Case Study 3 – Working with Sensor Data

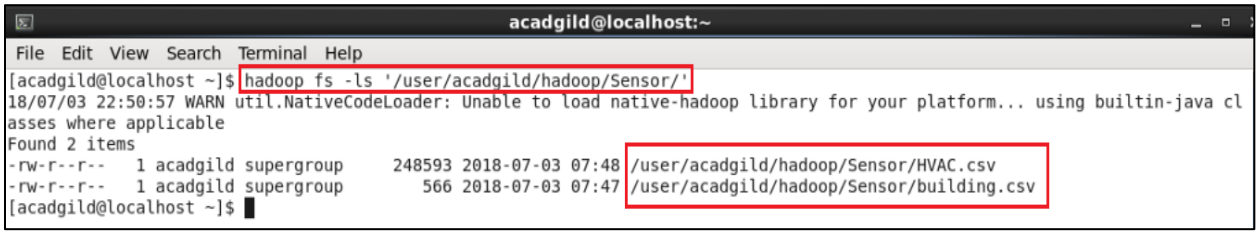
There are two data files provided.

1. Building.csv
2. HVAC.csv

Both these files are loaded into HDFS first.

hadoop fs -put building.csv '/user/acadgild/hadoop/Sensor/'

hadoop fs -put hvac.csv '/user/acadgild/hadoop/Sensor/'



On listing the contents of the Sensor folder on HDFS, we can see that the 2 csv files are present.

Objective 1:

- **Load HVAC.csv file into temporary table.**
- **Add a new column, tempchange -set to 1, if there is a change of greater than +/-5 between actual and target temperature**

Loading HVAC.csv into Temporary table:

Creating a rdd out of the textfile.

val hrdd = sc.textFile("/user/acadgild/hadoop/Sensor/HVAC.csv")

Importing all the required classes.

import org.apache.spark.sql.types.StructType
import org.apache.spark.sql.types.{StructField,StringType}

import org.apache.spark.sql.Row

Extracting the schema out of the header in the csv file and create a dataframe out of the data with this schema.

val hheader = hrdd.first
val hfs = hheader.split(",").map(f=> StructField(f,StringType))
val hschema = StructType(hfs)
val hnoheader = hrdd.filter(_!=hheader)
val hrows = hnoheader.map(_.split(",")).map(a => Row.fromSeq(a))
val hvacdf = spark.createDataFrame(hrows,hschema)

```
scala>

scala> val hrdd = sc.textFile("/user/acadgild/hadoop/Sensor/HVAC.csv")
hrdd: org.apache.spark.rdd.RDD[String] = /user/acadgild/hadoop/Sensor/HVAC.csv MapPartitionsRDD[9] at textFile at <console>:2
7

scala> import org.apache.spark.sql.types.StructType
import org.apache.spark.sql.types.StructType

scala> import org.apache.spark.sql.types.{StructField,StringType}
import org.apache.spark.sql.types.{StructField, StringType}

scala> import org.apache.spark.sql.Row
import org.apache.spark.sql.Row

scala> val hheader = hrdd.first
hheader: String = Date,Time,TargetTemp,ActualTemp,System,SystemAge,BuildingID

scala> val hfs = hheader.split(",").map(f=> StructField(f,StringType))
hfs: Array[org.apache.spark.sql.types.StructField] = Array(StructField(Date,StringType,true), StructField(Time,StringType,tru
e), StructField(TargetTemp,StringType,true), StructField(ActualTemp,StringType,true), StructField(System,StringType,true), St
ructField(SystemAge,StringType,true), StructField(BuildingID,StringType,true))

scala> val hschema = StructType(hfs)
hschema: org.apache.spark.sql.types.StructType = StructType(StructField(Date,StringType,true), StructField(Time,StringType,tr
ue), StructField(TargetTemp,StringType,true), StructField(ActualTemp,StringType,true), StructField(System,StringType,true), S
tructField(SystemAge,StringType,true), StructField(BuildingID,StringType,true))

scala> val hnoheader = hrdd.filter(_!=hheader)
hnoheader: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[10] at filter at <console>:34

scala> val hrows = hnoheader.map(_.split(",")).map(a => Row.fromSeq(a))
hrows: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[12] at map at <console>:36

scala> val hvacdf = spark.createDataFrame(hrows,hschema)
hvacdf: org.apache.spark.sql.DataFrame = [Date: string, Time: string ... 5 more fields]
```

