Three ways to generate RDD:

1. Using parallelize collection

```
val PCRDD =
spark.sparkContext.parallelize(Array("Monday","Tuesday","Thursday","Friday","Saturday","Sunday"),
2)

PCRDD.collect.foreach(println)
```

2. From external storage like hdfs,hive etc.

```
val Sparkfile = spark.read.textFile("/user/hadoop/wordcount/input/file1.txt")

Sparkfile.collect().foreach(println)
```

3. From existing RDDs

```
val words = spark.sparkContext.parallelize(Seq("Spark","is","very","powerful"))

val wordpair = words.map(w => (w.charAt(0),w))

wordpair.collect().foreach(println)
(S,Spark)
(i,is)
(v,very)
(p,powerful)
```

Examples:

matches.csv file containes IPL cricket data and is stored on hdfs at /user/hadoop path.

```
val ckfile = sc.textFile("/user/hadoop/matches.csv")

//Loads data in ckfile RDD

ckfile.collect.foreach(println)

//prints line by line

ckfile.first()

// prints schema

val states =ckfile.map(_.split(",")(2))
// finds all cities where match was conducted
```

```
states.collect.foreach(println)

// prints all cities where match was conducted

val scount = states.map(scount => (scount,1))

val statecount = scount.reduceByKey((x,y) => x+y).map(tup => (tup._2,tup._1))sortByKey(false)
//match count in all cities, sorted by count

statecount.take(10).foreach(println)

val filRDD = ckfile.flatMap(line => line.split("Hyderabad"))
//matches conducted in cities excluding Hyderabad

filRDD.collect.foreach(println)

val fil = ckfile.filter(line => line.contains("2017"))
//matches conducted in 2017

fil.collect.foreach(println)

val fil2 = ckfile.filter(line => line.contains("2016"))
//matches conducted in 2016

val uninRDD = fil.union(fil2)
//union of two RDDs

uninRDD.collect.foreach(println)

val ManOfTheMatch = ckfile.map(_.split(",")(13))

val MOTMcount = ManOfTheMatch.map(WINcount => (WINcount,1))

val ManOTH = MOTMcount.reduceByKey((x,y) => x+y).map(tup => (tup._2,tup._1))
```