Steps to load data from different sources

1. **From HDFS:**

   ```
   val textFile=sc.textFile("hdfs://localhost:9000/data/inputfile.txt")
   ```

2. **Create as a Sample data: (Parallelized collection)**

   ```
   val rdd1 = sc.parallelize(Seq((1,"jan",2016),(3,"nov",2014, (16,"feb",2014)))
   val data = sc.parallelize(Array(('k',5),('s',3),('s',4),('p',7),('p',5),('t',8),('k',6)),3)
   val words = Array("one","two","two","four","five","six","six","eight","nine","ten")
   ```

3. **Extract data from different database:**

   You need to save the driver file in the location where the application runs

   Syntax:

   ```
   val dataframe = sqlContext.read.format("jdbc").option("url", "url to accces your database
   ").option("driver", "Name_of_the_driver").option("dbtable", "table_name").option("user",
   "username").option("password", "password").load()
   ```

   Example:

   ```
   val dataframe = sqlContext.read.format("jdbc").option("url",
   "jdbc:mysql//localhost/sparksql").option("driver",
   "com.mysql.jdbc.Driver").option("dbtable", “Employee”).option("user",
   "root").option("password", "root").load()

   //Create a temporary table from the dataframe.
   dataframe.registerTempTable("Employee")

   //Process the sql queries
   dataframe.sqlContext.sql("select * from Employee where dummy_flag = 1 " )
   .collect.foreach(println)
   ```

4. **Steps to extract from a csv file**

   ```
   val csvFile=sc.textFile(“/usr/local/spark-application/College.csv”)
   ```

5. **Steps to extract from a text file**

   ```
   val txtFile=sc.read.csv(“/usr/local/spark-application/sparksample.txt”)
   ```

6. **Steps to extract from JSON file**

   ```
   val sqlcontext = new org.apache.spark.sql.SQLContext(sc)
   val dfs=sqlcontext.read.json("hdfs://localhost:9000/data/stocks.json")
   ```

Fields used from stocks.json

Company name	Describes the company's name
Country	Describes the country where company is located
Sector	Sample data includes Finance, Insurance, Banking, Healthcare
Profit Margin	Profit acquired by the company
Debt	Current liability of the company
Earnings Date	The date on which company presents its financial data
Price	Price of one share
Shares outstanding	Number of shares available for purchase
Gap	Difference between current share value and previous share value, more the gap you find risk to invest
20 Day simple moving average	Average value of recent 20 days share price

<u>Simple Transformations & Actions</u>

Example 1:

```
val fltr=textFile.filter(_.length>0) // val fltr = file.filter( x => x.length >  0
fltr.collect().foreach(println)
```

Example 2:

Possibility 1:

```
val sqlcontext = new org.apache.spark.sql.SQLContext(sc)

val dfs=sqlcontext.read.json("hdfs://localhost:9000/data/stocks.json")

dfs.show()

dfs.filter(dfs(“Price”)>500).collect().show()

dfs.printSchema() //printing the structure
```

Possibility 2:

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

val sqlcontext=new SQLContext(sc)
val dfs=sqlcontext.read.json("hdfs://localhost:9000/data/stocks.json")
dfs.registerTempTable("stocks")
```

//To rename the source fields

```
val stocksData=sqlcontext.sql("select '_id' as ComapnyId,'Profit Margin' as ProfitMargin,'Total
Debt/Equity' as TotalDebt, Sector,'20-Day Simple Moving Average' as
SimpleMovingAverage,'Shares Outstanding' as SharesOutstanding, 'Earnings Date' as
EarningDate,Price,Country,Company,Gap from stocks")
```

```
val jsonRead=sqlcontext.sql("select * from stocksData where Price > 500 ")
```

Other Possibilities for data querying :
Top 5 - Companies where you've less risks to invest

```
sql("Select ComapnyId,TotalDebt from stocksData order by TotalDebt asc").take(5).foreach(println)
```

Sector wise maximum price

```
val jsonRead=sqlcontext.sql("Select Sector,Max(Price) from stocksData group by Sector
").collect().foreach(println)
```

Next Possible Price(Prediction)

```
val jsonRead=sqlcontext.sql("Select Price,Gap,Price+Gap from stocksData where Price is not null
and gap is not null order by Gap ").take(5).foreach(println)
```

Companies where you've maximum shares to buy

```
val jsonRead=sqlcontext.sql("Select Comapny,Country from stocksData where country='USA'
").foreach(println)
```

Example 3:

```
val counts = sql("""
                    SELECT name, count FROM (
                     SELECT book, COUNT(*) as count FROM kjv_books GROUP BY book) bc
                              JOIN abbrevs_to_names an ON bc.book = an.abbrev
                              """).coalesce(1)
counts.registerTempTable("counts")
counts.printSchema
counts.queryExecution
counts.show(100)  // print all the lines; there are 66 books in the KJV.
```

//Load the results to hdfs
To write in json format:

```
stocksData.write.json("hdfs://localhost:9000/data/ouputjson")
```

To save in SparkSql :

```
stocksData.write.saveAsTable("stocksData")
```

To save in csv format:

```
stocksData.write.csv("/usr/local/spark-application/output/ouputcsv")
```

Note:
We cannot load complex data structures that has more than 1 colulmns to a text file, instead we can store it in parquet file format.Parquet is the default file format for writing data frames.

Other Transformations:

Group by

Groups by the first character of the name field

```
val x = sc.textFile("hdfs://localhost:9000/data/inputfile.txt")
val y = x.groupBy(w => w.charAt(0))
println(y.collect().mkString(", "))
```

Map

```
val data = sc.textFile("hdfs://localhost:9000/data/inputfile.txt")
val mapFile = data.map(line => (line,line.length))
mapFile.foreach(println)

def main(args: Array[String]) = {
val spark = SparkSession.builder.appName("mapExample").master("local").getOrCreate()
val data = spark.read.textFile("spark_test.txt").rdd
val mapFile = data.map(line => (line,line.length))
mapFile.foreach(println)
}
```

flatMap

```
val data = sc.textFile("hdfs://localhost:9000/data/inputfile.txt")
val flatmapFile = data.flatMap(lines => lines.split(" "))
flatmapFile.foreach(println)
```

filter

```
val data = sc.textFile("hdfs://localhost:9000/data/inputfile.txt")
val mapFile = data.flatMap(lines => lines.split(" ")).filter(value => value=="spark")
println(mapFile.count())
```

union

```
val rdd1 = sc.parallelize(Seq((1,"jan",2016),(3,"nov",2014),(16,"feb",2014)))
val rdd2 = spark.sparkContext.parallelize(Seq((5,"dec",2014),(17,"sep",2015)))
val rdd3 = spark.sparkContext.parallelize(Seq((6,"dec",2011),(16,"may",2015)))
val rddUnion = rdd1.union(rdd2).union(rdd3)
rddUnion.foreach(Println)
```

intersection

```
val rdd1 = sc.parallelize(Seq((1,"jan",2016),(3,"nov",2014, (16,"feb",2014)))
val rdd2 = spark.sparkContext.parallelize(Seq((5,"dec",2014),(1,"jan",2016)))
val comman = rdd1.intersection(rdd2)
comman.foreach(Println)
```

distinct

Example:
```
val rdd1 = spark.read.text("Sample.txt").as[String]
val result = rdd1.distinct()
println(result.collect().mkString(", "))
```

groupByKey

Example:
```
val data = sc.parallelize(Array(('k',5),('s',3),('s',4),('p',7),('p',5),('t',8),('k',6)),3)
val group = data.groupByKey().collect()
group.foreach(println)


scala> val data = spark.read.text("Sample.txt").as[String]
scala> data.flatMap(_.split(" ")).groupByKey(l=>l).count.show
```

reduceByKey

Example:
```
val words = Array("one","two","two","four","five","six","six","eight","nine","ten")
val data = sc.parallelize(words).map(w => (w,1)).reduceByKey(_+_)
data.foreach(println)
```

sortByKey

Example:
```
val data = sc.parallelize(Seq(("maths",52), ("english",75), ("science",82), ("computer",65),
("maths",85)))
val sorted = data.sortByKey() //key is the subject name as it takes in key-value pair format
sorted.foreach(println)
```

join

Example:
```
val data = sc.parallelize(Array(('A',1),('b',2),('c',3)))
val data2 =sc.parallelize(Array(('A',4),('A',6),('b',7),('c',3),('c',8)))
val result = data.join(data2)
println(result.collect().mkString(","))
```

//Reading through a text file
Code to find the number of rows with null values

```
object SparkWordCount {
  def main(args: Array[String]) {
    val sc = new SparkContext(new SparkConf().setAppName("Spark Count"))

    val files = sc.textFile(args(0)).map(_.split(","))

    def f(x:Array[String]) = {
      if (x.length > 0)
```

```
        x(3)
      else
        "NO NAME"
    }

    val names = files.map(f)

    val wordCounts = names.map((_, 1)).reduceByKey(_ + _).sortByKey()

    System.out.println(wordCounts.collect().mkString("\n"))
  }
}
```

<u>SparkSQl with Hive</u>

1. Create Hive context reference

```
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
```

2. Create a table using Hive QL

```
sqlContext.sql("CREATE TABLE IF NOT EXISTS employee(id INT, name STRING,
age INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES
TERMINATED BY '\n'")
```

3. Load the file, employee.txt from local disk to Hive table

```
sqlContext.sql("LOAD DATA LOCAL INPATH 'employee.txt' INTO TABLE
employee")
```

4. Querying the hive table

```
val result = sqlContext.sql("FROM employe SELECT id, name, age")
```

5. Viewing the hive table

```
result.show()
```