Experiment No: Group B-4

Aim: Write a simple program in SCALA using Apache Spark

Theory: 1. What is Scala

Scala is an acronym for “Scalable Language”. It is a general-purpose programming language designed for the programmers who want to write programs in a concise, elegant, and type-safe way. Scala enables programmers to be more productive. Scala is developed as an object-oriented and functional programming language.

If you write a code in Scala, you will see that the style is similar to a scripting language. Even though Scala is a new language, it has gained enough users and has a wide community support. It is one of the most user-friendly languages.

2. About Scala

The design of Scala started in 2001 in the programming methods laboratory at EPFL (École Polytechnique Fédérale de Lausanne). Scala made its first public appearance in January 2004 on the JVM platform and a few months later in June 2004, it was released on the .(dot)NET platform. The .(dot)NET support of Scala was officially dropped in 2012. A few more characteristics of Scala are:

2.1 Scala is pure Object-Oriented programming language

Scala is an object-oriented programming language. Everything in Scala is an object and any operations you perform is a method call. Scala, allow you to add new operations to existing classes with the help of implicit classes.

One of the advantages of Scala is that it makes it very easy to interact with Java code. You can also write a Java code inside Scala class. The Scala supports advanced component architectures through classes and traits.

2.2 Scala is a functional language

Scala is a programming language that has implemented major functional programming concepts. In Functional programming, every computation is treated as a mathematical function which avoids states and mutable data. The functional programming exhibits following characteristics:

- Power and flexibility
- Simplicity
- Suitable for parallel processing

Scala is not a pure functional language. Haskell is an example of a pure functional language. If you want to read more about functional programming, please refer to this article.

2.3 Scala is a compiler based language (and not interpreted)

Scala is a compiler based language which makes Scala execution very fast if you compare it with Python (which is an interpreted language). The compiler in Scala works in similar fashion as Java compiler. It gets the source code and generates Java byte-code that can be executed independently on any standard JVM (Java Virtual Machine). If you want to know more about the difference between complied vs interpreted language please refer this article.

There are more important points about Scala which I have not covered. Some of them are:

- Scala has thread based executors
- Scala is statically typed language
- Scala can execute Java code
- You can do concurrent and Synchronized processing in Scala

- Scala is JVM based languages

2.4 Companies using Scala

Scala is now big name. It is used by many companies to develop the commercial software. These are the following notable big companies which are using Scala as a programming alternative.

- LinkedIn
- Twitter
- Foursquare
- Netflix
- Tumblr
- The Guardian
- Precog
- Sony
- AirBnB
- Klout
- Apple

If you want to read more about how and when these companies started using Scala please refer this blog.

3. Installing Scala

Scala can be installed in any Unix or windows based system. Below are the steps to install for Ubuntu (14.04) for scala version 2.11.7. I am showing the steps for installing Scala (2.11.7) with Java version 7. It is necessary to install Java before installing Scala. You can also install latest version of Scala(2.12.1) as well.

Step 0: Open the terminal

Step 1: Install Java

```
 $ sudo apt-add-repository ppa:webupd8team/java
 $ sudo apt-get update
$ sudo apt-get install oracle-java7-installer
```

If you are asked to accept Java license terms, click on "Yes" and proceed. Once finished, let us check whether Java has installed successfully or not. To check the Java version and installation, you can type:

```
$ java -version
```

Step 2: Once Java is installed, we need to install Scala

```
$ cd ~/Downloads
$ wget http://www.scala-lang.org/files/archive/scala-2.11.7.deb
$ sudo dpkg -i scala-2.11.7.deb
$ scala –version
```

This will show you the version of Scala installed

4. Prerequisites for Learning Scala

Scala being an easy to learn language has minimal prerequisites. If you are someone with basic knowledge of C/C++, then you will be easily able to get started with Scala. Since Scala is developed on top of Java. Basic programming function in Scala is similar to Java. So, if you have some basic knowledge of Java syntax and OOPs concept, it would be helpful for you to work in Scala.

