# Phase 2 : Scala and Spark

Sample code

```
package pack
object obj {
  def main(args:Array[String]):Unit={
    println("===spark journey started==")
  }
}
```

================================

1) package pack

```
object obj {

          def main(args:Array[String]):Unit={

                              // variables
                              println("Started")
                              val a = 2
                              println("====raw data===")
                              println(a)
                              val b = a + 1
                              println("====proc data====")
                              println(b)

                               // strings
                               val c = "zeyobron"
                               println("===raw data====")
                               println(c)
                               val d = c + " ANALYTICS"
                               println("==proc data==")
                                println(d)
```

```
    }

        // arithmetic operations
        val a = 2
        println("====raw data===")
        println(a)
        val b = a + 1
        println("====proc data====")
        println(b)

        //list
        val lis = List( 1 , 2 , 3 , 4 )
        println(lis)
        lis.foreach(println)

        // list - map
        val lis = List( 1 , 2 , 3 , 4 )
        println(lis)
        val result = lis.map( x => x + 1 )
        println(result)

        // list filter
         val lisin = List("zeyobron","analytics","zeyo")
        lisin.foreach(println)
        println("===proc list=====")
        val fillis = lisin.filter( x => x.contains("zeyo") )
        fillis.foreach(println)

        //replace
        val lisin = List("zeyobron","analytics","zeyo")
        lisin.foreach(println)
        val maplis = lisin.map( x => x.replace("zeyo", "tera"))
        maplis.foreach(println)
    }
}
```

2)

```
package pack
object obj {
        def main(args:Array[String]):Unit={

                        println("===started Zeyo====")
                        val lisstr = List(
                                        "State->Telangana" ,
                                        "State->Gujarat" ,
                                        "State->Karnataka"
                                        )

                        lisstr.foreach(println)
                        val mapstr =  lisstr.map( x => x.replace("State->", ""))
                        println
                        println("=====replace list====")
                        println
                        mapstr.foreach(println)

                        val lisstr1  = List( "State~City" )
                        println
                        println("=====before flatten===")
                        lisstr1.foreach(println)

                        val flatdata = lisstr1.flatMap( x => x.split("~"))
                        println
                        println("=====after flatten===")
                        println
                        flatdata.foreach(println)
```

```
                    val lisstr2 = List(
                                    "State->Telangana" ,
                                    "City->Hyderabad" ,
                                    "State->Karnataka"
                                    )
                    println
                    println("=====before filter===")
                    println

                    lisstr2.foreach(println)
                    println
                    println("=====after filter===")
                    println
                    val filterdata = lisstr2.filter( x => x.contains("State"))
                    filterdata.foreach(println)

        }

}
```

---

3)

```
package pack

object obj {

        def main(args:Array[String]):Unit={
                    println("===started Zeyo====")
                    println
                    val lisstr = List(
                                  "state->Telangana~city->Hyderabad",
                                  "state->Karnataka~city->bangalore"
                                        )
                println
```

```
		println("==raw List===")
		println
		lisstr.foreach(println)

		println
		println("==flat List===")
		println
		val flatdata = lisstr.flatMap( x => x.split("~"))
		flatdata.foreach(println)
		println
		println("=====state filter List======")
		println
		val stlist = flatdata.filter( x => x.contains("state"))
		stlist.foreach(println)
		println
		println("=====city filter List======")
		println

		val clist = flatdata.filter( x => x.contains("city"))
		clist.foreach(println)
		println
		println("=====state replace  List======")
		println
		val statelist = stlist.map( x => x.replace("state->", ""))
		statelist.foreach(println)
		println
		println("=====city replace  List======")
		println
		val citylist = clist.map( x => x.replace("city->",""))
		citylist.foreach(println)
	}

}
```

4)

```
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf

object obj {
  def main(args:Array[String]):Unit={
    println("===started Zeyo====")

    val conf = new SparkConf().setAppName("first").setMaster("local[*]")
    val sc = new SparkContext(conf)
    sc.setLogLevel("ERROR")
    val data = sc.textFile("file:///D:/data/datatxns.txt")
          // ==> Change path according to your local file
    data.foreach(println)
  }

}
```

---

5)

```
package pack
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf

object obj {
          def main(args:Array[String]):Unit={

              println("===started Zeyo====")
              val conf = new SparkConf().setAppName("first").setMaster("local[*]")
              val sc = new SparkContext(conf)
              sc.setLogLevel("ERROR")
```

```
            val data = sc.textFile("file:///D:/data/usdata.csv")
            data.foreach(println)
            val lendata = data.filter( x => x.length>200)
            println
            println("======len data====")
            lendata.foreach(println)
            println
            println("======flatten data====")
            val flatten= lendata.flatMap( x => x.split(","))
            flatten.foreach(println)

        }

}
```

Phase 2

```
===============
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
object obj {
case class schema(id:String,category:String,product:String,mode:String)
def main(args:Array[String]):Unit={
        println("====started==")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
        val sc = new SparkContext(conf)
        sc.setLogLevel("ERROR")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._
        val data = sc.textFile("file:///D:/data/datatxns.txt")
        data.foreach(println)
        println
        println
        val mapsplit = data.map( x => x.split(","))
        val schemardd = mapsplit.map( x => schema(x(0),x(1),x(2),x(3)))
            val prodfilter = schemardd.filter( x =>
    x.product.contains("Gymnastics"))
        prodfilter.foreach(println)
        println
        println
        val dataframe = prodfilter.toDF()
        dataframe.show()
}
}
```