Now, this dataframe is registered as a temporary table. This temporary table can be then queried.

hvacdf.registerTempTable("HVAC")

We can see that the table HVAC can be queried.

```
scala> hvacdf.registerTempTable("HVAC")
warning: there was one deprecation warning; re-run with -deprecation for details

scala> val query = spark.sql("select * from HVAC")
query: org.apache.spark.sql.DataFrame = [Date: string, Time: string ... 5 more fields]

scala> query.show
+--------+--------+----------+----------+------+---------+----------+
|    Date|    Time|TargetTemp|ActualTemp|System|SystemAge|BuildingID|
+--------+--------+----------+----------+------+---------+----------+
| 6/1/13| 0:00:01|        66|        58|    13|       20|         4|
| 6/2/13| 1:00:01|        69|        68|     3|       20|        17|
| 6/3/13| 2:00:01|        70|        73|    17|       20|        18|
| 6/4/13| 3:00:01|        67|        63|     2|       23|        15|
| 6/5/13| 4:00:01|        68|        74|    16|        9|         3|
| 6/6/13| 5:00:01|        67|        56|    13|       28|         4|
| 6/7/13| 6:00:01|        70|        58|    12|       24|         2|
| 6/8/13| 7:00:01|        70|        73|    20|       26|        16|
| 6/9/13| 8:00:01|        66|        69|    16|        9|         9|
|6/10/13| 9:00:01|        65|        57|     6|        5|        12|
|6/11/13|10:00:01|        67|        70|    10|       17|        15|
|6/12/13|11:00:01|        69|        62|     2|       11|         7|
|6/13/13|12:00:01|        69|        73|    14|        2|        15|
|6/14/13|13:00:01|        65|        61|     3|        2|         6|
|6/15/13|14:00:01|        67|        59|    19|       22|        20|
|6/16/13|15:00:01|        65|        56|    19|       11|         8|
|6/17/13|16:00:01|        67|        57|    15|        7|         6|
|6/18/13|17:00:01|        66|        57|    12|        5|        13|
|6/19/13|18:00:01|        69|        58|     8|       22|         4|
|6/20/13|19:00:01|        67|        55|    17|        5|         7|
+--------+--------+----------+----------+------+---------+----------+
only showing top 20 rows
```

Add a new column, tempchange -set to 1, if there is a change of greater than +/-5 between actual and target temperature

```
scala> val hvacupdated = hvacdf.withColumn("tempchange", when($"targettemp"-$"actualtemp">5 or $"targettemp"-$"actualtemp"<(-
5),1).otherwise(0))
hvacupdated: org.apache.spark.sql.DataFrame = [Date: string, Time: string ... 6 more fields]

scala> hvacupdated.show
+--------+--------+----------+----------+------+---------+----------+----------+
|    Date|    Time|TargetTemp|ActualTemp|System|SystemAge|BuildingID|tempchange|
+--------+--------+----------+----------+------+---------+----------+----------+
| 6/1/13| 0:00:01|        66|        58|    13|       20|         4|         1|
| 6/2/13| 1:00:01|        69|        68|     3|       20|        17|         0|
| 6/3/13| 2:00:01|        70|        73|    17|       20|        18|         0|
| 6/4/13| 3:00:01|        67|        63|     2|       23|        15|         0|
| 6/5/13| 4:00:01|        68|        74|    16|        9|         3|         1|
| 6/6/13| 5:00:01|        67|        56|    13|       28|         4|         1|
| 6/7/13| 6:00:01|        70|        58|    12|       24|         2|         1|
| 6/8/13| 7:00:01|        70|        73|    20|       26|        16|         0|
| 6/9/13| 8:00:01|        66|        69|    16|        9|         9|         0|
|6/10/13| 9:00:01|        65|        57|     6|        5|        12|         1|
|6/11/13|10:00:01|        67|        70|    10|       17|        15|         0|
|6/12/13|11:00:01|        69|        62|     2|       11|         7|         1|
|6/13/13|12:00:01|        69|        73|    14|        2|        15|         0|
|6/14/13|13:00:01|        65|        61|     3|        2|         6|         0|
|6/15/13|14:00:01|        67|        59|    19|       22|        20|         1|
|6/16/13|15:00:01|        65|        56|    19|       11|         8|         1|
|6/17/13|16:00:01|        67|        57|    15|        7|         6|         1|
|6/18/13|17:00:01|        66|        57|    12|        5|        13|         1|
|6/19/13|18:00:01|        69|        58|     8|       22|         4|         1|
|6/20/13|19:00:01|        67|        55|    17|        5|         7|         1|
+--------+--------+----------+----------+------+---------+----------+----------+
only showing top 20 rows
```