5. Choosing a development environment

Once you have installed Scala, there are various options for choosing an environment. Here are the 3 most common options:

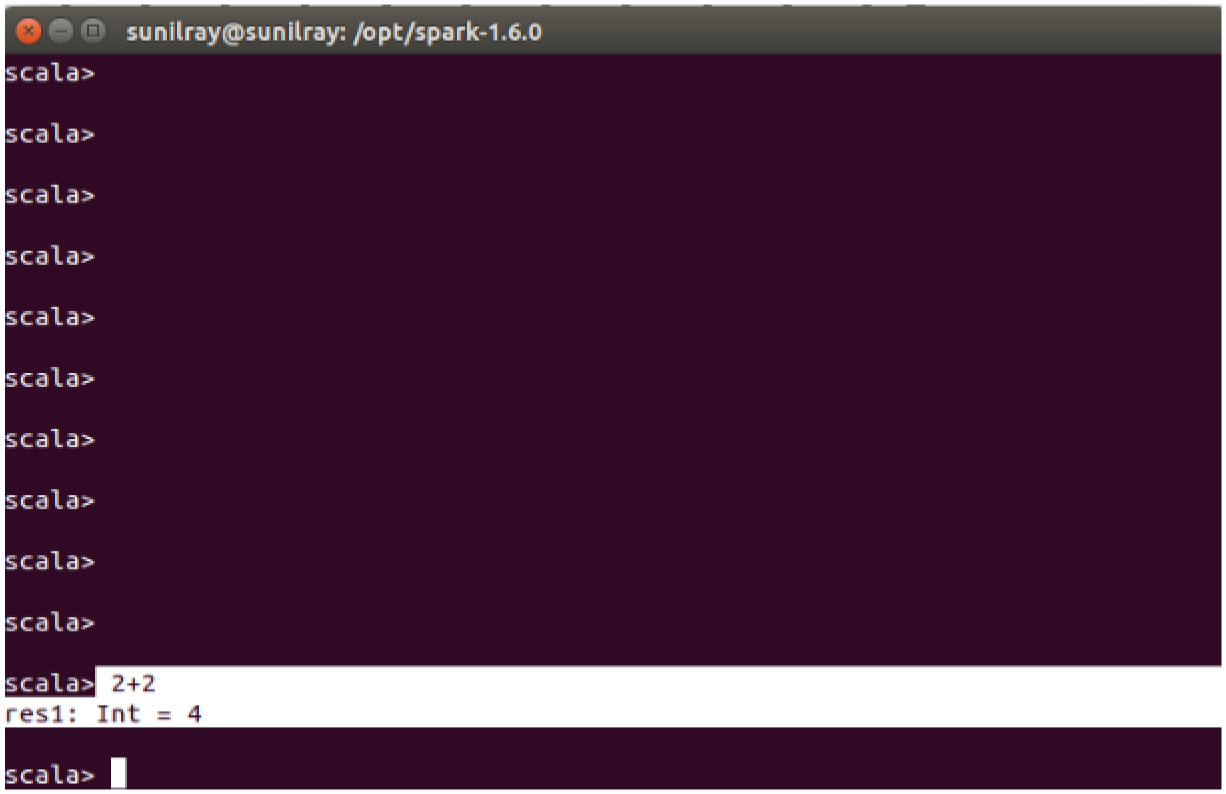
- Terminal / Shell based

- Notepad / Editor based
- IDE (Integrated development environment)

Choosing right environment depends on your preference and use case. I personally prefer writing a program on shell because it provides a lot of good features like suggestions for method call and you can also run your code while writing line by line.

Warming up: Running your first Scala program in Shell:

Let's write a first program which adds two numbers.



sunilray@sunilray: /opt/spark-1.6.0

```
scala>

scala>

scala>

scala>

scala>

scala>

scala>

scala>

scala>

scala>

scala> 2+2
res1: Int = 4

scala> 
```

6. Scala Basics Terms

Object: An entity that has state and behavior is known as an object. For example: table, person, car etc.

Class: A class can be defined as a blueprint or a template for creating different objects which defines its properties and behavior.

Method: It is a behavior of a class. A class can contain one or more than one method. For example: deposit can be considered a method of bank class.

Closure: Closure is any function that closes over the environment in which it's defined. A closure returns value depends on the value of one or more variables which is declared outside this closure.

Traits: Traits are used to define object types by specifying the signature of the supported methods. It is like interface in java.

7. Things to note about Scala

- It is case sensitive
- If you are writing a program in Scala, you should save this program using ".scala"
- Scala execution starts from main() methods
- Any identifier name cannot begin with numbers. For example, variable name "123salary" is invalid.
- You can not use Scala reserved keywords for variable declarations or constant or any identifiers.

