DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SCHOOL OF TECHNOLOGY

PANDIT DEENDAYAL ENERGY UNIVERSITY SESSION 2023-24

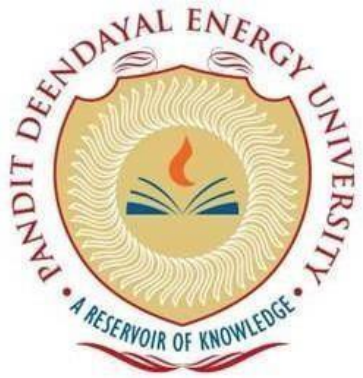
SUBMITTED BY

NAME	**:**	**Harsh Shah**
ROLL NO.	**:**	**21BCP359**
DIVISION	**:**	**6**
GROUP	**:**	**G11**
COURSE NAME	**:**	**Big Data Lab**
COURSE CODE	**:**	**23CP309P**

****

PANDIT DEENDAYAL ENERGY UNIVERSITY

Raysan, Gandhinagar – 382007, Gujarat, India



Certificate

This is to certify that

Mr./~~Ms.~~ Harsh Shah Roll no. 21BCP359

of 3rd Year B. TECH Degree in Computer Engineering has satisfactorily completed his/her term work in Big Data Analytics Lab subject during the

semester from January 2024 to

May 2024 at School of Technology, PDEU.

Date of Submission: April 27, 2024

Signature:

Faculty In-charge Head of Department

INDEX

Name : Harsh Shah

Roll No. : 21BCP359

(G11 Div6)

LAB 1 & 2

```
%scala
van nun = List(1,2,3,4)
```

```
num: List[Int] = List(1, 2, 3, 4)
```

```
%scala
num.head
```

```
res0: Int = 1
```

Start coding or generate with AI.

```
%scala
num.tail
```

```
nes1: List[Int] = List(2, 3, 4)
```

```
%scala
num.sum
```

```
rer2: Int = 10
```

```
%scala
num.take(3) //Important function
```

```
Yes3: List[Int] = List(1, 2, 3)
```

```
%scala
num.take(-1)
```

```
nes4: List[Int] = List()
```

```
%scala
var ani = List(1,1,1,12,2,2,2,2,2)
```

```
ani: List[Int] = List(1, 1, 1, 12, 2, 2, 2, 2, 2)
```

```
%scala
ani.distinct
```

```
Yes5: List[Int] = List(1, 12, 2)
```

```
%scala
ant(5)
```

```
rcs6: Int = 2
```

```
%scala
ani(-2)
```

```
        at scala.collection.LinearSeqOptimized.apply(LinearSeqOptimized.scala:67)
        at scala.collection.LinearSeqOptimized.apply$(LinearSeqOptimized.scala:65)
        at scala.collection.immutable.List.apply(List.scala:91)
        at  $1inec64bee687-F-6449b59e9-F-2ae0+e9a1d6c82.$read$$iw$$iw$$iw$$iw$$iw$$iw.‹init›(com
mand-1651621763864477:1)
        at  $linec64bee687f6449b59e9f2ae8fe9ald6c82.$read$$iw$$iw$$iw$$iw$$iw.‹init›(command
-1651821763864477:45)
        at $linec64bee687f6449b59e9f2aeBfe9ald6c82.$read$$iw$$iw$$iw$$iw.‹init›(command-165
1021763864477:47)
        at $linec64bee687f6449b59e9f2ae0fe9ald6c82.$read$$iw$$iw$$iw.<init>(command-1651021
763864477:49)
        at $linec64bee687f6449b59e9f2ae0fe9ald6c82.$read$$iw$$iw.<init>(command-16518217638
64477:51)
        at  $linec64bee687f6449b59e9f2ae0fe9ald6c82.$read$$iw.‹init›(command-165102176386447
7:53)
        at  $linec64bee687f6449b59e9f2ae0fe9ald6c82.$read.‹init›(command-1651021763864477:5

        at  $linec64bee687f6449b59e9f2aeBfe9ald6c82.$read$.‹init›(command-1651B21763864477:5
9)
        at  $linec64bee687f6449b59e9f2ae0fe9ald6c82.$read$.<clinit›(command-165102176386447
7)
        at $linec64bee687f6449b59e9f2ae0fe9ald6c82.$eval$.$print$lzycompute(‹notebooks:7)
        at $linec6Obee687f6449b59e9f2ae0fe9ald6c82.$eval$.$print(<notebook>:6)
        at $linec64bee687f6449b59e9f2ae8fe9ald6c82.$eval.$print(‹notebook>)
        at sun.reflect.NativeMethodAccessorImpl.invokeB(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.jav
a:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at scala.tools.nsc.interpreter.IMain$ReadEvalPrint.call(IMain.scala:747)
        at scala.tools.nsc.interpreter.IMain$Request.loadAndRun(IMain.scala:1020)
        at scala.tools.nsc.interpreter.IMain.$anonfun$interpret$1(IMain.scala:568)
        at scala.reflect.internal.util.ScalaClassLoader.asContext(ScalaClassLoader.scala:3
6)
        at scala.reflect.internal.uti1.ScalaClassLoader.asContext$(ScalaClassLoader.scala:1
16)
        at scala.reflect.internal.uti1.AbstractFileClassLoader.asContext(AbstractFileClassL
oader.scala:41)
        at scala.tools.nsc.interpreter.IMain.loadAndRunReq$1(IMain.scala:567)
        at scala.tools.nsc.interpreter.IMain.interpret(IMain.scala:594)
        at scala.tools.nsc.interpreter.IMain.interpret(IMain.scala:564)
        at com.databricks.backend.daemon.driver.DriverILoop.execute(DriverILoop.scala:223)
        at com.databricks.backend.daemon.driver.ScalaDriverLocal.$anonfun$repl$1(ScalaDrive
rLocal.scala:227)
        at scala.runtime.java8.JFunction0$mcV$sp.apply(JFunction0$mcV$sp.java:23)
        at com.databricks.backend.daemon.driver.DriverLocal$TrapExitInternal$.trapExit(Driv
erLocal.scala:1283)
        at com.databricks.backend.daemon.driver.DriverLocal$TrapExit$.apply(DriverLocal.sca
la:1236)
        at com.databricks.backend.daemon.driver.ScalaDriverLocal.repl(ScalaDriverLocal.scal
a:227)
        at com.databricks.backend.daemon.driver.DriverLocal.$anonfun$execute$24(DriverLoca
```

```
%scala
  ant(4)=2
```

```
%scala
num.reverse
```

```
res10: List[Int] = List(4, 3, 2, 1)
```

```
%scala
ani.min
```

```
resll: Int = 1
```

```
%scala
ant.1sEmpty
```

```
res12: Boolean = false
```

```
%scala
var arr1  = Array(10,11,12,13,14,15,16)
```

```
arr1: Array[Int] = Array(10, 11, 12, 13, 14, 15, 16)
```

```
%scala
var  arr2  = Array(1.2,3.2,4,5,6,7)
```

```
arr2: Array[Double] = Array(1.2, 3.2, 4.0, 5.8, 6.0, 7.0)
```

```
%scala
val lang = Array("Scala","Python","Java")
```

```
lang: Array[String] = Array(Scala, Python, Java)
```

```
%scala
lang.tail
```

```
res13: Array[String] = Array(Python, Java)
```

```
%scala
lang.head
```

```
res14: String = Scala
```

```
%scala
arr1(3)=30
```

```
%scala
arr1
```

```
res18: Array[Int] = Array 10, 11, 12, 30, 14, 15, 16)
```

```
%scala
import scala.collection.mutable.ArrayBuffer
```

```
import scala.collection.mutable.ArrayBuffer
```

```
%scala
var car = new ArrayBuffer[String]()
```

```
car: scala.collection.mutable.ArrayBuffer[String] = ArnayBuffer()
```

```
%scala
car.append("car1")
car.append("car2")
car.append("car3")
car.append("car4")
```

```
%scala
car
```

```
res21: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(car1, car1, car2, car3, car4)
```

```
%scala
car +="car4"
```

```
res22: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(car1, car1, car2, car3, car4, car4)
```

```
%scala
car
```

```
res23: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(car1, carl, car2, car3, car4, car4)
```

```
%scala
car.length
```

```
res24: Int = 6
```

```
%scala
car . t rlmEnd (2)
```

```
%scala
car
```

```
res26: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(carl, carl, car2, car3)
```

```
%scala
car• . I rifnst a rt (1)
```

```
%scala
car
```

```
res28: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(car1, car2, car3)
```

```
%scala
car+="car4"
car+="car5"
```

```
res38: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(carl, car2, car3, car4, car4, car5)
```

```
%scala
car.insert(4,“Porche“)
```

```
%scala
car
```

```
res32: scala.collection.mutable.ArrayBuffer[String] = ArrayBuffer(car1, car2, car3, car4, Porche, car4, car5)
```

```
%scala
//Map Transonmation Program

//Each variable “x“ is transformed to “x2“
arr1.map(x=>x*x)
```

```
res33: Array[Int] = Array(100, 121, 144, 900, 196, 225, 256)
```

```
%scala
// Program for cubing , sqrt and +3 add

arrl.map(x=>x*x*x)
```

```
res34: Array[Int] = Array(1000, 1331, 1728, 27000, 2744, 3375, 4096)
```

```
%scala
arr1.map(x=>x+3)
```

```
res35 : Array[ Int] = Array(13, 14, 15, 33, 17, 18, 19)
```

```
%scala
import Math.sqrt
arrl.map(x=>Math.sqrt(x))
```

```
Import Nath . sqrt
res38: Array[Double] = Array(3.1622776601683795, 3.3166247903554, 3.4641016151377544, 5.477225575051661, 3.7416573867739413, 3.872983346207417,
```

```
%scala
arr1.map(y=>y*(y-1))
```

```
res39: Array[Int] = Array(90, 110, 132, 876, 182, 210, 240)
```

```
$scala
val b = arrl.nap(x=›x+1) .nap(b=›b*b)
```

```
b: Array[Int] = Array(121, 144, 169, 961, 225, 256, 289)
```

```
%scala
val fruit = List("Orange","Banana","Apple","Pineapple")
```

```
fruit: List[String] = List(Orange, Banana, Apple, Pineapple)
```

```
%scala
//program for (Key,value) operation for calculating word count
fruit.map(x=>(x,x.length))
```

```
res40: List[(String, Int)] = List((Orange,6), (Banana,6), (Apple,5), (Pineapple,9))
```

```
%scala
fruit.filter(x=>x.length>5)
```

```
res41: L1st[St ring] = List(Orange, Banana, Pineapp1e)
```

```
%scala
//Create one list "ratings" of type float of 5 numbers * 10 filter marks between 60 to 80 map to divide by 10

va1 rat ings = List(2.3,4.5,5.4,7.6,8.9)
```

```
ratings: L1st[Doub1e] = L1st(2.3, 4.5, 5.4, 7.6, 8.9)
```

```
   a
val l lten = natings.map(x=>x*10)
```

```
multen: List[Double] = List(23.0, 45.B, 54.8, 76.0, 89.0)
```

```
   a
val g ade = multen.filten(x=›x›60&&x‹80).map(x=›x/10)
```

```
grade: List[Double] = List(7.6)
```

```
%scala
grade

//end
```

```
res43: List[Double] = List(7.6)
```

```
%scala
//function implementation

def add(a:Double=100,b:Double=200):Double ={
  var sum:Double = 0
  sum= a+b
  return  sum
```

```
add: (a: Double, b: Double)Double
```

```
%scala
add (33, 55)
```

```
res45: Double = 88.0
```

```
%scala
//COnditional Statements

var x = 10
var b = if(x< 3){
  println("less than 3")
} else{
  println("Greater than 3")
```

```
Greater than 3
x: Int = 10
b: Unit = ()
```

```
%scala
def squ(a:Double=2):Double=

  return  a*a
```

```
squ: (a: Double)Double
```

```
%scala
squ(2)
```

```
res46: Double = 4.8
```

```
%scala
//Nested function call

def sqqu(a:Double,b:Double):Double={
  return squ(a)+squ(b)
```

```
sqqu: (a: Doubbe, b: Double)Double
```

```
%scala
sqqu(2, 4)
```

```
res48: Double = 20.0
```

```
%scala
//Loops

for(i<-1 to 10)

  println(i)
```

```
2
3

5
6
7
8
9
10
```

```
%scala
var x= Array(
       Array(1, 2, 3),
       Array(4, 5,6),
       Array(7, 8, 9)


var y= Array(
     Array(9, 8, 7),
     Array(7, 6, 5),
     Array(4, 2, 1)

var nes=Annay.ofDim[Int](3,3)
```

```
      x: Array[Array[Int]] = Array(Array(1, 2, 3), Array(4, 5, 6), Array(7, 8, 9))
      y: Array[Array[Int]] = Array(Array(9, 8, 7), Array(7, 6, 5), Array(4, 2, 1))
      res: Array[Array[Int]] = Array(Array(0, 0, 0), Array(0, 0, 0), Array(0, 0, 0))
```

```
%scala
//Matrix Multiplication
for(i‹-0 to 2){
  for(j‹-0 to 2){
    res(i)(j)=0
    for(k‹-0 to 2){
      nes(i)(j)+=x(i)(k)*y(k)(j)
```

```
%scala
 res
```

```
      res57: Array[Array[Int]] = Array(Array(35, 26, 20), Arnay(95, 74, 59), Array(155, 122, 98))
```

```
%scala
//Time related info code

def  time():Long=(
  pr1ntln(”Inside  T1me  Function“)
  return System.nanoTime()


def  exec(t:Long):Long={
  println(”Inside Exec Function “)
  println(”Time  :“+ t)
  println(”Exiting from time function“)
  return t


println(“Main Function :“+ exec(time()))
```

```
       Inside  T1me  Func t ton
       Inside    Exec    Function
       T1me : 82223636251778
       Exiting  -From  time  -L-unction
       Main  Functlon  : 82223636251778
       time:  ( ) Long
       exec:  (I:    Long) Long
```

```
%scala
var i=1
while(i‹10){
  println(i)
```

Name : Harsh Shah

Roll No. : 21BCP359

(G11 Div6)

LAB 3

```
%scala
/* parallelize, map, filter, contains , sample, uuion, intersection, distinct- transformations function

 => parallelize typically refers to the process of parallelizing operations on collections using parallel computing techniques. Parallelizati<

 1. When its executed it creates - RDD (Resilient Distributed Dataset - concept in Spark), when there is any failure this RDD take care of
  no data loss occurs.
 2. RDD generally consists of data i.e used frequently.
 Ex - when we create database one RDD is created and so on.
 3. Creations of RDD - 3 ways.
 4. Every transformation has result displayed when action is performed = Job scheduling - DAG scheduler - Lazy evaluation.
 5. all RDDs creates a DAG
 6. spank context can only be one and it's default created and it's panent of all , we can't cneate oun own sc
 7. spark's base lang is Scala.
 8. saveAsTextFile - saves a file in HDFS with name -> part_00000 -> default file name in Hadoop

 9. Loop-iterative methods (fine grain mode) vs Block approach (course grain mode)  - for allocation of data

 10. Data dependency - delays due to extra computation of extra data to wonk - data latency comes
  hence data d1str1bution 1s 1np.

 11. Self-scheduling mechanism - those processing elements which have high computing power , then they will take data on their own from s/m.
  hence the fast computing elements will take data at their own speed until data is present to make max utilization.

 12. Hauffman and adaptive hauffmann - compression algos
"f

val a = sc.parallelize(List("A","B","C","D")); //sc= spark context


      a: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at command-1828


%scala
val b = a.map(x=>(x,1));

      b: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[1] at map at command-1820806844


%scala
//collect    action cmd - Used to get the action(result) done by the job created by transformations
b.collect

      res8: Array[(String, Int)j = Array((A,1), (B,1), (C,1), (D,1))


%scala
//Shortcut  method  -F-or  the   above  transf-ormatlon

val b = a.map((_,1))

      b: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[2] at map at command-1820806844


%scala
val a = sc.parallelize(List("Apple","Banana","0range","Mango"));

      a: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[3] at parallelize  at command-1820


%scala
val b = a.map(x=>(x,x.length));

      b: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[4] at map at command-1820806844


%scala
```

```
b.collect

      res1: Array[(String, Int)] = Array((Apple,5), (Banana,6), (Orange,6), (Mango,5))
```

```
a
val a  sc.parallelize(List(1,2,3,4,5)).map(x=›List(x,x,x))
```

```
a: org.apache.spark.rdd.RDD[List[Int]] = MapPartitionsRDD[6] at map at command-18288068441239
```

```
%scala
a.collect.
```

```
res2: Array[List[Int]] = Array(List(1, 1, 1), List(2, 2, 2), List(3, 3, 3), List(4, 4, 4), Li
```

```
%scala
val a = sc.parallelize(List(1,2,3,4,5)).flatMap(x=>List(x,x))
```

```
a: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[8] at flatMap at command-182B88684412398:
```

```
%scala
a.collect
```

```
res3: Array[Int] = Array(1, 1, 2, 2, 3, 3, 4, 4, 5, 5)
```

```
%scala
val rdda = sc.parallelize(List(“aaaa“,“bbbb“,“cccc”))
```

```
rdda: org.apache.spark.rdd.ADD[String] = ParallelCollectionRDD[9] at parallelize at command-1
```

```
%scala
//checking whether the element exists or not

rdda.filter(_.equals(“aaaa“)).collect
```

```
res6: Array[String] = Array(aaaa)
```

```
%scala
//Checking whether the character is present in list or not
rdda.filter(_.contains(“a“)).collect
```

```
res7: Array[String] = Array(aaaa)
```

```
%scala
val a  = sc.parallelize(List((“Mumbai“,2000),(“Delhi”,3000),("Chennai“,1000),(“6ujarat“,7000)))
```

```
a: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[0] at parallelize at comma
```

```
%scala
// _2 = value , _1 = key

a.filter(_._2.equals(70B0)).collect
```

```
res4: Array[(String, Int)] = Array((Gujarat,7880))
```

```
%scala
a.filter(_._2>1000).collect
```

```
resll: Array[(String, Int)] = Array((Mumbai,2000), (Delhi,3000), (Gujarat,7800))
```

```
%scala
a.filter(_._2> 1000 ).filter(_._2 ‹ 6000).collect
```

```
res7: Array[(String, Int)] = Array((Mumbai,2000), (Delhi,3008))
```

```
%scala
/*
function - sample(true/false, fraction,seed)
  1. true have repetition
  2. false dou't have repetition
  3. fraction 0 to 1 , no.of sub-sample to be used from the sample in o/p
  4. seed - randomization - result same if it has the same value - could be any number -
     same seed value results in same set of value
’
```

```
var s=sc.parallelize(1 to 100)
var a=s.sample(true,1.2)
a.count
```

```
s: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[22] at parallelize at command-182088
684412486:10
a: org.apache.spark.rdd.RDD[Int] = PartitionwiseSampledRDD[23] at sample at command-182080684
```

```
%scala
val a = sc.parallelize(1 to 1000)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[21] at parallelize at command-182088
```

```
%scala
// false - no repetition allowed , 0.2 - 20% of data of 1000 , 5 - seed value fon randomizing

a.sample(false,0.2,24).collect
```

```
res11: Array[Int] = Array(7, 10, 14, 16, 17, 19, 21, 27, 29, 32, 35, 46, 48, 54, 61, 63, 77,
86, 83, 84, 85, 86, 87, 88, 92, 188, 111, 114, 115, 116, 127, 144, 149, 151, 161, 169, 172, 1
91, 199, 204, 207, 217, 225, 236, 243, 256, 252, 254, 259, 262, 269, 272, 274, 275, 283, 285,
289, 299, 300, 307, 309, 322, 327, 332, 333, 342, 343, 345, 349, 364, 371, 386, 393, 397, 40
5, 466, 467, 411, 413, 415, 417, 419, 420, 425, 439, 442, 451, 453, 454, 456, 464, 466, 467,
470, 473, 477, 485, 496, 495, 498, 511, 517, 518, 519, 521, 525, 532, 542, 546, 549, 551, 55
```

```
%scala
a.sample(false,0.1,1022).collect
```

```
res19: Array[Int] = Array(21, 23, 44, 48, 55, 59, 66, 67, 71, 82, 106, 113, 115, 128, 123, 12
8, 135, 153, 162, 172, 214, 247, 249, 255, 269, 287, 322, 326, 330, 355, 359, 385, 388, 399,
468, 429, 432, 438, 465, 478, 484, 487, 496, 503, 510, 514, 541, 543, 559, 564, 566, 571, 57
```

```
%scala
val a = sc.parallelize(List(1,1,1,2,1,2,1,2))
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[31] at parallelize at command-182080
```

```
%scala
a.sample(true,0.5,15).collect
```

```
res21: Array[Int] = Array(1, 1, 2, 2, 2)
```

```
%scala
val a = sc.parallelize(1 to 7)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[33] at parallelize at command-182088
```

```
%scala
val b = sc.parallelize(3 to 12)
```

```
b: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[50] at parallelize at command-182080
```

```
%scala
a.union(b).collect
```

```
res28: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
```

```
%scala
a.intersection(b).collect
```

```
res29: Array[Int] = Array(3, 4, 5, 6, 7)
```

```
%scala
a.union(b).distinct.collect
```

```
res30: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
```

```
%scala
// 3 - distribution of data on 3 cores , by default - no. of cores in s/m

val a = sc.parallelize(1 to 9,3)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[62] at parallelize at command-182088
```

```
%scala
/*
Block mode distribution starting no. 1 , 4 , 7
iterator -  used to check the indexes of starting of partitions on cores

OS - fork and join in data distribution by parent to child process  and concept of shared memory
*/

a.mapPartitions(x=›List(x.next).iterator).collect
```

```
res31: Array[Int] = Array(1, 4, 7)
```

```
%scala
//Getting the index of partition of each element

def pra(index:Int, iter:Iterator[(lut)]) : Iterator[String]

  Iter.toLIst.map(x=›x+" "+1ndex) . iterator
```

```
pra: (index: Int, iter: Iterator[Int])Iterator[String]
```

```
%scala
val a = sc.parallelize(List(1,2,3,4,5,6),4)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[66] at parallelize at command-182080
```

```
%scala
a.mapPartitiousWithIndex(pra).collect
```

```
res34: Array[String] = Array(1 0, 2 1, 3 1, 4 2, 5 3, 6 3)
```

```
%scala
val v = sc.parallelize(1 to 10)
```

```
v: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[68] at parallelize at command-182088
```

```
%scala
//1. square of even numbers
val even = v.filter(x=>(x%2==0)).map(x=›x*x).sum
```

```
odd: Double = 220.6
```

```
  a
      u e          u
V    dq ’ v.filddrn ’be %2!=0)).map(x=›x*x).sum
```

```
even: Double = 165.0
```

```
%scala
//3.Prime numbers ' square sum
def 1sPrime(num: Int):Boolean =(
  (num ›1) && !(2 to seala . nath. sqrt(nun) . toInt) . extsts(x=›
  num&x==0)
```

