```
%md
#

<h1></h1>
```

Read csv file and create paired RDD

```
%sh pwd
```

```
/home/ubuntu/test_spark/zeppelin-0.8.1-bin-netinst
```

```
// Input files
z.put("transactionFile", "/home/ubuntu/test_spark/sia/ch04_data_transactions.txt")
z.put("productFile", "/home/ubuntu/test_spark/sia/ch04_data_products.txt")
```

```
%sh
ls -l {transactionFile}
ls -l {productFile}
```

```
-rw-rw-r-- 1 ubuntu ubuntu 34996 Mar 12 15:46 /home/ubuntu/test_spark/sia/ch04_data_transactions.txt
-rw-rw-r-- 1 ubuntu ubuntu 3466 Mar 12 15:46 /home/ubuntu/test_spark/sia/ch04_data_products.txt
```

```
%sh
head {transactionFile}
echo
echo
head {productFile}
```

```
2015-03-30#6:55 AM#51#68#1#9506.21
2015-03-30#7:39 PM#99#86#5#4107.59
2015-03-30#11:57 AM#79#58#7#2987.22
2015-03-30#12:46 AM#51#50#6#7501.89
2015-03-30#11:39 AM#86#24#5#8370.2
2015-03-30#10:35 AM#63#19#5#1023.57
2015-03-30#2:30 AM#23#77#7#5892.41
2015-03-30#7:41 PM#49#58#4#9298.18
2015-03-30#9:18 AM#97#86#8#9462.89
2015-03-30#10:06 PM#94#26#4#4199.15


1#ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU#9721.89#10
2#Mattel Little Mommy Doctor Doll#6060.78#6
3#Cute baby doll, battery#1808.79#2
4#Bear doll#51.06#6
5#LEGO Legends of Chima#849.36#6
6#LEGO Castle#4777.51#10
7#LEGO Mixels#8720.91#1
8#LEGO Star Wars#7592.44#4
9#LEGO Lord of the Rings#851.67#2
10#LEGO The Hobbit#7314.55#9
```

```
// Read transaction file into RDD using csv format

val transFile = sc.textFile(z.get("transactionFile").toString)
```

```
transFile: org.apache.spark.rdd.RDD[String] = /home/ubuntu/test_spark/sia/ch04_data_transactions.txt MapPartitionsRDD[596] at textFile at <console>:153
```

```
transFile.take(5)
```

```
res118: Array[String] = Array(2015-03-30#6:55 AM#51#68#1#9506.21, 2015-03-30#7:39 PM#99#86#5#4107.59, 2015-03-30#11:57 AM#79#58#7#2987.22, 2015-03-30#12:46 AM#51#50#6#7501.89, 2015-03-30#11:39 AM#86#24#5
```

```
val transData = transFile.map(_.split("#"))
transData.take(5)
```

```
transData: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[597] at map at <console>:149
res119: Array[Array[String]] = Array(Array(2015-03-30, 6:55 AM, 51, 68, 1, 9506.21), Array(2015-03-30, 7:39 PM, 99, 86, 5, 4107.59), Array(2015-03-30, 11:57 AM, 79, 58, 7, 2987.22), Array(2015-03-30, 12:
```

