

US Road Accidents (2016 - 2020)

Data Analysis using Apache Hadoop

ENGINEERING BIG DATA SYSTEMS

INFO 7250 (SUMMER 2020)



**Northeastern
University**

Christy Joseph Anoop
NUID: 001465082

TABLE OF CONTENTS

<i>Topics</i>	<i>Page Number</i>
Overview about the dataset	3
Analysis using MapReduce on Hadoop	6
Automation using Bash Script	19
MapReduce on AWS EMR using AWS CLI Tool	21
Analysis using Apache PIG on Hadoop	25
Analysis using Apache HIVE on Hadoop	28
References	31
APPENDIX	32

OVERVIEW OF DATASET

This data set was downloaded from Kaggle: <https://www.kaggle.com/sobhanmoosavi/us-accidents/>

DESCRIPTION

This is a countrywide car accident dataset, which covers **49 states of the USA**. The accident data are collected from **February 2016 to June 2020**, using two APIs that provide streaming traffic incident (or event) data. These APIs broadcast traffic data captured by a variety of entities, such as the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road-networks. Currently, there are about **3.5 million** accident records in this dataset. Check [here](#) to learn more about this dataset.

ACKNOWLEDGEMENTS

Please cite the following papers if you use this dataset:

- Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, and Rajiv Ramnath. "[A Countrywide Traffic Accident Dataset](#).", 2019.
- Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, Radu Teodorescu, and Rajiv Ramnath. "[Accident Risk Prediction based on Heterogeneous Sparse Data: New Dataset and Insights](#)." In proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2019.

CONTENT

This dataset has been collected in real-time, using multiple Traffic APIs. Currently, it contains accident data that are collected from February 2016 to June 2020 for the Contiguous United States. Check [here](#) to learn more about this dataset.

INSPIRATION

US-Accidents can be used for numerous applications such as real-time car accident prediction, studying car accidents hotspot locations, casualty analysis and extracting cause and effect rules to predict car accidents, and studying the impact of precipitation or other environmental stimuli on accident occurrence.

USAGE POLICY AND LEGAL DISCLAIMER

This dataset is being distributed only for **Research** purposes, under Creative Commons Attribution-Noncommercial-ShareAlike license (CC BY-NC-SA 4.0). By clicking on download button(s) below, you are agreeing to use this data only for non-commercial, research, or academic applications. You may need to cite the above papers if you use this dataset.

COLUMNS

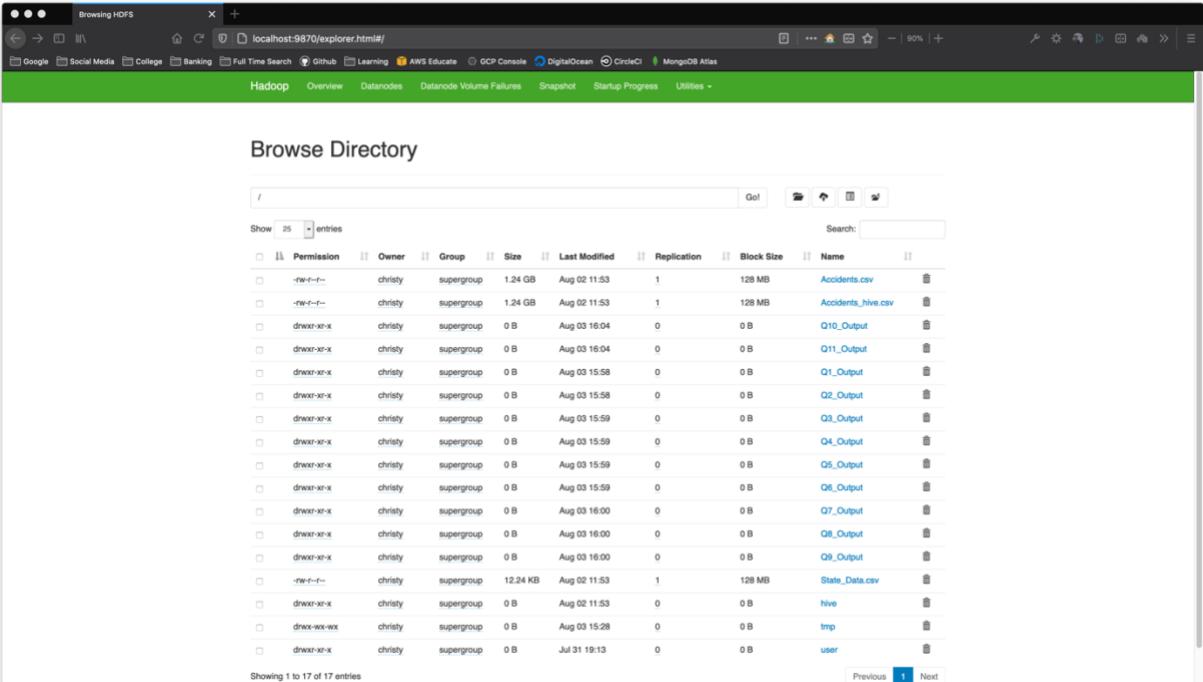
Column Name	Description
ID	This is a unique identifier of the accident record.
Source	Indicates source of the accident report (i.e. the API which reported the accident).
TMC	A traffic accident may have a Traffic Message Channel (TMC) code which provides more detailed description of the event.
Severity	Shows the severity of the accident, a number between 1 and 4, where 1 indicates the least impact on traffic (i.e., short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e., long delay).
Start_Time	Shows start time of the accident in local time zone.
End_Time	Shows end time of the accident in local time zone. End time here refers to when the impact of accident on traffic flow was dismissed.
Start_Lat	Shows latitude in GPS coordinate of the start point.
Start_Lng	Shows longitude in GPS coordinate of the start point.
End_Lat	Shows latitude in GPS coordinate of the end point.
End_Lng	Shows longitude in GPS coordinate of the end point.
Distance(mi)	The length of the road extent affected by the accident.
Description	Shows natural language description of the accident.
Number	Shows the street number in address record.
Street	Shows the street name in address record.
Side	Shows the relative side of the street (Right/Left) in address record.
City	Shows the city in address record.
County	Shows the county in address record.
State	Shows the state in address record.
Zipcode	Shows the zip code in address record.
Country	Shows the country in address record.
Timezone	Shows timezone based on the location of the accident (eastern, central, etc.).
Airport_Code	Denotes an airport-based weather station which is the closest one to location of the accident.

