Chethan Mahindrakar – SEC01 (NUID 002646783)

Big Data System Engineering with Scala
Spring 2023
Spark Assignment No 1

The Task:

Load the 'Titanic – Machine Learning from Disaster' found on Kaggle (Specifically, the train.csv) onto Apache spark and perform certain operations and analysis.

The operations, their outcomes and the takeaways from each of the operations has been detailed in the following sections of this document.

Task 0: Creating a spark Session and loading the csv file in the Databricks notebook as a DataFrame

In this screenshot, we load the spark session. In order to do this, we import the apache spark session and create a session with the name of 'Spark Assignment 1'

```
Cmd 1

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions

val spark = SparkSession
  .builder()
  .appName("Spark Assignment 1")
  .getOrCreate()

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@72e06171

Command took 0.59 seconds -- by mahindrakar.c@northeastern.edu at 2/5/2023, 1:43:51 AM on 4th feb
```

In this next screenshot, we have used "header" as an option and set it to true. This allows for the very first row in the dataset to behave as the header for the table. All columns can now be addressed with these headers making it more convenient to access.

```
Cmd 2

//Reading the Dataset
val df = spark.read.option("header",true).csv("/FileStore/tables/train_my.csv")
df.show()

(2) Spark Jobs
df: org.apache.spark.sql.DataFrame = [PassengerId: string, Survived: string ... 10 more fields]

+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+
|PassengerId|Survived|Pclass|                Name|   Sex| Age|SibSp|Parch|          Ticket|   Fare|Cabin|Embarked|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+
|          1|       0|     3|Braund, Mr. Owen ...|  male|  22|    1|    0|       A/5 21171|   7.25| null|       S|
|          2|       1|     1|Cumings, Mrs. Joh...|female|  38|    1|    0|        PC 17599|71.2833|  C85|       C|
|          3|       1|     3|Heikkinen, Miss. ...|female|  26|    0|    0|STON/O2. 3101282|  7.925| null|       S|
|          4|       1|     1|Futrelle, Mrs. Ja...|female|  35|    1|    0|          113803|   53.1| C123|       S|
|          5|       0|     3|Allen, Mr. Willia...|  male|  35|    0|    0|          373450|   8.05| null|       S|
|          6|       0|     3|    Moran, Mr. James|  male|null|    0|    0|          330877| 8.4583| null|       Q|
|          7|       0|     1|McCarthy, Mr. Tim...|  male|  54|    0|    0|           17463|51.8625|  E46|       S|
|          8|       0|     3|Palsson, Master. ...|  male|   2|    3|    1|          349909| 21.075| null|       S|
|          9|       1|     3|Johnson, Mrs. Osc...|female|  27|    0|    2|          347742|11.1333| null|       S|
|         10|       1|     2|Nasser, Mrs. Nich...|female|  14|    1|    0|          237736|30.0708| null|       C|
|         11|       1|     3|Sandstrom, Miss. ...|female|   4|    1|    1|         PP 9549|   16.7|   G6|       S|
|         12|       1|     1|Bonnell, Miss. El...|female|  58|    0|    0|          113783|  26.55| C103|       S|
|         13|       0|     3|Saundercock, Mr. ...|  male|  20|    0|    0|       A/5. 2151|   8.05| null|       S|
|         14|       0|     3|Andersson, Mr. An...|  male|  39|    1|    5|          347082| 31.275| null|       S|
|         15|       0|     3|Vestrom, Miss. Hu...|female|  14|    0|    0|          350406| 7.8542| null|       S|
|         16|       1|     2|Hewlett, Mrs. (Ma...|female|  55|    0|    0|          248706|     16| null|       S|
|         17|       0|     3|Rice, Master. Eugene|  male|   2|    4|    1|          382652| 29.125| null|       Q|
|         18|       1|     2|Williams, Mr. Cha...|  male|null|    0|    0|          244373|     13| null|       S|
Command took 1.37 seconds -- by mahindrakar.c@northeastern.edu at 2/5/2023, 1:43:54 AM on 4th feb
```

Task 1: Calculating the average price of tickets per class

In this task, we use the filter() to filter out all specific classes per query in the dataframe. We then use the Aggregator function- specifically the avg() over “Fare” and rename the output using the as(). The averages per ticket class can be seen in the screenshot.

The show() is to display the output after performing all of these queries.

