Girish Rajani
A20503736
CSP-554 Big Data Technologies
Homework 7

Exercise 1)

Step A

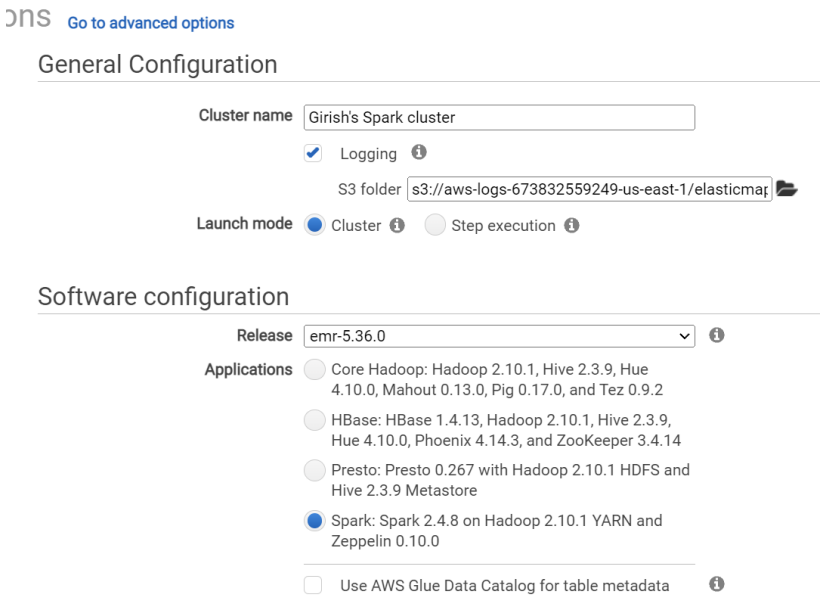

Figure 1 - Showing the Spark configuration selected when creating the EMR cluster

Step B

After successfully connecting to the master node via SSH command, the TestDataGen.class from assignment four was uploaded to the /home/hadoop directory via scp.

Figure 2 - Showing TestDataGen.class uploaded to /home/hadoop directory using scp

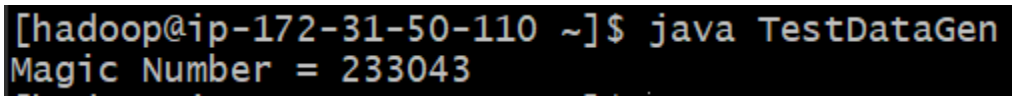

Figure 3 - Showing the file TestDataGen.class executed using “java TestDataGen” and generating a magic number, **233043**

Magic Number: **233043**

```
[hadoop@ip-172-31-50-110 ~]$ hadoop fs -put /home/hadoop/foodplaces233043.txt /h
ome/hadoop/foodratings233043.txt /user/hadoop
[hadoop@ip-172-31-50-110 ~]$
```

Figure 4 - Using -put command to copy foodratings233043.txt file and foodplaces233043.txt file from the /home/hadoop directory to HDFS /user/hadoop directory

Step C

```
[hadoop@ip-172-31-50-110 sparkdf]$ pyspark
Python 3.7.10 (default, Jun  3 2021, 00:02:01)
[GCC 7.3.1 20180712 (Red Hat 7.3.1-13)] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
22/10/10 19:38:06 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is
 set, falling back to uploading libraries under SPARK_HOME.
22/10/10 19:38:34 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to
request executors before the AM has registered!
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.8-amzn-2
      /_/

Using Python version 3.7.10 (default, Jun  3 2021 00:02:01)
SparkSession available as 'spark'.
```

Figure 5 - Running Spark

```
>>> from pyspark.sql.types import *
>>> struct1 = StructType().add("name", StringType(), True).add("food1",IntegerTy
pe(), True).add("food2",IntegerType(), True).add("food3",IntegerType(), True).ad
d("food4",IntegerType(), True).add("placeid",IntegerType(), True)
```

Figure 6 - Showing a schema struct1 specified which defines the columns and types for the dataframe which will be used when loading the foodratings233043.txt file

```
>>> foodratings = spark.read.schema(struct1).csv('hdfs:///user/hadoop/foodrating
s233043.txt')
>>> foodratings.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Joe|   44|   46|   37|   13|      5|
|Jill|   38|   35|   43|    2|      3|
| Joy|   33|    3|   10|   14|      4|
| Joy|   37|   48|   12|   12|      5|
| Mel|   18|   23|   47|   43|      3|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

Figure 7 - Creating a dataframe to load the foodratings233043.txt file as a ‘csv’ file using the schema from figure 6 and showing output of foodratings.printSchema() and foodratings.show(5)

Exercise 2)

```
>>> struct2 = StructType().add("placeid", IntegerType(), True).add("placename",S
tringType(), True)
>>> foodplaces = spark.read.schema(struct2).csv('hdfs:///user/hadoop/foodplaces2
33043.txt')
```

Figure 8 - Showing a schema struct2 specified, which defines the columns and types, which is then used to load the foodplaces233043.txt file as a 'csv' file