Lab Code Task -- Change your LABUSER

```
============================================
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.Row
import org.apache.spark.sql.types._
        println("====started==")
        case class schema(id:String,category:String,product:String,mode:String)
        val conf = new
        SparkConf().setAppName("first").setMaster("local[*]").set("spark.driver.allo
        wMultipleContexts","true")
        val sc = new SparkContext(conf)
        sc.setLogLevel("ERROR")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._
        val data = sc.textFile("/user/<LABUSER>/datatxns.txt")
        data.foreach(println)
        println
        val mapsplit = data.map( x => x.split(","))
        val rowrdd = mapsplit.map( x => Row(x(0),x(1),x(2),x(3)))
        val prodfilter = rowrdd.filter( x => x(2).toString().contains("Gymnastics"))
        prodfilter.foreach(println)

        val simpleSchema = StructType(Array(
        StructField("id",StringType),
        StructField("category",StringType),
        StructField("product",StringType),
        StructField("mode", StringType)
))
```

```
        val dataframe1 = spark.createDataFrame(prodfilter, simpleSchema)
        dataframe1.show()
=============================================================
================
Code
================
package pack
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.Row
import org.apache.spark.sql.types._

object obj {
        def main(args:Array[String]):Unit={
        System.setProperty("hadoop.home.dir", "C:\\hadoop")
        println("====started==")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
        val sc = new SparkContext(conf)
        sc.setLogLevel("ERROR")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._

        val df = spark.read.format("csv")
                        .option("header","true")
                        .load("file:///D:/data/usdata.csv") //  YOUR PATH HERE
        df.show()
        df.createOrReplaceTempView("ustab")
        val finaldf = spark.sql("  select * from ustab where state='LA' ")
        finaldf.show()
        }
}
=============================================================
```

===================

Solution Code

===================

```
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.Row
import org.apache.spark.sql.types._
object obj {
        def main(args:Array[String]):Unit={
        System.setProperty("hadoop.home.dir", "C:\\hadoop")
        println("====started==")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
        val sc = new SparkContext(conf)
        sc.setLogLevel("ERROR")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._
        val csvdf = spark
                .read
                .format("csv")
                .option("header","true")
                .load("file:///C:/data/sedata/usdata.csv")
                csvdf.show()
        val parquetdf = spark
                            .read
                            .format("parquet")
                            .option("header","true")
                            .load("file:///C:/data/sedata/parquetdata.parquet")
                        parquetdf.show()
```

```
            val jsonsdf = spark
                    .read
                    .format("json")
                    .load("file:///C:/data/sedata/devices.json")
            jsonsdf.show()
            val orcdf = spark
                      .read
                      .format("orc")
                       .load("file:///C:/data/sedata/part.orc")
                  orcdf.show()
      }
}
```

==================================================================

====================

Lab Folks Dataset

====================

```
cd
wget https://36buck.s3.amazonaws.com/df.csv
wget https://36buck.s3.amazonaws.com/df1.csv
wget https://36buck.s3.amazonaws.com/cust.csv
wget https://36buck.s3.amazonaws.com/prod.csv
wget https://36buck.s3.amazonaws.com/devices.json
wget https://36buck.s3.amazonaws.com/part.orc
wget https://36buck.s3.amazonaws.com/usdata.csv
wget https://36buck.s3.amazonaws.com/parquetdata.parquet
```

how to read the data sample

```
spark.read.format("json").load("file:///home/<LABUSER>/devices.json")
```

======================================================================

```
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
	def main(args:Array[String]):Unit={
		System.setProperty("hadoop.home.dir", "C:\\hadoop")
		println("====started==")
		val conf = new
SparkConf().setAppName("first").setMaster("local[*]")
		val sc = new SparkContext(conf)
		sc.setLogLevel("ERROR")
		val spark = SparkSession.builder().getOrCreate()
	import spark.implicits._
		val df = spark
			.read
			.format("csv")
			.option("header","true")
			.load("file:///D:/data/dt.txt")
			df.show()
val filterdf = df.filter(col("category")==="Exercise")
			filterdf.show()
	}
}
```
==========================================================

lab folks dataset

https://liyabuck.s3.amazonaws.com/dt.txt

lab folks dataset --- Terminal Command

```
wget https://liyabuck.s3.amazonaws.com/dt.txt
```

Lab Code

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

        println("====started==")
        val conf = new
        SparkConf().setAppName("first").setMaster("local[*]").set("spark.driver.allo
        wMultipleContexts", "true")
        val sc = new SparkContext(conf)
        sc.setLogLevel("ERROR")
        val spark = SparkSession.builder().getOrCreate()

        import spark.implicits._
        val df = spark
                .read
                .format("csv")
                .option("header","true")
                .load("file:///home/<LABUSER>/dt.txt")
                df.show()
val filterdf = df.filter(col("category")==="Exercise")
filterdf.show()
val df1 = df.select("tdate", "category")
df1.show()
val df2 = df.drop("tdate","category")
df2.show()

================================================================
                val df1 = df.filter(col("category")==="Exercise")
                df1.show()

                //  Multi Column filter  and
                     val df2 = df.filter( col("category")==="Exercise"
                          &&           //and operator
                          col("spendby") === "cash" )
                     df2.show()


                //  Multi Column filter  or
                     val df3 = df.filter( col("category")==="Exercise"
                          ||           //or operator
                          col("spendby") === "cash" )
                     df3.show()