4

```
isPrime: (num: Int)Boolean
```

```
%scala
val prime = v.filter(x=>(isPrime(x))).map(x=>(x*x)).sum
```

```
prime: Double = 87.0
```

```
%scala
println("Sum of odd square is :"+odd)
println("Sum Of even square is :"+even)
println("Sum of prime square is :"+prime)
```

```
Sum of odd square is :228.0
Sum of even squane is :165.0
Sum of prime square is :87.0
```

```
%scala
val fr = a.foreach(isPrime)
```

```
fr: Unit = ()
```

```
%scala
// frequency of occurence of words in the file

val wordList=sc.parallelize(List(
  "apple", "banana", "orange", "grape", "banana", "apple", "kiwi", "orange", "apple", "grape",
  "pear", "kiwi", "banana", "orange", "apple", "grape", "kiwi", "orange", "apple", "pear",
  "kiwi", "banana", "grape", "orange", "apple", "kiwi", "banana", "apple", "grape", "orange",
  "pean", "kiwi", "banana", "apple", "grape", "onange", "kiwi", "banana", "apple", "grape",
  "kiwi", "orange", "pear", "banana", "grape", "apple", "kiwi", "bauaua", "orauge", "grape",
  "pear", "kiwi", "banana", "apple", "grape", "orange", "kiwi", "banaua", "apple", "grape",
  "kiwi", "orange", "pear", "banana", "grape", "apple", "kiwi", "bauaua", "orauge", "grape",
  "pear", "kiwi", "banana", "apple", "grape", "orange", "kiwi", "banana", "apple", "grape",
  "kiwi", "orange", "pear", "banana", "grape", "apple", "kiwi", "banana", "orange", "grape",
  "pean", "kiwi", "banana", "apple", "grape", "onange", "kiwi", "banana", "apple", "gnape"
```

```
wordList: org.apache.spark.rdd.RDD[String] = ParallelCollectiouRDD[0] at parallelize at comma
```

```
%scala
wordList.map(x=>(x,1)).reduceByKey(_+_).collect
```

```
resl: Array[(String, Int)] = Array((kiwi,19), (apple,18), (grape,19), (banana,19), (pear,9),
```

Name : Harsh Shah

Roll No. : 21BCP359

(G11 Div6)

LAB 4

```
%scala
// Wordcount program for a file

val data = sc.textFile("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/input.txt")
   a
dal sp  tdata = data.flatMap(line => line.split(" "))
val  apdat     splitdata.map(word => (word, 1))
   d'      a,e-
: d :    :d'      'apdata.reduceByKey(_+_)
   u    t    ilec
```

```
data: org.apache.spark.rdd.RDD[String] - dbfs:/FileStone/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/input.txt  MapPartitionsRDD[1] at textFile  at
command-877532722711277:3
splitdata: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at command-877532722711277:5
mapdata: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at command-877532722711277:6
reducedata: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at command-877532722711277:8
res0: Array[(String, Int)] = Array((Birds,1), (orange,1), (dog.,1), (with,1), (lazy,1), (pink.,1), (draped,1), (over,2), (brown,1), (sunset,1),
(merrily,1), (fox,1), (nature.,1), (The,1), (chirped,1), (a,1), (painted,1), (quick,1), (creating,1), (A,2), (symphony,1), (in,1), (of,2), (tree
                    C..ww M u    wM               W4 wl.      ri..        W        ... u
```

•*Ascai a*

```
// Character couut program for file

val data = sc.textFile("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/input.txt")
data.collect
val splitdata = data.flatMap(q => q.split(""))
val mapdata = splitdata.map(word => (word, 1))
mapdata.collect
val reducedata = mapdata.reduceByKey(_+_) // ReduceByKey is a kind of transformation not an action
reducedata.collect
```

```
data: org.apache.spark.rdd.RDD[String] = dbfs:/FileStone/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/input.txt MapPartitionsRDD[6] at textFile
at command-877532722711278:3
splitdata: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at flatMap at command-877532722711278:5
mapdata: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[8] at map at command-877532722711278:6
reducedata: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[9] at reduceByKey at command-877532722711278:8
resl: Array[(String, Int)] = Array((T,1), (d,8), (z,2), (p,6), (x,1), (B,1), (t,12), (.,3), (b,3), (h,9), (" ",38), (n,11), (f,4), (j,1), (v,2),
       ra      e ,o    .„ o    r  w      o            ; ,c    ›.      .. u      o    ›. o      ,c            z      a      z
```

```
%scala
res3
```

```
res4: Array[(String, Int)] = Array((T,1), (d,8), (z,2), (p,6), (x,1), (B,1), (t,12), (.,3), (b,3), (h,9), (" ",38), (n,11), (f,4), (j,1), (v,2),
(„1), (1,4), (r,12), (w,2), (s,7), (e,21), (a,10), (i,10), (k,4), (y,6), (A,2), (u,7), (o,10), (q,1), (g,3), (m,3), (c,4))
```

```
%scala
//Getting number of cores used in the program
/*
1. rdds are keep on creating the lineage information and the DAG is continously updated
2. rdd itself is partitioned into blocks and if core has no space then this partitioned rdd is send to other cores
3. cone execution is always sequential
4. In case of node failure it cau look to the lineage info (metadata iufo)
5. Spark do sequential execution on a single core and all other cores then run parallely
6. Spark's objective is not to go for replication but focusing on processing just inverse of Hadoop
*/

val ndda=sc.panallelize(List(1,2,4,5),10) //List(),10 -› Number of cones
val rddb=rdda.collect //Transformed rdd //collect is also an "action"
println("Number of partitions: "+rdda.getNumPartitions)
println("Action: First element "+rdda.first())
rdda.foreach(println)
```

```
Number of partitions: 18
Action: First element 1
rdda: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[18] at parallelize at command-877532722711280:11
```

```
rddb: Array[Int] = Array(1, 2, 4, 5)
```

```
%scala
/*
Brodcasting the data to the all the rdd on multiple cores running the partitioned rdds
We can update rdds and list using ”SEQ“(sequantials) ,ideally we should avoid it
The broadcasting will be done to all the rdds where the rdd is stored
’
val broadCastVar=sc.broadcast(Array(1,2,4,5))

broadCastVar.value
```

```
broadCastvar: org.apache.spark.broadcast.Broadcast[Amay[Int]] = Broadcast(28)
res18: Array[Int] = Array(1, 2, 4, 5)
```

```
%scala
// Use of broadcast variable creation of Dataframe
/*
We can only create one context i.e sc - spark context

impont ong.apache.spark.sql.SpankSession

val states = Map((“NY“,“New York“),(“CA“,“California”),(“FL“,"FLorida“))
val countnie - Map((“USA“,“UNited States“),(“IN“,”India“),("CHN",“China“))
val bstates= spark.sparkContext.broadcast(states)
val bcountries=spark.sparkContext.broadcast(countrie)


val data = Seq((“James“,“Smith“,“USA“,“CA“),
(“Jamesl“,”Smith1“,“USA“,“CA“),
(“James2“,”Smith2“,“IN“,“CA“),
(“James3“,”Smith3“,“CHN“,“FL“),



val rdda =spark.sparkContext.parallelize(data)

val rdd2=rdda.map(f=>{
  val country=f._3
  val state=f._4
  val fullCountry=bcountries.value.get(country).get
  val fullstate=bstates.value.get(state).get
  (f._1,f._2,fullCountry,fullstate)


println(rdd2.collect().mkString(“\u“))
```

```
(James,Smith,UNited States,California)
(James1,Smith1,UNited States,California)
(James2,Smith2,India,California)
(James3,Smith3,China,FLorida)
import org.apache.spark.sql.SparkSession
states: scala.collection.immutable.Map[String,String] = Map(NY -> New York, CA -› California, FL -> FLorida)
countrie: scala.collection.immutable.Map[String,String] = Map(USA -› UNited States, IN -› India, CHN -› China)
bstates: org.apache.spark.broadcast.Broadcast[scala.collection.immutable.Map[String,String]] = Broadcast(40)
bcountries: org.apache.spark.broadcast.Broadcast[scala.collection.immutable.Map[String,String]] - Broadcast(41)
data: Seq[(String, String, String, String)] = List((James,Smith,USA,CA), (Jamesl,Smith1,USA,CA), (James2,Smith2,IN,CA), (James3,Smith3,CHN,FL))
rdda: org.apache.spark.rdd.RDD[(String, String, String, String)] = ParallelCollectionRDD[25] at parallelize at command-877532722711282:20
rdd2: org.apache.spark.rdd.RDD[(String, String, String, String)] = MapPartitionsRDD[26] at map at command-877532722711282:22
```

```
%scala
// Creating the columns in dataframe
import org.apache.spark.sql.SparkSession

val states = Map((“NY“,“New York“),(“CA“,“California“),(“FL“,"FLorida“))
val countrie = Map((“USA“,“UNited States“),(“IN“,”India“),("CHN","China“))
val bstates= spark.sparkContext.broadcast(states)
val bcountries=spark.sparkContext.broadcast(countrie)


val  data = Seq(("3anes", "Snlth", "USA", "CA"),
("3ames1",  "Snlth1", "USA", "CA"),
("3ames2",  "Snlth2", "IN", "CA"),
("James3", "Smith3", "CHN", "F L" ),



val columns = Seq(“firstname“,“lastname“,“country“,“state“)
impont spank.sqlContext.implicits._
val df=data.toDF(columns:_*)
val df2=df.map(row=>{
  val countny=now.getString(2)
  va1 state= row.getString(3)

  val fullCountny=bcountries.value.get(country).get
  val fullstate=bstates.value.get(state).get

  (row.getString(0),row.getString(1),fullCountry,fullstate)
}).toDF(columns:_*)

df2.collect
df2.show()
```

```
firstname|lastname|     country|      state

    James| Smith|UNited States|California|
   James1|  Smith1|UNited  States|California|
   James2|   Smith2|    India|California|
   James3| Smith3|     China|  FLorida|
```

```
import org.apache.spark.sql.SparkSession
states: scala.collection.immutable.Map[String,String] = Map(NY -› New York, CA -› California, FL -> FLorida)
countrie: scala.collection.immutable.Map[String,String] = Map(USA -> UNited States, IN -> India, CHN -> China)
bstates: org.apache.spark.broadcast.Broadcast[scala.collection.immutable.Map[String,String]] = Broadcast(55)
bcountries: org.apache.spark.broadcast.Broadcast[scala.collection.immutable.Map[String,String]] = Broadcast(s6)
data: Seq[(String, String, String, String)] = List((James,Smith,USA,CA), (Jamesl,Smith1,USA,CA), (James2,Smith2,IN,CA), (James3,Smith3,CHN,FL))
columns: Seq[String] = List(firstname, lastname, countny, state)
import spark.sqlContext.implicits._
df: org.apache.spark.sql.DataFrame = [firstname: string, lastname: string ... 2 more fields]
df2: org.apache.spark.sql.DataFrame = [firstname: string, lastname: string 2 more fields]
```

```
%scala
//Aggnegation of data fon high performance computing parallely
val accum = sc.longAccumulator(“My Accumulator”)
sc.parallelize(Array(2,4,4,3)).foreach(x => accum.add(x))
accum.value
```

```
accum: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 1982, name: Some(My Accumulator), value: 13)
res26: Long = 13
```

```
%scala
// Advanced aggregators -> transformation function

va1 accu = sc.1ongAccumu1ator("My Accumulator”)
val accul = sc.lougAccumulator(“My Accumulator”)
val accu2 = sc.longAccumulator(“My Accumulator”)
val accum = sc.longAccumulator(“My Accumulator”)

spark.sparkContext.setLogLevel(“ERROR“)
val inputRDD = spark.sparkContext.parallelize(List((“Z“,1),(“B“,1),(“C“,1),("D",1),(“E“,1)))

val listRDD = spark.sparkContext.parallelize(List(1,3,4,5,56,8,6))

//aggregate
def  paramO=(ac cu: Int,v:Int)=›ac cu+v
def   paraml=(ac cu1 : Int,ac cu2 : Int)=›accu1+a ccu2
println(“Aggregate for Num list:“+listRDD.aggregate(0)(param0,paraml))

//aggnegate
def param3=(accu:Int,v:(String,Int))=>accu+v._2
def param4=(accu1:Int,accu2:Int)=>accu1+accu2
println(“Aggregate for key val list:“+inputRDD.aggregate(0)(param3,param4))

//tree aggregate
def  param5=(accu: Int,v:(String,Int))=›accu+v._2
def- param6=(accu1:Int,accu2:Int)=›accu1+accu2
println(“Aggregate for key val usiug tree aggregate:“+inputRDD.treeAggregate(0)(param5,param6))

//ADVANCED ACTIONS


//Fold
println(“Fold for int list:“+listRDD.fold(0){
  (acc, v)=›val  sun=acc+v
  sum


//Reduce
println(“Reduce for int list:“+listRDD.reduce(_+_)) //Shortcut

//TneeReduce - neduces ndd to multilevel tree pattenn
println(“Tree Reduce for int list:“+listRDD.treeReduce(_+_))

//Count
println(“Count for int list:“+listRDD.count)

//CountApprox
println(“Count Approx for int list:“+listRDD.countApprox(1200))

//Count by value
println(”Count by value for int list:“+listRDD.countByValue())

//min
println(“Min for int list:“+listRDD.min())

//max
println(“Max for int list:“+listRDD.max())
```

```
Aggregate for Num list:83
Aggregate for key val list:5
Aggregate for key val using tree aggregate:5
Fold for int list:83
Reduce for int list:83
Tree Reduce for int list:83
Count for int list:7
Count Approx for int list:(final: [7.000, 7.800])
Count by value for int list:Map(5 -> 1, 56 -> 1, 1 -> 1, 6 -> 1, 3 -> 1, 8 -> 1, 4 -> 1)
Min for int list:1
Max for int list:56
accu: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 4995, name: Some(My Accumulator), value: 0)
accu1: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 4996, name: Some(My Accumulator), value: 0)
accu2: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 4997, name: Some(My Accumulator), value: 0)
accum: org.apache.spark.util.LongAccumulator = LongAccumulator(id: 4998, name: Some(My Accumulator), value: 0)
inputRDD: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[131] at parallelize at command-877532722711285:9
listRDD: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[132] at parallelize at command-877532722711285:11
param6: (Int, Int) => Int
param1: (Int, Int) => Int
parari3: (Int, (String, Int)) => Int
param4: (Int, Int) => Int
param5: (Int, (String, Int)) => Int
param6: (Int, Int) => Int
```

```
%scala
//Fetching and reading CSV
val df1 = spark.read.format("csv").option("delimiter", ";").option("headers","false").load("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.
// val columns = Seq("Username","Identifier")

// import spark.sqlContext.implicits._
// val df=data.toDF(columns:_*)
// val df2=df.map(row=>{
//   val country=row.getString(2)
//   val state= row.getString(3)

//   val fullCountry=bcountries.value.get(country).get
//   val fullstate=bstates.value.get(state).get

//   (row.getString(0),row.getString(1),fu11Country,fullstate)
// }).toDF(columns:_*)

// df2.collect
df1.show()
```

```
+---------+----------+---------------+-------------+----------+---------+----------+----------+
|      _c0|       _c1|            _c2|          _c3|       _c4|      _c5|       _c6|       _c7|
+---------+----------+---------------+-------------+----------+---------+----------+----------+
| Username|Identifier|One-time password|Recovery code|First name|Last name|Department|  Location|
|booker12|      9012|         12se74|      rb9012|    Rachel|   Booker|     Sales|Manchester|
|  grey07|      2070|         04ap67|      lg2070|     Laura|     Grey|     Depot|    London|
|johnson81|      4081|         30no86|      cj4081|     Craig|  Johnson|     Depot|    London|
|jenkins46|      9346|         14ju73|      mj9346|      Mary|  Jenkins|Engineering|Manchester|
| smith79|      5079|         09ja61|      js5B79|     Jamie|    Smith|Engineering|Nanchester|
+---------+----------+---------------+-------------+----------+---------+----------+----------+
```

```
df1: org.apache.spark.sql.DataFrame = [_c8: string, _c1: string ... 6 more fields]
```

```
%scala
dbutils.fs.ls("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/input.txt")
```

```
res52: Seq[com.databricks.backend.daemon.dbutils.FileInfo] = ArrayBuffer(FileInfo(dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/input.txt, input.txt, 209, 1706846202000))
```

```
%scala
dbutils.fs.help()
```

dbutils.ls provides utilities for working with FileSystems. Most methods in this package can take either a DBFS path (e.g., "/foo" or "dbfs:/foo"), or another FileSystem URI. For more info about a method, use dbutils.ls.help("methodName"). In notebooks, you can also use the %fs shorthand to access DBFS. The %fs shorthand maps straightforwardly onto dbutils calls. For example, "%fs head --maxBytes=10000 /file/path" translates into "dbutils.fs.head("/file/path", maxBytes = 10000)".

