timediff.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()


dates = [("1","2019-07-01 12:01:19.111"),
    ("2","2019-06-24 12:01:19.222"),
    ("3","2019-11-16 16:44:55.406"),
    ("4","2019-11-16 16:50:59.406")
    ]

df = spark.createDataFrame(data=dates, schema=["id","from_timestamp"])

from pyspark.sql.functions import *
df2=df.withColumn('from_timestamp',to_timestamp(col('from_timestamp')))\
  .withColumn('end_timestamp', current_timestamp())\
  .withColumn('DiffInSeconds',col("end_timestamp").cast("long") -
col('from_timestamp').cast("long"))
df2.show(truncate=False)

df.withColumn('from_timestamp',to_timestamp(col('from_timestamp')))\
  .withColumn('end_timestamp', current_timestamp())\
  .withColumn('DiffInSeconds',unix_timestamp("end_timestamp") -
unix_timestamp('from_timestamp')) \
  .show(truncate=False)

df2.withColumn('DiffInMinutes',round(col('DiffInSeconds')/60))\
  .show(truncate=False)

df2.withColumn('DiffInHours',round(col('DiffInSeconds')/3600))\
  .show(truncate=False)

#Difference between two timestamps when input has just timestamp

data= [("12:01:19.000","13:01:19.000"),
    ("12:01:19.000","12:02:19.000"),
    ("16:44:55.406","17:44:55.406"),
    ("16:50:59.406","16:44:59.406")]
df3 = spark.createDataFrame(data=data, schema=["from_timestamp","to_timestamp"])

df3.withColumn("from_timestamp",to_timestamp(col("from_timestamp"),"HH:mm:ss.SSS
")) \
   .withColumn("to_timestamp",to_timestamp(col("to_timestamp"),"HH:mm:ss.SSS"))
\
   .withColumn("DiffInSeconds", col("from_timestamp").cast("long") -
col("to_timestamp").cast("long")) \
   .withColumn("DiffInMinutes",round(col("DiffInSeconds")/60)) \
   .withColumn("DiffInHours",round(col("DiffInSeconds")/3600)) \
   .show(truncate=False)

#


df3 = spark.createDataFrame(
        data=[("1","07-01-2019 12:01:19.406")],
        schema=["id","input_timestamp"]
        )
df3.withColumn("input_timestamp",to_timestamp(col("input_timestamp"),"MM-dd-yyyy
HH:mm:ss.SSS")) \
    .withColumn("current_timestamp",current_timestamp().alias("current_timestamp
")) \
    .withColumn("DiffInSeconds",current_timestamp().cast("long") -
col("input_timestamp").cast("long")) \
    .withColumn("DiffInMinutes",round(col("DiffInSeconds")/60)) \
    .withColumn("DiffInHours",round(col("DiffInSeconds")/3600)) \
    .withColumn("DiffInDays",round(col("DiffInSeconds")/24*3600)) \
    .show(truncate=False)

#SQL

spark.sql("select unix_timestamp('2019-07-02 12:01:19') - unix_timestamp('2019-
07-01 12:01:19') DiffInSeconds").show()
spark.sql("select (unix_timestamp('2019-07-02 12:01:19') - unix_timestamp('2019-
07-01 12:01:19'))/60 DiffInMinutes").show()
spark.sql("select (unix_timestamp('2019-07-02 12:01:19') - unix_timestamp('2019-
07-01 12:01:19'))/3600 DiffInHours").show()
--------------------------------------------------------------------------------
---------------------------------
timestamp date.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()

df=spark.createDataFrame(
        data = [ ("1","2019-06-24 12:01:19.000")],
        schema=["id","input_timestamp"])
df.printSchema()


from pyspark.sql.functions import *

# Using Cast to convert Timestamp String to DateType
df.withColumn('date_type', col('input_timestamp').cast('date')) \
       .show(truncate=False)

# Using Cast to convert TimestampType to DateType
df.withColumn('date_type', to_timestamp('input_timestamp').cast('date')) \
  .show(truncate=False)

df.select(to_date(lit('06-24-2019 12:01:19.000'),'MM-dd-yyyy HH:mm:ss.SSSS')) \
  .show()

#Timestamp String to DateType
df.withColumn("date_type",to_date("input_timestamp")) \
  .show(truncate=False)

#Timestamp Type to DateType
df.withColumn("date_type",to_date(current_timestamp())) \
  .show(truncate=False)

df.withColumn("ts",to_timestamp(col("input_timestamp"))) \
  .withColumn("datetype",to_date(col("ts"))) \
  .show(truncate=False)
```