                //  Multi value filter
                     val df4 = df.filter(col("category") isin ("Exercise","Team
                     Sports"))
                     df4.show()
```

================================================================================
Full Code
================================

```
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
```

```
object obj {
	def main(args:Array[String]):Unit={
		System.setProperty("hadoop.home.dir", "C:\\hadoop")
		println("====started==")
		val conf = new SparkConf().setAppName("first").setMaster("local[*]")
		val sc = new SparkContext(conf)
		sc.setLogLevel("ERROR")
		val spark = SparkSession.builder().getOrCreate()

		import spark.implicits._
		val df = spark
			.read
			.format("csv")
			.option("header","true")
			.load("file:///D:/data/dt.txt")
		df.show()

		//  One Column Filter  category = 'Exercise'
		val df1 = df.filter(col("category")==="Exercise")
		df1.show()

		//  Multi Column filter  and
		val df2 = df.filter( col("category")==="Exercise"
		                     &&              //and operator
		                     col("spendby") === "cash")
		df2.show()

		//  Multi Column filter  or
		val df3 = df.filter( col("category")==="Exercise"
		                     ||               //or operator
		                     col("spendby") === "cash")
		df3.show()
```

```
        // Multi value filter
        val df4 = df.filter(col("category") isin ("Exercise","Team Sports"))
        df4.show()
    }
}
```

==================================================================

Task 1 ----
test like operator

```
val df4 = df.filter(col("product") like ("%Gymnastics%"))
df4.show()
```

Task 2 -----

Read devices.json as json
val df = spark read devices.json
df1= df  select only device_id,device_name
df2= df drop temp
df3= df filter lat>40
df4= df filter long<40
df5= df filter lat>40 and temp<30
df6 = df filter long>40 or temp>=20
df7 = df filter device_id>20
df8 = df filter device_name contains %am%
==================================================================

```
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
	def main(args:Array[String]):Unit={
	System.setProperty("hadoop.home.dir", "C:\\hadoop")
	println("====started==")
	val conf = new SparkConf().setAppName("first").setMaster("local[*]")
	val sc = new SparkContext(conf)
	sc.setLogLevel("ERROR")
	val spark = SparkSession.builder().getOrCreate()
	import spark.implicits._
	val df = spark.read
			.format("csv")
			.option("header","true")
			.load("file:///D:/data/dt.txt")
			df.show()

	println
	println("======one column filter=====")
	println
	val df1 = df.filter(col("category")==="Exercise")
	df1.show()
	println
	println("======Multi column filter=====")
	println
	val df2 = df.filter(col("category")==="Exercise" && col("spendby")==="cash")
	df2.show()
```

```
        println
        println("======Multi or  filter=====")
        println

        val df3 = df.filter(col("category")==="Exercise" || col("spendby")==="cash")
        df3.show()
        println
        println("======Multi value=====")
        println

        val df4 = df.filter(col("category") isin ("Exercise","Team Sports"))
        df4.show()

        println
        println("======like filter=====")
        println

        val df5 = df.filter(col("product") like "%Gymnastics%")
        df5.show()
        println
        println("======Not filter=====")
        println

val df6 = df.filter(!(col("category")==="Exercise") && col("spendby")==="cash")
        df6.show()

        println
        println("======null filter=====")
        println
        val df7 = df.filter(col("product") isNull )
        df7.show()
        println
        println("======Not null filter=====")
        println
        val df8 = df.filter(col("product") isNotNull)
        df8.show()
        }
}

=============================
category = Exercise
category = Exercise && spendby=cash
category = Exercise or spendby=cash
category = Exercise, Team Sports (both)
product like %Gymnastics%
category != Exercise && spendby=cash
product is null
product is not null
============================================================================

val df2 = df.selectExpr("*", "case when spendby='cash' then 1 else 0 end as
status")
val df1 = df.selectExpr("id", "product","lower(category) as lower")
df1.show()

val df1 = df.selectExpr( "id",
                        "split(tdate,'-')[2] as year",
                        "amount",
                        "category",
                        "product",
                        "spendby" )
df1.show()
============================================================================
        val df1 = df.withColumn("tdate", expr("split(tdate,'-')[2]"))
                                .withColumnRenamed("tdate", "year")
        df1.show()

        Task 1 ----  withColumn
        Task 2 ------ Read datatxns.txt
```

Filter _c1 == "Gymnastics"
From the above output case when _c2 contains Gymnastics then 'yes' else 'No'

```
        Task 3 ----- Complete the Final SQL Document

        Task 1 ---
                val df = spark.read
                        .format("csv")
                        .load("file:///D:/data/datatxns.txt")
                df.show()

                val df1 = df.withColumn("tdate", expr("split(tdate,'-')[2]"))
                                .withColumnRenamed("tdate", "year")
                df1.show()
```

==================================================================

Task 1  & 2 Solution
=======================

```
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
```

```
object obj {
	def main(args:Array[String]):Unit={
		System.setProperty("hadoop.home.dir", "C:\\hadoop")
		println("====started==")

		val conf = new SparkConf().setAppName("first").setMaster("local[*]")
		val sc = new SparkContext(conf)
		sc.setLogLevel("ERROR")
		val spark = SparkSession.builder().getOrCreate()
		import spark.implicits._
		val df = spark.read
			.format("csv")
			.option("header","true")
			.load("file:///D:/data/dt.txt")  // your path
		df.show()