Weather_Timestamp	Shows the timestamp of weather observation record (in local time).
Temperature (F)	Shows the temperature (in Fahrenheit).
Wind_Chill (F)	Shows the wind chill (in Fahrenheit).
Humidity (%)	Shows the humidity (in percentage).
Pressure (in)	Shows the air pressure (in inches).
Visibility (mi)	Shows visibility (in miles).
Wind_Direction	Shows wind direction.
Wind_Speed (mph)	Shows wind speed (in miles per hour).
Precipitation (in)	Shows precipitation amount in inches, if there is any.
Weather_Condition	Shows the weather condition (rain, snow, thunderstorm, fog, etc.)
Amenity	A POI annotation which indicates presence of amenity in a nearby location.
Bump	A POI annotation which indicates presence of speed bump or hump in a nearby location.
Crossing	A POI annotation which indicates presence of crossing in a nearby location.
Give_Way	A POI annotation which indicates presence of give_way in a nearby location.
Junction	A POI annotation which indicates presence of junction in a nearby location.
No_Exit	A POI annotation which indicates presence of no_exit in a nearby location.
Railway	A POI annotation which indicates presence of railway in a nearby location.
Roundabout	A POI annotation which indicates presence of roundabout in a nearby location.
Station	A POI annotation which indicates presence of station in a nearby location.
Stop	A POI annotation which indicates presence of stop in a nearby location.
Traffic_Calming	A POI annotation which indicates presence of traffic_calming in a nearby location.
Traffic_Signal	A POI annotation which indicates presence of traffic_signal in a nearby location.
Turning_Loop	A POI annotation which indicates presence of turning_loop in a nearby location.
Sunrise_Sunset	Shows the period of day (i.e. day or night) based on sunrise/sunset.
Civil_Twilight	Shows the period of day (i.e. day or night) based on civil twilight.
Nautical_Twilight	Shows the period of day (i.e. day or night) based on nautical twilight.
Astronomical_Twilight	Shows the period of day (i.e. day or night) based on astronomical twilight.

ANALYSIS OF DATA USING MAPREDUCE

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly available service on top of a cluster of computers, each of which may be prone to failures.



A screenshot of a web browser displaying the HDFS (Hadoop Distributed File System) browser interface. The URL is `localhost:9870/explorer.html#/`. The page title is "Browsing HDFS". The main content area is titled "Browse Directory" and shows a list of entries under the root directory "/". The columns in the table include: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. The data table contains 17 entries, mostly named "Qn_Output" where n is a number from 1 to 10, and other entries like "Accidents.csv", "Accidents_hive.csv", "State_Data.csv", "hive", "tmp", and "user". The "Size" column shows values such as 1.24 GB, 12.24 KB, and 0 B. The "Last Modified" column shows dates like Aug 02 11:53 and Jul 31 19:13. The "Replication" column shows values like 1 and 0. The "Block Size" column shows values like 128 MB and 0 B. The "Name" column lists the file or directory names. At the bottom of the table, it says "Showing 1 to 17 of 17 entries".

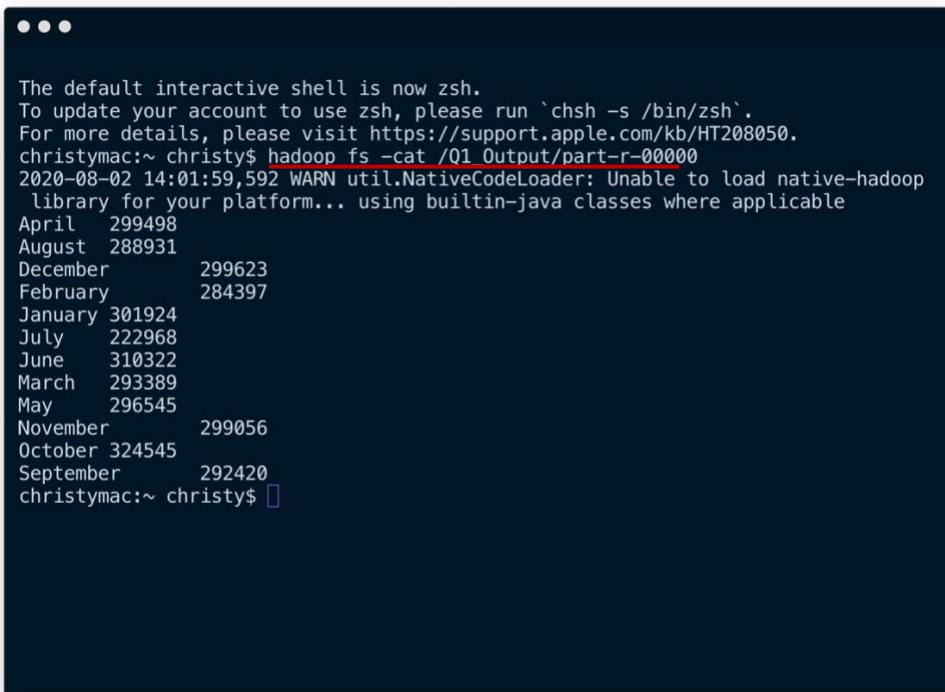
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	christy	supergroup	1.24 GB	Aug 02 11:53	1	128 MB	Accidents.csv
-rw-r--r--	christy	supergroup	1.24 GB	Aug 02 11:53	1	128 MB	Accidents_hive.csv
drwxr-xr-x	christy	supergroup	0 B	Aug 03 16:04	0	0 B	Q10_Output
drwxr-xr-x	christy	supergroup	0 B	Aug 03 16:04	0	0 B	Q11_Output
drwxr-xr-x	christy	supergroup	0 B	Aug 03 15:58	0	0 B	Q1_Output
drwxr-xr-x	christy	supergroup	0 B	Aug 03 15:58	0	0 B	Q2_Output
drwxr-xr-x	christy	supergroup	0 B	Aug 03 15:59	0	0 B	Q3_Output
drwxr-xr-x	christy	supergroup	0 B	Aug 03 15:59	0	0 B	Q4_Output
drwxr-xr-x	christy	supergroup	0 B	Aug 03 15:59	0	0 B	Q5_Output
drwxr-xr-x	christy	supergroup	0 B	Aug 03 15:59	0	0 B	Q6_Output
drwxr-xr-x	christy	supergroup	0 B	Aug 03 16:00	0	0 B	Q7_Output
drwxr-xr-x	christy	supergroup	0 B	Aug 03 16:00	0	0 B	Q8_Output
drwxr-xr-x	christy	supergroup	0 B	Aug 03 16:00	0	0 B	Q9_Output
-rw-r--r--	christy	supergroup	12.24 KB	Aug 02 11:53	1	128 MB	State_Data.csv
drwxr-xr-x	christy	supergroup	0 B	Aug 02 11:53	0	0 B	hive
drwxrwxrwt	christy	supergroup	0 B	Aug 03 15:28	0	0 B	tmp
drwxr-xr-x	christy	supergroup	0 B	Jul 31 19:13	0	0 B	user

I. NUMBER OF ACCIDENTS PER MONTH

A Simple MapReduce Program to retrieve the number of accidents per month.

```
# Number of Accidents Per Month
hadoop jar
~/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-
SNAPSHOT.jar com.hadoop.finalProject.Q1.DriverClass /Accidents.csv
/Q1_Output
```

Based on the analysis we can conclude that majority of the accidents happen in the months of October (during Fall), June (Summer) and January (Winter) throughout the US.