mount

mount(source: String, mountPoint: String, encryptionType: String = "", **owner: String** = **null, extraConfigs: Map** = **Map.empty[String, String]):** boolean -> Mounts the given source directory into DBFS at the given mount point
mounts: Seq -> Displays information about what is mounted within DBFS
refreshMounts: boolean -> Forces all machines in this cluster to refresh their mount cache, ensuring they receive the most recent information
unmount(mountPoint: String): boolean -> Deletes a DBFS mount point
updateMount(source: String, mountPoint: String, encryptionType: String = "", **owner:** String = null, extraConfigs: Map = **Map.empty[String, String]): boolean** -> Similar to mount(), but updates an existing mount point (if present) instead of creating a new one

fsutils

cp(from: String, to: String, recurse: boolean = false): boolean -> Copies a file or directory, possibly across FileSystems
head(file: String, maxBytes: int = 65536): String -> Returns up to the first 'maxBytes' bytes of the given file as a String encoded in UTF-8
lsfdir: String\): Sea -> Lists the rmntc'nts nf a rJirr'otnrv

Name : Harsh Shah

Roll No. : 21BCP359

(G11 Div6)

LAB 5

```
%scala
// from pyspark import SparkContext, SparkConf

// conf = SparkConf().setAppName("LogExample")
// sc = SpankContext(conf=conf)

// sc.setLogLevel("Error")
```

```
%scala
// in databricks this library has been deprecated and in databricks we have the flexibility to directly use sql
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

```
commaud-4077768786815913:2: warning: constructor SQLContext iu class SQLContext is deprecated (since 2.0.0): Use SparkSession.builder instead
val sqlContext = new org.apache.spank.sql.SQLContext(sc)

sqlContext:  org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@5fc7e659
```

```
%scala
val a = sc.parallelize(1 to 10)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at command-4077768786815914:1
```

```
%scala
a.collect
```

```
res1: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
$scala
val  b  =  a . nap (x=›(x, x*1) )
```

```
b: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[1] at map at command-4077768786815916:1
```

```
%scala
b.collect
```

```
res2: Array[(Int, Int)] = Array((1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,18), (10,11))
```

```
%scala
val df = b.toDF("First","Second")
```

```
%scala
df.show()
```

```
First |Second

    1|     2|
    2|     3|
    3|     4|
    4|     5|
    5|     6|
    6|     7|
    7|     8|
    8|     9|
    9|    10|
   10|    11|
```

```
%scala
val a = List((“Tom“,5),(“Jerry“,2),(“Donald“,7))
```

```
a: List[(String, Int)] = List((Tom,5), ( erry,2), (Dona1d,7))
```

```
%scala
val df = a.toDF(“Name“,“Age“)
```

```
df: org.apache.spark.sql.DataFrame = [Name: string, Age: int]
```

```
%scala
df.show
```

```
  Name|Age|

   Tom|  5|
 Jerry|  2|
Donald|  7|
```

```
%scala
// df is internally converted to rdd and nodes for parallization and then task is done by dag schedular
```

```
%scala
val a = Seq((“Tom“,5),(“Jerry“,2),(“Donald“,7))
```

```
a: Seq[(String, Int)] = List((Tom,5), (Jerry,2), (Donald,7))
```

```
%scala
val df = a.toDF(“Name“,“Age“)
```

```
df: org.apache.spark.sql.DataFrame = [Name: string, Age: int]
```

```
%scala
df.show
```

```
  Name|Age|

   Tom|  5|
 3erry|  2
Donald|  7|
```

```
%scala
df.registerTempTable(“Cartoon”) // metadata information - since we have the cartoon character names
// negistenTempTable is depnecated instead use - cneate0rReplaceTempView - this will cneate a table fon you.
// This command createOrReplaceTempView - by default work for all file systems
```

```
command-4077768786815927:1: warning: method registerTempTable iu class Dataset is deprecated (since 2.8.0): Use createOrReplaceTempView(viewName)
instead.
df.registerTempTable(“Cartoon“) // metadata information - since we have the cartoon character names
```

•*As cai a*

```
df.create0rReplaceTempView(“Cartoon“)
```

```
%scala
sqlContext.sql(“select * from Cartoon where Name = 'Tom'“).show
// whatever you do with sqlContext you can do with dataframe
```

```
Name|Age

 Tom|  5|
```

```
%scala
sqlContext.sql("select * from Cartoon").show
```

```
 Narre|Age

   Tom|  5|
 Jerry|  2|
Donald|  7|
```

```
%scala
// Question: To create a JSON File and penform the following openations
// select query with all names
// filter and identify age > 23
// groupBy Age couut it and show it

val df1 = spark.read.format("json").load("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/JS0NDatabricks.json")
```

```
df1: org.apache.spark.sql.DataFrame = [Age: string, id: string ... 1 more field]
```

```
%scala
df1.show
```

```
Age|  Id |    name

 25|1261| Sat i sh
 28|1262|Krishna
 39|1263|   Amith
 23|1264|  3aved
 23|1205|Pruthvi
```

```
%scala
df1.printSchema() // the information of the json file - field details
```

```
0OOt
-- Age: string (nullable = true)
-- id: string (nullable = true)
-- name: string (nullable = true)
```

```
%scala
df1.select("name","Age").show()
```

```
   name|Age|

 Satish|  25
Krishna|  28
  Amith|  39
  3aved|  23
Pruthvi|  23
```

```
%scala
dfl.create0rReplaceTempView("Employee")
```

```
%scala
// sqlContext - are faster than traditional mysql or oracle operations
// here since it is distributed environment
sq1Context.sql("select name from employee“).show
```

```
name|

Satish
Krishna
Amith
3aved
Pruthvi
```

```
%scala
df1.filter(“age > '23'“).show()
// altennate of doing the same thing:
// df1.filter(df1(“age“) > 23).show()
```

```
Age|  id |    name

25|1201| Satish|
28|1202|Km shna
39|1263|  Amith
```

```
%scala
df1.groupBy(“age“).count().show
```

```
age|count|

28|    1|
23|    2|
25|    1|
39|    1|
```

```
%scala
val rdda = sc.parallelize(l to 1000)
```

```
rdda: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[29] at parallelize at command-4077768786815939:1
```

```
%scala
rdda.collect
```

```
res28: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 5B, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 6
9, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 8B, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 10
4, 1B5, 106, 107, 108, 1B9, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 12B, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 13
3, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 16
2, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 19B, 19
1, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 264, 205, 206, 207, 268, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 22
0, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 246, 241, 242, 243, 244, 245, 246, 247, 248, 24
9, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 27
8, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 30
7, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 33
6, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 366, 361, 362, 363, 364, 36
5, 366, 367, 368, 369, 376, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 39
4, 395, 396, 397, 398, 399, 400, 401, 402, 463, 404, 405, 406, 467, 408, 409, 416, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 42
3, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 45
2, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 48
1, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 588, 501, 502, 503, 504, 505, 506, 567, 508, 509, 51
0, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 536, 531, 532, 533, 534, 535, 536, 537, 538, 53
9, 546, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 56
8, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 588, 581, 582, 583, 584, 585, 586, 587, 588, 589, 598, 591, 592, 593, 594, 595, 596, 59
7, 598, 599, 600, 601, 682, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 62
6, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 658, 651, 652, 653, 654, 65
5, 656, 657, 658, 659, 668, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 68
4, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 708, 701, 702, 703, 704, 705, 706, 707, 788, 709, 710, 711, 712, 71
3, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 74
2, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 77
1, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 80
0, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 82
9, 836, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 85
```

```
%scala
val rddb = sc.parallelize(List("BMW","Mercedes","Toyota","Audi"))
```

```
rddb: org.apache.spark.rdd.RDD[String] - ParallelCollectionRDD[30] at parallelize at command-4077768786815941:1
```

•*Ascai a*

```
rddb.collect
```

```
res29: Array[String] = Array(BMW, Mercedes, Toyota, Audi)
```

```
%scala
rdda.partitions.length
```

```
res33: Int = 8
```

```
%scala
rddb.partitions.length
```

```
nes34: Int = 8
```

```
%scala
val rdda = sc.parallelize(1 to 1000,10)
```

```
rdda: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[31] at parallelize at command-4877768786815945:1
```

```
%scala
rdda.partitions.length
```

```
res35: Int = 10
```

•*Ascai a*

```
rdda.count
```

```
res36: Long = 1000
```

```
%scala
rdda.first
```

```
res37: Int = 1
```

```
%scala
rdda.take(10)
```

```
res38: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
%scala
rdda.saveAsTextFile("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/parallelEX.txt")
```

```
%scala
// reduce - return only 1 number - it is an action, merges into a single number
// reduceByKey - transformation
```

```
%scala
// this can be useful if there is huge data and we have to store and analyze the intermediate data to perform analysis as and when required
val nddnead = sc.textFile("dbfs:/FileStone/shaned_uploads/kushagra.dce21@sot.pdpu.ac.in/panallelEX.txt")
```

```
rddread: org.apache.spark.rdd.RDD[String] = dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/para1lelEX.txt MapPartitionsRDD[5] at
textF ile at command-581507801517287:2
```

```
%scala
rddnead.count()
```

```
        at org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:304)
        at org.apache.hadoop.mapred.FileInputFormat.listStatus(FileInputFormat.java:244)
        at org.apache.hadoop.mapred.FileInputFormat.getSplits(FileInputFormat.java:332)
        at org.apache.spark.rdd.HadoopRDD.getPartitions(HadoopRDD.scala:223)
        at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:336)
        at scala.Option.getOrElse(Option.scala:189)
        at org.apache.spark.rdd.RDD.partitions(RDD.scala:332)
        at org.apache.spark.rdd.MapPartitionsRDD.getPartitions(MapPartitionsRDD.scala:57)
        at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:336)
        at scala.Option.get0rElse(Option.scala:189)
        at org.apache.spark.rdd.RDD.partitions(RDD.scala:332)
        at org.apache.spark.SparkContext.runJob(SparkContext.scala:2815)
        at org.apache.spark.rdd.RDD.count(RDD.scala:1324)
        at $lined04d35c7elBd477a8c04f1e4al83f59a39.$read$$iw$$iw$$iw$$iw$$iw$$iw.‹init›(command-5815078B1517288:1)
        at $lined04d35c7elBd477a8c04fle4al83f59a39.$read$$iw$$iw$$iw$$iw$$iw.‹init>(command-581587801517288:45)
        at $lined04d35c7el0d477a8c04fle4al83f59a39.$read$$iw$$iw$$iw$$iw.‹init›(command-581507801517288:47)
        at $lined04d35c7el0d477a8c04fle4al83f59a39.$read$$iw$$iw$$iw.‹init>(command-581507801517288:49)
        at $linedB4d35c7el8d477a8cB4fle4al83f59a39.$read$$iw$$iw.‹init›(command-581507801517288:51)
        at $linedB4d35c7el8d477a8c04fle4a183f59a39.$read$$iw.‹init›(command-581507881517288:53)
        at $lined04d35c7el0d477a8c04fle4a183f59a39.$read.<init>(command-581507801517288:55)
        at $lined04d35c7elBd477a8c04fle4al83f59a39.$read$.‹init›(command-581507801517288:59)
        at $lined04d35c7elBd477a8c04fle4a183f59a39.$read$.‹clinit>(command-581507801517288)
        at $lined04d35c7elBd477a8c04fle4a183f59a39.$eval$.$print$lzycompute(‹notebook›:7)
        at $lined04d35c7el8d477a8c04fle4al83f59a39.$eval$.$priut(‹notebook›:6)
        at $lined04d35c7el8d477a8c04fle4al83f59a39.$eval.$print(<notebook›)
        at sun.reflect.NativeMethodAccessorImpl.invoke8(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingNethodAccessorImpl.invoke(DelegatingNethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at scala.tools.nsc.interpreter.IMaiu$ReadEvalPrint.call(IMain.scala:747)
        at scala.tools.nsc.interpreter.IMain$Request.loadAndRun(IMain.scala:1020)
        at scala.tools.nsc.interpreter.IMaiu.$anonfun$interpret$1(IMain.scala:568)
        at scala.reflect.internal.util.ScalaClassLoader.asContext(ScalaClassLoader.scala:36)
        at scala.reflect.internal.util.ScalaClassLoader.asContext$(ScalaClassLoader.scala:116)
        at scala.reflect.internal.util.AbstractFileClassLoader.asContext(AbstractFileClassLoader.scala:41)
        at scala.tools.nsc.interpreter.IMain.loadAndRunReq$1(INain.scala:567)
        at sca1a.tools.nsc.Interpreter.IMaln.Interpret(IMaln.scala:594)
        at scala.tools.nsc.interpreter.IMain.interpret(IMain.scala:564)
        at com.databricks.backend.daemon.driver.DriverILoop.execute(DriverILoop.scala:223)
        at com.databricks.backend.daemon.driver.ScalaDriverLocal.$anonfun$repl$1(ScalaDriverLocal.scala:227)
        at scala.runtime.java8.JFunction0$mcV$sp.apply(JFunction8$mcV$sp.java:23)
        at com.databricks.backend.daemon.driver.DriverLocal$TrapExitInternal$.trapExit(DriverLocal.scala:1283)
        at com.databricks.backend.daemon.driver.DriverLocal$TrapExit$.apply(DriverLocal.scala:1236)
        at com.databricks.backend.daemon.driver.ScalaDriverLocal.repl(ScalaDriverLocal.scala:227)
        at com.databricks.backend.daemon.driver.DriverLocal.$anonfun$execute$24(DriverLocal.scala:889)
        at com.databricks.unity.EmptyHandle$.runWith(UCSHandle.scala:124)
        at com.databricks.backend.daemon.driver.DriverLocal.$anonfun$execute$21(DriverLocal.scala:872)
        at com.databricks.logging.UsageLogging.$anonfun$withAttributionContext$1(UsageLogging.scala:414)
        at scala.util.DyuamicVariable.withValue(DynamicVariable.scala:62)
        at com.databricks.logging.AttributionContext$.withValue(AttributionContext.scala:158)
        at com.databricks.logging.UsageLogging.withAttributionContext(UsageLogging.scala:412)
        at com.databricks.logging.UsageLogging.withAttributionContext$(UsageLogging.scala:409)
        at com.databricks.backend.daemon.driver.DriverLocal.withAttnibutionContext(DriverLocal.scala:69)
        at com.databrlc ks.logging.UsageLogging.wlthAtt ributlonTags(UsageLogglng. scala: 457)
        at com.databricks.logging.UsageLogging.withAttributionTags$(UsageLogging.scala:442)
        at com.databricks.backend.daemon.driver.DriverLocal.withAttributionTags(DriverLocal.scala:69)
        at com.databricks.backend.daemon.driver.DriverLocal.execute(DriverLocal.scala:849)
        at com.databricks.backend.daemon.driver.DriverWrapper.$anonfun$tryExecutingCommand$1(DriverWrappen.scala:660)
        at scala.util.Try$.apply(Try.scala:213)
        at com.databricks.backend.daemon.driver.DriverWrapper.tryExecutingCommand(DriverWrapper.scala:652)
        at com.databricks.backend.daemon.driver.DriverWrapper.executeCommandAndGetError(DriverWrapper.scala:571)
        at com.databricks.backend.daemon.driver.DriverWrapper.executeCommand(DriverWrapper.scala:606)
        at com.databricks.backend.daemon.driver.DriverWrapper.runInnerLoop(DriverWrapper.scala:448)
        at com.databricks.backend.daemon.driver.DriverWrapper.runInner(DriverWrapper.scala:389)
        at com.databricks.backend.daemon.driver.DriverWrapper.run(DniverWrapper.scala:247)
        at java.lang.Thread.run(Thread.java:750)
Caused by: java.io.IOException: Input path does not exist:
        dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/parallelEX.txt at
        org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileIuputFormat.java:278)
        at org.apache.hadoop.mapred.FileInputFormat.listStatus(FileInputFormat.java:244)
        at org.apache.hadoop.mapred.FileInputFormat.getSplits(FileInputFormat.java:332)
        at org.apache.spark.rdd.HadoopRDD.getPartitions(HadoopRDD.scala:223)
        at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:336)
        at scala.Option.getOrElse(Option.scala:189)
        at org.apache.spark.rdd.RDD.partitions(RDD.scala:332)
        at org.apache.spark.rdd.NapPartitionsRDD.getPartitions(MapPartitionsRDD.scala:57)
        at org.apache.spark.rdd.RDD.$anonfun$partitions82(RDD.scala:336)
```

```
at scala.Option.getOrElse(Option.scala:189)
```

```
at org.apache.spark.rdd.RDD.partitions(RDD.scala:332)
at org.apache.spark.SparkContext.runJob(SparkContext.scala:2815)
at org.apache.spark.rdd.RDD.count(RDD.scala:1324)
at $lined04d35c7el0d477a8c04f1e4al83f59a39.$read$$iw$$iw$$iw$$iw$$iw$$iw.‹init›(command-581507801517288:1)
at $lined04d35c7el0d477a8c04fle4al83f59a39.$read$$iw$$iw$$iw$$iw$$iw.‹init>(command-581507801517288:45)
at $lined84d35c7elBd477a8c04fle4al83f59a39.$read$$iw$$iw$$iw$$iw.‹init›(command-5815B7881517288:47)
at $lined04d35c7el0d477a8c04f1e4al83f59a39.$read$$iw$$iw$$iw.‹init>(command-581507801517288:49)
at $linedB4d35c7el8d477a8c04fle4al83f59a39.$read$$iw$$iw.‹init›(command-581507801517288:51)
at $linedB4d35c7el8d477a8c04f1e4al83f59a39.$read$$iw.‹init›(command-581507881517288:53)
at $lined04d35c7el8d477a8c04f1e4al83f59a39.$read.‹init›(command-581587801517288:55)
at $lined04d35c7elBd477a8c04fle4al83f59a39.$read$.‹init›(command-581587801517288:59)
at $lined04d35c7elBd477a8c04fle4al83f59a39.$read$.‹clinit>(command-581507801517288)
at $lined04d35c7elBd477a8c04fle4al83f59a39.$eval$.$print$lzycompute(‹notebook›:7)
at $lined04d35c7el8d477a8c04fle4al83f59a39.$eval$.$priut(‹notebook>:6)
at $lined04d35c7el0d477a8c04fle4al83f59a39.$eval.$print(<notebook>)
at  sun.reflect.NativeMethodAccessorImpl.invokeB(Native  Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at scala.tools.nsc.interpreter.INain$ReadEvalPrint.call(IMain.scala:747)
at scala.tools.nsc.interpreter.IMain$Request.loadAndRun(IMain.scala:1020)
at scala.tools.nsc.interpreter.IMaiu.$anonfun$interpret$1(IMain.scala:568)
at scala.reflect.internal.util.ScalaClassLoader.asContext(ScalaClassLoader.scala:36)
at scala.reflect.internal.util.ScalaClassLoader.asContext$(ScalaClassLoader.scala:116)
at scala.reflect.internal.util.AbstractFileClassLoader.asContext(AbstractFileClassLoader.scala:41)
at scala.tools.nsc.interpreter.IMain.loadAndRunReq$1(IMain.scala:567)
at  s cala.  tools . nsc . Interpreter  .IMain. Interpret(IMaln. scala: 594)
at scala.tools.nsc.interpreter.IMain.interpret(IMain.scala:564)
at com.databricks.backend.daemon.driver.DriverILoop.execute(DriverILoop.scala:223)
at com.databricks.backend.daemon.driver.ScalaDriverLocal.$anonfun$repl$1(ScalaDriverLocal.scala:227)
at scala.runtime.java8.JFunction0$mcV$sp.apply(JFunction8$mcV$sp.java:23)
at com.databricks.backend.daemon.driver.DriverLocal$TrapExitInternal$.trapExit(DriverLocal.scala:1283)
at com.databricks.backend.daemon.driver.DriverLocal$TrapExit$.apply(DriverLocal.scala:1236)
at com.databricks.backend.daemon.driver.ScalaDriverLocal.repl(ScalaDriverLocal.scala:227)
at com.databricks.backend.daemon.driver.DriverLocal.$anonfun$execute$24(DriverLocal.scala:889)
at com.databricks.unity.EmptyHandle$.runWith(UCSHandle.scala:124)
at com.databricks.backend.daemon.driver.DriverLocal.$anonfun$execute$21(DriverLocal.scala:872)
at  com. databrlc ks . logging. UsageLogging. $anon  fun$wlthAtt ributlonContext$1(UsageLogging. scala :414)
at scala.util.DyuamicVariable.withValue(DynamicVariable.scala:62)
at  com.databricks.logging.AttributionContext$.withValue(AttributionContext.scala:158)
at  com.databricks.logging.UsageLoggiug.withAttributionContext(UsageLogging.scala:412)
at com.databricks.logging.UsageLogging.withAttributionContext$(UsageLogging.scala:409)
at com.databricks.backend.daemon.driver.DriverLocal.withAttributionContext(DriverLocal.scala:69)
at  com. databrIe ks . logging. UsageLogging. wlthAtt ributlonTags(UsageLogging.  scala: 457)
at com.databricks.logging.UsageLogging.withAttributionTags$(UsageLogging.scala:442)
at com.databricks.backend.daemon.driver.DriverLocal.withAttributionTags(DriverLocal.scala:69)
at com.databricks.backend.daemon.driver.DriverLocal.execute(DriverLocal.scala:849)
at com.databricks.backend.daemon.driver.DriverWrapper.$anonfun$tryExecutingCommand$1(DriverWrappen.scala:660)
at scala.util.Try$.apply(Try.scala:213)
at com.databricks.backend.daemon.driver.DriverWrapper.tryExecutingCommand(DriverWrapper.scala:652)
at com.databricks.backend.daemon.driver.DriverWrapper.executeCommandAndGetError(DriverWrapper.scala:571)
at com.databricks.backend.daemon.driver.DriverWrapper.executeCommand(DriverWrapper.scala:606)
at com.databricks.backend.daemon.driver.DriverWrapper.runInnerLoop(DriverWrapper.scala:448)
at com.databricks.backend.daemon.driver.DriverWrapper.runInner(DriverWrapper.scala:389)
at com.databricks.backend.daemon.driver.DriverWrapper.run(DniverWrapper.scala:247)
at java.lang.Thread.run(Thread.java:75B)
```