Create paired RDD

Paired RDD Functions

```
var transByCust = sc.textFile(z.get("transactionFile").toString)
                        .map(_.split("#"))
                        .map(trans => (trans(2).toInt, trans))
transByCust.take(5)
```

```
transByCust: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[601] at map at <console>:153
res120: Array[(Int, Array[String])] = Array((51,Array(2015-03-30, 6:55 AM, 51, 68, 1, 9506.21)), (99,Array(2015-03-30, 7:39 PM, 99, 86, 5, 4107.59)), (79,Array(2015-03-30, 11:57 AM, 79, 58, 7, 2987.22)),
```

```
// Another way

val transByCust1 = transData.keyBy(_(2).toInt)
transByCust1.take(5)
```

```
transByCust1: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[602] at keyBy at <console>:151
res121: Array[(Int, Array[String])] = Array((51,Array(2015-03-30, 6:55 AM, 51, 68, 1, 9506.21)), (99,Array(2015-03-30, 7:39 PM, 99, 86, 5, 4107.59)), (79,Array(2015-03-30, 11:57 AM, 79, 58, 7, 2987.22)),
```

```
transByCust1
```

```
res127: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[602] at keyBy at <console>:151
```

```
// Number of unique customers

val uniqueCustomerIds = transByCust.keys.distinct.count
```

```
uniqueCustomerIds: Long = 100
```

```
transByCust.keys
```

```
res122: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[607] at keys at <console>:150
```

```
// Group by customerId
transByCust.countByKey.take(10)
```

```
res123: scala.collection.Map[Int,Long] = Map(69 -> 7, 88 -> 5, 5 -> 11, 10 -> 7, 56 -> 17, 42 -> 7, 24 -> 9, 37 -> 7, 25 -> 12, 52 -> 9)
```

```
transByCust.countByKey.values.sum
```

```
res124: Long = 1000
```

```
transByCust.values.take(5)
```

```
res126: Array[Array[String]] = Array(Array(2015-03-30, 6:55 AM, 51, 68, 1, 9506.21), Array(2015-03-30, 7:39 PM, 99, 86, 5, 4107.59), Array(2015-03-30, 11:57 AM, 79, 58, 7, 2987.22), Array(2015-03-30, 12:
```

```
transByCust.countByKey.toSeq
```

```
res5: Seq[(Int, Long)] = ArrayBuffer((69,7), (88,5), (5,11), (10,7), (56,17), (42,7), (24,9), (37,7), (25,12), (52,9), (14,8), (20,8), (46,9), (93,12), (57,8), (78,11), (29,9), (84,9), (61,8), (89,9), (1
```

```
// Customer who purchased most items

transByCust.countByKey.toSeq.sortBy(_._2).last
```

```
res6: (Int, Long) = (53,19)
```

countByKey returns a scala object. If we want to group and sort keys in RDD

Operations which can cause a shuffle include repartition operations like repartition and coalesce, â€˜ByKey operations (except for counting) like groupByKey and reduceByKey, and join operations like cogroup and join.

Reference: RDD Programming guide

```
transByCust.map(t => (t._1, 1))
        .reduceByKey(_ + _)
        .sortBy(_._2, ascending=false)
        .take(1)
```

```
res128: Array[(Int, Int)] = Array((53,19))
```

```
// sorting doesn't cause shuffling
```

```
transByCust.map(t => (t._1, 1))
        .sortBy(t => t._1)
        .take(5)

res133: Array[(Int, Int)] = Array((1,1), (1,1), (1,1), (1,1), (1,1))

// All transactions by customerId 53
// NOTE: lookup() transfers results to driver

transByCust.lookup(53)

res14: Seq[Array[String]] = WrappedArray(Array(2015-03-30, 6:18 AM, 53, 42, 5, 2197.85), Array(2015-03-30, 4:42 AM, 53, 44, 6, 9182.08), Array(2015-03-30, 2:51 AM, 53, 59, 5, 3154.43), Array(2015-03-30, 

// Using filter() instead of lookup()

val transByCust53 = transByCust.filter(_._1 == 53)
transByCust53.take(10)

transByCust53: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[18] at filter at <console>:27
res7: Array[(Int, Array[String])] = Array((53,Array(2015-03-30, 6:18 AM, 53, 42, 5, 2197.85)), (53,Array(2015-03-30, 4:42 AM, 53, 44, 6, 9182.08)), (53,Array(2015-03-30, 2:51 AM, 53, 59, 5, 3154.43)), (5
```

Modifying values in paired RDDs

```
transByCust.filter(_._2(3).toInt == 25).take(10)

res13: Array[(Int, Array[String])] = Array((25,Array(2015-03-30, 5:55 AM, 25, 25, 1, 5089.02)), (17,Array(2015-03-30, 6:26 PM, 17, 25, 6, 7193.11)), (93,Array(2015-03-30, 7:27 AM, 93, 25, 7, 2749.15)), (

// Modify only transactions with product id 25 and multiple purchased quantity.

transByCust = transByCust.mapValues(tran => {
    if(tran(3).toInt == 25 && tran(4).toInt > 1) {
        val newPrice = tran(5).toDouble * 0.95
        tran(5) = f"$newPrice%1.2f"
    }
    tran
})

transByCust.filter(_._2(3).toInt == 25).take(10)

transByCust: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[4] at mapValues at <console>:27
res3: Array[(Int, Array[String])] = Array((25,Array(2015-03-30, 5:55 AM, 25, 25, 1, 5089.02)), (17,Array(2015-03-30, 6:26 PM, 17, 25, 6, 6833.45)), (93,Array(2015-03-30, 7:27 AM, 93, 25, 7, 2611.69)), (1
```

Adding or removing values for a key

flatMapValues() ***Pass each value in the key-value pair RDD through a flatMap function without changing the keys; this also retains the original RDD's partitioning.***