		//Task 1
		val df1 = df.withColumn("tdate1", expr("split(tdate,'-')[2]"))
				.withColumnRenamed("tdate", "year")
		df1.show()

		val df2=spark.read
			.format("csv")
			.load("file:///D:/data/datatxns.txt")  // your path
		df2.show()

		//Task 2
		val df3= df2.selectExpr("*",
	"case when _c2 like '%Gymnastics%' then 'yes' else 'no' end as status")
		df3.show()
	}
}
=================================================================
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
        def main(args:Array[String]):Unit={
                System.setProperty("hadoop.home.dir", "C:\\hadoop")
                println("====started==")

                val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                val sc = new SparkContext(conf)
                sc.setLogLevel("ERROR")
                val spark = SparkSession.builder().getOrCreate()
                import spark.implicits._
                val df = spark.read
                        .format("csv")
                        .option("header", "true")
                        .load("file:///C:/data/dt.txt")
                df.show()

                val df1 = df.withColumn("category",expr("upper(category)"))
                            .withColumn("status", expr("case when spendby='cash'
                            then 1 else 0 end"))
                df1.show()
        }
}

==============================================================
```

===============
Others
===============

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
```

Lab Folks Dataset
====================
Terminal Commands
====================

```
wget https://liyabuck.s3.amazonaws.com/prod.csv
wget https://liyabuck.s3.amazonaws.com/cust.csv
```

============================================================================
=====================
Whole Code
=====================

```
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
	def main(args:Array[String]):Unit={
		System.setProperty("hadoop.home.dir", "C:\\hadoop")
		println("====started==")
		val conf = new SparkConf().setAppName("first").setMaster("local[*]")
				.set("spark.driver.allowMultipleContexts","true")
```

```
        val sc = new SparkContext(conf)
                sc.setLogLevel("ERROR")
                val spark = SparkSession.builder().getOrCreate()
                import spark.implicits._

        val cust =
        spark.read.format("csv").option("header","true").load("file:///C:/data
        /cust.csv")
        cust.show()

        val prod =
spark.read.format("csv").option("header","true").load("file:///C:/data/prod.csv")
        prod.show()

        println
        println("""========inner join===========""")
        println
        val innerjoin = cust.join(prod,Seq("id"),"inner")
        innerjoin.show()

        println
        println("""========left join===========""")
        println
        val left = cust.join(prod,Seq("id"),"left")
        left.show()

        println
        println("""========right join===========""")
        println
        val right = cust.join(prod,Seq("id"),"right")
        right.show()
```

```
		println
		println("""========full join===========""")
		println

		val full = cust.join(prod,Seq("id"),"full").orderBy("id")
		full.show()
	}
}
================================================================
```

Scenario Code

```
=================================
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
	def main(args:Array[String]):Unit={
	System.setProperty("hadoop.home.dir", "C:\\hadoop")
	println("====started==")

	val conf = new SparkConf().setAppName("first").setMaster("local[*]")
	                         .set("spark.driver.allowMultipleContexts","true")
	val sc = new SparkContext(conf)
	sc.setLogLevel("ERROR")
	val spark = SparkSession.builder().getOrCreate()
	import spark.implicits._
```

```
        val source =
spark.read.format("csv").option("header","true").load("file:///C:/data/source.csv")
        source.show()

        val target =
        spark.read.format("csv").option("header","true").load("file:///C:/data/targe
        t.csv").withColumnRenamed("name","name1")
        target.show()

        val full = source.join( target ,Seq("id") , "full" ).orderBy("id")
        full.show()

        val match_mis = full.withColumn("comment", expr("case when name=name1
then 'Match' else 'Mismatch' end"))
        match_mis.show()

        val remvmatch = match_mis.filter(! (col("comment")==="Match"))
        remvmatch.show()

        val finaldfpre = remvmatch.withColumn("comment", expr("case when name1 is
null then 'New in Source' when name is null then 'New in Target' else comment
end"))
        finaldfpre.show()

        val finaldf = finaldfpre.drop("name","name1")
        finaldf.show()
        }
}

========================================================================
```

=================
Join Code
=================

```
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
	def main(args:Array[String]):Unit={
	System.setProperty("hadoop.home.dir", "C:\\hadoop")
	println("====started==")

	val conf = new SparkConf().setAppName("first").setMaster("local[*]")
			.set("spark.driver.allowMultipleContexts","true")

	val sc = new SparkContext(conf)
			sc.setLogLevel("ERROR")

	val spark = SparkSession.builder().getOrCreate()
	import spark.implicits._

	val cust =
spark.read.format("csv").option("header","true").load("file:///C:/data/cust.csv")

	cust.show()
	val prod = spark.read.format("csv")
			.option("header","true")
			.load("file:///C:/data/prod.csv")
	prod.show()
```

```
println
println("======inner join=======")
println
val inner  = cust.join(prod,Seq("id"),"inner")
inner.show()

println
println("======left join=======")
println
val left  = cust.join(prod,Seq("id"),"left")
left.show()

println
println("======right join=======")
println
val right  = cust.join(prod,Seq("id"),"right")
right.show()

println
println("======full join=======")
println
val full  = cust.join(prod,Seq("id"),"full")
full.show()

println
println("======left anti join=======")
println
val left_anti  = cust.join(prod,Seq("id"),"left_anti")
left_anti.show()

println
println("======cross join=======")
println
```

```
		val cross = cust.crossJoin(prod)
		cross.show()
	}
}
=========================================================================
```