```
%scala
rddread.collect()
```

Name : Harsh Shah

Roll No. : 21BCP359

(G11 Div6)

LAB 6

```
# import SparkSession
from pyspark.sql import SparkSession


# create spark session object , totally optional
spank = SpankSession.builden.appName('data_pnocessing').get0rCreate()


# load csv dataset
#  df = spark.read.csv("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/sample_data.csv",inferSchema=True,header=True)

df=   spark.read.format(“csv“).option(“header“,  "true“).load("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/new-4.csv")


df.count()

     Out[8]: 50


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

     50 5


 df.printSchema()

     root
      |-- ratings: string (nullable = true)
      |-- age: string (nullable = true)
      |-- experience: string (nullable = true)
      |-- family: string (nullable = true)
      |-- mobile: string (nullable = true)



df.show(5)


     |ratings|age|experience|family|  mobile|

          4.5| 32|         8|     3|Samsung
          3.8| 25|         4|     2|  Apple|
          4.2| 40|        15|     4|OnePlus|
          3.9| 28|         6|     1| Xiaomi|
          4.1| 35|        10      2|Samsung|

     only showing top 5 rows


df.select('age','mobile').show(5)


       age|  mobile

        32|Samsung
        25|  Apple|
        40|OneP1us
        28| X1aon1
        35|Samsung

     ouly showing top 5 rows


df.describe().show()


       summa ry             ratings               age|       experience|            family|mobile|

         count|                  50|               50|               50|                50|    50|
```

```
 mean|            4.016|           35.58|           11.72|             2.36|  null|
stddev|0.3919391872862577|7.108747126905395|5.962604553483694|1.0052921191862387|  null|
  min|              3.3|               23|               10|                1| Apple|
  max|              4.7|               49|                9|                4|Xiaomi|
```

```
from pyspark.sql.types import StringType,DoubleType,IntegerType


# dataframe is sort of evolutionary, we can add or remove fields also
# withColumn - to add fields into the dataframe
df.withColumn("age_aften_10  ns",(df["age"]+10)).show(10,False) # hene the column will not be added but just showndf.
```

```
|ratings|age|experience|family|mobile |age_after_10   rs|

 4.5    |32 |8         |3    |Samsung|42.0
|3.8    |25 |4         |2    |Apple  |35.0
|4.2    |40 |15        |4    |0nePlus|50.0
|3.9    |28 |6         |1    |Xiaomi |38.0
|4.1    |35 |10        |2    |Samsung|45.0
|4.6    |45 |20        |3    |Apple  |55.0
|3.5    |30 |5         |2    |Google |40.0
|4.0    |38 |12        |4    |Xiaomi |48.0
|4.3    |42 |18        |3    |OnePlus|52.0
|3.7    |27 |7         |2    |Samsung|37.0

ouly showing top 10 rows
```

```
# .drop() can be used to drop the column
df.withColumn('age_double',df['age'].cast(DoubleType())).show(l0,False)
```

```
|ratings|age|expenience|family|mobile  |age_double|

|4.5    |32 |8         |3    |Samsung|32.0
|3.8    |25 |4         |2    |App1e  |25.0
|4.2    |40 |15        |4    |0nePlus|40.0
|3.9    |28 |6         |1    |Xiaomi |28.0
|4.i    |35 |10        |2    |Samsung|35.0
|4.6    |45 |20        |3    |Apple  |45.0
|3.5    |30 |5         |2    |Google |30.0
|4.0    |38 |12        |4    |Xiaomi |38.0
|4.3    |42 |18        |3    |0neP1us|42.0
|3.7    |27 |7         |2    |Samsung|27.0

only showing top 10 rows
```

```
df.filter(df['mobile']=='Apple').show()
```

```
|ratings|age|experience|family|mobi1e|

     3.8| 25|         4|    2| Apple|
     4.6| 45|        20|    3| App1e|
     4.4| 33|         9|    1| Apple|
     3.8| 31|         8|    3| Apple|
     4.0| 39|        14|    2| Apple|
     4.6| 46|        20|    l| Apple|
     3.8| 30|         7|    3| Apple|
     4.2| 40|        15|    2| Apple|
     4.1| 39|        14|    2| App1e|
     3.6| 32|         7|    2| Apple|
```

```
df.filter(df['mobile']=='Apple').select('age','ratings','mobile').show()
```

```
|age|ratings|mobile|

  25|    3.8| Apple|
  45|    4.6| Apple|
  33|    4.4| Apple|
  31|    3.8| Apple|
  39|    4.0| Apple|
  46|    4.6| Apple|
  30|    3.8| Apple|
  40|    4.2| Apple|
  39|    4.1| Apple|
  32|    3.6| Apple|
```

```
df.filter(df['mobile']=='Apple').filter(df['experience']>10).show()
# df.filter((df['mobile']=='Apple')&(df['experience']>10)).show()
```

```
|ratings|age|experience|family|mobile|

    4.6| 45|        20|     3| Apple|
    4.0| 39|        14|     2| App1e|
    4.6| 46|        20|     1| Apple|
    4.2| 40|        15|     2| Apple|
    4.1| 39|        14|     2| App1e|
```

```
df.select('mobile').distinct().show()
```

```
 mobile|

 X1aom1
Samsung
Google
onePlus
  App1e|
```

```
df.groupBy('mobile').count().show(5,False)
```

```
 mobile  |count

|Huawe1 |21
|Samsung| 15
|Goog1e |17
|onePlus |24
|App1e   |23
```

```
df.groupBy('mobile').count().orderBy('count',ascending=False).show(5,False)
```

```
|mobile |count|

|OneP1us |24
|Apple   |23
|Huawe1  |21
|Google  |17
|Samsung| 15
```

```
df. groupBy( 'nobl1e' ) .mean ( ) . show(5, Fa1se)
```

```
|mobile |avg(ratings)        avg(age)            |avg(expenience)   |avg(family)       |

|Huawei |6.328571428571427 |36.714285714285715|11.59047619047619 |3.4761904761904763|
|Samsung|6.3533333333333335|45.2                |9.446666666666667 |3.4
|Google |5.58235294117647  |43.470588235294116|12.48235294117647 |3.0588235294117645|
|OnePlus|5.704166666666666  |45.833333333333336|7.758333333333334  |3.6666666666666665|
|Apple   |6.234782608695651 |30.608695652173914|11.343478260869563|3.6956521739130435|
```

```
df.groupBy('mobile').sum().show(5,False)
```

```
|mobile |sum(ratings)      |sum(age)|sum(experience)  |sum(family)|

|Huawei |132.89999999999998|771    |243.39999999999998|73        |
|Samsung|95.3               |678     |141.70000000000002|51        |
|Google |94.89999999999999 |739     |212.2              |52        |
|OnePlus|136.89999999999998|1100    |186.20000000000002|88        |
```

```
|Apple  |143.39999999999998|704      |260.9              |85
```

Date - 16-82-2824

tfUDF

```
from pyspark.sql.functions import udf


den p’  e_u nge(brand) :
    elsb     dn         ee' amsung']:

             n         e
```

```
# create udf using python function
# the function is there inside the buffer and for every row the
brand_udf = udf(price_range, StringType())
# apply udf ou dataframe
df.withColumn('price_range',brand_udf(df['mobile'])).show(10,False)
```

```
|ratiugs|age|experience|family|mobile |price_range|

|6.0    |21 |4.5       |6     |0nePlus|Mid Price
|7.4    |64 |11.6      |1     |Goog1e |Low Price
|4.7    |50 |3.0        2     |OneP1us|N1d Price
 2. 3   |68 |4. 8       1     |Samsung|H1gh Price
 2. I   |34 |5. 4      |1     |OnePlus|Mid Price
|9.4    |17 |16.6       3     |Huawei |Low Price
|8.0    |29 |10.1       3     |Samsung|H1gh  Price
|9.5    |29 |6.5       |6     |0nePlus|Mid Price
|8.3    |4  |19.6      |5     |Apple  |High Price
|9.7    |77 |4.7       |1     |Samsung|High  Price

only showing top 10 rows
```

```
# using lambda function
def age(a):
    if a<='30':
        return  "Young"
    else:
        return  "Senior"

# age_udf = udf(lambda age: "Young" if age‹=30 else ”Senior“,StringType())
age_udf=udf(age,StringType())

df.withColumn('Age_descniption',age_udf(df.age)).show(10,False)
```

```
|ratings|age|experience|family|mobile |Age_description|

|4.5    |32 |8         |3     |Samsung|Senior         ⋮
|3.8    |25 |4         |2     |Apple  |Young          ⋮
|4.2    |40 |15        |4     |0neP1us|Senior         ⋮
|3.9    |28 |6         |1     |Xiaomi |Youug          ⋮
|4.1    |35 |10        |2     |Samsung|Senior         ⋮
|4.6    |45 |20        |3     |Apple  |Senior         ⋮
|3.5    |30 |5         |2     |Google |Young          ⋮
|4.0    |38 |12        |4     |Xiaomi |Senior         ⋮
|4.3    |42 |18        |3     |OnePlus|Senior         ⋮
|3.7    |27 |7         |2     |Samsung|Young          ⋮

only showing top 10 rows
```

```
#pandas udf
from pyspark.sql.functions import pandas_udf, PandasUDFType
```

```
# create python function
def remaining rs(age):
   yrs_left=100-age

    return yrs_left


#create udf using python function
length_udf = pandas_udf(remaining rs,IntegerType())
#apply pandas udf on dataframe
df.withColumn("yrs_left",length_udf(df['age'])).show(10,False)
```

```
|ratings|age|experience|family|mobile |yrs_left|

|6.0    |21 |4.5       |6     |0nePlus|79      |
|7.4    |64 |11.6      |1     |Google |36      |
|4.7    |50 |3.0       |2     |0nePlus|50      |
|2.3    |68 |4.8       |1     |Samsung|32      |
 2.I    |34 |5. 4      |1     |0nePlus|66      |
|9.4    |17 |16.6      |3     |Huawei |83      |
|8.0    |29 |10.1      |3     |Samsung|71      |
|9.5    |29 |6.5       |6     |OnePlus|71      |
|8.3    |4  |19.6      |5     |Apple  |96      |
|9.7    |77 |4.7       |1     |Samsung|23      |

only showing top 10 rows
```

```
It ud-£ using two co1umns
def  prod(rating, exp) :
    x=rat1ng*exp
    return x


# create udf using python function
prod_udf = pandas_udf(prod,DoubleType())
# apply pandas udf on multiple columns of dataframe
df.withColumn("product",prod_udf(df['ratings'],df['experience'])).show(10,False)
```

```
|natings|age|expenience|family|mobile  |product            |

|6.0    |21 |4.5       |6     |OnePlus|27.0
|7.4    |64 |11.6      |1     |6oog1e |85.84
|4.7    |50 |3.0       |2     |0nePlus|14.100000000000001|
|2.3    |68 |4.8       |1     |Samsung|11.04
 2.I    |34 |S. 4      |l     |0nePlus|11.340000000000002|
|9.4    |17 |16.6      |3     |Huawei |156.04000000000002|
|8.0    |29 |10.1      |3     |Samsung|80.8
|9.5    |29 |6.5       |6     |OnePlus|61.75
|8.3    |4  |19.6      |5     |Apple  |162.68000000000004|
|9.7    |77 |4.7        1     |Samsung|45.589999999999996

only showing top 10 rows
```

```
#duplicate values
df.count()
```

```
Out[24]: 50
```

```
# drop duplicate vales
df=df.drop_duplicates()


df.count()
```

```
Out[40]: 100
```

```
# drop columu of dataframe
df_new = df.drop('mobile')


df_new.show(10)
```

```
|ratings|age|experience|family|
```

```
          2.1| 34|        5. 4|     I
          7.4| 64|       11.6|     1|
          6.0| 21|        4.5|     6|
          4.7| 50|        3.0|     2|
          2.3| 68|        4.8|     1|
          9.7| 77|        4.7|     1|
          8.0| 29|       10.1|     3|
          8.3|  4|       19.6|     5|
          9.4| 17|       16.6|     3|
          9.5| 29|        6.5|     6|

     only showing top 10 rows
```

```
# savung file (csv)
```

```
!pwd
# current workI ng d1rectory
# dbutils.fs.1s(dir)
# dbutils.fs.1s
```

```
     /databnicks/driven
```

```
# target directory
write_uri  = 'dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/SavedFiles'
```

```
# save the dataframe as single csv
# coalesce - reducer will bring the data at one place and do compression, even if the data is at different nodes
df.coalesce(1).write.format(“csv“).option(“header“,“true“).save(write_uri)
```

```
# target location
parquet_uni='dbfs:/FileStone/shaned_uploads/kushagra.dce21@sot.pdpu.ac.in/SavedFiles_Parquet'
```

```
# save the data into parquet format
df.write.format('parquet').save(parquet_uri)
```

Name : Harsh Shah

Roll No. : 21BCP359

(G11 Div6)

LAB 7

```
# Machine learning
import pyspark.ml.feature as feat
import pyspark.sql.functions as f

some_text = spank.createDataFname(
    [[''' Apache Spark achieves high performance for both batch and streaming data, using a state-of-art DAG scheduler, a query optimizer, am
    [''' Apache Spark is a fast and general purpose clustered computing system. It provides high-level APIs in java, scala, python and R and :
    [''' Spark SQL adapts the execution plan at runtime, such as automatically setting the number of reducers and joiu algorithms. Use the saT
# for each sentences we are expecting a feature vector to be generated


splitter = feat.RegexTokenizer (
    inputCol = 'text'
    ,outputCol = 'text_split'
    ,pattenn - '\s+|[,.\"]'



splitten.tnansfonm(some_text).select('text_split').take(1)
```

```
Out[25]: [Row(text_split=['apache', 'spark', 'achieves', 'high', 'penfonmance', 'for', 'both', 'batch', 'and', 'streaming', 'data', 'usi
```

```
# stop wonds will nemove the words like, I , am , etc.
sw_remover = feat.StopWordsRemover(
inputCol = splitter.get0utputCol()
,outputCol = 'no_stopWords'



sw_remover.transform(splitter.transform(some_text)).select('no_stopWords').take(1)
```

```
Out[28]: [Row(no_stopWords=['apache', 'spark', 'achieves', 'high', 'performance', 'batch', 'streaming', 'data', 'using', 'state-of-ant',
```

```
# Hash1ng Term Frequency
hasher = -Feat.HashlngTF(
inputCol = sw_nemover.get0utputCol()
,outputCol = 'hashed'
,numFeatures=20

 # number of features can be specified of what we want from the text, a feature vector of 20 vectors will be created


hasher.transform(sw_remover.transform(splitter.transform(some_text))).select('hashed').take(1)
```

```
Out[32]: [Row(hashed=SparseVector(20, {0: 1.0, 3: 1.0, 6: 1.0, 8: 2.0, 9: 1.0, 11: 1.0, 12: 1.0, 13: 1.0, 15: 2.0, 16: 2.0, 17: 2.0, 18:
```

```
1df = I-eat.IDF(
    lnputCo1 = hasher.getOutputCo1()
    ,outputCo1 = 'features'



idfModel = idf.fit(hasher.transform(sw_remover.transform(splitter.transform(some_text))))

idfModel.transform(hasher.transform(sw_remover.transform(splitter.transform(some_text)))).select('features').take(1)
```

```
Out[35]: [Row(features=SparseVector(20, {0: 0.6931, 3: 0.0, 6: 0.0, 8: 0.0, 9: 0.0, 11: 0.6931, 12: 0.0, 13: 0.2877, 15: 0.0, 16: 0.5754
```

```
from pyspark.ml import Pipeline
```

```
pipeline = Pipeline(stages=[splitter,sw_remover,hasher,idf])
pipelineModel = pipeline.fit(some_text)
```

```
pipelineModel.transform(some_text).select('text','features').take(1)
```

```
Out[43]: [Row(text=' Apache Spark achieves high performance for both batch and streaming data, using a state-of-art DAG scheduler, a quE
```

```
# word to vector
w2v = feat.Word2Vec (
vectorSize=5
,minCount=2
,inputCol = sw_remover.get0utputCol()
,outputCol='vector'



model=w2v.fit(sw_remover.transform(splitter.transform(some_text)))
model.transform(sw_remover.transform(splitter.transform(some_text))).select('vector').take(1)
```

```
Out[47]: [Row(vector=DenseVector([0.007, 0.003, -0.0064, -0.0105, -0.0003]))]
```

Name : Harsh Shah

Roll No. : 21BCP359

(G11 Div6)

LAB 8

```
%scala
// from pyspark import SparkContext, SparkConf

// conf = SparkConf().setAppName("LogExample")
// sc = SpankContext(conf=conf)

// sc.setLogLevel("Error")
```

```
%scala
// in databricks this library has been deprecated and in databricks we have the flexibility to directly use sql
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

```
commaud-4077768786815913:2: warning: constructor SQLContext iu class SQLContext is deprecated (since 2.0.0): Use SparkSession.builder instead
val sqlContext = new org.apache.spank.sql.SQLContext(sc)

sqlContext:  org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@5fc7e659
```

```
%scala
val a = sc.parallelize(1 to 10)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at command-4077768786815914:1
```

```
%scala
a.collect
```

```
res1: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
$scala
val  b  =  a . nap (x=›(x, x*1) )
```

```
b: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[1] at map at command-4077768786815916:1
```

```
%scala
b.collect
```

```
res2: Array[(Int, Int)] = Array((1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,18), (10,11))
```

```
%scala
val df = b.toDF("First","Second")
```

```
df: org.apache.spark.sql.DataFrame = [First: int, Second: int]
```

```
%scala
df.show
```

```
First|Second

    1|     2|
    2|     3|
    3|     4|
    4|     5|
    5|     6|
    6|     7|
    7|     8|
    8|      I
    9|    10|
   10|    11|
```

```
%scala
val a = List((“Tom“,5),(“Jerry“,2),(“Donald“,7))
```

```
a: List[(String, Int)] = List((Tom,5), ( erry,2), (Dona1d,7))
```

```
%scala
val df = a.toDF(“Name“,“Age“)
```

```
df: org.apache.spark.sql.DataFrame = [Name: string, Age: int]
```

```
%scala
df.show
```

```
  Name|Age|

   Tom|  5|
 Jerry|  2|
Donald|  7|
```

```
%scala
// df is internally converted to rdd and nodes for parallization and then task is done by dag schedular
```

```
%scala
val a = Seq((“Tom“,5),(“Jerry“,2),(“Donald“,7))
```

```
a: Seq[(String, Int)] = L1st((Tom,5), ( erry,2), (Donaid ,7))
```

```
%scala
val df = a.toDF(“Name“,“Age“)
```

```
df: org.apache.spark.sql.DataFrame = [Name: string, Age: int]
```

```
%scala
df.show
```

```
  Name|Age|

   Tom|  5|
 3erry|  2
Donald|  7|
```

```
%scala
df.registerTempTable(“Cartoon”) // metadata information - since we have the cartoon character names
// negistenTempTable is deprecated instead use - cneate0rReplaceTempView - this will cneate a table fon you.
// This command createOrReplaceTempView - by default work for all file systems
```

```
command-4077768786815927:1: warning: method registerTempTable iu class Dataset is deprecated (since 2.8.0): Use createOrReplaceTempView(viewName)
instead.
df.registerTempTable(“Cartoon“) // metadata information - since we have the cartoon character names
```