Cmd 3

```
//Sub-question 1 -------- The average ticket fare for each ticket class
df.filter("Pclass = 1").agg(avg("Fare").as("Upper Average")).show()
df.filter("Pclass = 2").agg(avg("Fare").as("Middle Average")).show()
df.filter("Pclass = 3").agg(avg("Fare").as("Lower Average")).show()
```

▸ (6) Spark Jobs

```
+-----------------+
|    Upper Average|
+-----------------+
|84.15468749999992|
+-----------------+

+-----------------+
|   Middle Average|
+-----------------+
|20.66218315217391|
+-----------------+

+------------------+
|     Lower Average|
+------------------+
|13.675550101832997|
+------------------+
```

Command took 1.56 seconds -- by mahindrakar.c@northeastern.edu at 2/5/2023, 1:43:59 AM on 4th feb

Alternatively, the following approach can be taken as well to use a single query to show the averages of the fare for each of the classes. In this approach, we use groupBy() to separate identical data into groups. We need the Average Aggregator function to calculate the average of the ‘Fares’ (Same as last version) and we use the orderBy() to sort (default is in ascending order) the results based on Pclass.

```
Cmd 4

df.groupBy("Pclass").agg(avg("Fare")).orderBy("Pclass").show()

▸ (2) Spark Jobs
+------+------------------+
|Pclass|         avg(Fare)|
+------+------------------+
|     1| 84.15468749999992|
|     2| 20.66218315217391|
|     3|13.675550101832997|
+------+------------------+

Command took 2.88 seconds -- by mahindrakar.c@northeastern.edu at 2/7/2023, 12:37:20 AM on 6th feb - 3
```

Task 2: Calculating the survival percentage for each ticket class and figuring out the class with the highest survival rate

```
Cmd 5

//Sub Question 2 ------- Survival Percentage for each Ticket class
val totalPassenger = df.select("PassengerId").count()
val survived1 : Double = (df.filter("Pclass = 1").filter("Survived = 1").count().toDouble / totalPassenger.toDouble) * 100
val survived2 : Double = (df.filter("Pclass = 2").filter("Survived = 1").count().toDouble / totalPassenger.toDouble) * 100
val survived3 : Double = (df.filter("Pclass = 3").filter("Survived = 1").count().toDouble / totalPassenger.toDouble) * 100

import scala.math._
print(max(survived1, max(survived2, survived3)))


▸ (8) Spark Jobs
15.26374859708193totalPassenger: Long = 891
survived1: Double = 15.26374859708193
survived2: Double = 9.764309764309765
survived3: Double = 13.35578002244669
import scala.math._

Command took 3.42 seconds -- by mahindrakar.c@northeastern.edu at 2/7/2023, 12:37:24 AM on 6th feb - 3
```

In order to calculate the survival percentage, we first need to calculate the total number of survivors. This can be done by using the count() on the “PassengerId” column (or any of the columns really…). As can be seen, the total no of passengers is 891.

The next step is to use the formula to calculate the percentage survival rate of each class of the passenger. To do this, we first need to find the number of survivors in each class for which we use 2 filter() on the dataframe – one for filtering based on class and the second to filter for the survivors.

Both of the counts need to explicitly be converted to Double type since we are expecting a percentage.

The screenshot shows the survival rate for each of the classes. The class with the maximum survival rate as can be seen is Pclass 1.

Task 3: Finding the number of passengers that could potentially be Rose

```
Cmd 5

//Sub Question 3 --------- find the number of passengers who could possibly be Rose
// Survival = 1, Pclass = 1, Parch = 1, Gender = Female, age = 17

//df.filter("Pclass = 1").filter("Survived = 1").filter("sex = 'female'").filter("Parch = 1").filter("age = 17").count
df.filter(expr("sex = 'female' and Survived = 1 and Pclass = 1 and Parch = 1 and age = 17")).count

(2) Spark Jobs
res175: Long = 0
1
Command took 0.59 seconds -- by mahindrakar.c@northeastern.edu at 2/5/2023, 1:44:14 AM on 4th feb
```

The conditions for a person to potentially be Rose are mentioned in the comments. In order to evaluate this, I first came up with a query that had multiple filter(). This seemed highly inefficient which is when I came across the expr(). This allows me to put the entire expression that is to be evaluated and after the evaluation is complete, the filter() works on the entire expression in one go. This seems to more efficient.

The number of candidates that could be Rose is 0.