8. Variable declaration in Scala

In Scala, you can declare a variable using 'var' or 'val' keyword. The decision is based on whether it is a constant or a variable. If you use 'var' keyword, you define a variable as mutable variable. On the other hand, if you use 'val', you define it as immutable. Let's first declare a variable using "var" and then using "val".

8.1 Declare using var

```
var Var1 : String = "Ankit"
```

In the above Scala statement, you declare a mutable variable called “Var1” which takes a string value. You can also write the above statement without specifying the type of variable. Scala will automatically identify it. For example:

```
var Var1 = "Gupta"
```

8.2 Declare using val

```
val Var2 : String = "Ankit"
```

In the above Scala statement, we have declared an immutable variable “Var2” which takes a string “Ankit”. Try it for without specifying the type of variable. If you want to read about mutable and immutable please refer this link.

9. Operations on variables

You can perform various operations on variables. There are various kinds of operators defined in Scala. For example: Arithmetic Operators, Relational Operators, Logical Operators, Bitwise Operators, Assignment Operators.

Lets see “+” , “==” operators on two variables ‘Var4’, “Var5”. But, before that, let us first assign values to “Var4” and “Var5”.

```
scala> var Var4 = 2
Output: Var4: Int = 2
scala> var Var5 = 3
Output: Var5: Int = 3
```

Now, let us apply some operations using operators in Scala.

Apply ‘+’ operator

```
Var4+Var5
```

```
Output:
res1: Int = 5
```

Apply "==" operator

```
Var4==Var5
Output:
res2: Boolean = false
```

If you want to know complete list of operators in Scala refer this link:

10. The if-else expression in Scala

In Scala, if-else expression is used for conditional statements. You can write one or more conditions inside "if". Let's declare a variable called "Var3" with a value 1 and then compare "Var3" using if-else expression.

```
var Var3 =1
if (Var3 ==1){
 println("True")}else{
 println("False")}
Output: True
```

In the above snippet, the condition evaluates to True and hence True will be printed in the output.

11. Iteration in Scala

Like most languages, Scala also has a FOR-loop which is the most widely used method for iteration. It has a simple syntax too.

```
for( a <- 1 to 10){
 println( "Value of a: " + a );
 }
```

Output:

```
Value of a: 1
Value of a: 2
Value of a: 3
Value of a: 4
Value of a: 5
Value of a: 6
Value of a: 7
Value of a: 8
Value of a: 9
Value of a: 10
```

Scala also supports “while” and “do while” loops. If you want to know how both work, please refer this link.

12. Declare a simple function in Scala and call it by passing value

You can define a function in Scala using “def” keyword. Let’s define a function called “mul2” which will take a number and multiply it by 10. You need to define the return type of function, if a function not returning any value you should use the “Unit” keyword.

In the below example, the function returns an integer value. Let’s define the function “mul2”:

```
def mul2(m: Int): Int = m * 10
```

Output: mul2: (m: Int)Int

Now let’s pass a value 2 into mul2

```
mul2(2)
```

Output:

```
res9: Int = 20
```

If you want to read more about the function, please refer this tutorial.

13. Few Data Structures in Scala

- Arrays
- Lists
- Sets
- Tuple
- Maps
- Option

13.1 Arrays in Scala

In Scala, an array is a collection of similar elements. It can contain duplicates. Arrays are also immutable in nature. Further, you can access elements of an array using an index:

Declaring Array in Scala

To declare any array in Scala, you can define it either using a new keyword or you can directly assign some values to an array.

Declare an array by assigning it some values

```
var name = Array("Faizan","Swati","Kavya", "Deepak", "Deepak")
```

Output:

```
name: Array[String] = Array(Faizan, Swati, Kavya, Deepak, Deepak)
```

In the above program, we have defined an array called name with 5 string values.

