Case Study 3 Sensor

For this data analysis, you can download the necessary dataset from this link.

In the above link there are two datasets; building.csv contains the details of the top 20 buildings all over the world and HVAC.csv contains the target temperature and the actual temperature along with the building Id.

HVAC (heating, ventilating/ventilation, and air conditioning) is the technology of indoor and vehicular environmental comfort. Its goal is to provide thermal comfort and acceptable indoor air quality. Through the HVAC sensors, we will get the temperature of the buildings.

Here are the columns that are present in the datasets:

Building.csv – BuildingID, BuildingMgr, BuildingAge, HVACproduct,Country

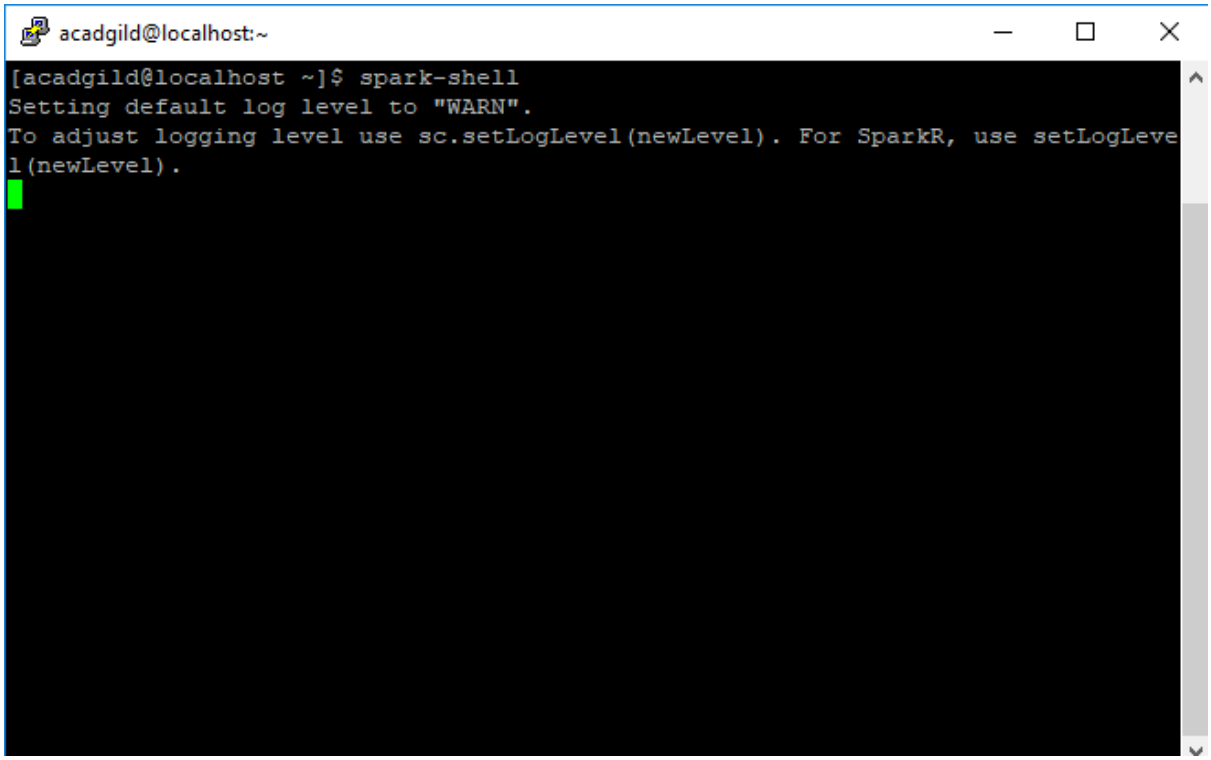
HVAC.csv – Date, Time, TargetTemp, ActualTemp, System, SystemAge, BuildingID

Objective 1

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

Step 1:

Start Spark –shell



Step 2:

//Load data into spark

val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val data = sc.textFile("/user/acadgild/HVAC.csv")

//Remove Header

val header = data.first()