Agg Code

```
======================
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
	def main(args:Array[String]):Unit={
		System.setProperty("hadoop.home.dir", "C:\\hadoop")
		println("====started==")
		val conf = new SparkConf().setAppName("first").setMaster("local[*]")
			.set("spark.driver.allowMultipleContexts","true")

		val sc = new SparkContext(conf)
		sc.setLogLevel("ERROR")
		val spark = SparkSession.builder().getOrCreate()
		import spark.implicits._
		val df = spark.read.format("csv")
				.option("header","true")
				.load("file:///C:/data/agg1.csv")
		df.show()
		val aggdf = df.groupBy("name","product")
				.agg(sum("amt").cast(IntegerType).as("total"),
					count("amt").as("cnt"))
				 .orderBy(col("total") desc)
		aggdf.show()
		df.createOrReplaceTempView("df")

		val finald = spark.sql("select name,product,cast(sum(amt) as int) as
	total, count(amt) as cnt from df group by name,product order by total")
		finald.show()
	}
}
```

==================================================================

AWS s3 Integration

====================

```
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
	def main(args:Array[String]):Unit={
		System.setProperty("hadoop.home.dir", "C:\\hadoop")
		println("====started==")

		val conf = new SparkConf().setAppName("first").setMaster("local[*]")
			.set("spark.driver.allowMultipleContexts","true")
		val sc = new SparkContext(conf)
		sc.setLogLevel("ERROR")
```

```
                val spark = SparkSession.builder().getOrCreate()
                import spark.implicits._
                val df = spark
                          .read
                          .format("json")
                          .option("fs.s3a.access.key","AKIAS3H27Y6URIBF3P4T")
                        .option("fs.s3a.secret.key","OwT38krhkde2OZNBYcNryzt7B3+
                        dpDKyjI2Ud8Zl")
                        .load("s3a://liyabuck/devices.json")
                df.show()
        }
}
===============================================================================

Salary Code
==============================
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
        def main(args:Array[String]):Unit={
                System.setProperty("hadoop.home.dir", "C:\\hadoop")
                println("====started==")
                val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                        .set("spark.driver.allowMultipleContexts","true")

                val sc = new SparkContext(conf)
                sc.setLogLevel("ERROR")
                val spark = SparkSession.builder().getOrCreate()
                import spark.implicits._

                val df1 = spark.read.format("csv")
                        .option("header","true")
                        .load("file:///C:/data/d1.csv")
                df1.show()

                val df2 = spark.read.format("csv")
                        .option("header","true")
                        .load("file:///C:/data/d2.csv")
                df2.show()

                val df3 = spark.read.format("csv")
                        .option("header","true")
                        .load("file:///C:/data/d3.csv")
                df3.show()

                val joindf1 = df1.join(df2,Seq("id"),"left").join(df3,Seq("id"),"left")
                joindf1.show()

                val joinwith = joindf1
        .withColumn("salary",expr("case when salary is null then 0 else salary end"))
.withColumn("salary1",expr("case when salary1 is null then 0 else salary1 end"))
         .withColumn("ns", expr("salary+salary1"))
                joinwith.show()

                val finaldf = joinwith.drop("salary","salary1")
                                            .withColumnRenamed("ns", "salary")
                finaldf.show()
        }
}
==================================================================
```

Write Code

```
======================
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object obj {
        def main(args:Array[String]):Unit={
        System.setProperty("hadoop.home.dir", "D:\\hadoop")
        // change your path accordingly for winutils
        println("====started==")

        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                        .set("spark.driver.allowMultipleContexts","true")

        val sc = new SparkContext(conf)
        sc.setLogLevel("ERROR")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._
        val df = spark.read.format("csv")
                                .option("header","true")
                                .load("file:///C:/data/usdata.csv")
                        // change your path accordingly for source data
        df.show()

        val rowdf = df.withColumn("row", monotonically_increasing_id()+1)
        rowdf.show()
        val filterdata = rowdf.filter(col("state")==="LA")
        filterdata.show()
        filterdata.write.format("parquet").partitionBy("county").mode("overwrite").s
ave("file:///C:/data/usprocdir1")
        }
}
================================================================================
df.show()
df.printSchema()
================================================================================
```

