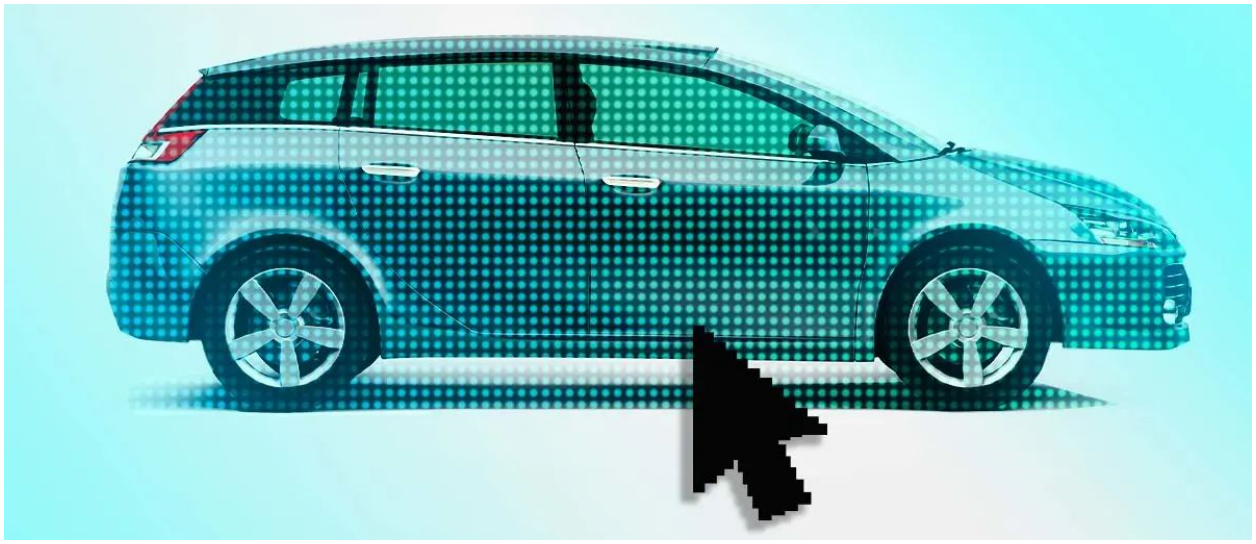
Hadoop Mini Project

Post-Sale Automobile Report

Estimated Time: 3-5 hours



In this project, you will utilize data from an automobile tracking platform that tracks the history of important incidents after the initial sale of a new vehicle. Such incidents include subsequent private sales, repairs, and accident reports. The platform provides a good reference for second-hand buyers to understand the vehicles they are interested in.

In this project, you will receive a dataset with a history report of various vehicles. Your goal is to write a MapReduce program 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 utilize MapReduce jobs in Python to create a solution for a real-life data problem. This will strengthen your mindset for leveraging a MapReduce processing model to process large scale data, and your capability to break down a complex problem into smaller tasks.

Hadoop Setup

Unless you already 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.

The setup is similar for both 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 steps 3 & 4 below.

Instructions

1. Filter out accidents with make and year
Since we only care about accident records, we should filter out records having Incident Type I and R. However, accident records don't carry the make and year fields as they're designed to remove redundancy. Your first step is propagating make and year info from Incident Type I into all other record types.

1.1 Create the mapper
Follow the mapper template to create a mapper that reads the input data and propagates the data you want. To propagate make and year to the accident records (Incident Type A), you need to use vin_number as the aggregate key. Therefore, the mapper output key should be vin_number, and the value should be the incident type, make, and year.

```
#!/usr/bin/env python
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # [derive mapper output key values]
    print '%s\t%s' % (key, value)
```

1.2 Create the reducer

Follow the reducer template to create a reducer that reads the mapper output. Within a vin_number group, you must iterate through all the records to find the one that has the make and year available. When you do, capture it in your group-level master info. As you filter and output the accident records, those records must be modified by adding the master info that you captured in the first iteration.

```
#!/usr/bin/env python

import sys

# [Define group level master information]

def reset():
    # [Logic to reset master info for every new group]

# Run for end of every group
def flush():
    # [Write the output]

# input comes from STDIN
for line in sys.stdin:

    # [parse the input we got from mapper and update the master info]

    # [detect key changes]
    if current_vin != vin:
        if current_vin != None:
            # write result to STDOUT
            flush()
        reset()

    # [update more master info after the key change handling]

    current_vin = vin
# do not forget to output the last group if needed!
flush()
```

2. Count number of accident occurrences for the vehicle make and year

2.1 Create the mapper

Follow the previous mapper template to create a mapper that reads the previous reducer output. The output key should be the composite key made up of the concatenation of vehicle make and year. The value should be the count of 1.

2.2 Create reducer

Follow the previous reducer template to create a reducer that reads the mapper output. Within the group of make and year, sum all the 1s to produce the total count as the output. The output key should also be the combination of make and year.

3. Test the MapReduce jobs using bash pipeline

You can use the Bash pipeline function to simulate what happens in MapReduce. This will not work in distributed mode, but it can be used to test your functionality.

```
cat data.csv | autoinc_mapper1.py | sort | autoinc_reducer1.py | autoinc_mapper2.py | sort |
autoinc_reducer2.py
```

Step 4. Write a shell script to run the two MapReduce jobs

To run your MapReduce code in the real distributed mode, you need to use the Hadoop streaming library to call the mappers and reducers. Create a shell script to call these two MapReduce jobs in sequence

```
hadoop jar /usr/local/hadoop/contrib/streaming/hadoop-*streaming*.jar \
-file autoinc_mapper1.py -mapper autoinc_mapper1.py \
-file autoinc_reducer1.py -reducer autoinc_reducer1.py \
-input input/data.csv -output output/all_accidents

hadoop jar /usr/local/hadoop/contrib/streaming/hadoop-*streaming*.jar \
-file autoinc_mapper2.py -mapper autoinc_mapper2.py \
-file autoinc_reducer2.py -reducer autoinc_reducer2.py \
-input output/all_accidents -output output/make_year_count
```

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.