The terminal window shows the output of a Hadoop MapReduce job. It starts with a warning about the default interactive shell being zsh and provides instructions to switch to zsh. The main output is a list of months and their corresponding accident counts from a file named 'Q1_Output/part-r-00000'. The data is as follows:

Month	Accidents
April	299498
August	288931
December	299623
February	284397
January	301924
July	222968
June	310322
March	293389
May	296545
November	299056
October	324545
September	292420

2. NUMBER OF ACCIDENTS VS HOUR OF DAY

This simple MapReduce job performs analysis to determine the number of accidents vs. the hour of day.

```
# Number of Accidents vs Hour of Day
hadoop jar
~/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-
SNAPSHOT.jar com.hadoop.finalProject.Q2.DriverClass /Accidents.csv
/Q2_Output
```

Based on the analysis we can conclude that majority of the accidents happen between **7:00 AM to 8:00 AM** and in the evening during **3:00 PM to 5:00 PM**. These are the times that people travel to and from their workplaces.

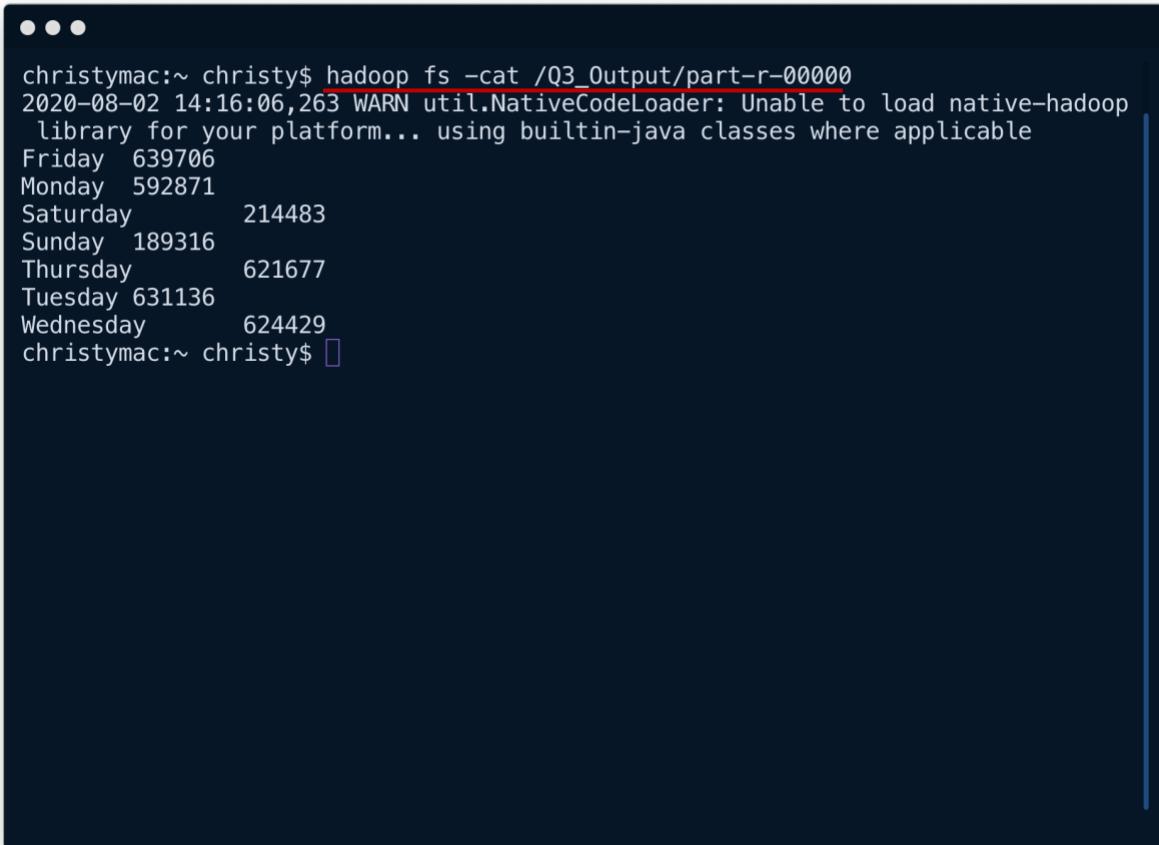
```
christymac:~ christy$ hadoop fs -cat /Q2_Output/part-r-00000
2020-08-02 14:05:24,215 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
00      29109
01      22401
02      23120
03      22702
04      64706
05      99244
06      198679
07      316352
08      326257
09      202664
10      182608
11      180849
12      163099
13      168260
14      180699
15      215566
16      256542
17      264893
18      201968
19      136253
20      96177
21      69771
22      57885
23      33814
```

3. NUMBER OF ACCIDENTS PER WEEKDAY

This simple MapReduce job performs analysis to determine the number of accidents per weekday

```
# Number of Accidents Per Weekday
hadoop jar
~/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-
SNAPSHOT.jar com.hadoop.finalProject.Q3.DriverClass /Accidents.csv
/Q3_Output
```

Based on the analysis we can conclude that majority of the accidents occur during weekdays. And it seems like people prefer to stay at home during weekends.