Full Revision Code

```
==========================
package pack

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.Row
import scala.io.Source

object obj {
case class schema(
                txnno:String,
                txndate:String,
                custno:String,
                amount:String,
                category:String,
                product:String,
                city:String,
                state:String,
                spendby:String
                )
```

```
        def main(args:Array[String]):Unit={
        System.setProperty("hadoop.home.dir", "D:\\hadoop")  // change your path
accordingly for winutils
        println("====started==")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                .set("spark.driver.allowMultipleContexts","true")

        val sc = new SparkContext(conf)
        sc.setLogLevel("ERROR")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._
        val listcol=
        List("txnno","txndate","custno","amount","category","product","city","state",
        "spendby")

        val data = sc.textFile("file:///C:/data/revdata/file1.txt")
        data.take(5).foreach(println)
        println
        println("==============Gymnastics rows============")
        println
        val gymdata = data.filter( x => x.contains("Gymnastics"))
        gymdata.take(5).foreach(println)
        val mapsplit = gymdata.map( x => x.split(","))

        val schemardd = mapsplit.map( x =>
        schema(x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8)))

        val prodfilter = schemardd.filter( x => x.product.contains("Gymnastics"))
        println
        println("==============prod column filter============")
        println
        prodfilter.take(5).foreach(println)
        println
        println("==============schema rdd to dataframe============")
        println
        val schemadf = prodfilter.toDF().select(listcol.map(col): _*)
        schemadf.show(5)
        val file2 = sc.textFile("file:///C:/data/revdata/file2.txt")
        val mapsplit1 = file2.map( x => x.split(","))

        val rowrdd = mapsplit1.map( x =>
        Row(x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8)))

        println
        println("==============Row rdd============")
        println
        rowrdd.take(5).foreach(println)
        val rowschema = StructType(Array(
                              StructField("txnno",StringType,true),
                              StructField("txndate",StringType,true),
                              StructField("custno",StringType,true),
                              StructField("amount", StringType, true),
                              StructField("category", StringType, true),
                              StructField("product", StringType, true),
                              StructField("city", StringType, true),
                              StructField("state", StringType, true),
                              StructField("spendby", StringType, true)
                              ))

        val rowdf = spark.createDataFrame(rowrdd,
rowschema).select(listcol.map(col): _*)

        println
        println("==============Row df============")
        println
        rowdf.show(5)
        val csvdf = spark.read.format("csv").option("header","true")
                .load("file:///C:/data/revdata/file3.txt").select(listcol.map(col): _*)
        println
        println("==============csv df============")
        println
        csvdf.show(5)

        val jsondf = spark.read.format("json")
                .load("file:///C:/data/revdata/file4.json").select(listcol.map(col): _*)

        println
        println("==============jsondf============")
        println
        jsondf.show(5)

        println
        println("==============parquetdf============")
        println

        val parquetdf =
spark.read.load("file:///C:/data/revdata/file5.parquet").select(listcol.map(col): _*)
        parquetdf.show(5)

        val xmldf = spark.read.format("xml").option("rowtag","txndata")
                .load("file:///C:/data/revdata/file6").select(listcol.map(col): _*)
        println
        println("=============xmldf============")
        println
        xmldf.show(5)

        println
        println("=============uniondf============")
        println
        val uniondf =
schemadf.union(rowdf).union(csvdf).union(jsondf).union(parquetdf).union(xmldf)
        uniondf.show(5)

        println
        println("==============proc df============")
        println
        val procdf = uniondf.withColumn("txndate", expr("split(txndate,'-')[2]"))
                .withColumnRenamed("txndate","year")
        .withColumn("status",expr("case when spendby='cash' then 1 else 0 end"))
        .filter(col("txnno")>50000)

        procdf.show(5)
        println
        println("==============agg df============")
        println

        val aggdf =
procdf.groupBy("category").agg(sum("amount").cast(IntegerType).as("total"))
        aggdf.show(5)

/*      uniondf
                .write
                .format("avro")
                .mode("append")
                .partitionBy("category")
                .save("file:///C:/data/revavrodata")   */

        val cust = spark.read.format("csv")
                        .option("header","true")
                        .load("file:///C:/data/revdata/cust.csv")
        cust.show()
        val prod = spark.read.format("csv").option("header","true")
                .load("file:///C:/data/revdata/prod.csv")
        prod.show()

        println
        println("==============inner df============")
        println

        val inner = cust.join(prod,Seq("id"),"inner")
        inner.show()
        println
        println("==============left df============")
        println
        val left = cust.join(prod,Seq("id"),"left")
        left.show()
        println
        println("==============right df============")
        println
        val right = cust.join(prod,Seq("id"),"right")
        right.show()
        println
        println("==============full df============")
        println
        val full = cust.join(prod,Seq("id"),"full")
        full.show()
        println
        println("==============anti df============")
        println
        val anti = cust.join(prod,Seq("id"),"left_anti")
        anti.show()
        }
}
=====================================================================
```

Task1 -- Complete Revision Slides
Task 2 -- Pending sql and Scala tutorials
Task 3 (optional)-- Complete the relationship Code
Task 4 (optional) --- Give a try of scenario next
====================================================

Struct Code
====================================================
```
package pack
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.sql.functions._

object obj {
	def main(args: Array[String]): Unit = {
	val conf = new SparkConf().setAppName("first").setMaster("local[*]")
			val sc = new SparkContext(conf)
			sc.setLogLevel("Error")
			val spark = SparkSession.builder().getOrCreate()
			import spark.implicits._
			val df = spark.read.format("json")
				.option("multiline","true")
				.load("file:///C:/data/jl.json")

			df.show()
			df.printSchema()
			val flattendf = df.select("id", "institute", "trainer",
"location.permanentLocation", "location.temporaryLocation")
			flattendf.show()
			flattendf.printSchema()
	}
}
```
============================================================================

