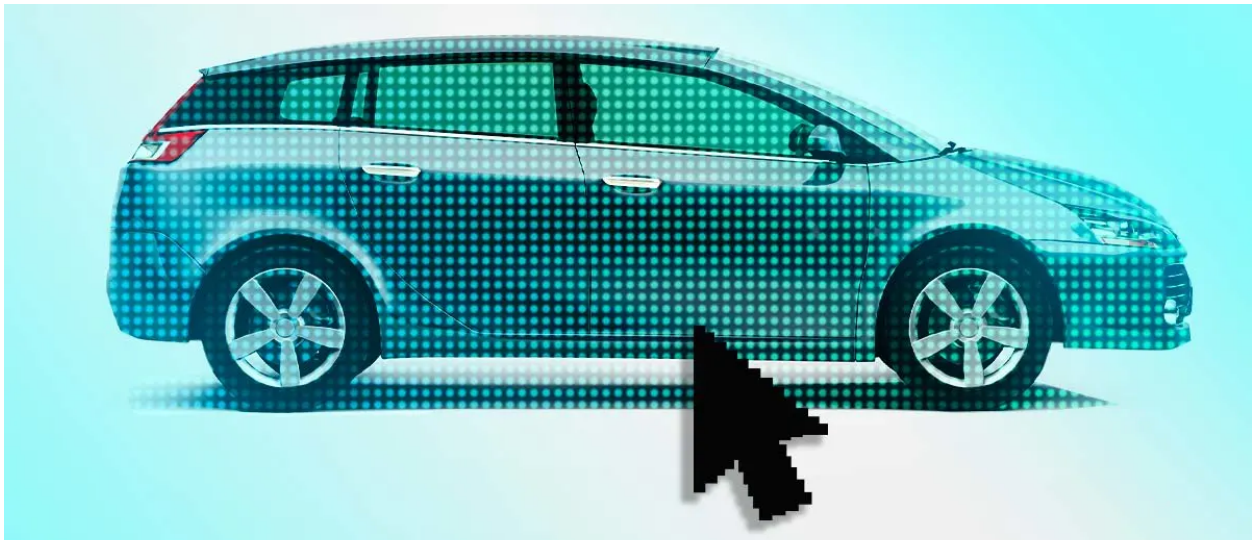
Spark Mini Project

Automobile post-sales report (Redesign)

Estimated Time: 3-5 hours



Note: For this project, we are leveraging Spark as the new engine to redesign the Hadoop auto post-sales report project that was previously done in MapReduce.

Consider an automobile tracking platform that keeps track of history of incidents after a new vehicle is sold by the dealer. Such incidents include further private sales, repairs and accident reports. This provides a good reference for second hand buyers to understand the vehicles they are interested in.

The same dataset of a history report of various vehicles is provided. Your goal is to write a Spark job to produce a report of the total number of accidents per make and year of the car.

The report is stored as CSV files in HDFS with the following schema.

Column	Type
incident_id	INT
incident_type	STRING (I: initial sale, A: accident, R: repair)
vin_number	STRING
make	STRING (The brand of the car, only populated with incident type “I”)
model	STRING (The model of the car, only populated with incident type “I”)
year	STRING (The year of the car, only populated with incident type “I”)
Incident_date	DATE (Date of the incident occurrence)

description	STRING

Learning Objectives

With this mini project, you will exercise using Spark transformations to solve traditional MapReduce data problems. It demonstrates Spark having a significant advantage against Hadoop MapReduce framework, in both code simplicity and its in-memory processing performance, which best suit for the chain of MapReduce use cases.

Hadoop Setup

Unless you have Hadoop setup elsewhere, we recommend using Hortonworks Hadoop Sandbox to run and test your code. This sandbox is a pre-configured virtual machine that has all necessary installation completed. You can follow the instructions from this video.

Similar setup can be done in MacOs and Windows systems.

As you setup Hadoop, add this data.csv into your file structure where you will be able to access it in the steps below.

Instructions

Step 1. Filter out accident incidents with make and year
Since we only care about accident records, we should filter out records having incident type other than “I”. However, accident records don't carry make and year fields due to the design to remove redundancy. So our first step is propagating make and year info from record type I into all other record types.

1.1 Read the input data CSV file
Use the Spark context object to read the input file to create the input RDD.

```
sc = SparkContext("local", "My Application")
raw_rdd = sc.textFile("data.csv")
```

1.1 Perform map operation
We need to propagate make and year to the accident records (incident type A), using vin_number as the aggregate key. Therefore the map output key should be vin_number, value should be the make and year, along with the incident type. In Spark, in order to proceed with the “groupByKey” function, we need the map operation to produce PairRDD, with tuple type as each record.

```
vin_kv = raw_rdd.map(lambda x: extract_vin_key_value(x))
# Please implement method extract_vin_key_value()
```

1.2 Perform group aggregation to populate make and year to all the records

Like the reducer in MapReduce framework, Spark provides a “groupByKey” function to achieve shuffle and sort in order to aggregate all records sharing the same key to the same groups. Within a group of vin_number, we need to iterate through all the records and find the one that has the make and year available and capture it in group level master info. As we filter and output accident records, those records need to be modified adding the master info that we captured in the first iteration.

```
enhance_make = vin_kv.groupByKey().flatMap(lambda kv: populate_make(kv[1]))
# Please implement method populate_make()
```

Step 2. Count number of occurrence for accidents for the vehicle make and year

2.1 Perform map operation

The goal of this step is to count the number of records for each make and year combination, given the result we derived previously. The output key should be the combination of vehicle make and year. The value should be the count of 1.

```
make_kv = enhance_make.map(lambda x: extract_make_key_value(x))
# Please implement method extract_make_key_value()
```

2.2 Aggregate the key and count the number of records in total per key

Use Spark provided “reduceByKey” function to perform the sum of all the values (1) from each record. As a result, we get the make and year combination key along with its total record count.

Step 3. Save the result to HDFS as text

The output file should look similar to this.

```
Nissan-2003,1
BMW-2008,10
MERCEDES-2013,2
```

Step 4. Shell script to run the Spark jobs

To run your Spark code, we need to use command “spark-submit” to start a Spark application and specify your Python script. Suppose your script is named autoinc_spark.py, your shell script should look like this.

```
spark-submit autoinc_spark.py
```

Instruction for Submission:

- Push the Python code and shell script to github.
- Add a readme file to include steps to run your code and verify the result. Your mentor should be able to run it by following your instructions.
- Readings about readme file: Example 1, Example 2
- Attach the command line execution log for the successful job run. You can capture it in a text file.