Customers who bought 5 or more dictionaries (id : 81)

```
transByCust.filter{case (custId, tran) => tran(3).toInt == 81 && tran(4).toInt > 5}.take(10)

res25: Array[(Int, Array[String])] = Array((85,Array(2015-03-30, 1:55 PM, 85, 81, 7, 9648.24)), (82,Array(2015-03-30, 3:40 AM, 82, 81, 6, 3665.45)), (47,Array(2015-03-30, 10:39 PM, 47, 81, 10, 1122.89)),

transByCust = transByCust.flatMapValues(tran => {
    if (tran(3).toInt == 81 && tran(4).toInt > 5) {
        val cloned = tran.clone()
        // Add a free toothbrush (id 70)
        cloned(3) = "70"; cloned(4) = "1"; cloned(5) = "0.00"
        List(tran, cloned)
    } else
        List(tran)
})

transByCust.filter{case (custId, tran) => tran(3).toInt == 70 && tran(5).toFloat == 0}.take(10)

transByCust: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[6] at flatMapValues at <console>:25
res4: Array[(Int, Array[String])] = Array((85,Array(2015-03-30, 1:55 PM, 85, 70, 1, 0.00)), (82,Array(2015-03-30, 3:40 AM, 82, 70, 1, 0.00)), (47,Array(2015-03-30, 10:39 PM, 47, 70, 1, 0.00)), (16,Array(
```

Reducing/Folding values by key

Find the customer who spent the most

```
// foldByKey causes shuffle

val custByTotalPrice = transByCust.mapValues(tran => tran(5).toFloat)
    .foldByKey(0)((p1, p2) => p1 + p2)
    .collect()

custByTotalPrice: Array[(Int, Float)] = Array((34,77332.586), (52,58348.02), (96,36928.57), (4,41801.35), (16,40696.02), (82,58722.58), (66,52130.01), (28,45534.297), (54,36307.043), (80,31794.62), (98,5

// Customer who spent most
custByTotalPrice.toSeq.sortBy(_._2).last._2

res29: Float = 100049.0
```

Merging multiple RDDs

```
val compTrans = Array(Array("2015-03-30", "11:59 PM", "76", "63", "1", "0.00"))

val compTransRDD = sc.parallelize(compTrans).map(tran => (tran(2).toInt, tran))


transByCust = transByCust.union(compTransRDD)
// compTransRDD.count
transByCust.filter{case (c, t) => t(2).toInt == 76 && t(3).toInt == 63}.take(10)

compTrans: Array[Array[String]] = Array(Array(2015-03-30, 11:59 PM, 76, 63, 1, 0.00))
compTransRDD: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[9] at map at <console>:29
transByCust: org.apache.spark.rdd.RDD[(Int, Array[String])] = UnionRDD[10] at union at <console>:32
res5: Array[(Int, Array[String])] = Array((76,Array(2015-03-30, 11:59 PM, 76, 63, 1, 0.00)))
```

Transforming and aggregatin values

aggregateByKey() **: similar to** ***foldByKey*** **and** ***reduceByKeyin*** **that it merges values and takes a zero value, but it also transforms values to another type.**

Function currying **is transforming a function with multiple arguments to a function with single argument.**

def aggregateByKey[U](zeroValue: U)(seqOp: (U, V) â‡' U, combOp: (U, U) â‡' U)(implicit arg0: ClassTag[U]): RDD[(K, U)]

```
// Find all products that customers purchased

val products = transByCust.aggregateByKey(List[String]())((prods, tran) => prods ::: List(tran(3)), (prods1, prods2) => prods1 ::: prods2)

products.take(10)

products: org.apache.spark.rdd.RDD[(Int, List[String])] = ShuffledRDD[635] at aggregateByKey at <console>:151
res134: Array[(Int, List[String])] = Array((34,List(34, 99, 66, 58, 59, 17, 58, 5, 38, 58, 93, 22, 48, 50)), (52,List(37, 69, 82, 11, 19, 58, 67, 51, 93)), (96,List(37, 31, 17, 88, 97, 81, 58, 78)), (4,L
```

Find the number of partitions in RDD

```
products.partitions.size

res13: Int = 10
```