A screenshot of a terminal window on a Mac OS X system. The title bar shows three dots. The terminal displays the command "hadoop fs -cat /Q3_Output/part-r-00000" followed by a warning message about the NativeCodeLoader. Below the warning, a table of accident counts by day of the week is shown. The data is as follows:

Day	Count
Friday	639706
Monday	592871
Saturday	214483
Sunday	189316
Thursday	621677
Tuesday	631136
Wednesday	624429

The command "christymac:~ christy\$" is visible at the bottom of the terminal window.

```
christymac:~ christy$ hadoop fs -cat /Q3_Output/part-r-00000
2020-08-02 14:16:06,263 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
Friday 639706
Monday 592871
Saturday 214483
Sunday 189316
Thursday 621677
Tuesday 631136
Wednesday 624429
christymac:~ christy$
```

4. PERCENTAGE OF ACCIDENTS PER SIDE OF THE ROAD

This simple MapReduce **Numerical Summarization** job performs analysis to determine the percentage of accidents per side of the road

```
# Percentage of Accidents per side of the road  
hadoop jar  
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalP  
roject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q4.DriverClass  
/Accidents.csv /Q4_Output
```

Based on the analysis we can conclude that vast majority of the accidents happens on the right side of the road.

More details can be found here: https://www.picknbuy24.com/column_331.html

```
● ● ●  
christymac:~ christy$ hadoop fs -cat /Q4_Output/part-r-00000  
2020-08-02 14:20:47,275 WARN util.NativeCodeLoader: Unable to load native-hadoop  
library for your platform... using builtin-java classes where applicable  
Left    18.04%  
Right   81.96%  
christymac:~ christy$
```

5. NUMBER OF ACCIDENTS PER STATE

This simple MapReduce job performs analysis to determine the number of accidents per State.

```
# Accidents Per State
hadoop jar
~/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-
SNAPSHOT.jar com.hadoop.finalProject.Q5.DriverClass /Accidents.csv
/Q5_Output
```

Further Analysis done in the next part

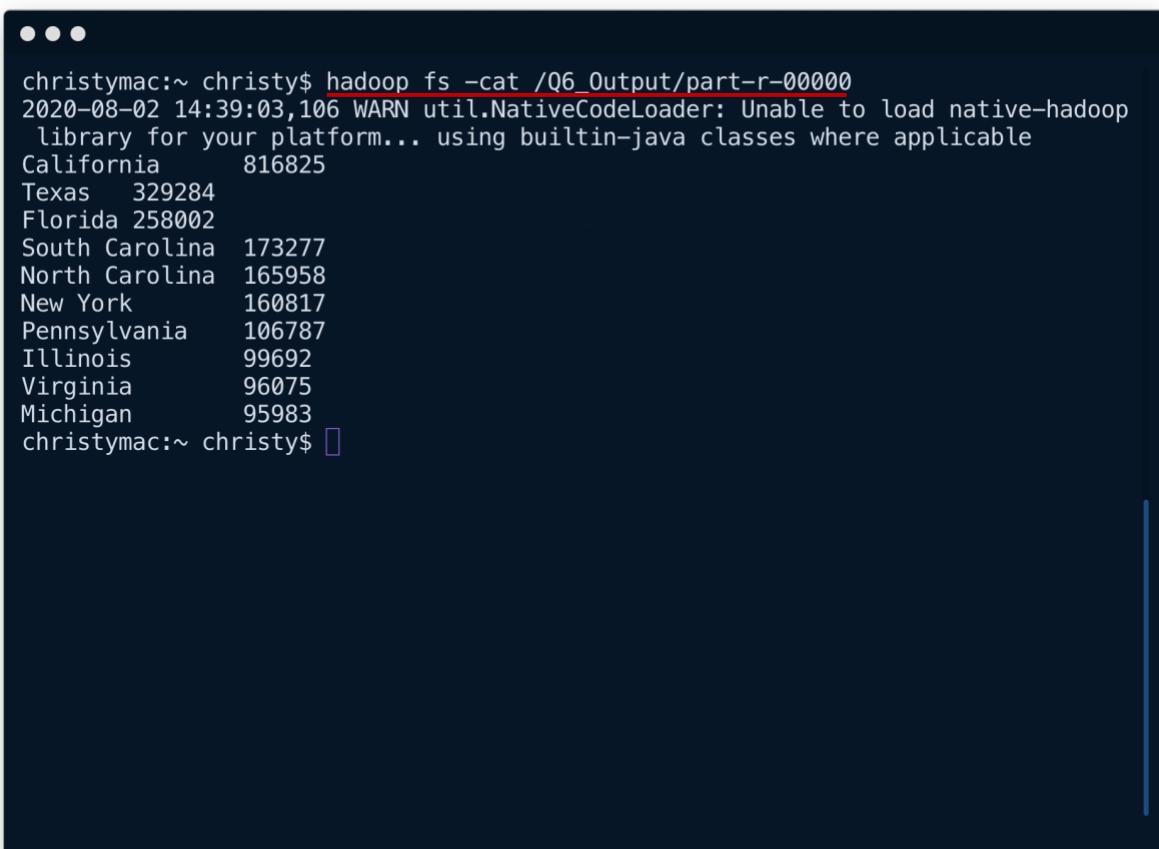
```
christymac:~ christy$ hadoop fs -cat /Q5_Output/part-r-00000 | head -n 20
2020-08-02 14:30:26,548 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
1
Alabama 44625
Arizona 78584
Arkansas 2012
California 816825
Colorado 49731
Connecticut 25901
Delaware 5739
District Of Columbia 4820
Florida 258002
Georgia 93614
Idaho 2044
Illinois 99692
Indiana 33746
Iowa 11475
Kansas 7939
Kentucky 22553
Louisiana 61515
Maine 2243
Maryland 53593
christymac:~ christy$
```

6. TOP 10 ACCIDENT PRONE STATES

This MapReduce Job uses **Top n Filtering Pattern** to filter the states from the previous job to find the states with the top 10 states with the greatest number of accidents.

```
# Top 10 Accident Prone States
hadoop jar
~/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-
SNAPSHOT.jar com.hadoop.finalProject.Q6.DriverClass /Q5_Output/part-r-
00000 /Q6_Output
```

Based on the analysis we can conclude that **California** leads the country in Road Accidents followed by few states in the south.



```
christymac:~ christy$ hadoop fs -cat /Q6_Output/part-r-00000
2020-08-02 14:39:03,106 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
California      816825
Texas        329284
Florida       258002
South Carolina 173277
North Carolina 165958
New York        160817
Pennsylvania    106787
Illinois        99692
Virginia         96075
Michigan         95983
christymac:~ christy$
```

7. CITIES WHERE DATA WAS RECORDED – INVERTED INDEX

This MapReduce Job uses ***Inverted Index Algorithm*** to find the cities per state (delimited by a `|`) where the accident data was collected.

```
# State - Cities = Inverted Index
hadoop jar
~/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-
SNAPSHOT.jar com.hadoop.finalProject.Q7.DriverClass /Accidents.csv
/Q7_Output
```