•*As cai a*

```
df.create0rReplaceTempView(“Cartoon“)
```

```
%scala
sqlContext.sql(“select * from Cartoon where Name = 'Tom'“).show
// whatever you do with sqlContext you can do with dataframe
```

```
Name|Age

 Tom|  5|
```

```
%scala
sqlContext.sql("select * from Cartoon").show
```

```
 Narre|Age

   Tom|  5|
 Jerry|  2|
Donald|  7|
```

```
%scala
// Question: To create a JSON File and penform the following openations
// select query with all names
// filter and identify age > 23
// groupBy Age count it and show it

val df1 = spark.read.format("json").load("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/JS0NDatabricks.json")
```

```
df1: org.apache.spark.sql.DataFrame = [Age: string, id: string ... 1 more field]
```

```
%scala
df1.show
```

```
Age|  Id |   narre

 25|1261| Sat i sh
 28|1262|Km  shna
 39|1263|   Amith
 23|1264|  3aved
 23|1205|Pruthvi
```

```
%scala
df1.printSchema() // the information of the json file - field details
```

```
POOt
-- Age: string (nullable = true)
-- id: string (nullable = true)
-- name: string (nullable = true)
```

```
%scala
df1.select("name","Age").show()
```

```
   name|Age|

 Satish| 25|
Krishna| 28|
 Amith| 39|
 Javed| 23|
Pruthvi| 23|
```

```
%scala
dfl.create0rReplaceTempView("Employee")
```

```
%scala
// sqlContext - are faster than traditional mysql or oracle operations
// here since it is distributed environment
sq1Context.sql("select name from employee“).show
```

```
name|

Satish
Krishna
Amith
3aved
Pruthvi
```

```
%scala
df1.filter(“age > '23'“).show()
// altennate of doing the same thing:
// df1.filter(df1(“age“) > 23).show()
```

```
Age|  id |    name

25|1201| Satish|
28|1202|Km shna
39|1263|  Amith
```

```
%scala
df1.groupBy(“age“).count().show
```

```
age|count|

28|    1|
23|    2|
25|    1|
39|    1|
```

```
%scala
val rdda = sc.parallelize(1 to 1000)
```

```
rdda: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[29] at parallelize at command-4077768786815939:1
```

```
%scala
rdda.collect
```

```
res28: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 5B, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 6
9, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 8B, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 10
4, 1B5, 106, 107, 108, 1B9, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 12B, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 13
3, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 16
2, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 196, 19
1, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 264, 205, 206, 207, 268, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 22
0, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 246, 241, 242, 243, 244, 245, 246, 247, 248, 24
9, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 27
8, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 30
7, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 33
6, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 366, 361, 362, 363, 364, 36
5, 366, 367, 368, 369, 376, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 39
4, 395, 396, 397, 398, 399, 400, 401, 402, 463, 404, 405, 406, 467, 408, 409, 416, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 42
3, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 45
2, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 48
1, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 560, 501, 502, 503, 504, 505, 506, 567, 508, 509, 51
0, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 536, 531, 532, 533, 534, 535, 536, 537, 538, 53
9, 546, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 56
8, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 586, 581, 582, 583, 584, 585, 586, 587, 588, 589, 596, 591, 592, 593, 594, 595, 596, 59
7, 598, 599, 600, 601, 682, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 62
6, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 658, 651, 652, 653, 654, 65
5, 656, 657, 658, 659, 668, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 68
4, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 708, 701, 702, 703, 704, 705, 706, 707, 788, 709, 710, 711, 712, 71
3, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 74
2, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 77
1, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 80
0, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 82
9, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 85
```

```
%scala
val rddb = sc.parallelize(List("BMW","Mercedes","Toyota","Audi"))
```

```
rddb: org.apache.spark.rdd.RDD[String] - ParallelCollectionRDD[30] at parallelize at command-4077768786815941:1
```

```
%scala
rddb.collect
```

```
res29: Array[String] = Array(BMW, Mercedes, Toyota, Audi)
```

```
%scala
rdda.partitions.length
```

```
res33: Int = 8
```

```
%scala
rddb.partitions.length
```

```
nes34: Int = 8
```

```
%scala
val rdda = sc.parallelize(1 to 1000,10)
```

```
rdda: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[31] at parallelize at command-4077768786815945:1
```

```
%scala
rdda.partitions.length
```

```
res35: Int = 10
```

```
%scala
rdda.count
```

```
res36: Long = 1000
```

```
%scala
rdda.first
```

```
res37: Int = 1
```

```
%scala
rdda.take(10)
```

```
res38: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
%scala
rdda.saveAsTextFile("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/parallelEX.txt")
```

```
%scala
// reduce - return only 1 number - it is an action, merges into a single number
// reduceByKey - transformation
```

```
%scala
// this can be useful if there is huge data and we have to store and analyze the intermediate data to perform analysis as and when required
val nddnead = sc.textFile("dbfs:/FileStone/shaned_uploads/kushagra.dce21@sot.pdpu.ac.in/panallelEX.txt")
```

```
rddread: org.apache.spark.rdd.RDD[String] = dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/para1lelEX.txt MapPartitionsRDD[34] at text
File at command-4077768786815952:1
```

```
%scala
rddnead.count()
```

```
res42: Long = 1000
```

```
%scala
rddread.collect()
```

```
res43: Array[String] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 3
2, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 6
8, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 96, 91, 92, 93, 94, 95, 96, 97, 98, 99, 166, 101, 162, 10
3, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 13
2, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 168, 16
1, 162, 163, 164, 165, 166, 167, 168, 169, 178, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 19
0, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 283, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 21
9, 228, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 24
8, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 268, 261, 262, 263, 264, 265, 266, 267, 268, 269, 278, 271, 272, 273, 274, 275, 276, 27
7, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 363, 304, 305, 30
6, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 33
5, 336, 337, 338, 339, 346, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 36
4, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 39
3, 394, 395, 396, 397, 398, 399, 400, 401, 462, 403, 404, 405, 406, 467, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 42
2, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 456, 45
1, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 48
0, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 566, 501, 502, 503, 564, 505, 566, 507, 508, 50
9, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 53
8, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 566, 561, 562, 563, 564, 565, 566, 56
7, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 59
6, 597, 598, 599, 606, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 626, 621, 622, 623, 624, 62
5, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 65
4, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 676, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 68
3, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 767, 708, 709, 710, 711, 71
2, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 74
1, 742, 743, 744, 745, 746, 747, 748, 749, 756, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 77
0, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 796, 791, 792, 793, 794, 795, 796, 797, 798, 79
9, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 82
8, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 846, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 85
```

```
%scala
/* Assignment
Read the jsou file agaiu */
val readjson = sc.textFile("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/JS0NDatabricks.json")
```

```
readjson: org.apache.spark.rdd.RDD[String] = dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/JS0NDatabricks.json
MapPartitionsRDD[36] a t textFile at command-4077768786815955:3
```

```
%scala
readjson.collect()
```

```
res44: Array[String] = Array({“id":“1201“,“name“:“Satish“,“Age":“25“}, {“id“:“1282”,“name“:"Krishna“,”Age“:“28“}, (“id“:"1203“,“name“:“Amith“,“Age“:“39”}, {“id":“1204“,”name“:“Javed“,“Age":”23“}, {"id“:“1205","name“:"Pruthvi“,"Age“:“23"})
```

a

```
val op1 = neadjson.flatMap(x=›x.split(“:“)).collect()
```

```
op1: Array[String] = Array((“id“, “12B1“,“name“, “Satish“,“Age", “25“}, {“id“, “1202“,“name", “Krishua“,“Age“, “28“}, {"id“, “1283“,“name“, “Amith“,“Age“, “39“}, (“id“, "1204“,“name“, “Javed“,“Age“, “23“}, ("id“, “1285“,“name“, “Pruthvi",“Age“, "23“})
```

Sscala

```
va1 op2 = opt.map(y=›(y, 1))
```

```
op2: Array[(String, Int)] = Array(({“id“,1), (“1201“,“name“,1), (“Satish“,“Age“,1), (“25“},1), ({“id”,1), (“1202“,“name”,1), (“Krishna“,“Age“,1), (“28”},1), ((“id“,1), (”1283“,“name",1), ("Amith“,“Age“,1), ("39“},1), ((“id“,1), (“1204“,"name“,1), (“Javed“,“Age“,1), (“23“},1), ((“id“,1), (“1
                ("Pruthvi" •^„• ,; /•o,•i ,;;
```

```
%scala
val onelineop = readjson.flatMap(x=›x.split(“:“)).map(y=›(y,1)).reduceByKey((x,y)=›(x+y)).collect
```

```
onelineop: Array[(String, Int)] = Array((“1282“,“name",1), (“23”},2), (“25“},1), (”1204“,“name“,1), ({“id“,5), (“Amith“,“Age“,1), (“28“},1), (“Satish”,”Age“,1), (“Krishna”,“Age“,1), (“39“},1), (“Javed“,“Age“,1), (“1201“,“name“,1), (“1205“,“name“,1), (“1203“,“name“,1), (“Pruthvi“,“Age“,1))
```

```
%scala
val onelineop = readjson.flatMap(x=›x.split(“:“)).map(y=›(y,1)).reduceByKey((x,y)=›(x+y)).sortBy(_._1,false).collect // sortBy - default asc
```

```
onelineop: Array[(String, Int)] = Anray(({“id“,5), (“Satish“,“Age“,1), (“Pruthvi“,”Age“,l), (“Krishna”,“Age“,1), (“Javed”,“Age“,1), (“Amith”,“Age“,1), (“39“},1), (“28”},1), (“25“},1), (”23"},2), (“1205“,“name",1), (“1284“,“name",1), (“1283“,“name",1), (“1282“,“name”,1), (“1281“,“name”,1))
```

```
%scala
val onelineop2 = readjson.flatMap(x=›x.split(“:“)).map(y=›(y,1)).reduceByKey((x,y)=›(x+y)).sortBy(_._2,false).collect
```

```
                                                                    ("1202" "name" 1)              ("1204" "name" 1)  ("Amith" "Age" 1)
```

Name : Harsh Shah

Roll No. : 21BCP359

(G11 Div6)

LAB 9

```
file_locatiou = “dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/vertex.csv“
file_type=”csv“
```

```
file_location_2=“dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/edges.csv“
```

JfCSV option

```
infer_schema =“false“
first_row_is_header=“false”
delimiter=”,”
```

```
df =spank.nead.format(file_type)\
.option('inferSchema',infer_schema)\
.option('header',first_row_is_header)\
.option('sep',delimiter)\
.load(file_location)
```

```
display(df)
```

c0	_c1	_c2
1	Jacob	48
2	Jessica	45
3	Andrew	25
4	Ryan	53
5	Emily	22
6	Lily	52

```
file_location=“dbfs:/FileStone/shaned_uploads/kushagra.dce21@sot.pdpu.ac.in/edges.csv“
df1 =spark.read.format(file_type)\
.option('inferSchema',infer_schema)\
.option('headen',first_row_is_headen)\
.option('sep',delimiter)\
.load(file_location)
```

```
display(df1)
```

_c0	_c1	_c2
6	1	Sister
1	2	Husband
2	1	Wife
5	1	Daughter
5	2	Daughter
3	1	Son
3	2	Son
4	1	Friend
1	5	Father

```
%scala
import org.apache.spark.rdd.RDD
import org.apache.spark.graphx._
```

```
import org.apache.spark.rdd.RDD
import org.apache.spark.graphx._
```

```
%scala
val vertexRDD =sc.textFile("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/vertex.csv")
```

```
kvertexRDD: org.apache.spark.rdd.RDD[String] = dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/vertex.csv MapPartitionsRDD[1314] at
tex tFile at command-694813824391304:1
```

```
%scala
val edgeRDD =sc.textFile("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/edges.csv")
```

```
edgeRDD: org.apache.spark.rdd.RDD[String] = dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/edges.csv MapPartitionsRDD[1316] at
textFi le at command-694813824391305:1
```

```
%scala
vertexRDD.collect()
```

```
res35: Array[String] = Array(1,Jacob,48, 2,Jessica,45, 3,Andrew,25, 4,Ryan,53, 5,Emily,22, 6,Lily,52)
```

```
%scala
edgeRDD.collect()
```

```
res36: Array[String] = Array(6,1,Sister, 1,2,Husband, 2,1,Wife, 5,1,Daughter, 5,2,Daughter, 3,1,Son, 3,2,Son, 4,1,Friend, 1,5,Father, 1,3,Father,
```

```
%scala
val vertices : RDD[(VertexId,(String,String))]=vertexRDD.map{
  line=›val fields=line.split(",")
  (fields(0).toLong,(fields(1),fields(2)))
```

```
vertices: org.apache.spark.rdd.RDD[(org.apache.spark.gnaphx.VertexId, (String, String))] = MapPartitionsRDD[1317] at map at command-6948138243913
```

```
%scala
vertices.collect
```

```
res37: Array[(org.apache.spark.graphx.VertexId, (String, String))] = Array((1,(Jacob,48)), (2,(Jessica,45)), (3,(Andrew,25)), (4,(Ryan,53)), (5,
            (6,(Lily,52)))
```

```
%scala
val edges : RDD[Edge[String]]=edgeRDD.map{
  1ine=›va1 fields=line.split(",")
  (Edge(fields(0).toLoug,fields(1).toLong,fields(2)))
```

```
edges: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[String]] = MapPartitionsRDD[1318] at map at command-694813824391310:1
```

```
%scala
edges.collect
```

```
res38: Array[org.apache.spark.graphx.Edge[String]] = Array(Edge(6,1,Sister), Edge(1,2,Husband), Edge(2,1,Wife), Edge(5,1,Daughter), Edge(5,2,Daug
hter), Edge(3,1,Son), Edge(3,2,Son), Edge(4,l,Friend), Edge(1,5,Father), Edge(1,3,Father), Edge(2,5,Mother), Edge(2,3,Mother))
```

```
%scala
val defualt=("unknown","missing")
```

```
defualt: (String, String) = (unknown,missing)
```

```
%scala
val graph = Graph(vertices,edges,defualt)
```

```
graph: org.apache.spark.graphx.Graph[(String, String),String] = org.apache.spark.graphx.impl.GraphImpl@490bd7f4
```

```
%scala
case class MoviesWatched(Movie:String , Genre:Striug)
val movies:RDD[(VertexId,MoviesWatched)]=sc.parallelize(List(
  (1,MoviesWatched(”Toy Story 3”,”Kids”)),
  (2,MoviesWatched(“Titanic“,“Love“)),
  (3,MoviesWatched(“The  Hangover“,“Comedy“)),
```

```
defined class MoviesWatched
movies: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.Vertexld, MoviesWatched)] = ParallelCollectionRDD[1331] at parallelize at command-69481
```

```
%scala
val movieOuterJoinedGraph=graph.outerJoinVertices(movies)((_,name,movies)=>(name,movies))
```

```
movie0uterJoinedGraph: org.apache.spark.graphx.Graph[((String, String), Option[MoviesWatched]),String] = org.apache.spark.graphx.impl.GraphImpl@5
```

```
%scala
movieOuterJoinedGraph.vertices.map(t=›t).collect.foreach(println)
```

```
(4, ((Ryan,53),None))
(6,((Lily,52),None))
(2,((Jessica,45),Some(Movieswatched(Titanic,Love))))
(1,((Jacob,48),Some(MoviesWatched(Toy Story 3,Kids))))
(3,((Andrew,25),Some(MoviesWatched(The Hangover,Comedy))))
(5,((Emily,22),None))
```

```
%scala
val movie0uterJoined6raph=graph.outerJoinVertices(movies)((_,name,movies)=›(name,movies.get0rElse(MoviesWatched(“NA“,“NA“))))
```

```
movieOuterJoinedGraph: org.apache.spark.graphx.Graph[((String, String), MoviesWatched),String] = org.apache.spark.graphx.impl.GraphImpl@6789e70c
```

```
%scala
movie0uterJoiuedGraph.vertices.map(t=>t).collect.foreach(println)
```

```
(4, ((Ryan,53),Mov1esNat ched(NA,NA)))
(6,((Lily,52),MoviesWatched(NA,NA)))
(2,((3essi ca,45),Movi esNatched(Tit ani c,Love)))
(1,((Jacob,48),MoviesWatched(Toy Story 3,Kids)))
(3,((Andrew,25),MoviesWatched(The Hangover,Comedy)))
(5,((Emily,22),MoviesWatched(NA,NA)))
```

```
%scala
val tCount = graph.triangleCount().vertices // how many triangles are created
```

```
tCouut: org.apache.spark.graphx.VertexRDD[lut] = VertexRDDImpl[1393] at RDD at VertexRDD.scala:57
```

```
%scala
val iterations =1000

val connected= graph.connectedComponents().vertices

val connecteds=graph.stronglyConnectedComponents(iterations).vertices

val connByPerson=vertices.join(connected).map{case(id,((person,age),conn))=>(conn,id,person)}

val connByPersouS=vertices.joiu(connectedS).map{case(id,((person,age),conn))=›(couu,id,person)}

connByPerson.collect().foreach{case(conn,id,person)=›println(f"Weak $conn $id $person“)}
```

```
Weak 1 4 Ryan
Weak 1 6 Lily
Weak 1 2 Jessica
Weak 1 1 Jacob
Weak 1 3 Andrew
Weak 1 5 Emily
iterations: Int = 1000
connected: org.apache.spark.graphx.VertexRDD[org.apache.spark.graphx.VertexId] = VertexRDDImpl[1417] at RDD at VertexRDD.scala:57
connecteds: org.apache.spark.graphx.VertexRDD[org.apache.spark.graphx.VertexId] = VertexRDDImpl[1668] at RDD at VertexRDD.sca1a:57
connByPerson: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, org.apache.spark.graphx.VertexId, String)] = MapPartitionsRDD[1718] at
map at command-694813824391321:7
connByPersonS: ong.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, org.apache.spark.graphx.VertexId, String)] = MapPartitionsRDD[1722] at
map at command-694813824391321:9
```

```
%scala
println("Vertices Count:"+graph.vertices.count)
```

```
Vertices Count : 6
```

```
%scala
println("Edges COunt:"+graph.edges.count)
```

```
Edges COunt:12
```

```
%scala
va1 cnt =graph.vertices.fi1ter{case (id,(name,age))=>age.toLong ›40}.count
//Master and transcation entry ?
```

```
cnt: Long = 4
```

```
%scala
val cnt2 =graph.edges.filter{case Edge(from,to,property)=›property=="Father" | property=="Mother"}.count
```

```
cnt2: Long = 4
```

```
%scala
def max(a:(VertexId,Int),b:(VertexId,Int)):(VertexId,Int)={
  if(a._2 > b._2) a else b
```

```
max: (a: (org.apache.spark.graphx.VertexId, Int), b: (org.apache.spark.graphx.VertexId, Int))(org.apache.spark.graphx.VertexId, Int)
```

```
%scala
val maxInDegrees :(VertexId,Int)=graph.inDegrees.reduce(max)
val max0utDegnees :(VentexId,Int)=gnaph.outDegnees.reduce(max) // 0 indicated not strongly connected likely to be weakly connected
val maxDegrees :(VertexId,Int)=graph.degrees.reduce(max)
```

```
maxInDegrees: (ong.apache.spark.graphx.VertexId, Int) = (1,5)
maxoutDegrees : (org.apache.spark. graphx.VertexId, Int) = (1, 3)
maxDegrees : (org.apache.spark.graphx.VertexId, Int) = (1,8)
```

```
%scala
val minDegrees =graph.outDegrees.filter(_._2 <= 1)
minDegrees.collect()
```

```
minDegrees: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[1742] at RDD at VertexRDD.scala:57
res44: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,1), (6,1))
```