Task 4: Finding the number of passengers that could potentially be Jack

```
Cmd 6

//Sub Question 4 --------- Number of passengers who could possibly be Jack?
//Survival = 0, Pclass = 3, Gender = Male, age = 19 or 20, SibSp = 0, Parch = 0

var temp_df = df.filter(expr("sex = 'male'  and Pclass = 3 and SibSP = 0 and Parch = 0 and survived = 0"))
temp_df = temp_df.filter(expr("age >= 19 and age <= 20"))
temp_df.count()


(2) Spark Jobs
temp_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [PassengerId: integer, Survived: integer ... 10 more fields]
temp_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [PassengerId: int, Survived: int ... 10 more fields]
temp_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [PassengerId: int, Survived: int ... 10 more fields]
res11: Long = 21
Command took 1.65 seconds -- by mahindrakar.c@northeastern.edu at 2/6/2023, 3:56:39 PM on 6th feb -2
```

This query is actually similar to the previous query, but only it is processed in two parts. The first part evaluates all the conditions except age and the second part evaluates just the age. This can however be done as a single query itself as shows below.

```
Cmd 7

//Sub Question 4 --------- Number of passengers who could possibly be Jack?
//Survival = 0, Pclass = 3, Gender = Male, age = 19 or 20, SibSp = 0, Parch = 0

//var temp_df = df.filter(expr("sex = 'male'  and Pclass = 3 and SibSP = 0 and Parch = 0 and survived = 0"))
//temp_df = temp_df.filter(expr("age >= 19 and age <= 20"))
//temp_df.count()

df.filter(expr("(sex = 'male'  and Pclass = 3 and SibSP = 0 and Parch = 0 and survived = 0) and (age >= 19 and age <= 20)" )).count()


(2) Spark Jobs
res11: Long = 21
Command took 0.98 seconds -- by mahindrakar.c@northeastern.edu at 2/7/2023, 12:59:45 AM on 6th feb - 3
```

The next screenshot shows all possible candidates who could be ‘Jack’

Assumption: In the query, I have assumed that survival = 0 for Jack since he died in the movie.

```
//Sub Question 4 ------- Continued --- Displaying potential Jacks
temp_df.show(22)  //Since show only displays the first 20 rows
```

▸ (1) Spark Jobs

```
|         13|       0|     3|Saundercock, Mr. ...|male|20.0|    0|    0|         A/5. 2151|   8.05| null|        S|
|         68|       0|     3|Crease, Mr. Ernes...|male|19.0|    0|    0|         S.P. 3464| 8.1583| null|        S|
|         92|       0|     3|Andreasson, Mr. P...|male|20.0|    0|    0|            347466| 7.8542| null|        S|
|        132|       0|     3|Coelho, Mr. Domin...|male|20.0|    0|    0|SOTON/O.Q. 3101307|   7.05| null|        S|
|        144|       0|     3| Burke, Mr. Jeremiah|male|19.0|    0|    0|            365222|   6.75| null|        Q|
|        303|       0|     3|Johnson, Mr. Will...|male|19.0|    0|    0|              LINE|    0.0| null|        S|
|        373|       0|     3|Beavan, Mr. Willi...|male|19.0|    0|    0|            323951|   8.05| null|        S|
|        379|       0|     3| Betros, Mr. Tannous|male|20.0|    0|    0|              2648| 4.0125| null|        C|
|        380|       0|     3|Gustafsson, Mr. K...|male|19.0|    0|    0|            347069|  7.775| null|        S|
|        442|       0|     3|     Hampe, Mr. Leon|male|20.0|    0|    0|            345769|    9.5| null|        S|
|        567|       0|     3|Stoytcheff, Mr. Ilia|male|19.0|    0|    0|            349205| 7.8958| null|        S|
|        576|       0|     3|Patchett, Mr. George|male|19.0|    0|    0|            358585|   14.5| null|        S|
|        641|       0|     3|Jensen, Mr. Hans ...|male|20.0|    0|    0|            350050| 7.8542| null|        S|
|        647|       0|     3|   Cor, Mr. Liudevit|male|19.0|    0|    0|            349231| 7.8958| null|        S|
|        683|       0|     3|Olsvigen, Mr. Tho...|male|20.0|    0|    0|              6563|  9.225| null|        S|
|        688|       0|     3|   Dakic, Mr. Branko|male|19.0|    0|    0|            349228|10.1708| null|        S|
|        716|       0|     3|Soholt, Mr. Peter...|male|19.0|    0|    0|            348124|   7.65|F G73|        S|
|        726|       0|     3| Oreskovic, Mr. Luka|male|20.0|    0|    0|            315094| 8.6625| null|        S|
|        841|       0|     3|Alhomaki, Mr. Ilm...|male|20.0|    0|    0|  SOTON/O2 3101287|  7.925| null|        S|
|        877|       0|     3|Gustafsson, Mr. A...|male|20.0|    0|    0|              7534| 9.8458| null|        S|
|        878|       0|     3|Petroff, Mr. Nedelio|male|19.0|    0|    0|            349212| 7.8958| null|        S|
```