Based on the analysis we get all the cities per state where the accident data was collected. This allows us to perform an inverted index search on the data set to get very specific city specific data. (Data below shows the cities in Alabama)

```
christymac:~ christy$ hadoop fs -cat /Q7_Output/part-r-00000 | head -c 1600
2020-08-02 14:49:48,864 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
|
AL Ethelsville | Carrollton | Valhermoso Springs | Lynn | Fayette | Fort Da
vis | Prichard | Bryant | Slocomb | Arab | Steele | Union Springs | Quinton | Ho
llywood | Gulf Shores | Moody | Springville | Vinemont | Dora | Fort Mitchell |
Reform | Meridianville | Chancellor | Wetumpka | Salem | Somerville | Red Level
| Elrod | Red Bay | Creola | Cedar Bluff | Horton | Shorter | Cuba | Delta | Pel
ham | Stapleton | Hayneville | Mathews | Bayou la Batre | Cullman | Mc Calla | C
ollinsville | Sumiton | Childersburg | Lisman | Ashford | Goodwater | Jackson |
Perdido | Lester | Pleasant Grove | Daleville | Foley | Alexandria | Randolph |
Jacksonville | Sweet Water | Semmes | Taylor | Buhl | Letohatchee | Columbia | S
heffield | Shelby | Rogersville | Eufaula | Ragland | Pansey | Higdon | Bayou La
Batre | Bremen | Fultondale | Eutaw | Flomaton | Haleyville | Alabaster | Grant
| Jemison | Moundville | Fitzpatrick | Vernon | Chunchula | Lillian | Wedowee |
Lafayette | Tuscaloosa | Forest Home | Ozark | Pell City | Shorterville | Clant
on | Brantley | Echola | Dozier | Lincoln | Elkmont | Wellington | Jacksons Gap
| Paint Rock | Marbury | Frisco City | Mc Kenzie | Millport | Berry | Seale | Al
pine | Clayton | Troy | Pike Road | Fort Deposit | Town Creek | Abbeville | Newt
on | Gainesville | Theodore | Rockford | Calera | Elberta | Thomasville | Parris
h | Cropwell | Silverhill | New Brockton | Bankston | Empire | Brilliant | Kille
n | Mount Hope | Sylvania | Billingsley | Double Springs | Gaylesville | Mobile
| Hanceville | Alexander City | Enterprise | Boaz | Ariton | Ranburne | Ashville
cat: Unable to write to output stream.
christymac:~ christy$
```

8. AVERAGE, MIN AND MAX TEMPERATURE (F) PER SERVERITY

This MapReduce **Numerical Summarization** job performs analysis to determine the minimum, maximum and average temperature (F) per severity of accident. (Severity of an accident ranges from 1 to 4)

```
# Average, Min and Max Temperature per Severity
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProj
ect-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q8.DriverClass /Accidents.csv
/Q8_Output
```

Based on the analysis we can see that the minimum temperature per severity is quite low (maybe due to snow), but the average temp is normal which means that majority of the accidents happens during clear weather.

```
...
christymac:~ christy$ hadoop fs -cat /Q8_Output/part-r-00000
2020-08-02 14:57:08,894 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform.
.. using builtin-java classes where applicable
1      Max Temperature = 111.0 | Min Temperature = 3.9 | Average Temperature = 70.74177685233377
2      Max Temperature = 170.6 | Min Temperature = -89.0 | Average Temperature = 61.994931063614594
3      Max Temperature = 167.0 | Min Temperature = -89.0 | Average Temperature = 61.85957242491215
4      Max Temperature = 117.0 | Min Temperature = -40.0 | Average Temperature = 59.02189844306035
christymac:~ christy$
```

9. COUNT OF ACCIDENTS PER STATE PER YEAR – SECONDARY SORTED + PARTITIONED

This MapReduce *Secondary Sorting* job uses finds the number of accidents per state per job **partitioned** per year. Since the data was collected from 2016 to 2020 (5 years), we have 5 partitions.

```
# Count of Accidents Per State Per Year (SecondarySorted with 5
Partitions - Per Year)
hadoop jar
~/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-
1.0-SNAPSHOT.jar com.hadoop.finalProject.Q9.DriverClass /Accidents.csv
/Q9_Output
```

PARTITIONS:

```
christymac:~ christy$ hadoop fs -ls /Q9_Output
2020-08-02 15:04:12,281 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform.
.. using builtin-java classes where applicable
Found 6 items
-rw-r--r-- 1 christy supergroup      0 2020-08-02 11:56 /Q9_Output/_SUCCESS
-rw-r--r-- 1 christy supergroup  1702 2020-08-02 11:56 /Q9_Output/part-r-00000
-rw-r--r-- 1 christy supergroup  1687 2020-08-02 11:56 /Q9_Output/part-r-00001
-rw-r--r-- 1 christy supergroup  1710 2020-08-02 11:56 /Q9_Output/part-r-00002
-rw-r--r-- 1 christy supergroup  1718 2020-08-02 11:56 /Q9_Output/part-r-00003
-rw-r--r-- 1 christy supergroup  1718 2020-08-02 11:56 /Q9_Output/part-r-00004
christymac:~ christy$
```

Below is the data for accidents per state for the year 2020;

```
christymac:~ christy$ hadoop fs -cat /Q9_Output/part-r-00000
2020-08-02 15:04:55,024 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform.
.. using builtin-java classes where applicable
Alabama - in Year - 2020 :      8251
Arkansas - in Year - 2020 :       263
Arizona - in Year - 2020 :     16257
California - in Year - 2020 :  153560
Colorado - in Year - 2020 :      9604
Connecticut - in Year - 2020 :    3097
District Of Columbia - in Year - 2020 :      1168
Delaware - in Year - 2020 :      1305
Florida - in Year - 2020 :    34251
Georgia - in Year - 2020 :      9990
Iowa - in Year - 2020 :        1129
Idaho - in Year - 2020 :        285
Illinois - in Year - 2020 :   13304
Indiana - in Year - 2020 :     3709
Kansas - in Year - 2020 :      1052
Kentucky - in Year - 2020 :     3429
Louisiana - in Year - 2020 :     9032
Massachusetts - in Year - 2020 :    6033
Maryland - in Year - 2020 :    10267
Maine - in Year - 2020 :        178
Michigan - in Year - 2020 :     7294
Minnesota - in Year - 2020 :   19131
Missouri - in Year - 2020 :     4629
Mississippi - in Year - 2020 :     625
```