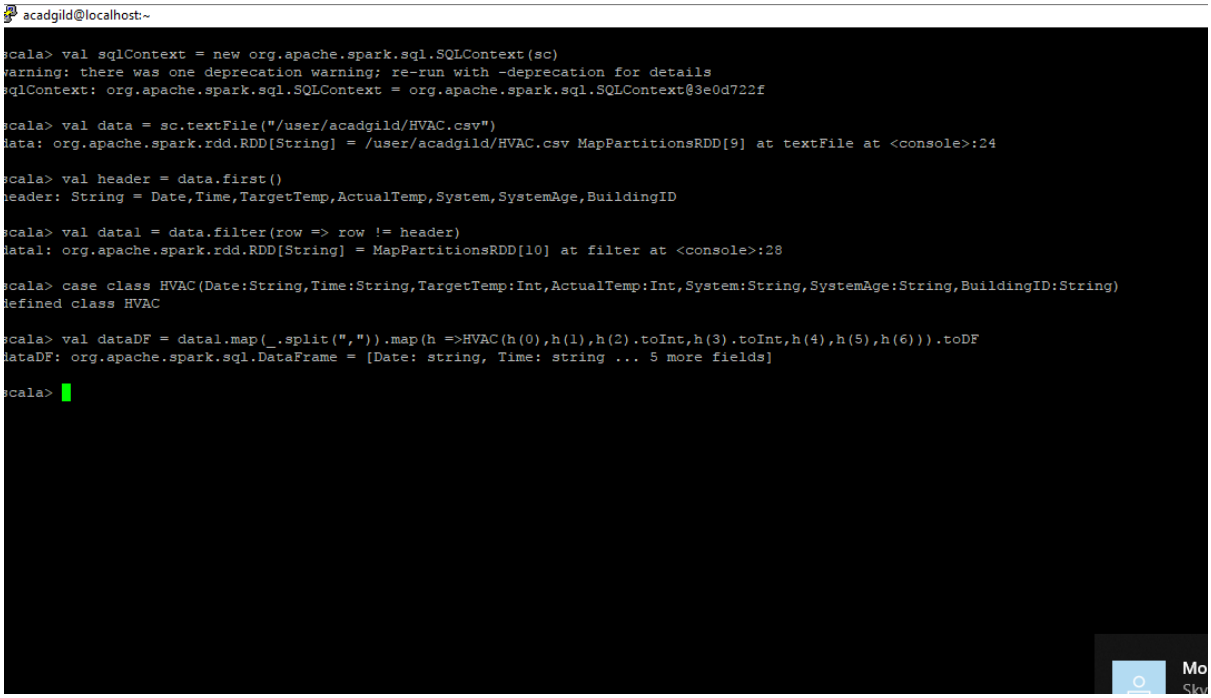
val data1 = data.filter(row => row != header)

//Define Case Class HVAC

case class HVAC(Date:String,Time:String,TargetTemp:Int,ActualTemp:Int,System:String,SystemAge:String,BuildingID:String)

//Convert to DataFrame

val dataDF = data1.map(_.split(",")).map(h=>HVAC(h(0),h(1),h(2).toInt,h(3).toInt,h(4),h(5),h(6))).toDF



Step 3:

//Adding new column NewTemp to find difference between TargetTemp and ActualTemp

```
val temp = udf((TargetTemp:Int,ActualTemp:Int) =>
(TargetTemp,ActualTemp)match

{

case(TargetTemp,ActualTemp) => {TargetTemp-ActualTemp}

})

spark.udf.register("NewTemp",temp)

val tempRDD
=dataDF.withColumn("NewTemp",temp(dataDF.col("TargetTemp"),dataDF.col(
"ActualTemp"))).toDF
```