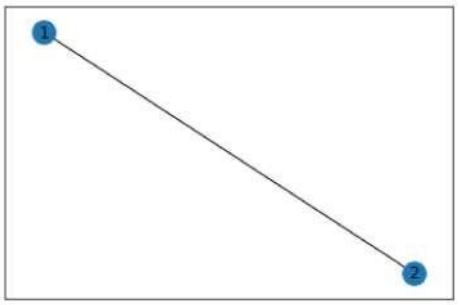
```
%scala
graph.trip1ets.map(
  triplet=>triplet.srcAttr._1+ " is the " + triplet.attr +" of "+triplet.dstAttr._1
).collect.foreach(println)
```

```
Jacob is the Husband of Jessica
Jessica is the Wife of Jacob
Andrew is the Son of Jacob
Emily is the Daughter of Jacob
Emily is the Daughter of Jessica
Lily is the Sister of Jacob
Jacob is the Father of Audrew
Jacob is the Father of Emily
Jessica is the Mother of Andrew
Jessica is the Mother of Emily
Andrew is the Son of Jessica
Ryan is the Friend of Jacob
```

```
# SECOND HALF
!pip install networkx
```

```
Collecting networkx
  Downloading networkx-3.2.1-py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6 MB 4.7 MB/s
Installing collected packages: networkx
Successfully installed networkx-3.2.1
WARNING: You are using pip version 21.2.4; however, version 24.0 is available.
You should consider upgrading via the '/local_disk0/.ephemeral_nfs/envs/pythonEnv-15a998d6-6d73-4c37-a3d6-76ce03c700f7/bin/python -m pip
```

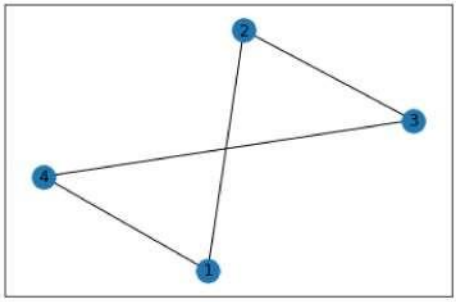
```
import networkx as nx
import matplotlib.pyplot as plt


# Create ans empty undlrected graph
G =nx. Graph ( )

tfaddlng edge In Graph
G . add_edge(1, 2)

nx.draw_networkx(G)
plt.show()
```




```
G. add_nodes_f rom( [3, 4] )
nx. draw_networkx(G)
p1t.show
```

```
Out[47]: ‹function matplotlib.pyp1ot.show(close=None, block=None)›
```


```
G . add_edge ( 3, 4)
G. add_edges_-I rom ( [ ( 2, 3) , (4, 1) ] )
nx . d raw_networkx (G )
plt . show( )
```




```
G.nodes #getting the number of nodes
```

```
Out [ 50] : NodeVlew( ( 1, 2, 3, 4) )
```

```
G.edges
```

```
Out[51]: EdgeView([(1, 2), (1, 4), (2, 3), (3, 4)])
```

```
list(nx.generate_adjlist(G))
```

Out[52]: ['1 2 4', '2 3', '3 4', '4']

```
nx.to_dict_of_lists(G)
```

Out[53]: {1: [2, 4], 2: [1, 3], 3: [4, 2], 4: [3, 1]}

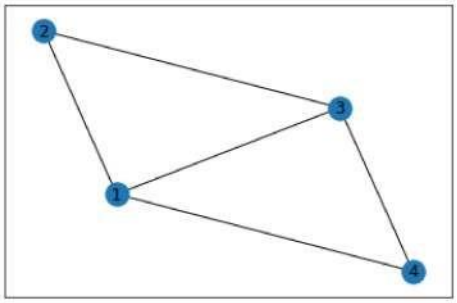
```
nx.to_edgelist(G)
```

```
Out[54]: EdgeDataView([(1, 2, {}), (1, 4, {}), (2, 3, {}), (3, 4, {})])
```

```
nx.to_pandas_adjacency(G)
```

	1	2	3	4
1	0.0	1.0	0.0	1.0
2	1.0	0.0	1.0	0.0
3	0.0	1.0	0.0	1.0
4	1.0	0.0	1.0	0.0

```
G. add_edge(1, 3)
nx. draw_networkx(G)
p1t . show( )
```

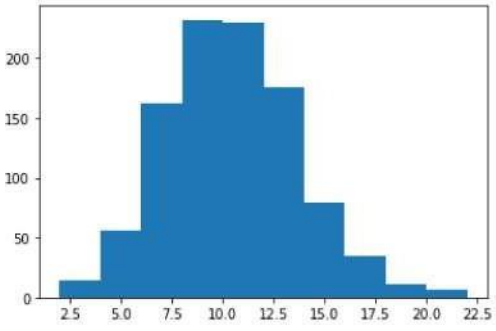



```
G. degree
```

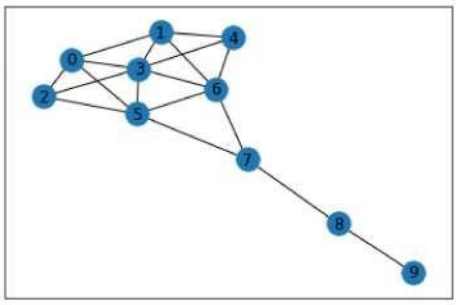
Out[57]: DegreeV1ew({1: 3, 2: 2, 3: 3, 4: 2})

```
k=nx.fast  np_random_graph(1000,0.01).degree()
plt.hist(list(dict(k).values()))
```

```
Out[59]: (array([ 14.,  56., 162., 232., 229., 176.,  79.,  35.,  11.,   6.]),
 array([ 2.,  4.,  6.,  8., 10., 12., 14., 16., 18., 20., 22.]),
 ‹BarContainer object of 10 artists›)
```




```
G=nx . krac khardt_kIt  e_graph ( )
nx. draw_networkx(G)
p1t . show( )
```




```
print(nx.has_path(G,sounce=1,target=9))
print(nx.shortest_path(G,source=1,target=9))
print(ux.shortest_path_length(G,source=1,target=9))
print(list(nx.shortest_simple_paths(G,source=1,target=9)))
paths=list(nx.all_pairs_shortest_path(G))
paths[5][1]
```

```
True
[1, 6, 7, 8, 9]
4
[[1, 6, 7, 8, 9], [1, 0, 5, 7, 8, 9], [1, 6, 5, 7, 8, 9], [1, 3, 5, 7, 8, 9], [1, 4, 6, 7, 8, 9], [I, 3, 6, 7, 8, 9], [1, 0, 2, 5, 7, 8,
Out[64]: (5: [5],

 2: [5, 2],
 3: [5, 3],
 6: [5, 6],
 7: [5, 7],
 1: [s, e, 11,
 4: [5, 3, 4],
 8: [5, 7, 8],
 9: [5, 7, 8, 9]}
```

```
# importance of node inside the network - centrality
nx.betweenness_centrality(G)
```

```
Out[65]: {0: 0.023148148148148143,
 1: 0.023148148148148143,

 3: 0. 10185185185185183,
 4: 0.0,
 5: 0.23148148148148148,
 6: 0.23148148148148148,
 7: 0.38888888888888884,
 8: 0.2222222222222222,
 9: 0.0}
```

```
nx.degree_ceutrality(G)
```

```
Out[66]: {0: 0.^^^^^""444444444,
 1: 0. 444444444^^^^^^^,
 2: 0.3333333333333333,
 3: 0. 6666666666666666,
 4: 0.3333333333333333,
 5: 0.5555555555555556,
 6: 0.5555555555555556,
 7: 0.3333333333333333,
 8: 0.2222222222222222,
 9: 0. 1111111111111111}
```

```
nx.closeness_centrality(6)
```

```
Out[67]: {0: 0.5294117647058824,
 1: 0.5294117647058824,
 2: 0.5,
 3: 0. 6,
 4: 0.5,
 5: 0. 6428571428571429,
 6: 0. 6428571428571429,
 7: 0. 6,
 8: 0. 42857142857142855,
 9: 0.3163448275862069}
```

```
nx. cI usterlng(G)
```

```
Out[68]: {0: 0.6666666666666666,
 1: 0.6666666666666666,
 2: 1.0,
 3: 0.5333333333333333,
 4: 1.0,

 6: 0.5,
 7: 0.3333333333333333,
 8: 0,
 9: 0}
```

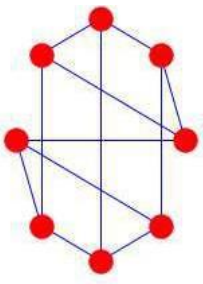
```
nx.e1genvector_centra l 1ty(G)
```

```
Out[69]: {0: 0.3522089813920359,
 1: 0.3522089813920358,
 2: 0.28583473531632403,
 3: 0.48102048812210046,
 4: 0.28583473531632403,
 5: 0.3976910106255469,
 6: 0.39769101062554685,
 7: 0.19586185175360382,
 8: 0.048074775014202924,
 9: 0.011164058575824235}
```

```
nx.harmonic_centrality(G)
```

```
Out[71]: {0: 6.083333333333333,
 1: 6.083333333333333,
 2: 5.583333333333333,
 3: 7.083333333333333,
 4: 5.583333333333333,
 5: 6.833333333333333,
 6: 6.833333333333333,
 7: 6.0,
 8: 4.666666666666666,
 9: 3.4166666666666665}
```

```
#plotting the
import matplotlib.pyplot as plt

JtCreatlng cubica1 empty graph
G=nx.cubical_graph()

#plotting graph
plt.subplot(122)

#drawing the graph with node as red and edges as blue
nx.dnaw(G,pos=nx.circulan_layout(G),
        node_color='r',

        edge_co for=' b ' )
```



```
tf pr1 nt1 ng the adjancency vert1ces
print(G . adj)
```

```
(O: (1: {}, 3: {}, 4: {}}, 1: (O: {}, 2: {}, 7: {}), 2: (1 : {}, 3: {}, 6: {}}, 3: (0 : {}, 2: {}, 5: {}}, 4: (O: {}, 5: {}, 7: {}}, 5: (*
```

Start coding or generate with AI.

Name : Harsh Shah

Roll No. : 21BCP359

(G11 Div6)

LAB 10

```
import pandas as pd

csv_string = dbutils.fs.head("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/Housing.csv") data = pd.read_csv(io.StringI0(csv_string))
#pr1nt(data)
data.head(4)
```

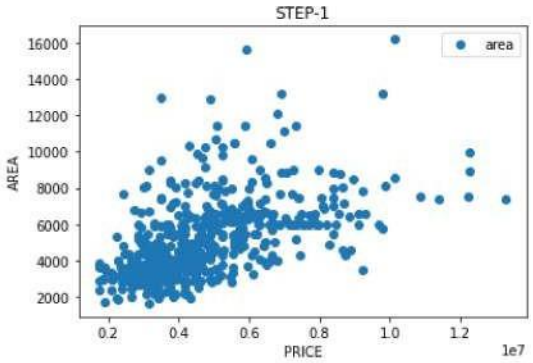
	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furn
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	

-I- Code -I- Text

```
df_1 = data.loc[:,['price', 'area']]
df_1.head()
#csv_string.head()
```

	price	area
0	13300000	7420
1	12250000	8960
2	12250000	9960
3	12215000	7500
4	11410000	7420

```
impont matplotlib.pyplot as plt
df_l.plot(x='price', y='area', style='o')
plt.xlabel('PRICE')
plt.ylabel('AREA')
plt.title('STEP-1')
plt.show()
```




```
X = pd.DataFrame(df_1['price'])
y = pd.DataFrame(df_l['area'])

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)
```

```
print(X_tnain.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(381, 1)
(164, I)
(381, 1)
(164, I)
```

```
from skleann.linean_model import LineanRegnession
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
Out[10]: LinearRegression()
```

```
print(regressor.intercept_)
```

```
[2107.90864244]
```

```
print(regressor.coef_)
```

```
[[0.0006545]]
```

```
y_pred = regressor.predict(X_test)
print(y_pred[:10])
```

```
[[6735.21572472]
 [5B85.87854688]
 [5773.10237098]
 [6116.71428303]
 [4032.13534992]
 [4581.9144092 ]
 [3711.43089867]
 [7010.10525436]
 [4032.13534992]
 [6322.88143026]]
```

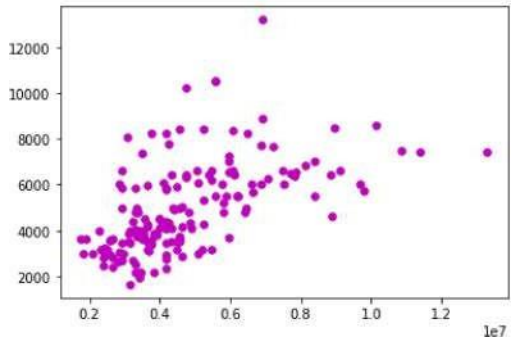
```
from sklearu import metrics
lnport numpy as np
print("Mean Absolute Error: ", metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error: ", metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error: ", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Nean Absolute Error:  1254.691752727886
Nean Squared Error:  2699402.6789812935
Root Mean Squared Error:  1642.9859034639626
```

```
def plot_regression_line(X, y, b):
    plt.scatter(X, y, color = "m",
            marker = "o", s = 30)

    plt.plot(X, y_pred, color = "g")

    plt.xlabel('x')
    plt.ylabel('y')

    plt.show()
```

```
lnport numpy as np
import matplotlib.pyplot as plt

del  est imate_coef(X, y):

    return (regressor.intercept_, regressor.coef_)


b = estimate_coef(X_test, y_test)
print("Estimated coefficients:\nb_0 = {} \
    \nb_1 = {}".format(regressor.intercept_, regressor.coef_))

plot_negnession_line(X_test, y_test, b)
```

```
Estimated coefficients:
b_0 = [2107.90864244]
b_1 = [[0.0006545]]
```




```
Inva1Id I ndexE r ror                          Traceback (most recent call last)
File <command-646436246386047›:5
      1 b = estimate_coef(X_test, y_test)
      2 print("Estimated coefficients:\nb_0 = {} \
      3         \nb_1 = {}".format(regressor.intercept_, regressor.coef_))
----› 5 plot_regression_liue(X_test, y_test, b)

File ‹command-646436246386046›:5, in plot_regressiou_line(X, y, b)
      1 def plot_regression_line(X, y, b):
      2         plt.scatter(X, y, color = "m",
      3                     marker = "o", s = 30)
----› 5         plt.plot(X, y_pred, color  = "g")
      7         plt.xlabel('x')
      8         pit.ylabel('y')

File /databricks/python/lib/python3.9/site-packages/matplotlib/pyplot.py:2757, iu plot(scalex, scaley, data, *args, **kwargs)
   2755 @_copy_docstring_aud_deprecators(Axes.plot)
   2756 def plot(*args, scalex=True, scaley=True, data=None, **kwargs):
-> 2757     return gca().plot(
   2758         *args, scalex=scalex, scaley=scaley,
   2759         **({"data": data} if data is not None else {}), **kwargs)

File /databricks/python/lib/python3.9/site-packages/matplotlib/axes/_axes.py:1632, in Axes.plot(self, scalex, scaley, data, *args,
**kwargs)
   139d """
   1391 PLot y versus x as LInes and/on martens.
   1392

   1629 (‘‘ 'green'  ) or hex str i figs (‘‘ '#888888"‘ ).

   U3   kwargs = cbook.normalize kwargs(kwargs, mlines.Line2D)
-> 1632 linear= [*self._get_lines(*args, data=data, **kwargs)]

   1633     sene.add_ nee(line)

File /databricks/pythou/lib/python3.9/site-packages/matplotlib/axes/_base.py:312, in _process_plot_var_args.  call  (self, data,
*args, **kwargs)
    31e     this += args[0],
    311     args = args[1:]
--> 312 yield from self._plot_args(this,kwargs)

File /databricks/python/lib/python3.9/site-packages/matplotlib/axes/_base.py:487, in _process_plot_var_args._plot_args(self, tup,
kwargs, return_kwargs)
    #8#         kw[prop_name] = val
    486 if len(xy) == 2:
--> 487     x = _check_1d(xy[0])
    488     y = _check_1d(xy[1])
    489 else:

File /databricks/python/lib/python3.9/site-packages/matplotlib/cbook/finite.py:1327, in _check_1d(x)
   1321 with warnings.catch_warnings(record=True) as w:
   1322     warnings.filterwarnings(
   1323         "always",
   1324         category=Warning,
   1325         message='Support for multi-dimensional indexing')
-> 1327     ndim = x[:, None].ndim
   1328     # we hove de/inirety htm o pondos index or series object
   1329     # cost to O numpy orroy.
   z33e     if- len(w) › O:

File /databricks/python/lib/python3.9/site-packages/pandas/core/frame.py:3505, in DataFrame.  etitem  (self, key)
   35e3 if self.columns.nlevels > 1:
   35e4     return self._getitem_multilevel(key)
-> 3505 indexer = self.columns.get_loc(key)
   35e6 if is_integer(indexer):
```

lpython

```
PAGE RANK
0.15 - damping factor - default value for the edge
0.85- current value
this params are continously updated until the tolerance level is reached
```

```
Out[16]: '\nPAGE RANK\n0.15 - damping factor - default value for the edge\n0.85- current value\nthis params are continously updated unti
```

```
%python
# using the inbuilt pagerank
!pip install networkx scipy
```

```
Requirement already satisfied: networkx in /local_disk0/.ephemeral_nfs/envs/pythonEnv-392c7de1-b5e3-438b-a425-6c622b6b5276/lib/python3.?
Requirement already satisfied: scipy in /local_disk0/.ephemeral_nfs/envs/pythouEuv-392c7de1-b5e3-438b-a425-6c622b6b5276/lib/python3.9/si
Requirement already satisfied: numpy<2.3,›=1.22.4 in /local_disk0/.ephemeral_nfs/envs/pythonEnv-392c7de1-b5e3-438b-a425-6c622b6b5276/lit
WARNING: You are using pip version 21.2.4; however, version 24.0 is available.
You should consider upgrading via the '/local_disk0/.ephemeral_nfs/envs/pythonEnv-392c7de1-b5e3-438b-a425-6c622b6b5276/bin/python -m pip
```

Xpython

```
pip install --upgrade scipy
```

```
Python interpreter will be restarted.
Requirement already satisfied: scipy in /local_disk0/.ephemeral_nfs/envs/pythonEuv-392c7de1-b5e3-438b-a425-6c622b6b5276/lib/python3.9/si
Requirement already satisfied: numpy<2.3,›=1.22.4 in /local_disk0/.ephemeral_nfs/envs/pythonEnv-392c7de1-b5e3-438b-a425-6c622b6b5276/lit
Pythou interpreter will be restarted.
```

```
%python
#page rank inbuilt utility

Inport networkx as nx
G=nx.DIGraph()
G.add_edges_-£-rom([(1, 2), (1, 3), (2, 1), (3, 1), (3, 2)])

#calculating page rank
pageran k= nx.pagerank(G)

tfprInt1ng pageran k scores
print("PageRank soceres")
for node ,score In pagerank.items():
    print(f"Node {node}: {score}")
```

```
PageRank soceres
Node 1:0.43274880303664615
Node 2:0.33333333333333326
Node 3:0.23391786363002037
```

```
%python
from pylab import rcParams
rcParams['figure.figsize']=(3,3)
```

âpython

```
def n1ce_prInt (v, d1gits=3):
    -format = '&&.Pdf' & digits
    print(' ,'.join([fornat & e for e In v]))
```

```
File <command-646436246385979›:3
  point(' ,'. join([format $ e I-or e in v]))

SyutaxError: invalid uon-printable character U+00A0
```

```
%python
nice_print([.12333122,.1343221,.644442143])
nice_print([.12333122,.1343221,.644442143],digits=4)
```

```
NameError                                 Traceback (most recent call last)
F1le ‹connand-646436246385980›:1
----> I nice_print([.12333122,.1343221,.644442143])
      2 nice_print([.12333122,.1343221,.644442143],digits=4)

NameError: name 'nice_priut' is not defined
```