10. PARTITIONING DATA PER STATE – INNER JOIN

This MapReduce **Partitioning** Job divides the large ~1.3 GB file into smaller files where each file holds data for a particular state. It also replaces the state abbreviation to the full state name using **Inner Join**. *The other .csv file holds abbreviation -> State Name data.*

	A	B
1	WA	Washington
2	MN	Minnesota
3	WA	Washington
4	CO	Colorado
5	DC	District of Columbia
6	LA	Louisiana
7	TX	Texas
8	TX	Texas
9	AL	Alabama
10	KS	Kansas
11	LA	Louisiana
12	AL	Alabama
13	AL	Alabama
14	GA	Georgia
15	MO	Missouri
16	CA	California
17	GA	Georgia
18	CA	California
19	TX	Texas
20	TX	Texas
21	NC	North Carolina

```
# Divide file into partitions divided by state
hadoop jar
~Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-
SNAPSHOT.jar com.hadoop.finalProject.Q10.DriverClass /Accidents.csv
/State_Data.csv /Q10_Output
```

50 PARTITIONS:

```
christymac:~ christy$ hadoop fs -ls /Q10_Output
2020-08-02 15:17:33,992 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform.
.. using builtin-java classes where applicable
Found 50 items
-rw-r--r-- 1 christy supergroup 0 2020-08-02 11:59 /Q10_Output/_SUCCESS
-rw-r--r-- 1 christy supergroup 71468316 2020-08-02 11:56 /Q10_Output/part-r-00000
-rw-r--r-- 1 christy supergroup 474142179 2020-08-02 11:56 /Q10_Output/part-r-00001
-rw-r--r-- 1 christy supergroup 6827876 2020-08-02 11:56 /Q10_Output/part-r-00002
-rw-r--r-- 1 christy supergroup 963868138 2020-08-02 11:56 /Q10_Output/part-r-00003
-rw-r--r-- 1 christy supergroup 0 2020-08-02 11:56 /Q10_Output/part-r-00004
-rw-r--r-- 1 christy supergroup 97654005 2020-08-02 11:56 /Q10_Output/part-r-00005
-rw-r--r-- 1 christy supergroup 453032310 2020-08-02 11:56 /Q10_Output/part-r-00006
-rw-r--r-- 1 christy supergroup 47050698 2020-08-02 11:56 /Q10_Output/part-r-00007
-rw-r--r-- 1 christy supergroup 0 2020-08-02 11:56 /Q10_Output/part-r-00008
-rw-r--r-- 1 christy supergroup 3136624 2020-08-02 11:56 /Q10_Output/part-r-00009
-rw-r--r-- 1 christy supergroup 0 2020-08-02 11:56 /Q10_Output/part-r-00010
-rw-r--r-- 1 christy supergroup 0 2020-08-02 11:56 /Q10_Output/part-r-00011
-rw-r--r-- 1 christy supergroup 0 2020-08-02 11:56 /Q10_Output/part-r-00012
-rw-r--r-- 1 christy supergroup 408730644 2020-08-02 11:56 /Q10_Output/part-r-00013
-rw-r--r-- 1 christy supergroup 385286594 2020-08-02 11:56 /Q10_Output/part-r-00014
-rw-r--r-- 1 christy supergroup 0 2020-08-02 11:56 /Q10_Output/part-r-00015
-rw-r--r-- 1 christy supergroup 38347249026 2020-08-02 11:57 /Q10_Output/part-r-00016
-rw-r--r-- 1 christy supergroup 2539994615 2020-08-02 11:57 /Q10_Output/part-r-00017
-rw-r--r-- 1 christy supergroup 88525620 2020-08-02 11:57 /Q10_Output/part-r-00018
-rw-r--r-- 1 christy supergroup 206231792 2020-08-02 11:57 /Q10_Output/part-r-00019
-rw-r--r-- 1 christy supergroup 16098539 2020-08-02 11:57 /Q10_Output/part-r-00020
```

Below is the file that holds only data for District of Columbia (DC);

```
christymac:~ christy$ hadoop fs -cat /Q10_Output/part-r-00000 | head -n 3
2020-08-02 15:19:47,143 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
District of Columbia A-3209871,Bing,,2,2018-11-30 00:56:19,2018-11-30 01:25:04,38.96
208,-77.08029,38.96102,-77.08031,0.073,At Military Rd NW - Overturned vehicle.,,4050.0,
Legation St NW,L,Washington,District of Columbia,DC,20015-2920,US,US/Eastern,KCGS,2018-
11-30 01:02:00,34.9,,59.0,30.07,10.0,Calm,,,Overcast,False,False,False,False,False,Fals
e,False,False,False,False,False,Night,Night,Night,Night
District of Columbia A-3277048,Bing,,2,2018-09-24 08:47:16,2018-09-24 09:16:57,38.88
7959,-77.02807,38.88948,-77.02806,0.105,At 12th St NW - Accident. Lane blocked.,,12th S
t NW,R,Washington,District of Columbia,DC,20004,US,US/Eastern,KDCA,2018-09-24 08:52:00,
64.9,,90.0,30.34,4.0,NE,10.4,0.01,Light Rain,False,True,False,False,False,Day,Day,Day
District of Columbia A-3277049,Bing,,2,2018-09-24 08:47:16,2018-09-24 09:16:57,38.88
757,-77.02795,38.8919,-77.02805,0.299,At Constitution Ave NW - Accident. Lane blocked.,,
12th St NW,R,Washington,District of Columbia,DC,20004,US,US/Eastern,KDCA,2018-09-24 08
:52:00,64.9,,90.0,30.34,4.0,NE,10.4,0.01,Light Rain,False,True,False,False,Day,Day,Day
cat: Unable to write to output stream.
christymac:~ christy$
```