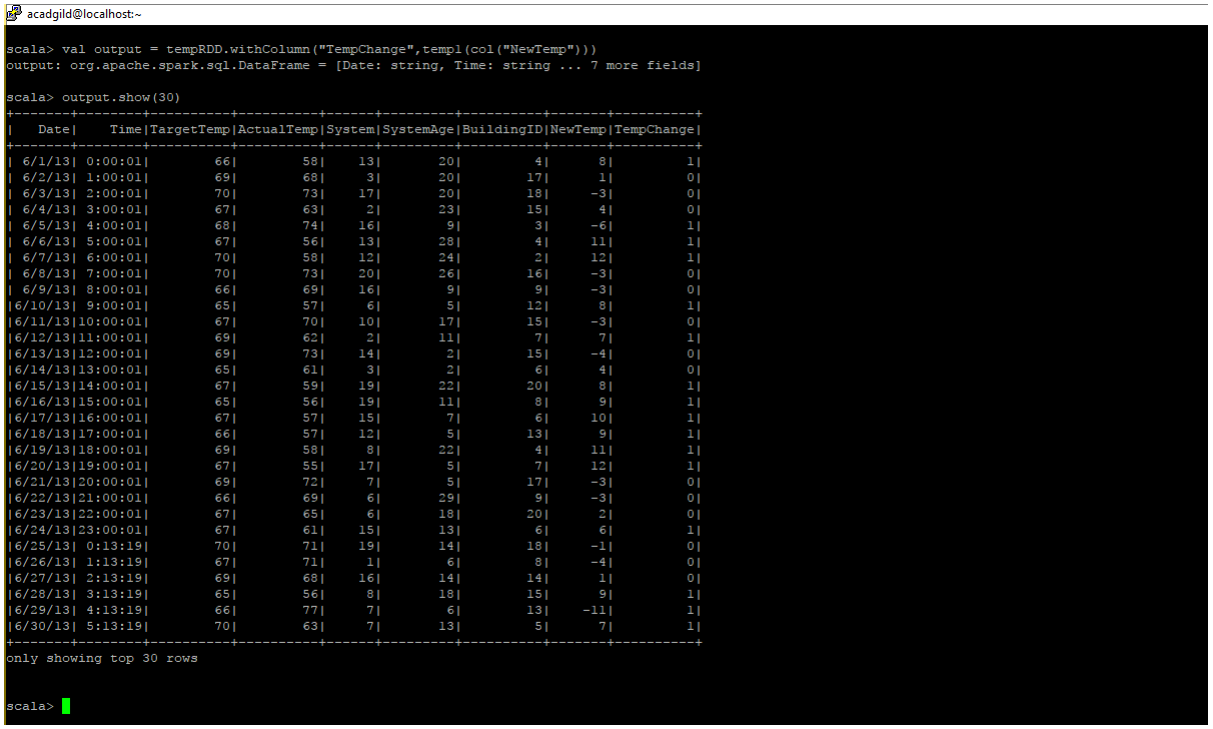
```
acadgild@localhost:~
scala> val temp = udf((TargetTemp:Int,ActualTemp:Int) => (TargetTemp,ActualTemp)match
     | {
     | case(TargetTemp,ActualTemp) => {TargetTemp-ActualTemp}
     | })
temp: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<function2>,IntegerType,Some(List(IntegerType, IntegerType)))

scala>

scala> spark.udf.register("NewTemp",temp)
res4: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<function2>,IntegerType,Some(List(IntegerType, IntegerType)))

scala> val tempRDD =dataDF.withColumn("NewTemp",temp(dataDF.col("TargetTemp"),dataDF.col("ActualTemp"))).toDF
tempRDD: org.apache.spark.sql.DataFrame = [Date: string, Time: string ... 6 more fields]

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

Step 4:

//Adding New Column tempchange - set to 1, if there is a change of greater than +/-5 between actual and target temperature

```
val temp1 = udf{(NewTemp:Int) =>

if(NewTemp > 5 ) {1} else if(NewTemp > -5 ) {1} else {0}

}

val output = tempRDD.withColumn("TempChange",temp1(col("NewTemp")))