The withColumn function returns a new dataframe with a new column appended to the existing dataframe.

val hvacupdated = hvacdf.withColumn("tempchange",when($"targettemp"-$"actualtemp">5 or $"targettemp"-$"actualtemp"<(-5),1).otherwise(0))

The name of the new column is ‘tempchange’. The values that are to be populated in the new column are specified in the second part of withColumn function.

Since the value of the new column is based on the difference in the other tow columns, we use a when condition. So when the specified condition is true, 1 is popoulated, in all the other cases, 0 is populated in the newly added column.

The new dataframe is shown in the screenshot above.

hvacupdated.show

Now this dataframe has to be saved as another temporary table so that it can be queried.

```
scala>

scala> hvacupdated.registerTempTable("HVACUpdated")
warning: there was one deprecation warning; re-run with -deprecation for details
```

Objective 2:

Load building.csv file into temporary table.

Creating a rdd out of the textfile.

val brdd = sc.textFile("/user/acadgild/hadoop/Sensor/building.csv")

Since all the necessary classes are already imported, we can move to the next step.

Extracting the schema out of the header in the csv file and create a dataframe out of the data with this schema.

val bheader = brdd.first
val fs = bheader.split(",").map(f=> StructField(f,StringType))
val schema = StructType(fs)
val noheader = brdd.filter(_ !=bheader)
import org.apache.spark.sql.Row
val rows = noheader.map(_.split(",")).map(a => Row.fromSeq(a))
val buildingdf = spark.createDataFrame(rows,schema)

```
scala>

scala> val brdd = sc.textFile("/user/acadgild/hadoop/Sensor/building.csv")
brdd: org.apache.spark.rdd.RDD[String] = /user/acadgild/hadoop/Sensor/building.csv MapPartitionsRDD[17] at textFile at <conso
le>:30

scala> val bheader = brdd.first
bheader: String = BuildingID,BuildingMgr,BuildingAge,HVACproduct,Country

scala> val fs = bheader.split(",").map(f=> StructField(f,StringType))
fs: Array[org.apache.spark.sql.types.StructField] = Array(StructField(BuildingID,StringType,true), StructField(BuildingMgr,St
ringType,true), StructField(BuildingAge,StringType,true), StructField(HVACproduct,StringType,true), StructField(Country,Strin
gType,true))

scala> val schema = StructType(fs)
schema: org.apache.spark.sql.types.StructType = StructType(StructField(BuildingID,StringType,true), StructField(BuildingMgr,S
tringType,true), StructField(BuildingAge,StringType,true), StructField(HVACproduct,StringType,true), StructField(Country,Stri
ngType,true))

scala> val noheader = brdd.filter(_ !=bheader)
noheader: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[18] at filter at <console>:34

scala> val rows = noheader.map(_.split(",")).map(a => Row.fromSeq(a))
rows: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[20] at map at <console>:36

scala> val buildingdf = spark.createDataFrame(rows,schema)
buildingdf: org.apache.spark.sql.DataFrame = [BuildingID: string, BuildingMgr: string ... 3 more fields]
```