Struct Generation Code

=============================

```
package pack
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.sql.functions._

object obj {
        def main(args: Array[String]): Unit = {
        System.setProperty("hadoop.home.dir","C:\\hadoop")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
        val sc = new SparkContext(conf)
        sc.setLogLevel("Error")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._

        val df= spark.read.format("json")
                .option("multiline","true")
                .load("file:///C:/data/jk.json")
        df.show
        df.printSchema
        val flattendf = df.select("id", "institute", "location.*",  "worklocation")
        flattendf.show()
        flattendf.printSchema()
        val complexdf = flattendf.select(col("id"), col("institute"),
        struct(
                col("permanentLocation"),
                col("temporaryLocation"),
                col("worklocation")
                ).as("allLocations")  )

                complexdf.show()
                complexdf.printSchema()
                val complexdf_withColumn = flattendf
                        .withColumn("allLocations",
                        expr("""struct(permanentLocation, temporaryLocation,
                                worklocation) """) )
                        .drop("permanentLocation","temporaryLocation","worklocation")

                complexdf_withColumn.show()
                complexdf_withColumn.printSchema()
        }
}
```
===========================================================================

jkn.json Code

----------------------------

```
package pack
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.sql.functions._

object obj {
        def main(args: Array[String]): Unit = {
        System.setProperty("hadoop.home.dir","C:\\hadoop")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
        val sc = new SparkContext(conf)
        sc.setLogLevel("Error")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._
        val df= spark.read.format("json")
                        .option("multiline","true").load("file:///C:/data/jkn.json")
        df.show(false)
        df.printSchema
        val arrayexplode = df.withColumn("Students",expr("explode(Students)"))
        arrayexplode.show()
        arrayexplode.printSchema()
        val finalflatten = arrayexplode.select(
                    "Students.user.*" ,"id" ,"institute")
        finalflatten.show()
        finalflatten.printSchema()
        }
}
================================================================
jln.json Code
======================

package pack
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.sql.functions._

object obj {
        def main(args: Array[String]): Unit = {
        System.setProperty("hadoop.home.dir","C:\\hadoop")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
        val sc = new SparkContext(conf)
        sc.setLogLevel("Error")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._
        val df= spark.read.format("json")
                        .option("multiline","true")
                        .load("file:///C:/data/jln.json")

        df.show(false)
        df.printSchema
        val flattendf = df.withColumn("Students",expr("explode(Students)"))
        flattendf.show(false)
        flattendf.printSchema
        }
}
========================================================================
```

Scenario Code

```
=============================
package pack
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.sql.functions._

object obj {
        def main(args: Array[String]): Unit = {
        System.setProperty("hadoop.home.dir","C:\\hadoop")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
        val sc = new SparkContext(conf)
        sc.setLogLevel("Error")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._
        val lis1 = List( 1 , 2 , 3 )
        println(lis1)
        val lis2 = List( "one"  , "two" , "three" )
        println(lis2)
        val lis1df = lis1.toDF("c1")
        lis1df.show()

        val lis2df  = lis2.toDF("c2")
        lis2df.show()
        val rolist1 = lis1df.withColumn("id", monotonically_increasing_id())
        rolist1.show()
        val rolist2 =lis2df.withColumn("id", monotonically_increasing_id())
        rolist2.show()
        val joindf = rolist1.join(rolist2,Seq("id"),"inner")
        joindf.show()
        val concat = joindf.withColumn("final", expr("concat(c1,' is ',c2)"))
        concat.show()
        val finaldf = concat.select("final")
        finaldf.show()
        }
}
==============================================================================
        val snowdf = spark
                        .read
                        .format("snowflake")
                .option("sfURL","https://vvapryv-sg46500.snowflakecomputing.com")
                        .option("sfAccount","vvapryv")
                        .option("sfUser","zeyobron")
                        .option("sfPassword","Zeyo@usa908")
                        .option("sfDatabase","zeyodb")
                        .option("sfSchema","zeyoschema")
                        .option("sfRole","ACCOUNTADMIN")
                        .option("sfWarehouse","COMPUTE_WH")
                .option("query","select a.,c.location from (select a.,b.prod from
zeyotab a join zeyoprod b on a.id=b.id) a  join zeyoloc c on a.id=c.id;")
                        .load()
                snowdf.show(false)
                val snowdf_delete = snowdf.withColumn("current_date",lit(today))
                        .withColumn("delete_ind",lit(0))

                snowdf_delete.show(false)
                snowdf_delete.printSchema()
        val schema = StructType(Array(
                StructField("name", StringType, nullable = false),
                StructField("Students", ArrayType(StringType), nullable = false) ) )

                val jsondf = snowdf_delete.withColumn("JDATA",
                        from_json(col("JDATA"),schema))
        jsondf.show(false)
        jsondf.printSchema()

        val exploded = jsondf.withColumn("name", expr("JDATA.name"))
        .withColumn("Students",expr("explode(JDATA.Students)")).drop("JDATA")

        exploded.show(false)
        exploded.printSchema()
        exploded.write.format("csv").partitionBy("current_date","delete_ind")
                            .mode("append").save("file:///C:/data/scnewdata")
```