output.show()
```

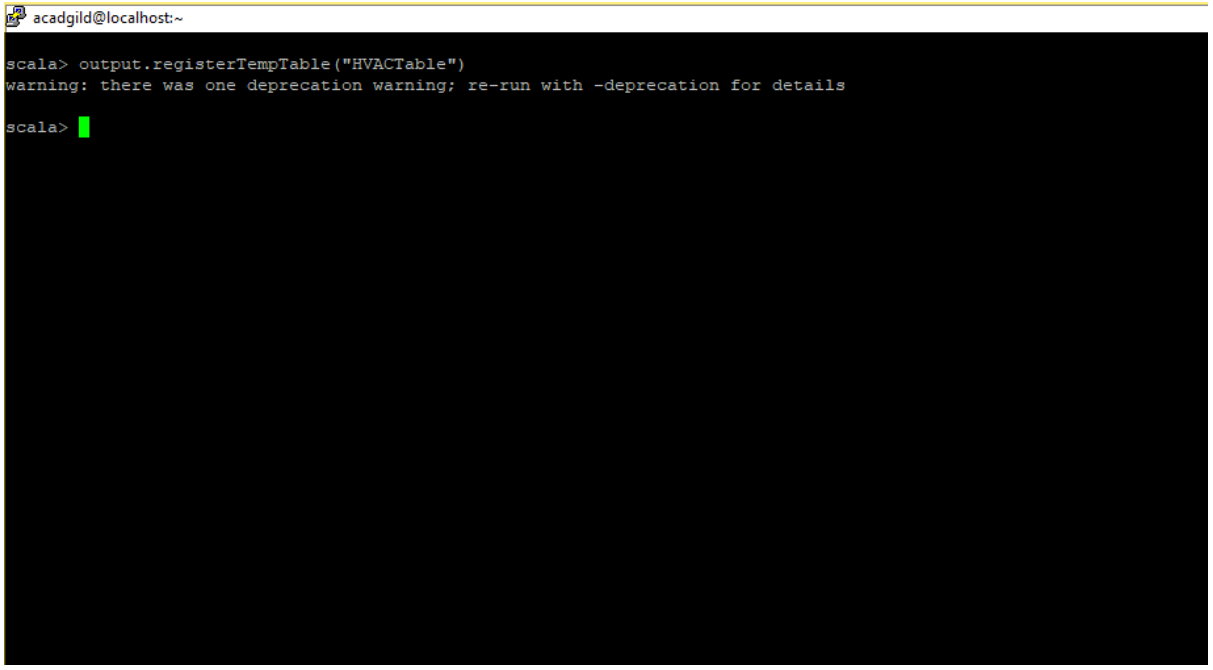
```
acadgild@localhost:~
scala> val temp1 = udf{(NewTemp:Int) =>
     | if(NewTemp > 5 ) {1} else if(NewTemp > -5 ) {1} else {0}
     | }
temp1: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<function1>,IntegerType,Some(List(IntegerType)))

scala>

scala> val output = tempRDD.withColumn("TempChange",temp1(col("NewTemp")))
```



Step 5:

//Loading HVAC.csv file into temporary table

output.registerTempTable("HVACTable")



Step 6:

val result1 = sqlContext.sql("select * from HVACTable")

result1.show()

```
scala> val result1 = sqlContext.sql("select * from HVACTable")
result1: org.apache.spark.sql.DataFrame = [Date: string, Time: string ... 7 more fields]