```
#SQL TimestampType to DateType
spark.sql("select to_date(current_timestamp) as date_type")
#SQL CAST TimestampType to DateType
spark.sql("select date(to_timestamp('2019-06-24 12:01:19.000')) as date_type")
#SQL CAST timestamp string to DateType
spark.sql("select date('2019-06-24 12:01:19.000') as date_type")
#SQL Timestamp String (default format) to DateType
spark.sql("select to_date('2019-06-24 12:01:19.000') as date_type")
#SQL Custom Timeformat to DateType
spark.sql("select to_date('06-24-2019 12:01:19.000','MM-dd-yyyy HH:mm:ss.SSSS')
as date_type")
--------------------------------------------------------------------------------
---------------------
types.py:

from pyspark.sql import SparkSession
from pyspark.sql.types import DataType
from pyspark.sql.types import StructType, StructField, StringType, ArrayType,
IntegerType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

from pyspark.sql.types import ArrayType,IntegerType
arrayType = ArrayType(IntegerType(),False)
print(arrayType.jsonValue())
print(arrayType.simpleString())
print(arrayType.typeName())


from pyspark.sql.types import MapType,StringType,IntegerType
mapType = MapType(StringType(),IntegerType())

print(mapType.keyType)
print(mapType.valueType)
print(mapType.valueContainsNull)

data = [("James","","Smith","36","M",3000),
    ("Michael","Rose","","40","M",4000),
    ("Robert","","Williams","42","M",4000),
    ("Maria","Anne","Jones","39","F",4000),
    ("Jen","Mary","Brown","","F",-1)
  ]

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


df = spark.createDataFrame(data=data,schema=schema)
df.printSchema()
df.show(truncate=False)
--------------------------------------------------------------------------------
-----
Udf.py:
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, udf
from pyspark.sql.types import StringType
```

```
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

columns = ["Seqno","Name"]
data = [("1", "john jones"),
    ("2", "tracey smith"),
    ("3", "amy sanders")]

df = spark.createDataFrame(data=data,schema=columns)

df.show(truncate=False)

def convertCase(str):
    resStr=""
    arr = str.split(" ")
    for x in arr:
       resStr= resStr + x[0:1].upper() + x[1:len(x)] + " "
    return resStr

""" Converting function to UDF """
convertUDF = udf(lambda z: convertCase(z))

df.select(col("Seqno"), \
    convertUDF(col("Name")).alias("Name") ) \
.show(truncate=False)


@udf(returnType=StringType())
def upperCase(str):
    return str.upper()

upperCaseUDF = udf(lambda z:upperCase(z),StringType())

df.withColumn("Cureated Name", upperCase(col("Name"))) \
.show(truncate=False)

""" Using UDF on SQL """
spark.udf.register("convertUDF", convertCase,StringType())
df.createOrReplaceTempView("NAME_TABLE")
spark.sql("select Seqno, convertUDF(Name) as Name from NAME_TABLE") \
     .show(truncate=False)

spark.sql("select Seqno, convertUDF(Name) as Name from NAME_TABLE " + \
          "where Name is not null and convertUDF(Name) like '%John%'") \
     .show(truncate=False)

""" null check """

columns = ["Seqno","Name"]
data = [("1", "john jones"),
    ("2", "tracey smith"),
    ("3", "amy sanders"),
    ('4',None)]

df2 = spark.createDataFrame(data=data,schema=columns)
df2.show(truncate=False)
df2.createOrReplaceTempView("NAME_TABLE2")

spark.udf.register("_nullsafeUDF", lambda str: convertCase(str) if not str is
None else "" , StringType())

spark.sql("select _nullsafeUDF(Name) from NAME_TABLE2") \
     .show(truncate=False)

spark.sql("select Seqno, _nullsafeUDF(Name) as Name from NAME_TABLE2 " + \
          " where Name is not null and _nullsafeUDF(Name) like '%John%'") \
     .show(truncate=False)
--------------------------------------------------------------------------------
---------
Unoin.py:

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = [("James","Sales","NY",90000,34,10000), \
    ("Michael","Sales","NY",86000,56,20000), \
    ("Robert","Sales","CA",81000,30,23000), \
    ("Maria","Finance","CA",90000,24,23000) \
  ]

columns= ["employee_name","department","state","salary","age","bonus"]
df = spark.createDataFrame(data = simpleData, schema = columns)
df.printSchema()
df.show(truncate=False)

simpleData2 = [("James","Sales","NY",90000,34,10000), \
    ("Maria","Finance","CA",90000,24,23000), \
    ("Jen","Finance","NY",79000,53,15000), \
    ("Jeff","Marketing","CA",80000,25,18000), \
    ("Kumar","Marketing","NY",91000,50,21000) \
  ]
columns2= ["employee_name","department","state","salary","age","bonus"]

df2 = spark.createDataFrame(data = simpleData2, schema = columns2)

df2.printSchema()
df2.show(truncate=False)

unionDF = df.union(df2)
unionDF.show(truncate=False)
disDF = df.union(df2).distinct()
disDF.show(truncate=False)

unionAllDF = df.unionAll(df2)
unionAllDF.show(truncate=False)
--------------------------------------------------------------------------------
----------
unix time.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()

inputData = [("2019-07-01 12:01:19",
            "07-01-2019 12:01:19",
            "07-01-2019")]
columns=["timestamp_1","timestamp_2","timestamp_3"]
df=spark.createDataFrame(
        data = inputData,
        schema = columns)
df.printSchema()
df.show(truncate=False)

from pyspark.sql.functions import *
```

```
df2 = df.select(
      unix_timestamp(col("timestamp_1")).alias("timestamp_1"),
      unix_timestamp(col("timestamp_2"),"MM-dd-yyyy
HH:mm:ss").alias("timestamp_2"),
      unix_timestamp(col("timestamp_3"),"MM-dd-yyyy").alias("timestamp_3"),
      unix_timestamp().alias("timestamp_4")
   )
df2.printSchema()
df2.show(truncate=False)

df3=df2.select(
    from_unixtime(col("timestamp_1")).alias("timestamp_1"),
    from_unixtime(col("timestamp_2"),"MM-dd-yyyy
HH:mm:ss").alias("timestamp_2"),
    from_unixtime(col("timestamp_3"),"MM-dd-yyyy").alias("timestamp_3"),
    from_unixtime(col("timestamp_4")).alias("timestamp_4")
  )
df3.printSchema()
df3.show(truncate=False)

--------------------------------------------------------------------------------
----------
update columns.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()

inputData = [("2019-07-01 12:01:19",
            "07-01-2019 12:01:19",
            "07-01-2019")]
columns=["timestamp_1","timestamp_2","timestamp_3"]
df=spark.createDataFrame(
        data = inputData,
        schema = columns)
df.printSchema()
df.show(truncate=False)

from pyspark.sql.functions import *
df2 = df.select(
      unix_timestamp(col("timestamp_1")).alias("timestamp_1"),
      unix_timestamp(col("timestamp_2"),"MM-dd-yyyy
HH:mm:ss").alias("timestamp_2"),
      unix_timestamp(col("timestamp_3"),"MM-dd-yyyy").alias("timestamp_3"),
      unix_timestamp().alias("timestamp_4")
   )
df2.printSchema()
df2.show(truncate=False)

df3=df2.select(
    from_unixtime(col("timestamp_1")).alias("timestamp_1"),
    from_unixtime(col("timestamp_2"),"MM-dd-yyyy
HH:mm:ss").alias("timestamp_2"),
    from_unixtime(col("timestamp_3"),"MM-dd-yyyy").alias("timestamp_3"),
    from_unixtime(col("timestamp_4")).alias("timestamp_4")
  )
df3.printSchema()
df3.show(truncate=False)

--------------------------------------------------------------------------------
----
```

```
when other wise.py:

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = [("James","","Smith","36636","M",60000),
        ("Michael","Rose","","40288","M",70000),
        ("Robert","","Williams","42114","",400000),
        ("Maria","Anne","Jones","39192","F",500000),
        ("Jen","Mary","Brown","","F",0)]

columns = ["first_name","middle_name","last_name","dob","gender","salary"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)

# Using when otherwise
from pyspark.sql.functions import col, when
df2 = df.withColumn("new_gender", when(col("gender") == "M","Male")
                                 .when(col("gender") == "F","Female")
                                 .otherwise("Unknown"))
df2.show(truncate=False)

df22=df.select(col("*"), when(col("gender") == "M","Male")
      .when(col("gender") == "F","Female")
      .otherwise("Unknown").alias("new_gender")).show(truncate=False)

# Using case when
from pyspark.sql.functions import expr
df3 = df.withColumn("new_gender", expr("case when gender = 'M' then 'Male' " +
                       "when gender = 'F' then 'Female' " +
                       "else 'Unknown' end"))
df3.show(truncate=False)