==================================================================

Mistake Corrected Code

======================

Updated Code

======================

```
package pack
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.SparkContext
import java.security.cert.X509Certificate
import javax.net.ssl._
import org.apache.http.client.methods.HttpGet
import org.apache.http.impl.client.HttpClients
import org.apache.http.util.EntityUtils
import scala.io.Source
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.Row
import org.apache.spark.sql.types._
import scala.io._
import org.apache.spark.sql.functions._

object obj {
        def main(args: Array[String]): Unit = {
        System.setProperty("hadoop.home.dir","C:\\hadoop")
        val conf = new SparkConf().setAppName("first").setMaster("local[*]")
        val sc = new SparkContext(conf)
        sc.setLogLevel("Error")
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._
        val sslContext = SSLContext.getInstance("TLS")
                sslContext.init(null, Array(new X509TrustManager {
                override def getAcceptedIssuers: Array[X509Certificate] =
        Array.empty[X509Certificate]
        override def checkClientTrusted(x509Certificates: Array[X509Certificate],
s: String): Unit = {}
        override def checkServerTrusted(x509Certificates:
        Array[X509Certificate], s: String): Unit = {}
                                        }), new java.security.SecureRandom())
        val hostnameVerifier = new HostnameVerifier {override def verify(s: String,
sslSession: SSLSession): Boolean = true}

        val httpClient =
HttpClients.custom().setSSLContext(sslContext).setSSLHostnameVerifier(hostna
meVerifier).build()
        val content = EntityUtils.toString(httpClient.execute(new
HttpGet("https://randomuser.me/api/0.8/?results=10")).getEntity)
```

```
        val urlstring = content.mkString
        println(urlstring)
        val df = spark.read.json(sc.parallelize(List(urlstring)))
        df.show()
        df.printSchema()
        val flatdf =
df.withColumn("results",explode(col("results"))).select("nationality","seed",
        "version","results.user.username","results.user.cell","results.user.dob","resul
        ts.user.email","results.user.gender","results.user.location.city","results.user.l
        ocation.state","results.user.location.street","results.user.location.zip","result
        s.user.md5","results.user.name.first","results.user.name.last","results.user.na
        me.title","results.user.password","results.user.phone","results.user.picture.la
        rge","results.user.picture.medium","results.user.picture.thumbnail","results.u
        ser.registered","results.user.salt","results.user.sha1","results.user.sha256")
flatdf.show()
        }
}
```

==============================================================================

Project Whole Code

===========================

```
package pack

import java.security.cert.X509Certificate
import javax.net.ssl._
import org.apache.http.client.methods.HttpGet
import org.apache.http.impl.client.HttpClients
import org.apache.http.util.EntityUtils
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._
```

```
object obj {
        def main(args: Array[String]): Unit = {
                val sslContext = SSLContext.getInstance("TLS")
                sslContext.init(null, Array(new X509TrustManager {
                override def getAcceptedIssuers: Array[X509Certificate] =
Array.empty[X509Certificate]
                override def checkClientTrusted(x509Certificates:
Array[X509Certificate], s: String): Unit = {}
                override def checkServerTrusted(x509Certificates:
Array[X509Certificate], s: String): Unit = {}}), new java.security.SecureRandom())
                val hostnameVerifier = new HostnameVerifier {override def verify(s:
String, sslSession: SSLSession): Boolean = true}
                val httpClient =
        HttpClients.custom().setSSLContext(sslContext).setSSLHostnameVerifier(h
        ostnameVerifier).build()
                val content = EntityUtils.toString(httpClient.execute(new
HttpGet("https://randomuser.me/api/0.8/?results=500")).getEntity)
                val urlstring = content.mkString

                val conf = new SparkConf().setAppName("first").setMaster("local[*]")
                                        .set("spark.driver.allowMultipleContexts","true")
                val sc = new SparkContext(conf)
                sc.setLogLevel("ERROR")
                val spark = SparkSession.builder().getOrCreate()
                import spark.implicits._
                val rdd = sc.parallelize(List(urlstring))

                val df = spark.read.json(rdd)
                println
                println
                println("======raw json api data")
                println
                df.show()
```

```
val arrayflatten = df.withColumn("results",expr("explode(results)"))
val finalflatten = arrayflatten.select(
        "nationality",
                "results.user.cell",
                "results.user.username",
                "results.user.dob",
                "results.user.email",
                "results.user.gender",
                "results.user.location.city",
                "results.user.location.state",
                "results.user.location.street",
                "results.user.location.zip",
                "results.user.md5",
                "results.user.name.first",
                "results.user.name.last",
                "results.user.name.title",
                "results.user.password",
                "results.user.phone",
                "results.user.picture.large",
                "results.user.picture.medium",
                "results.user.picture.thumbnail",
                "results.user.registered",
                "results.user.salt",
                "results.user.sha1",
                "results.user.sha256",
                "seed",
                "version"
                                )
                println
                println("======flatten data")
                println
                println
                finalflatten.show()
                val avrodf = spark.read.format("avro")
                        .load("file:///C:/data/projectsample.avro")
                println
                println
                println("======avro data")
                println
                println
                avrodf.show()
                avrodf.printSchema()
                val numdf =
        finalflatten.withColumn("username",regexp_replace(col("userna
        me"),  "([0-9])", ""))
                        println
                        println
                        println("======numericals removed data")
                        println
                        println
                        numdf.show()
                        println
                        println
                        println("======joined data")
                        println

        val joindf = avrodf.join(numdf,Seq("username"),"left")
        joindf.show()
                        println
                        println
                        println("======available data")
                        println
                        println

val availablecustomerinapi = joindf.filter(col("nationality").isNotNull)
availablecustomerinapi.show()
```

```
                                        println
                                        println
                                        println("======Not available data")
                                        println
                                        println
                    val notavailablecustomerinapi = joindf.filter(col("nationality").isNull)
                    notavailablecustomerinapi.show()

          availablecustomerinapi.write.format("parquet").mode("append")
                    .save("file:///C:/data/projwrite/available")

          notavailablecustomerinapi.write.format("parquet").mode("append")
                    .save("file:///C:/data/projwrite/notavailable")
          }
}
```

=================================================================

Lab Project Parquet

------------------------------

https://liyabuck.s3.amazonaws.com/projectsample.parquet

=============================================

Phase 3 : Pyspark, Aws, Kafka, Nifi