scala> result1.show()
+-------+--------+----------+----------+------+---------+----------+-------+----------+
|   Date|    Time|TargetTemp|ActualTemp|System|SystemAge|BuildingID|NewTemp|TempChange|
+-------+--------+----------+----------+------+---------+----------+-------+----------+
| 6/1/13| 0:00:01|        66|        58|    13|       20|         4|      8|         1|
| 6/2/13| 1:00:01|        69|        68|     3|       20|        17|      1|         0|
| 6/3/13| 2:00:01|        70|        73|    17|       20|        18|     -3|         0|
| 6/4/13| 3:00:01|        67|        63|     2|       23|        15|      4|         0|
| 6/5/13| 4:00:01|        68|        74|    16|        9|         3|     -6|         1|
| 6/6/13| 5:00:01|        67|        56|    13|       28|         4|     11|         1|
| 6/7/13| 6:00:01|        70|        58|    12|       24|         2|     12|         1|
| 6/8/13| 7:00:01|        70|        73|    20|       26|        16|     -3|         0|
| 6/9/13| 8:00:01|        66|        69|    16|        9|         9|     -3|         0|
|6/10/13| 9:00:01|        65|        57|     6|        5|        12|      8|         1|
|6/11/13|10:00:01|        67|        70|    10|       17|        15|     -3|         0|
|6/12/13|11:00:01|        69|        62|     2|       11|         7|      7|         1|
|6/13/13|12:00:01|        69|        73|    14|        2|        15|     -4|         0|
|6/14/13|13:00:01|        65|        61|     3|        2|         6|      4|         0|
|6/15/13|14:00:01|        67|        59|    19|       22|        20|      8|         1|
|6/16/13|15:00:01|        65|        56|    19|       11|         8|      9|         1|
|6/17/13|16:00:01|        67|        57|    15|        7|         6|     10|         1|
|6/18/13|17:00:01|        66|        57|    12|        5|        13|      9|         1|
|6/19/13|18:00:01|        69|        58|     8|       22|         4|     11|         1|
|6/20/13|19:00:01|        67|        55|    17|        5|         7|     12|         1|
+-------+--------+----------+----------+------+---------+----------+-------+----------+
only showing top 20 rows