Command took 1.04 seconds -- by mahindrakar.c@northeastern.edu at 2/6/2023, 3:56:42 PM on 6th feb -2

Task 5: Split the age in the dataframe into groups of 10 years

Inorder to achieve this, I used the WithColumn() that basically allows for the manipulation of columns within dataframes. I created a temporary dataframe and manipulated the ‘age’ function to include the range it belonged to as opposed to the actual ages. This would be an overwrite of the entire column but on a different dataframe. The when() otherwise() is similar to the ‘if-else’ and ‘case when’ constructs from several other programming languages and SQL. This allows to establish a condition and rewrite the value within the column. In this case, other() corresponded to all of the null values within the dataset.

```
Cmd 8

//Sub Question 5 ---------- Split the age across every 10 years.

val agedf = df.withColumn(
  "age",
  when(col("age").between(1, 10), "1-10").
  when(col("age").between(11, 20), "11-20").
  when(col("age").between(21, 30), "21-30").
  when(col("age").between(31, 40), "31-40").
  when(col("age").between(41, 50), "41-50").
  when(col("age").between(51, 60), "51-60").
  when(col("age").between(61, 70), "61-70").
  when(col("age").between(71, 80), "71-80").
  when(col("age").between(81, 90), "81-90").
  otherwise("other")).toDF

// "other" over here corresponds to all the null values that are in the dataset

agedf: org.apache.spark.sql.DataFrame = [PassengerId: string, Survived: string ... 10 more fields]
agedf: org.apache.spark.sql.DataFrame = [PassengerId: string, Survived: string ... 10 more fields]
Command took 0.41 seconds -- by mahindrakar.c@northeastern.edu at 2/5/2023, 3:11:13 AM on 4th feb
```

Figuring out the relationship between age and ticket fare. As can be seen from the screenshot, the Fares for the Ages between 1-30 had an average asking price of about 29 per ticket. This Fare changed for the age groups more than 30. From 30-70, the average was 42. The row ‘other’ indicates all the entries where the ‘age’ was not specified.

```
Cmd 9

// Relationship between age and ticket fare -- Need to calculate the mean to get a rough idea
agedf.groupBy("age").agg(avg("Fare")).orderBy("age").show()  // -- Avg price for ages 1 - 30 is about 29 whereas that for ages b/n 40-70 is 42

(2) Spark Jobs
+-----+------------------+
|  age|         avg(Fare)|
+-----+------------------+
| 1-10| 29.59597894736843|
|11-20|29.337466379310357|
|21-30|28.221191774891793|
|31-40|42.537712903225824|
|41-50|41.878376190476196|
|51-60| 44.77480238095238|
|61-70| 43.79073888888888|
|71-80|          30.48335|
|other| 22.73315000000001|
+-----+------------------+

Command took 0.70 seconds -- by mahindrakar.c@northeastern.edu at 2/5/2023, 3:11:17 AM on 4th feb
```

The age group that is most likely to have survived is indicated in the screenshot below. The ages of 21-30 had the most number of survivors. In order to show this, I used the OrderBy(desc()) that orders the survived_count column within a temporary dataframe called tempAgeDf. The first() prints out the very first row of the dataframe. Apart from this, the sum() is used to calculate the sum of survivors in each age category. Alias() is used to assign a name to the newly generated column.

Cmd 11

```
//Which age group most likely survived? -- Count the number of survived in each group
val tempAgeDf = agedf.groupBy("age").agg(sum(when(col("survived") === true, 1)).alias("survived_count")).orderBy(desc("survived_count"))
tempAgeDf.first() //Most survived age group is 21-30
```

▸ (2) Spark Jobs

▸ tempAgeDf: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [age: string, survived_count: long]

```
tempAgeDf: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [age: string, survived_count: bigint]
res16: org.apache.spark.sql.Row = [21-30,84]
```

Command took 2.83 seconds -- by mahindrakar.c@northeastern.edu at 2/7/2023, 1:02:46 AM on 6th feb - 3

Takeaways from this assignment

Using Databricks Notebook, DataFrames and SQL functions in Spark