Glom example

glom (the word means to grab) gathers elements of each partition into an array and returns a new RDD with those arrays as elements. The number of elements in the new RDD is equal to the number of its partitions.

```
val list = List.fill(500)(scala.util.Random.nextInt(100))
val glomRdd = sc.parallelize(list, 30).glom()
```

```
list: List[Int] = List(66, 67, 38, 86, 78, 12, 49, 76, 15, 36, 18, 40, 28, 87, 91, 16, 91, 46, 98, 45, 85, 24, 35, 12, 90, 85, 32, 40, 41, 43, 23, 16, 23, 54, 54, 72, 99, 91, 72, 0, 59, 89, 44, 57, 53, 8

glomRdd.count
glomRdd.collect()

res4: Array[Array[Int]] = Array(Array(66, 67, 38, 86, 78, 12, 49, 76, 15, 36, 18, 40, 28, 87, 91, 16), Array(91, 46, 98, 45, 85, 24, 35, 12, 90, 85, 32, 40, 41, 43, 23, 16, 23), Array(54, 54, 72, 99, 91,
```

Joining data

```
val transByProd = transData.map(tran => (tran(3).toInt, tran))

val totalByProd = transByProd.mapValues(t => t(5).toDouble).reduceByKey((t1, t2) => t1 + t2)

totalByProd.collect()

transByProd: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[7] at map at <console>:25
totalByProd: org.apache.spark.rdd.RDD[(Int, Double)] = ShuffledRDD[9] at reduceByKey at <console>:27
res4: Array[(Int, Double)] = Array((34,62592.43000000001), (52,57708.119999999995), (96,73536.78), (4,63520.21000000001), (16,46664.43000000001), (82,59612.96), (66,39755.84), (28,82055.45999999999), (54

val prodFile = sc.textFile(z.get("productFile").toString)
prodFile.take(5)

prodFile: org.apache.spark.rdd.RDD[String] = /home/ubuntu/test_spark/sia/ch04_data_products.txt MapPartitionsRDD[11] at textFile at <console>:27
res5: Array[String] = Array(1#ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU#9721.89#10, 2#Mattel Little Mommy Doctor Doll#6060.78#6, 3#Cute baby doll, battery#1808.79#2, 4#Bear doll#51.06#6, 5#LEGO Legend

val prodById = prodFile.map(_.split("#")).map(p => (p(0).toInt, p))
prodById.take(5)

prodById: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[13] at map at <console>:25
res6: Array[(Int, Array[String])] = Array((1,Array(1, ROBITUSSIN PEAK COLD NIGHTTIME COLD PLUS FLU, 9721.89, 10)), (2,Array(2, Mattel Little Mommy Doctor Doll, 6060.78, 6)), (3,Array(3, Cute baby doll, b

// Join totalByProd with prodById to find the details of each product and total sold.

val totalsAndProds = totalByProd.join(prodById)
totalsAndProds.take(5)

totalsAndProds: org.apache.spark.rdd.RDD[(Int, (Double, Array[String]))] = MapPartitionsRDD[16] at join at <console>:29
res7: Array[(Int, (Double, Array[String]))] = Array((34,(62592.43000000001,Array(34, GAM X360 Assassins Creed 3, 6363.95, 9))), (52,(57708.119999999995,Array(52, Essentials Crash Tag Team Racing PSP, 403

// Find the products that weren't sold
// leftOuterJoin() or rightOuterJoin()

var prodsAndTotals = prodById.leftOuterJoin(totalByProd)
prodsAndTotals.take(2)

prodsAndTotals: org.apache.spark.rdd.RDD[(Int, (Array[String], Option[Double]))] = MapPartitionsRDD[42] at leftOuterJoin at <console>:32
res19: Array[(Int, (Array[String], Option[Double]))] = Array((34,(Array(34, GAM X360 Assassins Creed 3, 6363.95, 9),Some(62592.43000000001))), (52,(Array(52, Essentials Crash Tag Team Racing PSP, 4037.85

var notSoldProds = prodsAndTotals
    .filter{case(prodId, productTotalTuple) => productTotalTuple._2 == None}
    .mapValues(_._1)


notSoldProds.take(2)

notSoldProds: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[44] at mapValues at <console>:29
res20: Array[(Int, Array[String])] = Array((20,Array(20, LEGO Elves, 4589.79, 4)), (63,Array(63, Pajamas, 8131.85, 3)))

// Another way

val notSoldProds1 = prodById.subtractByKey(totalByProd)

notSoldProds1.count

notSoldProds1: org.apache.spark.rdd.RDD[(Int, Array[String])] = SubtractedRDD[22] at subtractByKey at <console>:31
res12: Long = 4

// Another way

val prodCogroup = totalByProd.cogroup(prodById)

prodCogroup: org.apache.spark.rdd.RDD[(Int, (Iterable[Double], Iterable[Array[String]]))] = MapPartitionsRDD[19] at cogroup at <console>:29

val notSoldProds2 = prodCogroup.filter{case(prodId, productTotalTuple) => productTotalTuple._1.isEmpty}
notSoldProds2.count

notSoldProds2: org.apache.spark.rdd.RDD[(Int, (Iterable[Double], Iterable[Array[String]]))] = MapPartitionsRDD[23] at filter at <console>:27
res18: Long = 4

notSoldProds2.foreach(x => println(x._2._2.head.mkString(", ")))
```