scala>
```

Objective 2

Load building.csv file into temporary table

Step 1 :

//Load data

val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val data = sc.textFile("/user/acadgild/building.csv")

//Remove Header

val header = data.first()

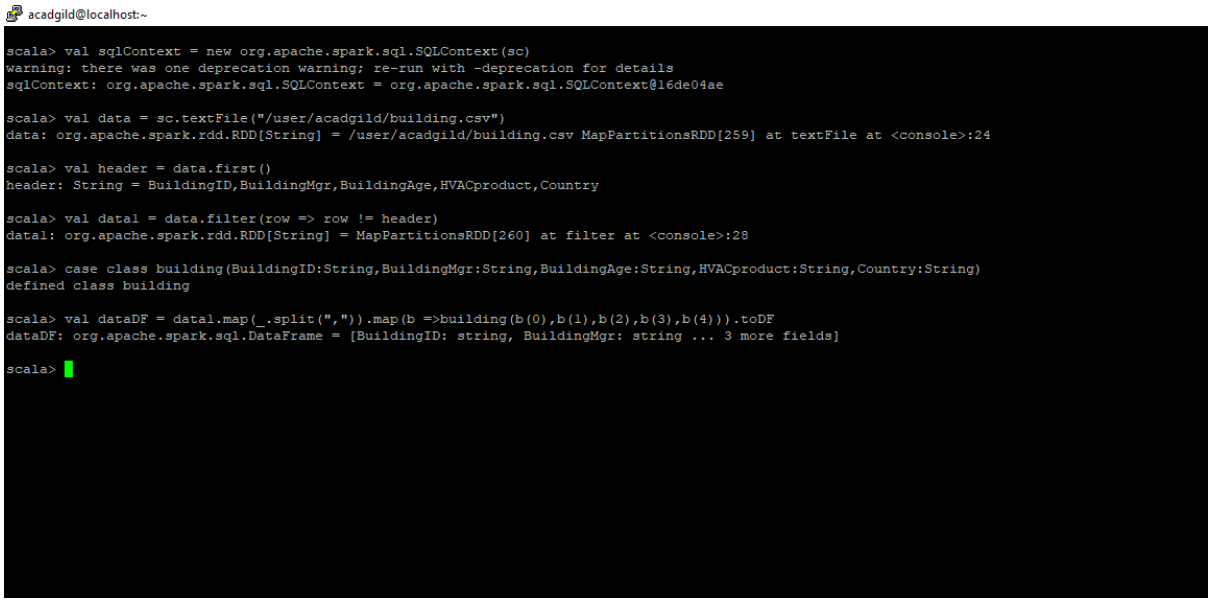
val data1 = data.filter(row => row != header)

//Define Building Case class

case class building(BuildingID:String,BuildingMgr:String,BuildingAge:String,HVACproduct:String,Country:String)

//convert to dataFrame

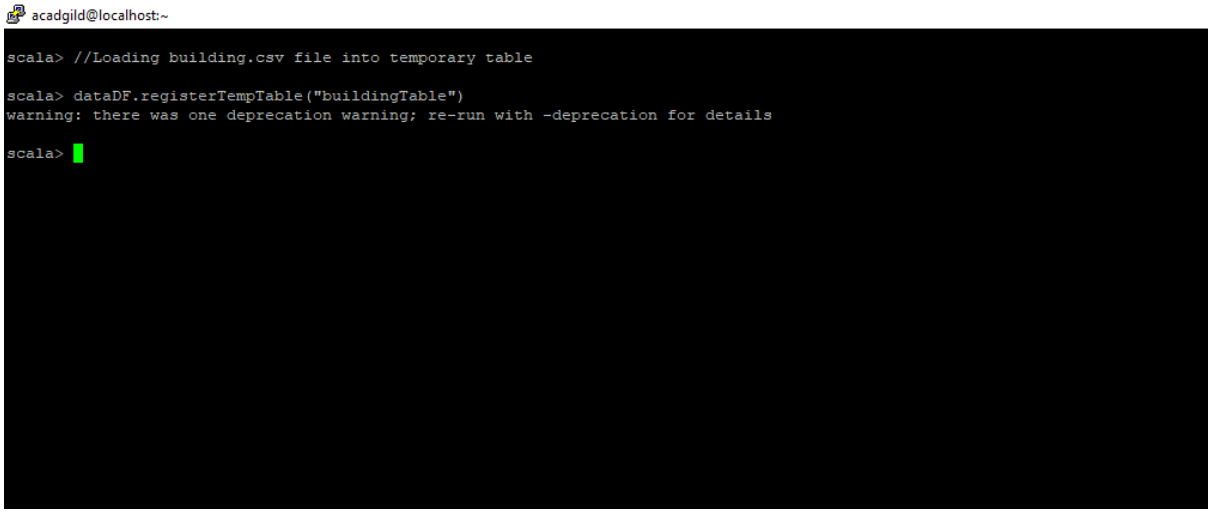