Declaring an array using “new” keywords

The following is the syntax for declaring an array variable using a new keyword.

```
var name:Array[String] = new Array[String](3)
```

or

```
var name = new Array[String](3)
```

Output:

```
name: Array[String] = Array(null, null, null)
```

Here you have declared an array of Strings called “name” that can hold up to three elements. You can also assign values to “name” by using an index.

```
scala> name(0) = "jal"
scala> name(1) = "Faizy"
scala> name(2) = "Expert in deep learning"
```

Let’s print contents of “name” array.

```
scala> name
res3: Array[String] = Array(jal, Faizy, Expert in deep learning)
```

Accessing an array

You can access the element of an array by index. Lets access the first element of array “name”. By giving index 0. Index in Scala starts from 0.

```
name(0)
Output:
res11: String = jal
```

13.2 List in Scala

Lists are one of the most versatile data structure in Scala. Lists contain items of different types in Python, but in Scala the items all have the same type. Scala lists are immutable.

Here is a quick example to define a list and then access it.

Declaring List in Scala

You can define list simply by comma separated values inside the “List” method.

```
scala> val numbers = List(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

```
numbers: List[Int] = List(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

You can also define multi dimensional list in Scala. Lets define a two dimensional list:

```
val number1 = List( List(1, 0, 0), List(0, 1, 0), List(0, 0, 1) )
number1: List[List[Int]] = List(List(1, 0, 0), List(0, 1, 0), List(0, 0, 1))
```

Accessing a list

Let's get the third element of the list "numbers" . The index should 2 because index in Scala start from 0.

```
scala> numbers(2)
res6: Int = 3
```

We have discussed two of the most used data Structures. You can learn more from this link.

14. Writing & Running a program in Scala using an editor

Let us start with a "Hello World!" program. It is a good simple way to understand how to write, compile and run codes in Scala. No prizes for telling the outcome of this code!

```
object HelloWorld {
 def main(args: Array[String]) {
 println("Hello, world!")
 }
 }
```

As mentioned before, if you are familiar with Java, it will be easier for you to understand Scala. If you know Java, you can easily see that the structure of above "HelloWorld" program is very similar to Java program.

This program contains a method “main” (not returning any value) which takes an argument – a string array through command line. Next, it calls a predefined method called “Println” and passes the argument “Hello, world!”.

You can define the main method as static in Java but in Scala, the static method is no longer available. Scala programmer can’t use static methods because they use singleton objects. To read more about singleton object you can refer this article.

14.1 Compile a Scala Program

To run any Scala program, you first need to compile it. “Scalac” is the compiler which takes source program as an argument and generates object files as output.

Let’s start compiling your “HelloWorld” program using the following steps:

1. For compiling it, you first need to paste this program into a text file then you need to save this program as HelloWorld.scala
2. Now you need change your working directory to the directory where your program is saved
3. After changing the directory you can compile the program by issuing the command.

```
scalac HelloWorld.scala
```

4. After compiling, you will get Helloworld.class as an output in the same directory. If you can see the file, you have successfully compiled the above program.

14.2 Running Scala Program

After compiling, you can now run the program using following command:

```
scala HelloWorld
```

You will get an output if the above command runs successfully. The program will print "Hello, world!"

15. Advantages of using Scala for Apache Spark

If you are working with Apache Spark then you would know that it has 4 different APIs support for different languages: **Scala, Java, Python and R**.

Each of these languages have their own unique advantages. But using Scala is more advantageous than other languages. These are the following reasons why Scala is taking over big data world.

- Working with Scala is more productive than working with Java
- Scala is faster than Python and R because it is compiled language
- Scala is a functional language

16. Comparing Scala, Java, Python and R APIs in Apache Spark

Let's compare 4 major languages which are supported by Apache Spark API.

Metrics	Scala	Java	Python	R
Type	Compiled	Compiled	Interpreted	Interpreted
JVM based	Yes	Yes	No	No
Verbosity	Less	More	Less	Less
Code Length	Less	More	Less	Less
Productivity	High	Less	High	High
Scalability	High	High	Less	Less

OOPS Support	Yes	Yes	Yes	Yes

17. Install Apache Spark & some basic concepts about Apache Spark

To know the basics of Apache Spark and installation, please refer to my first article on Pyspark. I have introduced basic terminologies used in Apache Spark like big data, cluster computing, driver, worker, spark context, In-memory computation, lazy evaluation, DAG, memory hierarchy and Apache Spark architecture in the previous article.

As a quick refresher, I will be explaining some of the topics which are very useful to proceed further. If you are a beginner, then I strongly recommend you to go through my first article before proceeding further.

- **Lazy operation:** Operations which do not execute until we require results.
- **Spark Context:** holds a connection with Spark cluster manager.
- **Driver and Worker:** A driver is in charge of the process of running the main() function of an application and creating the SparkContext.
- **In-memory computation:** Keeping the data in RAM instead of Hard Disk for fast processing.