8python

```
labels=[
    'A' , 'B' , 'C' , 'D' , 'E' , 'F' , 'G'


pages=range(len(labels))

positions= [
    (0,1),(0,2),(2,2),(0,0),(1,0),(2,0),(1,1)


page_labels = {p: 1 for p,l in zip(pages,labels)}
page_labels
```

Out[12]: (0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G'}

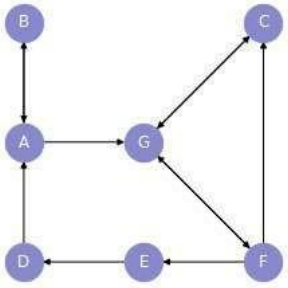
%python

```
lin ks =[
    (1, B) , (3, 8) , (8, 1) , (5, 2), (6, 2), (6, 5) , (5, 6) , (2, 6), (B, 6) , (S, 4), (4, 3)
```

```
%python
impont networkx as nx
import matplotlib.pyplot as plt
g=nx.DiGraph()

for p in pages:
    node =g.add_node(p)

for (a,b) in links:
    g.add_edge(pages[a],pages[b])
```

8python

```
plt.clf()

display(nx.draw(g,with_labels=True,labels =page_labels,
                uode_size=800,node_color='#8888CC',
                font_color='white',
                pos=positions
```



%scala

```
lnport org. apache . spark.HashPartltloner

val links =sc.parallelize(List((“MapR“,List(“Baidu”,“Blogger“)),(“Baidu“,List(“MapR“)),(“Blogger“,List(“Google“,“Baidus“)),(“Google”,List(“fi
partitionBy(new HashPartitioner(4)).persist()

var ranks=links.mapValues(v=>1.0)
```

```
import org.apache.spark.HashPartitioner
links: org.apache.spark.rdd.RDD[(Stning, List[String])] = ShuffledRDD[8] at partitionBy at command-646436246385985:4
ranks: org.apache.spark.rdd.RDD[(Stning, Double)] = MapPartitionsRDD[9] at mapValues at command-646436246385985:6
```

```
%scala
val contributions = links.join(ranks).flatMap {case (url,(links,rank))=›
links.map(dest=›(dest,rank/links.size)) }
```

```
contributions: ong.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[13] at flatMap at command-646436246385986:1
```

lscala
contrlbutlons . co11ect

```
res2: Array[(String, Double)] = Array((MapR,1.0), (Baidu,0.5), (Blogger,B.5), (Google,0.5), (Baidus,0.5), (MapR,1.0))
```

```
%scala
val ranks=contributions.reduceByKey((x,y)=> x+y).mapValues(v=› 0.15+0.85*v)
```

```
ranks: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitiousRDD[15] at mapValues at command-646436246385988:1
```

```
%scala
ranks.collect
```

```
res3: Array[(String, Double)] = Array((Baidus,0.575), (Google,8.575), (MapR,1.8499999999999999), (Blogger,0.575), (Baidu,B.575))
```

```
%scala
val lines = spark.read.textFile("dbfs:/FileStore/shared_uploads/kushagra.dce21@sot.pdpu.ac.in/links-1.txt").rdd
val iters = 20
val links = lines.map{ s=>
val parts = s.split("\\s+")
  (parts(0),parts(1))
  }.distinct().groupByKey().cache()

var ranks = links.mapValues(v =› 1.0)

for (i ‹- 1 to iters) {
  val contribs = links.join(ranks).values.flatMap{ case (urls,rank) =› val size = urls.size
  urls.map(url =› (url,rank/size))

  ranks = contribs.reduceByKey(_ + _).mapValues(0.15 + 0.85 * _)


val output = rauks.collect()

output.foreach(tup =› println(tup._1+" has rank: " + tup._2))
println("==================================")
output.foreach(tup =› println(tup._1+" has rank: " + f"${tup._2}%.3f"))
println("==================================")

ranks.collect()
val r = ranks.toDF("URL","PageRank")
r.show()
```

```
B has rank: 1.2982456036167454
A has rank: 1.4561335524686925
C has rank: 6.7688567446637524
D has rank: 0.4767641005174078

B has rank: 1.298
A has rank: 1.456
C has rank: 0.769
D has rank: 0.477


URL             PageRank|

  B|1.2982456036107454
  A|1. 4561335 524086925
  C|0.7688567440637524|
  D|0.4767641005174078|


lines: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1887] at rdd at command-646436246385990:1
iters: Int = 20
links: org.apache.spark.rdd.RDD[(Stning, Iterable[String])] = ShuffledRDD[1092] at groupByKey at command-646436246385998:6
ranks: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[1233] at mapValues at command-646436246385998:14
output: Array[(String, Double)] = Array((B,1.2982456830187454), (A,1.4561335524088925), (C,8.7688567448637524), (D,0.4767641005174878))
r: org.apache.spark.sql.DataFrame = [URL: string, PageRank: double]
```

```
%scala
import org.apache.spark.sql.SparkSession
r.cneate0nReplaceTempView(“Table_2“)

val r1=sqlContext.sql(“select PageRank from Table_2 where PageRank ‹ 1 “ )

r1.show()
```

```
            PageRank

0.7688567448637524|
0.4767641005174078|


import org.apache.spark.sql.SparkSession
r1: org.apache.spark.sql.DataFrame = [PageRank: double]
```

Name : Harsh Shah

Roll No. : 21BCP359

(G11 Div6)

LAB 11

```
#Structured Streaming using the Python DataFrames API
#Apache Spark includes a high-level stream processing API, Structured Streaming. Iu this notebook we take a quick look at how to use the Datal

#Sample Data
#we have some sample action data as files in /databricks-datasets/structured-streaming/events/ which we are going to use to build this appica
#To run this notebook, import it and attach it to a Spark cluster.
dbutils.fs.1s("/databricks-datasets/structured-streaming/events/")
```

```
Out[39]: [FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-0.json', name='file-0.json', size=72530, modificatic
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-I.jsou', name='file-1.json', size=72961, modificationTime=14E
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-10.json', name='file-10.json', size=73025, modificationTime=2
 FileInfo(path='dbfs:/databnicks-datasets/stnuctuned-stneaming/events/file-11.json', name='file-11.json', size=72999, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-12.json', name='file-12.json', size=72987, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-13.jsou', name='file-13.json', size=73006, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-14.json', name='file-14.json', size=73003, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-15.jsou', name='file-15.json', size=73007, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-16.json', name='file-16.json', size=72978, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-17.jsou', name='file-17.json', size=73008, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-18.json', name='file-18.json', size=73002, modificationTime=2
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-19.jsou', name='file-19.json', size=73014, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-2.json', name='file-2.json', size=73007, modificationTime=14f
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-20.json', name='file-20.json', size=72987, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-21.json', name='file-21.json', size=72983, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-22.jsou', name='file-22.json', size=73009, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-23.json', name='file-23.json', size=72985, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-24.jsou', name='file-24.json', size=73020, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-25.json', name='file-25.json', size=72980, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-26.jsou', name='file-26.json', size=73002, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-27.json', name='file-27.json', size=73013, modificationTime=2
 FileInfo(path='dbfs:/databnicks-datasets/stnuctuned-stneaming/events/file-28.json', name='file-28.json', size=73005, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-29.json', name='file-29.json', size=72977, modificationTime=2
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-3.json', name='file-3.json', size=72996, modificationTime=14f
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-30.json', name='file-30.json', size=73009, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-31.jsou', name='file-31.json', size=73008, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-32.json', name='file-32.json', size=72982, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-33.jsou', name='file-33.json', size=73033, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-34.json', name='file-34.json', size=72985, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-35.jsou', name='file-35.json', size=72974, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-36.json', name='file-36.json', size=73013, modificationTime=2
 FileInfo(path='dbfs:/databnicks-datasets/stnuctuned-stneaming/events/file-37.json', name='file-37.json', size=72989, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-38.json', name='file-38.json', size=72999, modificationTime=2
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-39.json', name='file-39.json', size=73013, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-4.json', name='file-4.json', size=72992, modificationTime=14f
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-40.jsou', name='file-40.json', size=72986, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-41.json', name='file-41.json', size=73019, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-42.jsou', name='file-42.json', size=72986, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-43.jsou', name='file-43.json', size=72990, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-44.jsou', name='file-44.json', size=73018, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-45.json', name='file-45.json', size=72997, modificationTime=2
 FileInfo(path='dbfs:/databnicks-datasets/stnuctuned-stneaming/events/file-46.json', name='file-46.json', size=72991, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-47.json', name='file-47.json', size=73009, modificationTime=2
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-48.json', name='file-48.json', size=72993, modificationTime=1
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-49.json', name='file-49.json', size=73496, modificationTime=1
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-5.jsou', name='file-5.json', size=72998, modificatiorTime=14E
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-6.jsou', name='file-6.json', size=72997, modificationTime=14f
 FileIufo(path='dbfs:/databricks-datasets/structured-streaming/events/file-7.jsou', name='file-7.json', size=73022, modificationTime=14f
 FileInfo(path='dbfs:/databricks-datasets/structured-streaming/events/file-8.jsou', name='file-8.json', size=72997, modificationTime=14f
 FileInfo(path='dbfs:/databnicks-datasets/structured-stneaming/events/file-9.json', name='file-9.json', size=72970, modificationTime=14f
```

```
#There are about 50 JSON files in the directory. Let's see what each JSON file contains.
dbutils.fs.head("/databricks-datasets/structured-streaming/events/file-0.json")
#Each line in the file contains JSON record with two fields - time and action. Let's try to analyze these files interactively.
```

```
[Truncated to first 65536 bytes]
Out[40]: '{"time":1469501107,"action":"Open"}\n{"time":1469501147,"action":"Opeu"}\n{"time":1469501202,"action":"Open"}\n{"time":146950]
```

*** WARNING: max output size exceeded, skipping output. ***

“Opeu“}\n{”time“:1469504710,“action“:“Open"}\n{“time“:1469504710,“action“:“Open“}\n{“time":1469504710,“action“:“Open“}\n{“time“:14695047

```
%python
#Batch/Interactive Processing
#The usual first step in attempting to process the data is to interactively query the data. Let's defiue a static DataFrame on the files, an

from pyspark.sql.types import *
```

```
&python
inputPath  =  "/databricks-datasets/structured-streaming/events/"
```

```
%python
# Since we know the data format already, let's define the schema to speed up processing (no need for Spark to infer schema)
jsonschema = StructType([ StructField("time", TimestampType(), True), StructField("action", StriugType(), True) ])
```

```
&python
# Static DataFname representing data in the JSON files
statlcInputDF = (
  spark
    .read
    .schema(jsonSchema)
    .json(inputPath)
```

```
%python
display(staticInputDF)
```

time	action
2016-07-28T04:19:28.000+0000	Close
2016-07-28T04:19:28.000+0000	'Close
2016-07-28T04:19:29.000+0000	'Open
2016-07-28T04:19:31.000+0000	'Close
2016-07-28T04:19:31.000+0000	'Open
2016-07-28T04:19:31.000+0000	'Open
2016-07-28T04:19:32.000+0000	'Close
2016-07-28T04:19:33.000+0000	'Close
2016-07-28T04:19:35.000+0000	'Close
3n1n.n7.39Tnd-dQ-2n nnn+nnnn	"mnc•n

```
%python
#Now we can compute the number of "open" and "close" actions with one hour windows. To do this, we will group by the action column and 1 hou
from pyspark.sql.functions import *    # for window() function

staticCountsDF = (
  statlcInputDF
    .groupBy(
       stat1cInputDF.act ton,  w1ndoW(stat1cInputDF.tlne,  "1  hour"))
    .count()

staticCountsDF.cache()

# Register the DataFrame as table 'static_counts'
staticCountsDF.create0rReplaceTempView("static_counts")
```

```
%sql select action, sum(count) as total_count fnom static_counts gnoup by action
```

action	total_count
Close	50000
Open	50000

```
%sql select action, date_format(window.end, "MMM-dd HH:mm") as time, count from static_counts order by time, action
```

action	time	count
Close	Jul-26 03:00	11
Open	Jul-26 03:00”	179
Close	Jul-26 04:00	344
Open	Jul-26 04:00	1001
Close	Jul-26 05:00”	815
Open	Jul-26 05:00	999
Close	Jul-26 06:00	1003
Open	Jul-26 06:00	1000
Close	Jul-26 07:00	1011
Open	Jul-26 07:00	993

&python
#Note the two ends of the graph. The close actions are generated such that they are after the corresponding open actions, so there are more

&python
tfstream Processi ng

```
#Now that we have analyzed the data interactively, let's convert this to a streaming query that continuously updates as data comes. Since we

from pyspank.sql.functions import *

# Similar to definition of staticInputDF above, just using ’readstream’ instead of ’read’

streamingInputDF = (
  spark
    . readst ream
    .schema(jsonSchema)               # Set the schema of the JSON data
    .optiou(“maxFilesPerTrigger“, 1)  # Treat a sequence of files as a stream by picking one file at a time
    .json(inputPath)


# Same query as staticInputDF
streamingCountsDF = (
  st reamIngInputDF
    . groupBy(
      st rearingInputDF . act ton ,
      window(streamingInputDF.time, “5 minutes“))
    .count()


# Just to check, is this DF actually a streaming DF?
streamingCountsDF.isStreaming
```

Out [50] : True

```
%python
#As you can see, streamingCountsDF is a streaming Dataframe (streamingCountsDF.isStreaming was true). You can start streaming computation, b

spark.conf.set(“spark.sql.shuffle.partitions", “2”)  # keep the size of shuffles small

query = (
  streamiugCouutsDF
    .writeStream
    .format(“memory“)       # memory = store iu-memory table
    .queryName(“counts“)    # counts = name of the in-memory table
    .outputMode(“complete“)   # complete = all the counts should be in the table
    .stant()


%python
#query is a handle to the streaming query that is running in the background. This query is continuously picking up files and updating the wi

#Note the status of query in the above cell. The progress bar shows that the query is active. Furthermore, if you expand the > counts above,

#Let's wait a bit for a few files to be processed and then interactively query the in-memory counts table.
from time import sleep
sleep(5)  # wait a bit for computation to start
```

```
%sql select action, date_format(window.end, “MMM-dd HH:mm“) as time, count from couuts order by time, action
```

action	time	count
Open	Jul-26 02:50	32
Close	Jul-26 02:55	5
Open	Jul-26 02:55	66
Close	Jul-26 03:00	6
Open	Jul-26 03:00	81
Close	Jul-26 03:05	5
Open	Jul-26 03:05	86
Close	Jul-26 03:10	14
Open	Jul-26 03:10	76
W.mom	liiK9C n2'16'	C

```
%python
sleep(5)  # wait a bit more for more data to be computed
```

```
%sql select action, date_format(wiudow.end, “MMM-dd HH:mm“) as time, count from couuts order by time, action
```

action	time	count
Open	Jul-26 02:50	32
Close	Jul-26 02:55	5
Open	Jul-26 02:55	66
Close	Jul-26 03:00	6
Open	Jul-26 03:00”	81
Close	Jul-26 03:05”	5
Open	Jul-26 03:05”	86
Close	Jul-26 03:10”	14
Open	Jul-26 03:10”	76
Close	Jul-26 03:15	16

```
#Also, let's see the total number of “opens“ and “closes“.
#%sql select action, sum(count) as total_count from counts group by action order by action

#If you keep running the above query repeatedly, you will always find that the number of “opens" is more than the number of “closes“, as exp

#Note that thene are only a few files, so consuming all of them there will be no updates to the counts. Rerun the queny if you want to inter

#Finally, you can stop the query running in the background, either by clicking on the 'Cancel' link in the cell of the query, or by executin
query.stop()
```

```
%scala
import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}

val sentenceData = spark.createDataFrame(Seq(
  (0.0, “Hi I heard about Spark“),
  (0.0, “I wish Java could use case classes"),
  (1.0, “Logistic regression models are neat”)
)).toDF(“label“, "seuteuce")

val tokenizer = new Tokenizer().setInputCol(“sentence“).set0utputCol(“words“)
val wordsData = tokenizer.transform(sentenceData)

val hashingTF = new HashingTF()
  .setIuputCol(“words“).set0utputCol(“rawFeatures“).setNumFeatures(20)

val featurizedData = hashingTF.transform(wordsData)
// alternatively, CountVectorizer can also be used to get term frequency vectors

val idf = new IDF().setInputCol(“rawFeatunes”).setOutputCol("featunes”)
val idfModel = idf.fit(featurizedData)

val rescaledData = idfModel.transform(featurizedData)
rescaledData.select(“label“, "features“).show()
```

```
label|             I-eatures

  6. 6|(20, [6, 8, 13, 16] , [. . .
  0. 6|  20, [0, 2, 7, 13, 15, . . .
  1. 0|(20, [3, 4, 6, 11, 19] . . .
```

```
import org.apache.spark.ml.feature.(HashingTF, IDF, Tokenizer}
sentenceData: org.apache.spark.sql.DataFrame = [label: double, sentence: string]
tokenizer: org.apache.spark.ml.feature.Tokenizer = tok_194117516f2e
wordsData: org.apache.spark.sql.DataFrame = [label: double, sentence: string ... 1 more field]
hashingTF: org.apache.spark.ml.feature.HashingTF = HashingTF: uid=hashingTF_b127e195334a, binary=false, numFeatures=20
featurizedData: org.apache.spark.sql.DataFrame = [label: double, sentence: string ... 2 more fields]
idf: org.apache.spark.ml.feature.IDF = idf_d47dd593f9c2
idfModel: org.apache.spark.ml.feature.IDFModel = IDFModel: uid=idf_d47dd593f9c2, numDocs=3, numFeatures=20
rescaledData: org.apache.spark.sql.DataFrame = [label: double, sentence: string ... 3 more fields]
```

```
%scala
import org.apache.spark.sql.fuuctious._
import org.apache.spark.sql.expressions.Window

val df = Seq(
  (“2021-01-01 00:00:00“, 100),
  (“2021-01-01 00:01:30", 150),
  (“2021-01-01 00:02:30“, 200),
  (“2021-01-01 00:03:00“, 50),
  (“2021-01-01 00:04:00“, 100),
  (“2021-01-01 00:05:30“, 150),
  (“2021-01-01 00:06:00“, 75),
  (“2021-01-01 00:07:00", 125),
  (“2021-01-01 00:08:00“, 50),
  (“2821-8l -8l  88: 89 : 38", 288),
  ("2021-01 -01  66 : 11 : 30",  2OO)
).toDF( "I lmestanp", "bytes_sent" )

val windowspec = Window.orderBy(“timestamp“)
  . rowsBetween ( -2,  2)

val rollingsum = sum(“bytes_sent“).over(windowspec)

val result = df.select(col(“timestamp“), col(“bytes_sent“), rollingSum.as("rolling_sum“))

result.show()
```

```
            timestamp|bytes_sent|rolling_sum|

2821-81-81 88:88:88|        tee         450
2021-01-01 00:01:30|        150|        500|
2021-01-01 00:02:38|        200|        600|
2021-81-01 00:03:0B|         50|        650|
2021-81-01 80:84:0B|        100|        575|
2021-01-01 00:05:30|        150|        500|
2021-01-01 00:06:00|         75|        500|
2021-81-01 00:07:00|        125|        600|
2021-01-01 00:08:00|         50|        650|
2021-81-01 00:09:38|        200|        575|
2021-01-01 00:11:30|       z00         450|
```

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
df: org.apache.spark.sql.DataFrame = [timestamp: string, bytes_sent: int]
windowspec: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@1afc31c3
rolliugsum: org.apache.spark.sql.Column = sum(bytes_sent) OVER (ORDER BY timestamp ASC NULLS FIRST ROWS BETWEEN -2 FOLLOWING AND 2 FOLLOWING)
result: org.apache.spark.sql.DataFrame = [timestamp: string, bytes_sent: int ... 1 more field]
```

```
%scala
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._

//  #create a sample dataframe
val df = Seq(
  (1, "a"),
  (2, "b"),
  (3, "c"),
  (4, "d"),
  (5, "e")
).toDF("Id", "va1ue")