Below is the file that holds only data for South Carolina;

```
christymac:~ christy$ hadoop fs -cat /Q10_Output/part-r-00024 | head -n 3
2020-08-02 15:21:10,248 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
South Carolina A-701917,MapQuest,201.0,2,2020-02-07 09:37:22,2020-02-07 11:07:16,34.16
8549,-79.84549,,,0.0,Accident on US-76 Palmetto St at Old Ebenezer Rd.,,1226.0,Old Ebene
zer Rd,L,Florence,Florence,SC,29501-5946,US,US/Eastern,KFL0,2020-02-07 09:53:00,53.0,53
.0,48.0,29.49,10.0,W,29.0,0.0,Fair / Windy,False,False,False,False,False,Day,Day,Day
South Carolina A-907631,MapQuest,201.0,2,2019-11-24 19:09:08,2019-11-24 20:59:26,33.41
4066,-80.667763,,,0.0,Accident on I-26 Eastbound at Exit 159 / SC-36 Homestead Rd.,,Hom
estead Rd,R,Bowman,Orangeburg,SC,29018,US,US/Eastern,KOGB,2019-11-24 18:53:00,49.0,49.0
,,77.0,29.65,10.0,CALM,0.0,0.0,Fair,False,False,False,False,False,Day,Day,Day
South Carolina A-907629,MapQuest,241.0,3,2019-11-24 15:57:05,2019-11-24 16:25:56,32.88
3392,-80.019562,,,0.0,Lane blocked due to accident on I-26 Westbound at Exits 212B 212B
-C 212C I-526 Exits 17 17A 17B,,,I-26 W,R,Charleston,Charleston,SC,29406,US,US/Eastern,
KCHS,2019-11-24 15:56:00,60.0,60.0,49.0,29.75,10.0,NW,10.0,0.0,Fair,False,False,Day,Day,Day
cat: Unable to write to output stream.
christymac:~ christy$
```

II. PERCENTAGE OF ACCIDENTS VS. PROXIMITY TO TRAFFIC OBJECT

This MapReduce Job finds the percentage of accidents vs. the proximity to traffic object. This **custom algorithm** uses a lot of columns from the data set to produce 2 columns in the end result. It also does simple calculations to find the percentage.

```
# Proximity to Traffic Object (Percentage / per all traffic)
hadoop jar
~Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-
SNAPSHOT.jar com.hadoop.finalProject.Q11.DriverClass /Accidents.csv
/Q11_Output
```

Based on the analysis we can conclude that majority of the accidents occur near the traffic signal followed by Crossings.

```
christymac:~ christy$ hadoop fs -cat /Q11_Output/part-r-00000
2020-08-02 15:29:27,871 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Amenity      1.20%
Bump         0.02%
Crossing     7.81%
Give_Way     0.27%
Junction    8.10%
No_Exit      0.12%
Railway      0.89%
Roundabout   0.01%
Station      2.00%
Stop         1.48%
Traffic_Calming  0.04%
Traffic_Signal  17.75%
christymac:~ christy$
```

AUTOMATION USING BASH SCRIPT

Everything from packaging the project to starting the Hadoop Cluster, importing files to HDFS, running MapReduce jobs and Hive/Pig Config were automated using a Bash Shell Script. By running the bash script, the entire MapReduce part of this project was executed in less than 10 minutes and the results were waiting in HDFS.

```
#!/bin/bash

# Generate JAR file
mvn package &&

#-----
# Start Hadoop
cd /usr/local/opt/hadoop/sbin &&
./start-dfs.sh

# Put files into HDFS
hadoop fs -put ~/Downloads/Accidents.csv / &&
hadoop fs -put ~/Downloads/State_Data.csv / &&

#-----
# Hive config
hadoop fs -mkdir /hive &&
hadoop fs -mkdir /hive/warehouse &&
hadoop fs -put ~/Downloads/Accidents_hive.csv / &&

#-----
# Number of Accidents Per Month
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q1.DriverClass
/Accidents.csv /Q1_Output &&

# Number of Accidents vs Hour of Day
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q2.DriverClass
/Accidents.csv /Q2_Output &&

# Number of Accidents Per Weekday
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q3.DriverClass
/Accidents.csv /Q3_Output &&

# Percentage of Accidents per side of the road
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q4.DriverClass
/Accidents.csv /Q4_Output &&
```

```

# Accidents Per State
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalP
roject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q5.DriverClass
/Accidents.csv /Q5_Output &&

# Top 10 Accident Prone States
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalP
roject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q6.DriverClass
/Q5_Output/part-r-00000 /Q6_Output &&

# State - Cities = Inverted Index
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalP
roject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q7.DriverClass
/Accidents.csv /Q7_Output &&

# Average, Min and Max Temperature per Severity
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalP
roject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q8.DriverClass
/Accidents.csv /Q8_Output &&

# Count of Accidents Per State Per Year (SecondarySorted with 5 Partitions
# - Per Year)
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalP
roject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q9.DriverClass
/Accidents.csv /Q9_Output &&

# Divide file into partitions divided by state
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalP
roject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q10.DriverClass
/Accidents.csv /State_Data.csv /Q10_Output &&

# Proximity to Traffic Object (Percentage / per all traffic)
hadoop jar
/Users/christy/Downloads/Projects/BigDataFinalProject/target/BigDataFinalP
roject-1.0-SNAPSHOT.jar com.hadoop.finalProject.Q11.DriverClass
/Accidents.csv /Q11_Output

```

MAPREDUCE ON AWS ELASTIC MAP REDUCE (EMR)

The MapReduce jobs can also be executed on an AWS EMR Cluster provided the Host machine has AWS CLI tool installed with the AWS credentials configured.

The below Bash script automatically executes the steps in the following order:

1. Package the project using maven to the latest changes made by the developer.
2. Uploads the jar file generated from the previous step to an S3 bucket (Make sure to variable values with respect to your project). The cluster will automatically be destroyed after all the MapReduce jobs are completed successfully.
3. Adds steps to run the MapReduce Jobs.