Spark has three data representations viz RDD, Dataframe, Dataset. To use Apache Spark functionality, we must use one of them for data manipulation. Let’s discuss each of them briefly:

- **RDD:** RDD (Resilient Distributed Database) is a collection of elements, that can be divided across multiple nodes in a cluster for parallel processing. It is also fault tolerant collection of elements, which means it can automatically recover from failures. RDD is immutable, we can create RDD once but can’t change it.
- **Dataset:** It is also a distributed collection of data. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.). As I have already discussed in my previous articles, dataset API is only available in Scala and Java. It is not available in Python and R.
- **DataFrame:** In Spark, a DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame. It is mostly used for structured data processing. In Scala, a DataFrame is represented by a Dataset of Rows. A DataFrame can be constructed by wide range of arrays for example, existing RDDs, Hive tables, database tables.
- **Transformation:** Transformation refers to the operation applied on a RDD to create new RDD.

- **Action:** Actions refer to an operation which also apply on RDD that perform computation and send the result back to driver.
- **Broadcast:** We can use the Broadcast variable to save the copy of data across all node.
- **Accumulator:** In Accumulator, variables are used for aggregating the information.

18. Working with RDD in Apache Spark using Scala

First step to use RDD functionality is to create a RDD. In Apache Spark, RDD can be created by two different ways. One is from existing Source and second is from an external source.

So before moving further let's open the Apache Spark Shell with Scala. Type the following command after switching into the home directory of Spark. It will also load the spark context as sc.

```
$ ./bin/spark-shell
```

After typing above command you can start programming of Apache Spark in Scala.

18.1 Creating a RDD from existing source

When you want to create a RDD from existing storage in driver program (which we would like to be parallelized). For example, converting an array to RDD, which is already created in a driver program.

```
val data = Array(1, 2, 3, 4, 5,6,7,8,9,10)
val distData = sc.parallelize(data)
```

In the above program, I first created an array for 10 elements and then I created a distributed data called RDD from that array using "parallelize" method. SparkContext has a parallelize method, which is used for creating the Spark RDD from an iterable already present in driver program.

To see the content of any RDD we can use "collect" method. Let's see the content of distData.

```
scala> distData.collect()
Output: res1: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

18.2 Creating a RDD from External sources

You can create a RDD through external sources such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format. So let’s create a RDD from the text file:

The name of the text file is text.txt. and it has only 4 lines given below.

I love solving data mining problems.

I don’t like solving data mining problems.

I love solving data science problems.

I don’t like solving data science problems.

Let’s create the RDD by loading it.

```
val lines = sc.textFile("text.txt");
```

Now let’s see first two lines in it.

```
lines.take(2)
```

The output is received is as below:

```
Output: Array(I love solving data mining problems., I don't like solving data mining problems)
```

18.3 Transformations and Actions on RDD

18.3.1. Map Transformations

A map transformation is useful when we need to transform a RDD by applying a function to each element. So how can we use map transformation on ‘rdd’ in our case? Let’s calculate the length (number of characters) of each line in “text.txt”

```
val Lenght = lines.map(s => s.length)
Length.collect()
```

After applying above map operation, we get the following output:

```
Output: res6: Array[Int] = Array(36, 42, 37, 43)
```

18.3.2 Count Action

Let’s count the number of lines in RDD “lines”.

```
lines.count()
res1: Long = 4
```

The above action on “lines1” will give 4 as the output.

18.3.3 Reduce Action

Let’s take the sum of total number of characters in text.txt.

```
val totalLength = Length.reduce((a, b) => a + b)
totalLength: Int = 158
```

18.3.4 flatMap transformation and reduceByKey Action

Let’s calculate frequency of each word in “text.txt”

```
val counts = lines.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
counts.collect()
Output:
res6: Array[(String, Int)] = Array((solving,4), (mining,2), (don't,2), (love,2), (problems.,4),
(data,4), (science,2), (I,4), (like,2))
```

18.3.5 filter Transformation

Let's filter out the words in "text.txt" whose length is more than 5.