Cartesian product

```
val c1 = sc.parallelize(List(1,2,3))
val c2 = sc.parallelize(List(4,5,6))

c1.cartesian(c2).collect

c1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[24] at parallelize at <console>:25
c2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[25] at parallelize at <console>:26
res20: Array[(Int, Int)] = Array((1,4), (1,5), (1,6), (2,4), (2,5), (2,6), (3,4), (3,5), (3,6))
```

Sorting data

```
// sortBy()

val sortedProdNames = totalsAndProds.sortBy(_._2._2(1))

sortedProdNames.take(5)

sortedProdNames: org.apache.spark.rdd.RDD[(Int, (Double, Array[String]))] = MapPartitionsRDD[38] at sortBy at <console>:29
res24: Array[(Int, (Double, Array[String]))] = Array((90,(48601.89,Array(90, AMBROSIA TRIFIDA POLLEN, 5887.49, 1))), (94,(31049.07,Array(94, ATOPALM MUSCLE AND JOINT, 1544.25, 7))), (87,(26047.72,Array(8
```

combineByKey()

- Very versatile and flexible.
- Used to implement *aggregateByKey(), foldByKey(),* and *reduceByKey()*.

```
def createCombiner = (t: Array[String]) => {
    val total = t(5).toDouble  // price
    val q = t(4).toInt  // quantity
    (total/q, total/q, q, total)  // min, max, count, total
}

def mergeVal: ((Double, Double, Int, Double), Array[String]) =>
    (Double, Double, Int, Double) = {
        case((mn, mx, c, tot), t) => {
            val total = t(5).toDouble
            val q = t(4).toInt
            (scala.math.min(mn, total/q), scala.math.max(mx, total/q), c + q, tot + total)
        }
    }

def mergeComb: ((Double, Double, Int, Double), (Double, Double, Int, Double)) => (Double, Double, Int, Double) = {
    case((mn1, mx1, c1, tot1), (mn2, mx2, c2, tot2)) => {
        (scala.math.min(mn1, mn2), scala.math.max(mx1, mx2), c1 + c2, tot1 + tot2)
    }
}

val avgByCust = transByCust.combineByKey(createCombiner, mergeVal, mergeComb, new org.apache.spark.HashPartitioner(transByCust.partitioner.size))
    .mapValues({case(mn, mx, cnt, tot) => (mn, mx, cnt, tot, tot/cnt)})

avgByCust.take(1)

createCombiner: Array[String] => (Double, Double, Int, Double)
mergeVal: ((Double, Double, Int, Double), Array[String]) => (Double, Double, Int, Double)
```

```
mergeComb: ((Double, Double, Int, Double), (Double, Double, Int, Double)) => (Double, Double, Int, Double)
avgByCust: org.apache.spark.rdd.RDD[(Int, (Double, Double, Int, Double, Double))] = MapPartitionsRDD[26] at mapValues at <console>:52
res11: Array[(Int, (Double, Double, Int, Double, Double))] = Array()
```

Converting RDD to DF by specifying schema using case class

```
case class Transaction(date: String, time: String, customerId: Int, productId: Int, quantity: Int, price: Float)

defined class Transaction

val transRDD = transData.map(trans => Transaction(trans(0), trans(1), trans(2).toInt, trans(3).toInt, trans(4).toInt, trans(5).toFloat))
transRDD.take(10)

transRDD: org.apache.spark.rdd.RDD[Transaction] = MapPartitionsRDD[17] at map at <console>:27
res7: Array[Transaction] = Array(Transaction(2015-03-30,6:55 AM,51,68,1,9506.21), Transaction(2015-03-30,7:39 PM,99,86,5,4107.59), Transaction(2015-03-30,11:57 AM,79,58,7,2987.22), Transaction(2015-03-30

val transDF = transRDD.toDF
transDF.show(10)

+----------+--------+----------+---------+--------+-------+
|      date|    time|customerId|productId|quantity|  price|
+----------+--------+----------+---------+--------+-------+
|2015-03-30| 6:55 AM|        51|       68|       1|9506.21|
|2015-03-30| 7:39 PM|        99|       86|       5|4107.59|
|2015-03-30|11:57 AM|        79|       58|       7|2987.22|
|2015-03-30|12:46 AM|        51|       50|       6|7501.89|
|2015-03-30|11:39 AM|        86|       24|       5| 8370.2|
|2015-03-30|10:35 AM|        63|       19|       5|1023.57|
|2015-03-30| 2:30 AM|        23|       77|       7|5892.41|
|2015-03-30| 7:41 PM|        49|       58|       4|9298.18|
|2015-03-30| 9:18 AM|        97|       86|       8|9462.89|
|2015-03-30|10:06 PM|        94|       26|       4|4199.15|
+----------+--------+----------+---------+--------+-------+
only showing top 10 rows

transDF: org.apache.spark.sql.DataFrame = [date: string, time: string ... 4 more fields]

transDF.printSchema

root
 |-- date: string (nullable = true)
 |-- time: string (nullable = true)
 |-- customerId: integer (nullable = false)
 |-- productId: integer (nullable = false)
 |-- quantity: integer (nullable = false)
 |-- price: float (nullable = false)
```