#Using case when
df4 = df.select(col("*"), expr("case when gender = 'M' then 'Male' " +
                       "when gender = 'F' then 'Female' " +
                       "else 'Unknown' end").alias("new_gender"))
df4.show(truncate=False)

data2 = [(66, "a", "4"), (67, "a", "0"), (70, "b", "4"), (71, "d", "4")]
df5 = spark.createDataFrame(data = data2, schema = ["id", "code", "amt"])


df5.withColumn("new_column", when(col("code") == "a" | col("code") == "d", "A")
      .when(col("code") == "b" & col("amt") == "4", "B")
      .otherwise("A1")).show()
--------------------------------------------------------------------------------
------
windows function.py;

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData = (("James", "Sales", 3000), \
    ("Michael", "Sales", 4600),  \
    ("Robert", "Sales", 4100),   \
    ("Maria", "Finance", 3000),  \
    ("James", "Sales", 3000),    \
    ("Scott", "Finance", 3300),  \
    ("Jen", "Finance", 3900),    \
    ("Jeff", "Marketing", 3000), \
    ("Kumar", "Marketing", 2000),\
    ("Saif", "Sales", 4100) \
  )

columns= ["employee_name", "department", "salary"]

df = spark.createDataFrame(data = simpleData, schema = columns)

df.printSchema()
df.show(truncate=False)

from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
windowSpec  = Window.partitionBy("department").orderBy("salary")

df.withColumn("row_number",row_number().over(windowSpec)) \
    .show(truncate=False)

from pyspark.sql.functions import rank
df.withColumn("rank",rank().over(windowSpec)) \
    .show()

from pyspark.sql.functions import dense_rank
df.withColumn("dense_rank",dense_rank().over(windowSpec)) \
    .show()

from pyspark.sql.functions import percent_rank
df.withColumn("percent_rank",percent_rank().over(windowSpec)) \
    .show()

from pyspark.sql.functions import ntile
df.withColumn("ntile",ntile(2).over(windowSpec)) \
    .show()

from pyspark.sql.functions import cume_dist
df.withColumn("cume_dist",cume_dist().over(windowSpec)) \
   .show()

from pyspark.sql.functions import lag
df.withColumn("lag",lag("salary",2).over(windowSpec)) \
      .show()

from pyspark.sql.functions import lead
df.withColumn("lead",lead("salary",2).over(windowSpec)) \
    .show()

windowSpecAgg  = Window.partitionBy("department")
from pyspark.sql.functions import col,avg,sum,min,max,row_number
df.withColumn("row",row_number().over(windowSpec)) \
  .withColumn("avg", avg(col("salary")).over(windowSpecAgg)) \
  .withColumn("sum", sum(col("salary")).over(windowSpecAgg)) \
  .withColumn("min", min(col("salary")).over(windowSpecAgg)) \
  .withColumn("max", max(col("salary")).over(windowSpecAgg)) \
  .where(col("row")==1).select("department","avg","sum","min","max") \
  .show()
--------------------------------------------------------------------------------
---------
with column.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField, StringType,IntegerType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = [('James','','Smith','1991-04-01','M',3000),
  ('Michael','Rose','','2000-05-19','M',4000),
  ('Robert','','Williams','1978-09-05','M',4000),
  ('Maria','Anne','Jones','1967-12-01','F',4000),
  ('Jen','Mary','Brown','1980-02-17','F',-1)
]

columns = ["firstname","middlename","lastname","dob","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.printSchema()
df.show(truncate=False)

df2 = df.withColumn("salary",col("salary").cast("Integer"))
df2.printSchema()
df2.show(truncate=False)

df3 = df.withColumn("salary",col("salary")*100)
df3.printSchema()
df3.show(truncate=False)

df4 = df.withColumn("CopiedColumn",col("salary")* -1)
df4.printSchema()

df5 = df.withColumn("Country", lit("USA"))
df5.printSchema()

df6 = df.withColumn("Country", lit("USA")) \
   .withColumn("anotherColumn",lit("anotherValue"))
df6.printSchema()

df.withColumnRenamed("gender","sex") \
  .show(truncate=False)

df4.drop("CopiedColumn") \
.show(truncate=False)

dataStruct = [(("James","","Smith"),"36636","M","3000"), \
      (("Michael","Rose",""),"40288","M","4000"), \
      (("Robert","","Williams"),"42114","M","4000"), \
      (("Maria","Anne","Jones"),"39192","F","4000"), \
      (("Jen","Mary","Brown"),"","F","-1") \
]

schemaStruct = StructType([
        StructField('name', StructType([
             StructField('firstname', StringType(), True),
             StructField('middlename', StringType(), True),
             StructField('lastname', StringType(), True)
             ])),
          StructField('dob', StringType(), True),
         StructField('gender', StringType(), True),
         StructField('salary', StringType(), True)
         ])


df7 = spark.createDataFrame(data=dataStruct, schema = schemaStruct)
df7.printSchema()
df7.show(truncate=False)
```