```
val lg5 = lines.flatMap(line => line.split(" ")).filter(_.length > 5)
```

Output:

```
res7: Array[String] = Array(solving, mining, problems., solving, mining, problems., solving,
science, problems., solving, science, problems.)
```

19. Working with DataFrame in Apache Spark using Scala

A DataFrame in Apache Spark can be created in multiple ways:

- It can be created using different data formats. For example, by loading the data from JSON, CSV
- Loading data from Existing RDD
- Programmatically specifying schema

Let's create a DataFrame using a csv file and perform some analysis on that.

For reading a csv file in Apache Spark, we need to specify a new library in our Scala shell. To perform this action, first, we need to download Spark-csv package (Latest version) and extract this package into the home directory of Spark. Then, we need to open a PySpark shell and include the package (I am using "spark-csv_2.10:1.3.0").

```
$ ./bin/spark-shell --packages com.databricks:spark-csv_2.10:1.3.0
```

Now let's load the csv file into a DataFrame df. You can download the file(train) from this link.

```
val df = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").load("train.csv")
```

19.1 Name of columns

Let's see the name of columns in df by using "columns" method.

```
df.columns
```

Output:

```
res0: Array[String] = Array(User_ID, Product_ID, Gender, Age, Occupation, City_Category,
Stay_In_Current_City_Years, Marital_Status, Product_Category_1, Product_Category_2,
Product_Category_3, Purchase)
```

19.2 Number of observations

To see the number of observation in df you can apply “count” method.

```
df.count()
```

Output:

```
res1: Long = 550068
```

19.3 Print the columns datatype

You can use “printSchema” method on df. Let’s print the schema of df.

```
df.printSchema()
```

Output:

```
root
 |-- User_ID: integer (nullable = true)
 |-- Product_ID: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- Occupation: integer (nullable = true)
 |-- City_Category: string (nullable = true)
 |-- Stay_In_Current_City_Years: string (nullable = true)
 |-- Marital_Status: integer (nullable = true)
 |-- Product_Category_1: integer (nullable = true)
 |-- Product_Category_2: integer (nullable = true)
 |-- Product_Category_3: integer (nullable = true)
 |-- Purchase: integer (nullable = true)
```

19.4 Show first n rows

You can use “show” method on DataFrame. Let’s print the first 2 rows of df.

```
df.show(2)
```

Output:

```
+-------+----------+------+----+----------+-------------+--------------------------+--------------+-----------
-------+------------------+------------------+--------+

|User_ID|Product_ID|Gender|
Age|Occupation|City_Category|Stay_In_Current_City_Years|Marital_Status|Product_Category_
1|Product_Category_2|Product_Category_3|Purchase|

+-------+----------+------+----+----------+-------------+--------------------------+--------------+-----------
-------+------------------+------------------+--------+

|1000001| P00069042| F|0-17| 10| A| 2| 0| 3| null| null| 8370|

|1000001| P00248942| F|0-17| 10| A| 2| 0| 1| 6| 14| 15200|

+-------+----------+------+----+----------+-------------+--------------------------+--------------+-----------
-------+------------------+------------------+--------+

only showing top 2 rows
```

19.5 Subsetting or select columns

To select columns you can use “select” method. Let’s apply select on df for “Age” columns.

```
df.select("Age").show(10)
```

Output:

```
+-----+

| Age|

+-----+

| 0-17|

| 0-17|

| 0-17|

| 0-17|
| 55+|
|26-35|
|46-50|
|46-50|
|46-50|
|26-35|
+-----+
only showing top 10 rows
```

19.6 Filter rows

To filter the rows you can use “filter” method. Let’s apply filter on “Purchase” column of df and get the purchase which is greater than 10000.

```
df.filter(df("Purchase") >= 10000).select("Purchase").show(10)
+--------+
|Purchase|
+--------+
| 15200|
| 15227|
| 19215|
| 15854|
| 15686|
| 15665|
| 13055|
| 11788|
| 19614|
| 11927|
+--------+
only showing top 10 rows
```

19.7 Group DataFrame

To groupby columns, you can use groupBy method on DataFrame. Let’s see the distribution on “Age” columns in df.

```
df.groupBy("Age").count().show()
```

Output:

```
+-----+------+
| Age| count|
+-----+------+
|51-55| 38501|
|46-50| 45701|
| 0-17| 15102|
|36-45|110013|
|26-35|219587|
| 55+| 21504|
|18-25| 99660|
+-----+------+
```

19.8 Apply SQL queries on DataFrame

To apply queries on DataFrame You need to register DataFrame(df) as table. Let’s first register df as temporary table called (B_friday).