Reading directly into DF

```
val transDF = spark.read.format("csv")
                    .option("header", false)
                    .option("delimiter", "#")
                    .load(z.get("transactionFile").toString)
transDF.show()

+----------+--------+---+---+---+-------+
|       _c0|     _c1|_c2|_c3|_c4|    _c5|
+----------+--------+---+---+---+-------+
|2015-03-30| 6:55 AM| 51| 68|  1|9506.21|
|2015-03-30| 7:39 PM| 99| 86|  5|4107.59|
|2015-03-30|11:57 AM| 79| 58|  7|2987.22|
|2015-03-30|12:46 AM| 51| 50|  6|7501.89|
|2015-03-30|11:39 AM| 86| 24|  5| 8370.2|
|2015-03-30|10:35 AM| 63| 19|  5|1023.57|
|2015-03-30| 2:30 AM| 23| 77|  7|5892.41|
|2015-03-30| 7:41 PM| 49| 58|  4|9298.18|
|2015-03-30| 9:18 AM| 97| 86|  8|9462.89|
|2015-03-30|10:06 PM| 94| 26|  4|4199.15|
|2015-03-30|10:57 AM| 91| 18|  1|3795.73|
|2015-03-30| 7:43 AM| 20| 86| 10|1477.35|
|2015-03-30| 5:58 PM| 38| 39|  6| 1090.0|
|2015-03-30| 1:08 PM| 46|  6| 10|1014.78|
|2015-03-30|12:18 AM| 56| 48|  9|8346.42|
|2015-03-30| 1:18 AM| 11| 58|  4| 364.59|
|2015-03-30| 3:01 AM| 59|  9|  5|5984.68|
|2015-03-30|11:44 AM|  8| 35|  6| 1859.2|
|2015-03-30|12:05 PM| 23|  8|  3|1527.04|
|2015-03-30| 4:10 AM| 85| 93|  9|3314.71|
+----------+--------+---+---+---+-------+
only showing top 20 rows

transDF: org.apache.spark.sql.DataFrame = [_c0: string, _c1: string ... 4 more fields]

transDF.printSchema

root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
```

Specifying schema while reading

```
val transDF = spark.read.format("csv")
                    .option("header", false)
                    .option("delimiter", "#")
                    .schema("date DATE, time STRING, customerId INT, productId INT, quantity INT, price DOUBLE")
                    .load(z.get("transactionFile").toString)
transDF.show()

+----------+--------+----------+---------+--------+-------+
|      date|    time|customerId|productId|quantity|  price|
+----------+--------+----------+---------+--------+-------+
|2015-03-30| 6:55 AM|        51|       68|       1|9506.21|
|2015-03-30| 7:39 PM|        99|       86|       5|4107.59|
|2015-03-30|11:57 AM|        79|       58|       7|2987.22|
|2015-03-30|12:46 AM|        51|       50|       6|7501.89|
|2015-03-30|11:39 AM|        86|       24|       5| 8370.2|
|2015-03-30|10:35 AM|        63|       19|       5|1023.57|
|2015-03-30| 2:30 AM|        23|       77|       7|5892.41|
|2015-03-30| 7:41 PM|        49|       58|       4|9298.18|
|2015-03-30| 9:18 AM|        97|       86|       8|9462.89|
|2015-03-30|10:06 PM|        94|       26|       4|4199.15|
|2015-03-30|10:57 AM|        91|       18|       1|3795.73|
|2015-03-30| 7:43 AM|        20|       86|      10|1477.35|
|2015-03-30| 5:58 PM|        38|       39|       6| 1090.0|
|2015-03-30| 1:08 PM|        46|        6|      10|1014.78|
|2015-03-30|12:18 AM|        56|       48|       9|8346.42|
|2015-03-30| 1:18 AM|        11|       58|       4| 364.59|
|2015-03-30| 3:01 AM|        59|        9|       5|5984.68|
|2015-03-30|11:44 AM|         8|       35|       6| 1859.2|
|2015-03-30|12:05 PM|        23|        8|       3|1527.04|
|2015-03-30| 4:10 AM|        85|       93|       9|3314.71|
+----------+--------+----------+---------+--------+-------+
only showing top 20 rows

transDF: org.apache.spark.sql.DataFrame = [date: date, time: string ... 4 more fields]

transDF.printSchema

root
 |-- date: date (nullable = true)
 |-- time: string (nullable = true)
```

```
 |-- customerId: integer (nullable = true)
 |-- productId: integer (nullable = true)
 |-- quantity: integer (nullable = true)
 |-- price: double (nullable = true)
```