//  def-1ne a window speclf-lcatlon
val windowspec = Window.orderBy("id").rowsBetween(-1, 1)

// define the window function to apply
val windowFunction = avg("id").over(windowspec)

// apply the wiudow function to the dataframe
val resultDF = df.withColumn("movingAvg", windowFunction)

resultDF.show()
```

```
 id|value|movingAvg|

  1|    a|      1.5|
  2|    b|      2.8|
  3|    c|      3.8|
  4|    d|      4.8|
  5|    e|      4.5|
```

```
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._
df: org.apache.spark.sql.DataFrame = [id: int, value: string]
windowspec: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@22c4f1e2
windowFunction: org.apache.spark.sql.Column = avg(id) OVER (ORDER BY id ASC NULLS FIRST R0wS BETWEEN -1 FOLLOWING AND 1 FOLLOWING)
resultDF: org.apache.spark.sql.DataFrame = [id: int, value: string ... 1 more field]
```

```
# Demonstration for range between
```

```
%scala
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._

// create a sample dataframe
val df = Seq(
  (1, "a", 10),
  (2, "b", 20),
  (3, "c", 30),
  (4, "d", 40),
  (5, "e", 50)
).toDF(“id”, "value", "amount")

// defiue a wiudow specification based on a range of values
val windowspec = Wiudow.orderBy(“id“).rangeBetween(-10, 10)

// define the window function to apply
val windowFunction = sum(“amount“).over(windowspec)

// apply the window function to the dataframe
val resultDF = df.withColumn(“runningSum“, windowFunction)

resultDF.show()
```

```
 id|value|amount|runningSum|

  1|    a|    10|       15O
  2|    b|    20|       150|
  3|    c|    38|       150|
  4|    d|    40|       150|
  5|    e|    50|       150|
```

```
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._
df: org.apache.spark.sql.DataFrame = [id: int, value: string ... 1 more field]
windowspec: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@3dda689e
windowFunction: org.apache.spark.sql.Column = sum(amount) OVER (ORDER BY id ASC NULLS FIRST RANGE BETWEEN -10 FOLLOWING AND 10 FOLLOWING)
resultDF: org.apache.spark.sql.DataFrame = [id: int, value: string ... 2 more fields]
```

```
%scala
import org.apache.spark.sql.fuuctious._
import org.apache.spark.sql.expressions.Window

val df = Seq(
  ("2021-01-01 09:00:00", 100.0),
  ("2021-01-01 09:1B:00", 120.0),
  ("2021-01-01 09:28:00", 110.0),
  ("2021-01-01 09:30:00", 90.0),
  ("2021-01-01 09:48:00", 95.0),
  ("2021-01-01 09:50:00", 105.0),
  ("2021-01-01 10:00:00", 125.0),
  ("2021-01-01 10:18:00", 130.0),
  ("2021-01-01 10:20:00", 140.0),
  ("2021-01-01 18:38:00", 135.0),
  ("2021-01-01 10:40:00", 130.0),
  ("2021-01-01 10:58:00", 125.0)
).toDF("timestamp“, "price")

val windowspec = Wiudow.orderBy(“timestamp“)
  .rangeBetweeu(-60 * 60, 0)

val rollingAvg = avg(“price“).over(windowspec)

val result = df.select(col(“timestamp“), col(“price”), rollingAvg.as(“rollin  avg”))

result.show()
```

```
'Project [timestamp#1697, price#1698, avg(price#1698) windowspecdefinition(timestamp#1697 ASC NULLS FIRST, specifiedwindowframe(RangeFrame, ca
st(-3688 as string), currentrow$())) AS rolling_avg#1702]
+- Project [_1#1692 AS timestamp#1697, _2#1693 AS price#1698]
   +- LocalRelation [_1#1692, _2#1693]

        at org.apache.spark.sql.catalyst.analysis.package$AnalysisErrorAt.dataTypeMismatch(package.scala:83)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$6(CheckAnalysis.scala:314)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$6$adapted(CheckAnalysis.scala:284)
        at org.apache.spark.sql.catalyst.trees.TneeNode.foreachUp(TneeNode.scala:302)
        at org.apache.spark.sql.catalyst.trees.TreeNode.$anonfun$foneachUp$1(TreeNode.scala:301)
        at org.apache.spark.sql.catalyst.trees.TreeNode.$anonfun$foneachUp$1$adapted(TreeNode.scala:301)
        at scala.collection.Iterator.foreach(Iterator.scala:943)
        at scala.collection.Iterator.foreach$(Iterator.scala:943)
        at scala.collection.AbstractIterator.foreach(Iterator.scala:1431)
        at scala.collection.IterableLike.foreach(IterableLike.scala:74)
        at scala.collection.IterableLike.foreach$(IterableLike.scala:73)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:56)
        at org.apache.spark.sql.catalyst.trees.TreeNode.foreachUp(TreeNode.scala:3B1)
        at org.apache.spark.sql.catalyst.trees.TreeNode.$anonfun$foneachUp$1(TreeNode.scala:301)
        at org.apache.spark.sql.catalyst.trees.TreeNode.$anonfun$foreachUp$1$adapted(TreeNode.scala:3B1)
        at scala.collection.Iterator.foreach(Iterator.scala:943)
        at scala.collection.Iterator.foreach$(Iterator.scala:943)
        at scala.collection.AbstractIterator.foreach(Iterator.scala:1431)
        at scala.collection.IterableLike.foreach(IterableLike.scala:74)
        at scala.collection.IterableLike.foreach$(IterableLike.scala:73)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:56)
        at org.apache.spark.sql.catalyst.trees.TneeNode.foreachUp(TneeNode.scala:301)
        at org.apache.spark.sql.catalyst.trees.TreeNode.$anonfun$foneachUp$1(TreeNode.scala:301)
        at org.apache.spark.sql.catalyst.trees.TreeNode.$anonfuu$foreachUp$1$adapted(TreeNode.scala:301)
        at scala.collection.Iterator.foreach(Iterator.scala:943)
        at scala.collection.Iterator.foreach$(Iterator.scala:943)
        at scala.collection.AbstractIterator.foreach(Iterator.scala:1431)
        at scala.collection.IterableLike.foreach(IterableLike.scala:74)
        at scala.collection.IterableLike.foreach$(IterableLike.scala:73)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:56)
        at org.apache.spark.sql.catalyst.trees.TneeNode.foreachUp(TneeNode.scala:301)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$5(CheckAnalysis.scala:284)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$5$adapted(CheckAnalysis.scala:284)
        at scala.collection.immutable.Stream.foreach(Stream.scala:533)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$1(CheckAnalysis.scala:284)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$1$adapted(CheckAnalysis.scala:170)
        at org.apache.spark.sql.catalyst.trees.TreeNode.foreachUp(TreeNode.scala:302)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.checkAnalysis8(CheckAnalysis.scala:17B)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.checkAnalysis8$(CheckAnalysis.scala:167)
        at org.apache.spark.sql.catalyst.analysis.Analyzer.checkAnalysis0(Analyzer.scala:289)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis$1(CheckAnalysis.scala:163)
        at scala.runtime.java8.JFunction0$mcV$sp.apply(JFunctionB$mcV$sp.java:23)
        at com.databricks.spark.util.FrameProfiler$.record(FrameProfiler.scala:80)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.checkAnalysis(CheckAnalysis.scala:153)
        at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.checkAnalysis$(CheckAnalysis.scala:153)
        at org.apache.spark.sql.catalyst.analysis.Analyzer.checkAnalysis(Analyzer.scala:289)
        at org.apache.spark.sql.catalyst.analysis.Analyzer.$anonfun$executeAndCheck$1(Analyzer.scala:343)
        at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper$.markInAnalyzer(AnalysisHelper.scala:402)
        at org.apache.spark.sql.catalyst.analysis.Analyzer.executeAndCheck(Analyzer.scala:34B)
        at org.apache.spark.sql.execution.QueryExecution.$anonfun$analyzed$1(QueryExecution.scala:171)
        at com.databricks.spark.util.FrameProfiler$.record(FrameProfiler.scala:80)
        at org.apache.spark.sql.catalyst.QueryPlanningTracker.measurePhase(QueryPlanningTracker.scala:352)
        at org.apache.spark.sql.execution.QueryExecution.$anonfun$executePhase$4(QueryExecution.scala:393)
        at org.apache.spark.sql.execution.QueryExecution$.withInternalError(QueryExecution.scala:841)
        at org.apache.spark.sql.execution.QueryExecution.$anonfun$executePhase$2(QueryExecution.scala:393)
        at com.databricks.util.LexicalThreadLocal$Handle.runWith(LexicalThreadLocal.scala:63)
        at org.apache.spark.sql.execution.QueryExecution.$anonfun$executePhase$1(QueryExecution.scala:389)
        at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:1073)
        at org.apache.spark.sql.execution.QueryExecution.executePhase(QueryExecution.scala:389)
        at org.apache.spark.sql.execution.QueryExecution.analyzed$lzycompute(QueryExecution.scala:165)
        at org.apache.spark.sql.execution.QueryExecution.analyzed(QueryExecution.scala:165)
        at org.apache.spark.sql.execution.QueryExecution.assertAnalyzed(QueryExecution.scala:155)
        at org.apache.spark.sql.Dataset$.$anonfun$ofRows$1(Dataset.scala:100)
        at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:1073)
        at org.apache.spark.sql.SparkSession.$anonfun$withActiveAndFrameProfiler$1(SparkSession.scala:1080)
        at com.databricks.spank.util.FrameProfiler$.record(FrameProfiler.scala:80)
        at org.apache.spark.sql.SparkSession.withActiveAndFrameProfiler(SparkSession.scala:1080)
        at org.apache.spark.sql.Dataset$.ofRows(Dataset.scala:98)
        at org.apache.spark.sql.Dataset.$anonfun$org$apache$spark$sql$Dataset$$withPlan$1(Dataset.scala:4414)
        at com.databricks.spank.util.FrameProfiler$.record(FrameProfiler.scala:80)
        at org.apache.spark.sql.Dataset.org$apache$spark$sql$Dataset$$withPlan(Dataset.scala:4414)
        at org.apache.spark.sql.Dataset.select(Dataset.scala:1621)
```

HADOOP SETUP AND INSTALLATION

Hadoop is an open-source framework used for distributed storage and processing of large data sets using simple programming models. Here's a detailed guide on setting up and installing Hadoop:

Prerequisites:

1. **Java 8 runtime environment (JRE):** Hadoop 3 requires a Java 8 installation. I prefer using the ofline installer

   https://www.java.com/en/download/windows_ofline.jsp

2. **Java 8 development Kit (JDK):** Download JDK from Oracles Website.

   https://www.oracle.com/java/technologies/downloads/#java8-windows

3. To unzip downloaded Hadoop binaries, we should **install 7zip.**

   https://www.7-zip.org/download.html

4. **Link for installing Hadoop**

   https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.2.4/hadoop-3.2.4.tar.gz

5. **Download Hadoop binaries:** Download Hadoop binaries from the Apache Hadoop website.

   https://hadoop.apache.org/releases.html

6. After unpacking the package, we should add the Hadoop native IO libraries.
7. **Hadoop Native IO Libraries: Download libraries from following**

   https://1drv.ms/f/s!ArSg3Xpur4Grml7l087JBp_4bzks?e=aSqIQV

SETUP STEPS :

Add environment variables:

- Set HADOOP_HOME to the Hadoop installation directory (C:\hadoopsetup\hadoop-3.2.4).
- Add %HADOOP_HOME%\bin and %JAVA_HOME%\bin to the PATH environment variable.

C:\hadoopsetup\hadoop-3.2.4

C:\Progra~1\Java\jdk-1.8

%HADOOP_HOME%\bin

%JAVA_HOME%\bin

Configure Core-Site (C:\hadoopsetup\hadoop-3.2.4\etc\hadoop\core-site.xml):

```
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9820</value>
```

</property>

Configure Hadoop Env (C:\hadoopsetup\hadoop-3.2.4\etc\hadoop\hadoop-env.cmd):

- Set JAVA_HOME to your JDK installation directory (C:\Progra~1\Java\jdk-1.8).

  set JAVA_HOME=C:\Progra~1\Java\jdk-1.8

Error - The system cannot find the path specified. Error: JAVA_HOME is incorrectly set. Please update C:\hadoopsetup\hadoop-3.2.4\etc\hadoop\hadoop-env.cmd '-Xmx512m' is not recognized as an internal or external command, operable program or batch file.

Solution

1. In the cmd line, charge the directory that contain the jdk (in my case C:\Program Files\Java\jdk1.8.0_73).
2. execute the following line "***for %I in (.) do echo %~sI***" to display the short name of your installed jdk (in my case C:\PROGRA~1\Java\JDK18~1.0_7)

Configure HDFS-Site (C:\hadoopsetup\hadoop-3.2.4\etc\hadoop\hdfs-site.xml):

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:///C:/hadoopsetup/hadoop-3.2.4/data/dfs/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:///C:/hadoopsetup/hadoop-3.2.4/data/dfs/datanode</value>

</property>

Configure Mapred-Site (C:\hadoopsetup\hadoop-3.2.4\etc\hadoop\mapred-site.xml):

<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
<description>MapReduce framework name</description>
</property>

Configure YARN-Site (C:\hadoopsetup\hadoop-3.2.4\etc\hadoop\yarn-site.xml):

<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shufle</value>
<description>Yarn Node Manager Aux Service</description>
</property>

STARTING TERMINALS(Starting Hadoop Services):

\start-dfs.cmd

./start-yarn.cmd

Jps

Or

Hadoop/Hadoop-3.4.2/sbi • start-all

- **LINKS TO VIEW STATUS:**

http://localhost:9870/dfshealth.html

http://localhost:9864/datanode.html

http://localhost:8088/cluster

This setup guide covers the basic installation and configuration of Hadoop. Adjust paths and configurations as per your environment.

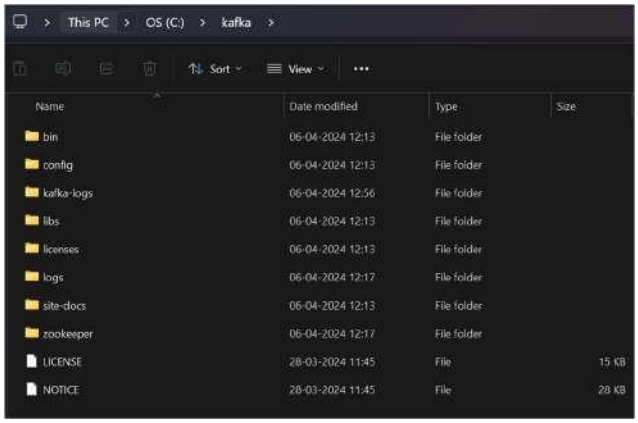
KAFKA CONFIG AND PRODUCER-CONSUMER PROGRAM

1. **Download Kafka:**
    - Download Kafka from Apache Kafka website.
    - Choose the appropriate version (e.g., 3.6.2) and Scala version (e.g., 2.13).

    Video Reference – https://www.youtube.com/watch?v=BwYFuhVhshI

2. **Extract and Copy Kafka Folder:**
    - Extract the downloaded Kafka folder.
    - Copy the extracted Kafka folder to the C: drive and rename it to "kafka".

NOTE: After downloading and extracting the kafka folder , copy it to the C: drive and name the folder “kafka”. This folder should contain



3. **Set Temp Dir Paths:**
    - Open the Kafka folder and navigate to the "config" directory.
    - Edit the "server.properties" file:
        - Set log.dirs = c:/kafka/kafka-logs.
    - Edit the "zookeeper.properties" file:
        - Set dataDir = c:/kafka/zookeeper.

4. **Start Zookeeper:**
- Open a terminal and navigate to the Kafka directory (cd c:/kafka).
- Start Zookeeper using the following command:

    .\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

5. **Start Kafka Server (Broker):**

- Open another terminal and navigate to the Kafka directory (cd c:/kafka).
- Start the Kafka server (broker) using the following command
  .\bin\windows\kaffia-server-start.bat .\config\server.properties

6. **Create a Topic:**

- Open a new terminal and navigate to the Kafka directory (cd c:/kafka).
- Create a topic using the following command:
  .\bin\windows\kaffia-topics.bat --create --bootstrap-server localhost:9092 --topic test –replication-factor 1 –partitions 3

7. **Start Producer:**

- Open a new terminal and navigate to the Kafka directory (cd c:/kafka).
- Start the producer using the following command
  .\bin\windows\kaffia-console-producer.bat --broker-list localhost:9092 --topic test

8. **Start Consumer:**

   Open a new terminal and navigate to the Kafka directory (cd c:/kafka).
   Start the consumer using the following command:

   >.\bin\windows\kaffia-console-consumer.bat --bootstrap-server localhost:9092 --topic test --from-beginning.

9. ***Send Messages:***

   - In the producer terminal, type a message and press Enter.
   - The message should appear in the consumer terminal.

Make sure to follow these steps in order and in separate terminals. This setup allows you to run a basic Kafka producer-consumer program locally.

PIG SETUP AND INSTALLATION ON HADOOP

Prerequisites:

1. Hadoop Setup in commodity hardware
2. Download Hadoop from here
3. Download Pig from here
4. Unpack the Hadoop and Pig packages in the C: drive

Setting Up Environment Variables:

1. In user variables, set PIG_HOME to C:\pigsetup\pig-0.17.0
2. In system variables, add %PIG_HOME% to the PATH

Edit Pig Configuration:

1. Go to the bin folder of the Pig extracted files
2. Edit the pig.cmd file and update set HADOOP_BIN_PATH = %HADOOP_HOME%\libexec

- **STARTING TERMINALS (Run as Admin)**

  - To run Pig on the local machine, type the command:

    pig -x local

    Change the path to the **bin** folder of Pig in the command prompt:

    C:\pigsetup\pig-0.17.0\bin

  - To run Pig on Hadoop, type the command:

    pig

MONGODB SETUP ON DATABRICKS

- **STEPS FOR INSTALLATION**

1. Create a Databricks Cluster and Add the Connector as a Library
2. Create a Databricks cluster.
3. Now go to the cluster detail page and select the Libraries tab.
4. Click the Install New button.
5. Select Maven as the Library Source.
6. Use the Search Packages feature, find 'mongo-spark'. This should point to

   org.mongodb.spark:mongo-spark-connector_2.12:3.0.1 or newer.

7. Click Install.
8. For any errors visit MongoDB documentation.

- **STEPS FOR RUNNING MONGODB ON DATABRICKS (on MongoDB atlas)**

1. Create a MongoDB Atlas Instance
2. Sign up for MongoDB Atlas.
3. Create the free tier MongoDB cluster.
4. Enable Databricks clusters to connect to the cluster by adding the external IP addresses for the Databricks cluster nodes to the ***whitelist in Atlas or allow access from anywhere***.
5. Now in MongoDB Atlas load the sample data set once the cluster is up and running.
6. Now to view the collection go to browse collection.

- **STEPS TO RUN IN DATABRICKS CLUSTERS**

1. Update Spark Configuration with the Atlas Connection String

2. Get the connect string under the Connect dialog in MongoDB Atlas. It looks like ***"mongodb+srv://<username>:<password>@<databasename>.xxxxx.mongodb.net/"***
3. Now in the Databricks in your cluster configuration, under **Advanced Options (bottom of page**), paste the connection string for both the ***spark.mongodb.output.uri*** and ***spark.mongodb.input.ur***i variables. Don't forgot to enter the username and password fields correctly. In this way all the workbooks you are running on the cluster will use this configuration.
4. Alternatively you can explicitly set the option when calling APIs like: *spark.read.format("mongodb").option("spark.mongodb.input.uri", connectionString).load().*
5. Now to get the sample data set on the cluster run the following command

connectionString='mongodb+srv://CONNECTION_STRING_HERE/

database="sample_movies"

collection="movies"

OR

df = spark.read.format("mongodb").option("database", database).option("spark.mongodb.input.uri", connectionString).option("collection","movies").load()

df.printSchema()

- **<u>Create a temp view</u>**
  df.createOrReplaceTempView("temp")
  filtered_df = spark.sql("SELECT customer FROM temp WHERE movies='New York'")

  display(filtered_df)