```
df.registerTempTable("B_friday")
```

Now you can apply SQL queries on “B_friday” table using sqlContext.sql. Lets select columns “Age” from the “B_friday” using SQL statement.

```
sqlContext.sql("select Age from B_friday").show(5)
+----+
| Age|
+----+
|0-17|
|0-17|
|0-17|
|0-17|
| 55+|
+----+
```

20. Building a machine learning model

If you have come this far, you are in for a treat! I’ll complete this tutorial by building a machine learning model.

I will use only three dependent features and the independent variable in df1. Let’s create a DataFrame df1 which has only 4 columns (3 dependent and 1 target).

```
val df1 = df.select("User_ID","Occupation","Marital_Status","Purchase")
```

In above DataFrame df1 “User_ID”,”Occupation” and “Marital_Status” are features and “Purchase” is target column.

Let’s try to create a formula for Machine learning model like we do in R. First, we need to import RFormula. Then we need to specify the dependent and independent column inside this formula. We also have to specify the names for features column and label column.

```
import org.apache.spark.ml.feature.RFormula
val formula = new RFormula().setFormula("Purchase ~
User_ID+Occupation+Marital_Status").setFeaturesCol("features").setLabelCol("label")
```

After creating the formula, we need to fit this formula on df1 and transform df1 through this formula. Let’s fit this formula.

```
val train = formula.fit(df1).transform(df1)
```

After applying the formula we can see that train dataset has 2 extra columns called features and label. These are the ones we have specified in the formula (featuresCol=”features” and labelCol=”label”)

20.1 Applying Linear Regression on train

After applying the RFormula and transforming the DataFrame, we now need to develop the machine learning model on this data. I want to apply a Linear Regression for this task. Let us import a Linear regression and apply on train. Before fitting the model, I am setting the hyperparameters.

```
import org.apache.spark.ml.regression.LinearRegression
val lr = new LinearRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
val lrModel = lr.fit(train)
```

You can also make predictions on unseen data. But I am not showing this here. Let’s print the coefficient and intercept for linear regression.

```
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
```

Output:

```
Coefficients: [0.015092115630330033,16.12117786898672,-10.520580986444338] Intercept: -
5999.754797883323
```

Let’s summarize the model over the training set and print out some metrics.

```
val trainingSummary = lrModel.summary
```

Now, See the residuals for train's first 10 rows.

```
trainingSummary.residuals.show(10)
+-------------------+
| residuals|
+-------------------+
| -883.5877032522076|
| 5946.412296747792|
| -7831.587703252208|
| -8196.587703252208|
|-1381.3298625817588|
| 5892.776223171599|
| 10020.251134994305|
| 6659.251134994305|
| 6491.251134994305|
|-1533.3392694181512|
+-------------------+
only showing top 10 rows
```

Now, let’s see RMSE on train.

```
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
```

Output:

```
RMSE: 5021.899441991144
```

Let’s repeat above procedure for taking the prediction on cross-validation set. Let’s read the train dataset again.

```
val train = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").load("train.csv")
```

Now, randomly divide the train in two part train_cv and test_cv

```
val splits = train.randomSplit(Array(0.7, 0.3))
val (train_cv,test_cv) = (splits(0), splits(1))
```

Now, Transform train_cv and test_cv using RFormula.

```
import org.apache.spark.ml.feature.RFormula
val formula = new RFormula().setFormula("Purchase ~
User_ID+Occupation+Marital_Status").setFeaturesCol("features").setLabelCol("label")
```

```
val train_cv1 = formula.fit(train_cv).transform(train_cv)
val test_cv1 = formula.fit(train_cv).transform(test_cv)
```

After transforming using RFormula, we can build a machine learning model and take the predictions. Let’s apply Linear Regression on training and testing data.

```
import org.apache.spark.ml.regression.LinearRegression
val lr = new LinearRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
val lrModel = lr.fit(train_cv1)
val train_cv_pred = lrModel.transform(train_cv1)
val test_cv_pred = lrModel.transform(test_cv1)
```

In train_cv_pred and test_cv_pred, you will find a new column for prediction.

Conclusion: Hence, We have thoroughly studied how to **program in SCALA using Apache Spark.**