Create schema manually

```
import org.apache.spark.sql.types._

val schema1 = StructType(List(
        StructField("date", DateType, false),
        StructField("time", StringType, false),
        StructField("customerId", IntegerType, false),
        StructField("productId", IntegerType, false),
        StructField("quantity", IntegerType, false),
        StructField("price", FloatType, false)
    ))

val transDF = spark.read.format("csv")
                    .option("header", false)
                    .option("delimiter", "#")
                    .schema(schema1)
                    .load(z.get("transactionFile").toString)
transDF.show()
```

```
+----------+--------+----------+---------+--------+-------+
|      date|    time|customerId|productId|quantity|  price|
+----------+--------+----------+---------+--------+-------+
|2015-03-30| 6:55 AM|        51|       68|       1|9506.21|
|2015-03-30| 7:39 PM|        99|       86|       5|4107.59|
|2015-03-30|11:57 AM|        79|       58|       7|2987.22|
|2015-03-30|12:46 AM|        51|       50|       6|7501.89|
|2015-03-30|11:39 AM|        86|       24|       5| 8370.2|
|2015-03-30|10:35 AM|        63|       19|       5|1023.57|
|2015-03-30| 2:30 AM|        23|       77|       7|5892.41|
|2015-03-30| 7:41 PM|        49|       58|       4|9298.18|
|2015-03-30| 9:18 AM|        97|       86|       8|9462.89|
|2015-03-30|10:06 PM|        94|       26|       4|4199.15|
|2015-03-30|10:57 AM|        91|       18|       1|3795.73|
|2015-03-30| 7:43 AM|        20|       86|      10|1477.35|
|2015-03-30| 5:58 PM|        38|       39|       6| 1090.0|
|2015-03-30| 1:08 PM|        46|        6|      10|1014.78|
|2015-03-30|12:18 AM|        56|       48|       9|8346.42|
|2015-03-30| 1:18 AM|        11|       58|       4| 364.59|
|2015-03-30| 3:01 AM|        59|        9|       5|5984.68|
|2015-03-30|11:44 AM|         8|       35|       6| 1859.2|
|2015-03-30|12:05 PM|        23|        8|       3|1527.04|
|2015-03-30| 4:10 AM|        85|       93|       9|3314.71|
+----------+--------+----------+---------+--------+-------+
only showing top 20 rows

import org.apache.spark.sql.types._
schema1: org.apache.spark.sql.types.StructType = StructType(StructField(date,DateType,false), StructField(time,StringType,false), StructField(customerId,IntegerType,false), StructField(productId,IntegerT
transDF: org.apache.spark.sql.DataFrame = [date: date, time: string ... 4 more fields]
```

```
transDF.printSchema
```

```
root
 |-- date: date (nullable = true)
 |-- time: string (nullable = true)
 |-- customerId: integer (nullable = true)
 |-- productId: integer (nullable = true)
 |-- quantity: integer (nullable = true)
 |-- price: float (nullable = true)
```