```
>>> foodplaces.printSchema()
root
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>> foodplaces.show(5)
+-------+------------+
|placeid|   placename|
+-------+------------+
|      1|China Bistro|
|      2|    Atlantic|
|      3|   Food Town|
|      4|      Jake's|
|      5|   Soup Bowl|
+-------+------------+
```

Figure 9 - Showing output after running foodplaces.printSchema() and foodplaces.show(5) commands

Exercise 3)

Step A

```
>>> foodratings.createOrReplaceTempView("foodratingsT")
>>> foodplaces.createOrReplaceTempView("foodplacesT")
```

Figure 10 - Registering the DataFrames created in exercise 1 and 2 as tables called "foodratingsT" and "foodplacesT"

Step B

```
>>> foodratings_ex3a = spark.sql("SELECT * FROM foodratingsT WHERE food2<25 AND
food4>40")
```

Figure 11 - Using a SQL query on the table "foodratingsT" to create a new DataFrame called foodratings_ex3a holding records where food2 < 25 and food4 > 40.

```
>>> foodratings_ex3a.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings_ex3a.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Mel|   18|   23|   47|   43|      3|
| Mel|   35|    5|    6|   42|      5|
| Sam|   47|    2|   33|   45|      4|
| Mel|   37|   16|   11|   45|      2|
| Joe|   48|    6|   34|   41|      2|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

Figure 12 - Showing output after running foodratings_ex3a.printSchema() and foodratings_ex3a.show(5) commands

Step C

```
>>> foodplaces_ex3b = spark.sql("SELECT * FROM foodplacesT WHERE placeid>3")
>>> foodplaces_ex3b.printSchema()
root
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>> foodplaces_ex3b.show(5)
+-------+---------+
|placeid|placename|
+-------+---------+
|      4|   Jake's|
|      5|Soup Bowl|
+-------+---------+
```

Figure 13 - Using a SQL query on the table “foodplacesT” to create a new DataFrame called foodplaces_ex3b holding records where placeid > 3 and then running the following commands: **foodplaces_ex3b.printSchema()** and **foodplaces_ex3b.show(5)**

Exercise 4)

```
>>> foodratings_ex4 = foodratings.filter(foodratings.name == "Mel").filter(foodr
atings.food3 < 25)
```

Figure 14 - Using a transformation on the DataFrame ‘foodratings’ created in exercise 1 to create a new DataFrame called foodratings_ex4 where ‘name’ is “Mel” and food3 < 25

```
>>> foodratings_ex4.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings_ex4.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Mel|   24|    1|   19|   32|      4|
| Mel|   35|    5|    6|   42|      5|
| Mel|    4|   31|    2|   33|      2|
| Mel|   37|   39|   21|   38|      2|
| Mel|   37|   16|   11|   45|      2|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

Figure 15 - Showing output after running foodratings_ex4.printSchema() and foodratings_ex4.show(5)

Exercise 5)

```
>>> foodratings_ex5 = foodratings.select(foodratings.name, foodratings.placeid)
```

Figure 16 - Using a transformation on the DataFrame ‘foodratings’ created in exercise 1 to create a new DataFrame called foodratings_ex5 that has only the columns ‘name’ and ‘placeid’

```
>>> foodratings_ex5.printSchema()
root
 |-- name: string (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings_ex5.show(5)
+----+-------+
|name|placeid|
+----+-------+
| Joe|      5|
|Jill|      3|
| Joy|      4|
| Joy|      5|
| Mel|      3|
+----+-------+
only showing top 5 rows
```

Figure 17 - Showing output after running foodratings_ex5.printSchema() and foodratings_ex5.show(5)

Exercise 6)

```
>>> ex6 = foodratings.join(foodplaces, foodratings.placeid == foodplaces.placeid
,'inner').drop(foodratings.placeid)
```

Figure 18 - Using a transformation to create a new DataFrame called ex6 which is the inner join, on placeid, of the DataFrames ‘foodratings’ and ‘foodplaces’ created in exercises 1 and 2

The placeid in foodratings was dropped because we already have palceid from foodplaces from the inner join so this would’ve just resulted in two placeid columns which meant having data redundancy.

```
>>> ex6.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>> ex6.show(5)
+----+-----+-----+-----+-----+-------+---------+
|name|food1|food2|food3|food4|placeid|placename|
+----+-----+-----+-----+-----+-------+---------+
| Joe|   44|   46|   37|   13|      5|Soup Bowl|
|Jill|   38|   35|   43|    2|      3|Food Town|
| Joy|   33|    3|   10|   14|      4|   Jake's|
| Joy|   37|   48|   12|   12|      5|Soup Bowl|
| Mel|   18|   23|   47|   43|      3|Food Town|
+----+-----+-----+-----+-----+-------+---------+
only showing top 5 rows
```

Figure 19 - Showing output after running ex6.printSchema() and ex6.show(5)