Now, this dataframe is registered as a temporary table. This temporary table can be then queried.

buildingdf.registerTempTable("Buildings")

We can see that the table Buildings can be queried.

```
scala>

scala> buildingdf.registerTempTable("Buildings")
warning: there was one deprecation warning; re-run with -deprecation for details

scala> val query = spark.sql("select * from buildings")
query: org.apache.spark.sql.DataFrame = [BuildingID: string, BuildingMgr: string ... 3 more fields]

scala> query.show
+----------+-----------+-----------+-----------+------------+
|BuildingID|BuildingMgr|BuildingAge|HVACproduct|     Country|
+----------+-----------+-----------+-----------+------------+
|         1|         M1|         25|     AC1000|         USA|
|         2|         M2|         27|     FN39TG|      France|
|         3|         M3|         28|     JDNS77|      Brazil|
|         4|         M4|         17|     GG1919|     Finland|
|         5|         M5|          3|    ACMAX22|   Hong Kong|
|         6|         M6|          9|     AC1000|   Singapore|
|         7|         M7|         13|     FN39TG|South Africa|
|         8|         M8|         25|     JDNS77|   Australia|
|         9|         M9|         11|     GG1919|      Mexico|
|        10|        M10|         23|    ACMAX22|       China|
|        11|        M11|         14|     AC1000|     Belgium|
|        12|        M12|         26|     FN39TG|     Finland|
|        13|        M13|         25|     JDNS77|Saudi Arabia|
|        14|        M14|         17|     GG1919|     Germany|
|        15|        M15|         19|    ACMAX22|      Israel|
|        16|        M16|         23|     AC1000|      Turkey|
|        17|        M17|         11|     FN39TG|       Egypt|
|        18|        M18|         25|     JDNS77|   Indonesia|
|        19|        M19|         14|     GG1919|      Canada|
|        20|        M20|         19|    ACMAX22|   Argentina|
+----------+-----------+-----------+-----------+------------+
```

Objective 3:

Figure out the number of times, temperature has changed by 5 degrees or more for each country:

- **Join both the tables.**
- **Select tempchange and country column**

- **Filter the rows where tempchange is 1 and count the number of occurrence for each country**

```
scala>

scala> val q3 = spark.sql("select count(h.tempchange),b.country from buildings b, hvacupdated h where h.buildingid=b.building
id and h.tempchange = 1 group by b.country")
q3: org.apache.spark.sql.DataFrame = [count(tempchange): bigint, country: string]

scala> q3.show
+-----------------+------------+
|count(tempchange)|     country|
+-----------------+------------+
|              230|   Singapore|
|              243|      Turkey|
|              196|     Germany|
|              251|      France|
|              230|   Argentina|
|              199|     Belgium|
|              473|     Finland|
|              241|       China|
|              248|   Hong Kong|
|              232|      Israel|
|              213|         USA|
|              228|      Mexico|
|              243|   Indonesia|
|              233|Saudi Arabia|
|              232|      Canada|
|              226|      Brazil|
|              225|   Australia|
|              236|       Egypt|
|              237|South Africa|
+-----------------+------------+
```

According to the steps mentioned, the query is

> *val q13 = spark.sql("select count(h.tempchange),b.country from buildings b, hvacupdated h where h.buildingid=b.buildingid and h.tempchange = 1 group by b.country")*

Both the tables Buildings and HVACUpdated are joined on the common column buildingID.

> *h.buildingid=b.buildingid*

The tempchange column and the country column are displayed.

> *select count(h.tempchange),b.country*

A condition that the value of tempchange must be 1 is added, thus filtering the data where the difference is +/-5.

> *h.tempchange = 1*

A count function is applied on this data to calculate how many times each country experienced a temperature change of +/-5.

> *select count(h.tempchange)*

group by b.country

The count in each country is displayed in the results.