```
"""
columns = ["name","address"]
data = [("Robert, Smith", "1 Main st, Newark, NJ, 92537"), \
        ("Maria, Garcia","3456 Walnut st, Newark, NJ, 94732")]
dfFromData = spark.createDataFrame(data=data, schema = schema)
newDF = dfFromData.map(f=>{
nameSplit = f.getAs[String](0).split(",")
addSplit = f.getAs[String](1).split(",")

(nameSplit(0),nameSplit(1),addSplit(0),addSplit(1),addSplit(2),addSplit(3))
    })
finalDF = newDF.toDF("First Name","Last Name",
             "Address Line1","City","State","zipCode")
finalDF.printSchema()
finalDF.show(false)
--------------------------------------------------------------------------------
-------------
pandas.py:

import pandas as pd
data = [["James","","Smith",30,"M",60000],
        ["Michael","Rose","",50,"M",70000],
        ["Robert","","Williams",42,"",400000],
        ["Maria","Anne","Jones",38,"F",500000],
        ["Jen","Mary","Brown",45,None,0]]
columns = ['First Name', 'Middle Name','Last Name','Age','Gender','Salary']

# Create the pandas DataFrame
pandasDF = pd.DataFrame(data=data, columns=columns)

# print dataframe.
print(pandasDF)

#Outputs below data on console

pdCount=pandasDF.count()
print(pdCount)

print(pandasDF.max())
print(pandasDF.mean())
--------------------------------------------------------------------------------
-----------
schema.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.functions import to_timestamp
from pyspark.sql.types import StructType, StructField, StringType,
IntegerType,DateType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

schema = StructType([
            StructField("city", StringType(), True),
            StructField("dates", StringType(), True),
            StructField("population", IntegerType(), True)])

dates = ["1991-02-25","1998-05-10", "1993/03/15", "1992/07/17"]
cities = ['Caracas', 'Ccs', '   São Paulo   ', '~Madrid']
population = [37800000, 19795791, 12341418, 6489162]

        # Dataframe:
df = spark.createDataFrame(list(zip(cities, dates, population)), schema=schema)
```

```
df.show(truncate=False)
--------------------------------------------------------------------------------
------
repartition.py:

from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
          .appName('SparkByExamples.com') \
          .getOrCreate()

df=spark.read.option("header",True) \
        .csv("C:/apps/sparkbyexamples/src/pyspark-examples/resources/simple-
zipcodes.csv")

newDF=df.repartition(3)
print(newDF.rdd.getNumPartitions())

newDF.write.option("header",True).mode("overwrite") \
        .csv("/tmp/zipcodes-state")

df2=df.repartition(4,"state")
df2.write.option("header",True).mode("overwrite") \
   .csv("/tmp/zipcodes-state-3states")

df3=df.repartition("state")
df3.write.option("header",True).mode("overwrite") \
   .csv("/tmp/zipcodes-state-allstates")
--------------------------------------------------------------------------------
-------
Timediff.py:

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, round
from pyspark.sql.functions import to_timestamp, current_timestamp
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,
LongType

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

schema = StructType([
            StructField("input_timestamp", StringType(), True)])

dates = ['2019-07-01 12:01:19.111',
    '2019-06-24 12:01:19.222',
    '2019-11-16 16:44:55.406',
    '2019-11-16 16:50:59.406']

df = spark.createDataFrame(list( zip(dates)), schema=schema)

df.withColumn('input_timestamp',to_timestamp(col('input_timestamp')))\
  .withColumn('current_timestamp',
current_timestamp().alias('current_timestamp'))\
  .withColumn('DiffInSeconds',current_timestamp().cast(LongType()) -
col('input_timestamp').cast(LongType()))\
  .withColumn('DiffInMinutes',round(col('DiffInSeconds')/60))\
  .withColumn('DiffInHours',round(col('DiffInSeconds')/3600))\
  .withColumn('DiffInDays',round(col('DiffInSeconds')/24*3600))\
  .show()
================================================================================
=================
```

Triggers mean:
----
triggers are used to schedule and execute pipelines based on a predefined recurrence or an event.

Triggers are used to automate the execution of pipelines without manual intervention.

There are several types of triggers available in ADF:

Schedule trigger: It allows you to run pipelines at a specific time, day, week, or month.

Tumbling window trigger: It allows you to run pipelines at specific intervals, such as every hour or every day.

Event-based trigger: It allows you to run pipelines based on an event, such as a file arriving in a storage account.

Custom trigger: It allows you to create a trigger using your own code or REST API calls.

==
==