Another way to specify schema

```
val schema2 = new StructType()
        .add("date", DateType, false)
        .add("time", StringType, false)
        .add("customerId", IntegerType, false)
        .add("productId", IntegerType, false)
        .add("quantity", IntegerType, false)
        .add("price", FloatType, false)

val transDF = spark.read.format("csv")
                    .option("header", false)
                    .option("delimiter", "#")
                    .schema(schema2)
                    .load(z.get("transactionFile").toString)
transDF.show()
```

```
+----------+--------+----------+---------+--------+-------+
|      date|    time|customerId|productId|quantity|  price|
+----------+--------+----------+---------+--------+-------+
|2015-03-30| 6:55 AM|        51|       68|       1|9506.21|
|2015-03-30| 7:39 PM|        99|       86|       5|4107.59|
|2015-03-30|11:57 AM|        79|       58|       7|2987.22|
|2015-03-30|12:46 AM|        51|       50|       6|7501.89|
|2015-03-30|11:39 AM|        86|       24|       5| 8370.2|
|2015-03-30|10:35 AM|        63|       19|       5|1023.57|
|2015-03-30| 2:30 AM|        23|       77|       7|5892.41|
|2015-03-30| 7:41 PM|        49|       58|       4|9298.18|
|2015-03-30| 9:18 AM|        97|       86|       8|9462.89|
|2015-03-30|10:06 PM|        94|       26|       4|4199.15|
|2015-03-30|10:57 AM|        91|       18|       1|3795.73|
|2015-03-30| 7:43 AM|        20|       86|      10|1477.35|
|2015-03-30| 5:58 PM|        38|       39|       6| 1090.0|
|2015-03-30| 1:08 PM|        46|        6|      10|1014.78|
|2015-03-30|12:18 AM|        56|       48|       9|8346.42|
|2015-03-30| 1:18 AM|        11|       58|       4| 364.59|
|2015-03-30| 3:01 AM|        59|        9|       5|5984.68|
|2015-03-30|11:44 AM|         8|       35|       6| 1859.2|
|2015-03-30|12:05 PM|        23|        8|       3|1527.04|
|2015-03-30| 4:10 AM|        85|       93|       9|3314.71|
+----------+--------+----------+---------+--------+-------+
only showing top 20 rows

schema2: org.apache.spark.sql.types.StructType = StructType(StructField(date,DateType,false), StructField(time,StringType,false), StructField(customerId,IntegerType,false), StructField(productId,IntegerT
transDF: org.apache.spark.sql.DataFrame = [date: date, time: string ... 4 more fields]
```

Another way to specify schema

```
val schema3 = StructType(StructField("date", DateType, false)
                    :: StructField("time", StringType, false)
                    :: StructField("customerId", IntegerType, false)
                    :: StructField("productId", IntegerType, false)
                    :: StructField("quantity", IntegerType, false)
                    :: StructField("price", FloatType, false)
                    :: Nil
                )

val transDF = spark.read.format("csv")
                    .option("header", false)
                    .option("delimiter", "#")
                    .schema(schema3)
                    .load(z.get("transactionFile").toString)
transDF.show()
```

```
+----------+--------+----------+---------+--------+-------+
|      date|    time|customerId|productId|quantity|  price|
+----------+--------+----------+---------+--------+-------+
|2015-03-30| 6:55 AM|        51|       68|       1|9506.21|
|2015-03-30| 7:39 PM|        99|       86|       5|4107.59|
```

```
|2015-03-30|11:57 AM|        79|       58|       7|2987.22|
|2015-03-30|12:46 AM|        51|       50|       6|7501.89|
|2015-03-30|11:39 AM|        86|       24|       5| 8370.2|
|2015-03-30|10:35 AM|        63|       19|       5|1023.57|
|2015-03-30| 2:30 AM|        23|       77|       7|5892.41|
|2015-03-30| 7:41 PM|        49|       58|       4|9298.18|
|2015-03-30| 9:18 AM|        97|       86|       8|9462.89|
|2015-03-30|10:06 PM|        94|       26|       4|4199.15|
|2015-03-30|10:57 AM|        91|       18|       1|3795.73|
|2015-03-30| 7:43 AM|        20|       86|      10|1477.35|
|2015-03-30| 5:58 PM|        38|       39|       6| 1090.0|
|2015-03-30| 1:08 PM|        46|        6|      10|1014.78|
|2015-03-30|12:18 AM|        56|       48|       9|8346.42|
|2015-03-30| 1:18 AM|        11|       58|       4| 364.59|
|2015-03-30| 3:01 AM|        59|        9|       5|5984.68|
|2015-03-30|11:44 AM|         8|       35|       6| 1859.2|
|2015-03-30|12:05 PM|        23|        8|       3|1527.04|
|2015-03-30| 4:10 AM|        85|       93|       9|3314.71|
+----------+--------+----------+---------+--------+-------+
only showing top 20 rows

schema3: org.apache.spark.sql.types.StructType = StructType(StructField(date,DateType,false), StructField(time,StringType,false), StructField(customerId,IntegerType,false), StructField(productId,IntegerT
transDF: org.apache.spark.sql.DataFrame = [date: date, time: string ... 4 more fields]
```