val dataDF = data1.map(_.split(",")).map(b=>building(b(0),b(1),b(2),b(3),b(4))).toDF



Step 2:

//Loading building.csv file into temporary table

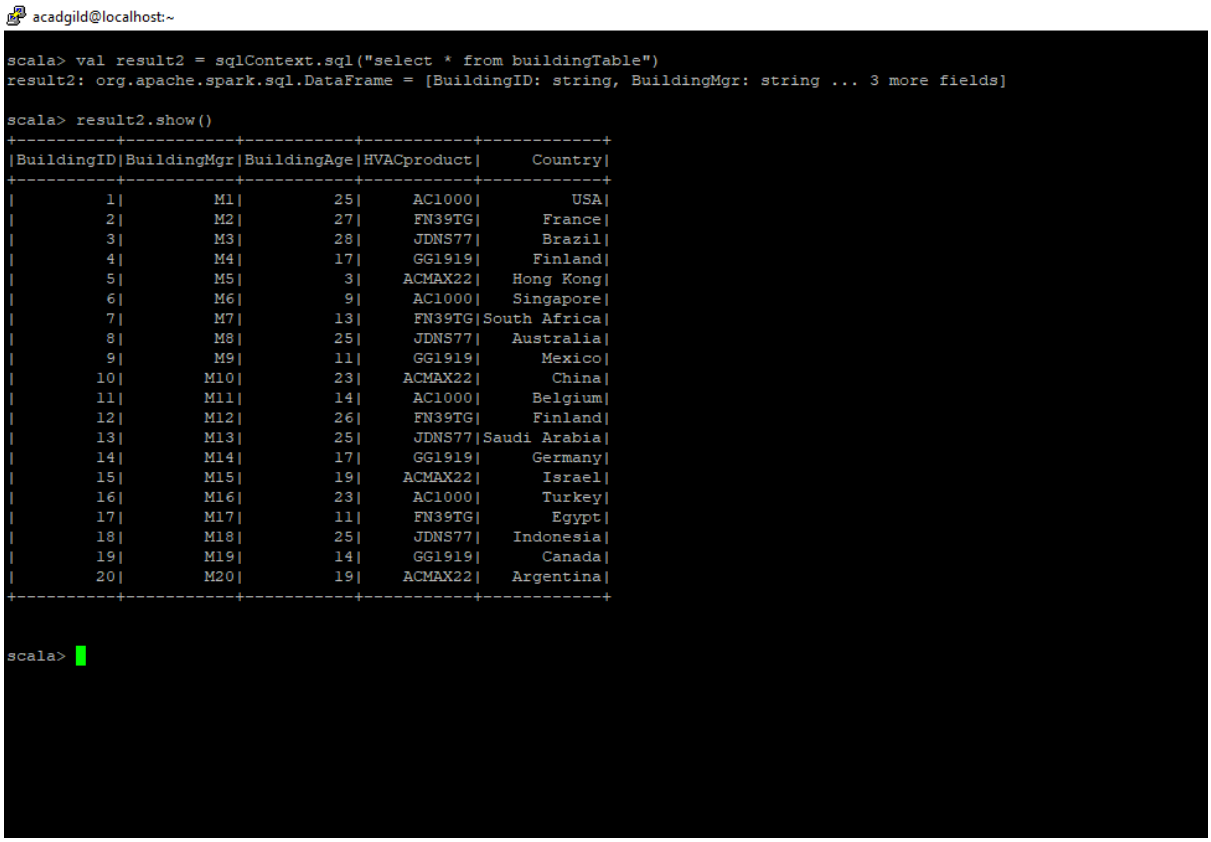
dataDF.registerTempTable("buildingTable")



Step 3:

val result2 = sqlContext.sql("select * from buildingTable")

result2.show()



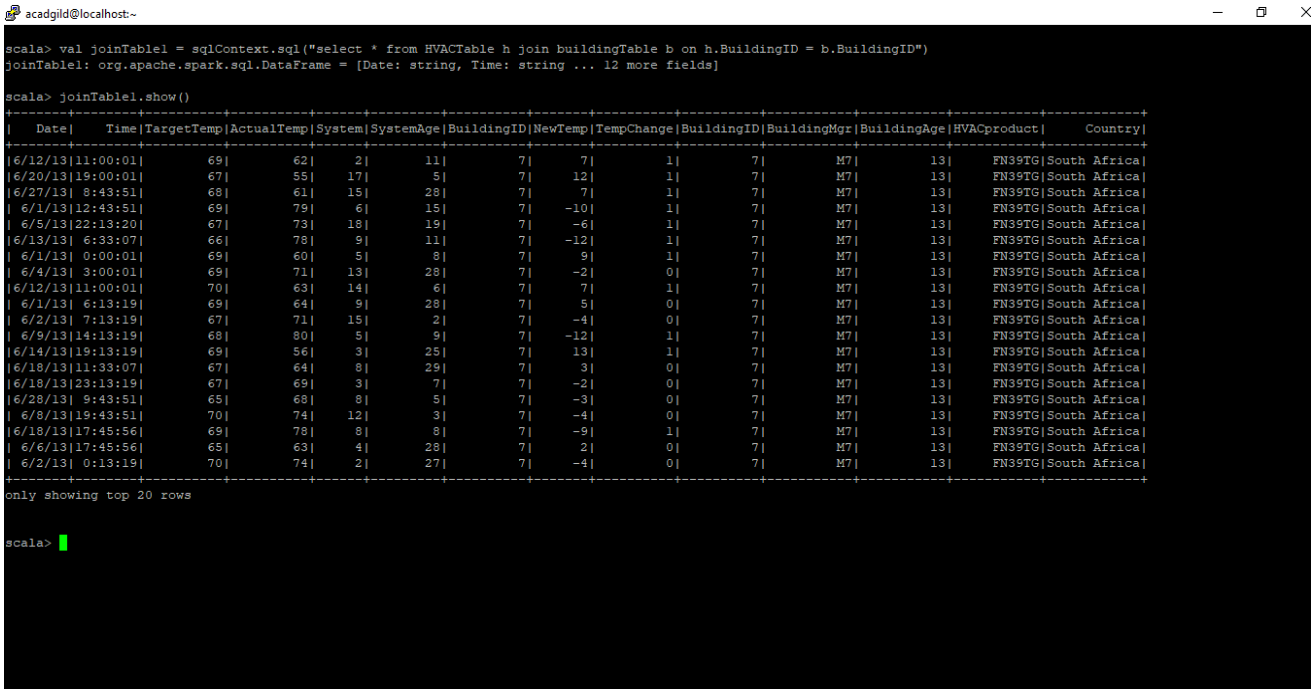
Objective 3:

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

1) Join both the tables.

val joinTable1 = sqlContext.sql("select * from HVACTable h join buildingTable b on h.BuildingID = b.BuildingID")

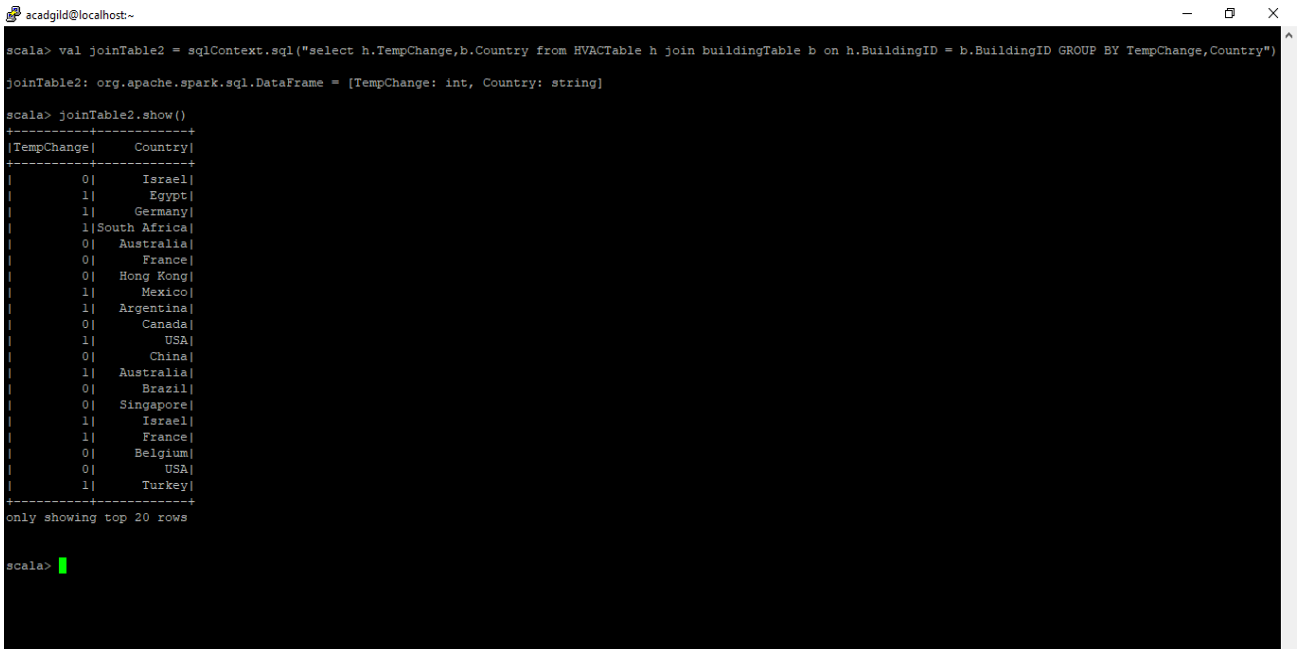
joinTable1.show()



2) Select tempchange and country column

val joinTable2 = sqlContext.sql("select h.TempChange,b.Country from HVACTable h join buildingTable b on h.BuildingID = b.BuildingID GROUP BY TempChange,Country")

joinTable2.show()



3) Filter the rows where tempchange is 1 and count the number of occurrence for each country

val joinTable3 = sqlContext.sql("select b.Country,Count(h.TempChange) as Count from HVACTable h join buildingTable b on h.BuildingID = b.BuildingID GROUP BY TempChange,Country HAVING TempChange = 1")

joinTable3.show()