After the MapReduce jobs are finished, you can SSH into the Master NameNode from the EC2 instances to check the output from the jobs.

```
#!/bin/bash

# Package project to jar file
mvn package

# Variables (TODO: Change)
keyName="Lab2"
projectName="ebds-cluster"
bucketName="ebds-final-project"

# Upload jar file to S3 bucket
aws s3 cp
~/Downloads/Projects/BigDataFinalProject/target/BigDataFinalProject-1.0-
SNAPSHOT.jar s3://$bucketName/ebds-runnable.jar

# Create a new Cluster
clusterId=$(aws emr create-cluster \
--name $projectName \
--use-default-roles \
--release-label emr-5.30.1 \
--instance-count 3 \
--instance-type m5.xlarge \
--applications Name=Pig Name=Hive Name=Hadoop \
--ec2-attributes KeyName=$keyName \
--log-uri s3://$bucketName/logs \
--auto-terminate | awk '{print $2}')

# Add Steps to cluster
aws emr add-steps \
--cluster-id $clusterId \
--steps
Type=CUSTOM_JAR,Name=Q1_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q1.DriverClass,s3://$bucketName/
Accidents.csv,/Q1_Output \
Type=CUSTOM_JAR,Name=Q2_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
```

```

/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q2.DriverClass,s3://$bucketName/
Accidents.csv,/Q2_Output \

Type=CUSTOM_JAR,Name=Q3_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q3.DriverClass,s3://$bucketName/
Accidents.csv,/Q3_Output \

Type=CUSTOM_JAR,Name=Q4_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q4.DriverClass,s3://$bucketName/
Accidents.csv,/Q4_Output \

Type=CUSTOM_JAR,Name=Q5_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q5.DriverClass,s3://$bucketName/
Accidents.csv,/Q5_Output \

Type=CUSTOM_JAR,Name=Q6_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q6.DriverClass,s3://$bucketName/
Accidents.csv,/Q5_Output/part-r-00000,/Q6_Output \

Type=CUSTOM_JAR,Name=Q7_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q7.DriverClass,s3://$bucketName/
Accidents.csv,/Q7_Output \

Type=CUSTOM_JAR,Name=Q8_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q8.DriverClass,s3://$bucketName/
Accidents.csv,/Q8_Output \

Type=CUSTOM_JAR,Name=Q9_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q9.DriverClass,s3://$bucketName/
Accidents.csv,/Q9_Output \

Type=CUSTOM_JAR,Name=Q10_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q10.DriverClass,s3://$bucketName
/Accidents.csv,s3://$bucketName/State_Data.csv,/Q10_Output \

Type=CUSTOM_JAR,Name=Q11_Step,ActionOnFailure=CONTINUE,Jar=s3://$bucketName
/ebds-
runnable.jar,Args=com.hadoop.finalProject.Q11.DriverClass,s3://$bucketName
/Accidents.csv,/Q11_Output

```

S3 bucket with files uploaded:

Name	Last modified	Size	Storage class
Q1_Output	--	--	--
hive_output	--	--	--
logs	--	--	--
Accidents.csv	Jul 30, 2020 10:26:15 AM GMT-0400	1.2 GB	Standard
Accidents_hive.csv	Jul 30, 2020 10:26:15 AM GMT-0400	1.2 GB	Standard
BigDataFinalProject-1.0-SNAPSHOT.jar	Jul 30, 2020 3:00:09 PM GMT-0400	58.9 KB	Standard
State_Data.csv	Jul 30, 2020 10:23:14 AM GMT-0400	12.2 KB	Standard
ebds-runnable.jar	Aug 3, 2020 4:27:41 PM GMT-0400	59.5 KB	Standard
queries.hive	Jul 30, 2020 11:26:18 AM GMT-0400	2.0 KB	Standard

AWS Provisioning an EMR Cluster:

EMR MapReduce Jobs Queued:

The screenshot shows the AWS EMR Cluster Details page for a cluster named 'ebds-cluster'. The status is 'Starting' and it is 'Configuring cluster software'. There is one change in concurrency. The 'Steps' tab is selected, showing 11 pending steps. Each step has a green circular icon, an ID, a name, a status of 'Pending', a start time of '...', an elapsed time of '...', and a 'View logs' link. A tooltip for each row indicates 'Debugging not configured'. The table has columns for ID, Name, Status, Start time (UTC-4), Elapsed time, and Log files.

ID	Name	Status	Start time (UTC-4)	Elapsed time	Log files
s-2ZG23E1X08D7H	Q1_Step	Pending	View logs Debugging not configured
s-3LSQ8M9RR5YG2	Q10_Step	Pending	View logs Debugging not configured
s-41PRBZ22HFEM	Q9_Step	Pending	View logs Debugging not configured
s-1X2JMVY1SMV15	Q8_Step	Pending	View logs Debugging not configured
s-k9DIWQ9VVN9M	Q7_Step	Pending	View logs Debugging not configured
s-2KBV5I7D1LW9E	Q6_Step	Pending	View logs Debugging not configured
s-1A4KQJ1BRQVYC	Q5_Step	Pending	View logs Debugging not configured
s-35Y37KLUNA9V7	Q4_Step	Pending	View logs Debugging not configured
s-3B21WZ2KGGRMB2	Q3_Step	Pending	View logs Debugging not configured
s-25FY4FJLUJ3FH	Q2_Step	Pending	View logs Debugging not configured
s-3PFV2LP0OU2L8	Q1_Step	Pending	View logs Debugging not configured

After MapReduce Jobs are executed and SSH into Master Machine:

```
christymac:keys christy$ ssh -i "Lab2.pem" ec2-user@ec2-18-234-197-29.compute-1.amazonaws.com
Last login: Mon Aug  3 20:51:32 2020 from pool-96-237-107-103.bstnma.fios.verizon.net
[ec2-user@ip-172-31-34-35 ~]$ ls
christymac:keys christy$ ssh -i "Lab2.pem" ec2-user@ec2-18-234-197-29.compute-1.amazonaws.com
Last login: Mon Aug  3 20:51:32 2020 from pool-96-237-107-103.bstnma.fios.verizon.net
[ec2-user@ip-172-31-34-35 ~]$ hadoop fs -ls /
Found 14 items
drwxr-xr-x  - hadoop hadoop      0 2020-08-03 20:51 /010_Output
drwxr-xr-x  - hadoop hadoop      0 2020-08-03 20:52 /011_Output
drwxr-xr-x  - hadoop hadoop      0 2020-08-03 20:34 /01_Output
drwxr-xr-x  - hadoop hadoop      0 2020-08-03 20:36 /02_Output
drwxr-xr-x  - hadoop hadoop      0 2020-08-03 20:37 /03_Output
drwxr-xr-x  - hadoop hadoop      0 2020-08-03 20:38 /04_Output
drwxr-xr-x  - hadoop hadoop      0 2020-08-03 20:40 /05_Output
drwxr-xr-x  - hadoop hadoop      0 2020-08-03 20:43 /07_Output
drwxr-xr-x  - hadoop hadoop      0 2020-08-03 20:44 /08_Output
drwxr-xr-x  - hadoop hadoop      0 2020-08-03 20:45 /09_Output
drwxr-xr-x  - hdfs  hadoop      0 2020-08-03 20:30 /apps
drwxrwxrwt  - hdfs  hadoop      0 2020-08-03 20:31 /tmp
drwxr-xr-x  - hdfs  hadoop      0 2020-08-03 20:30 /user
drwxr-xr-x  - hdfs  hadoop      0 2020-08-03 20:30 /var
[ec2-user@ip-172-31-34-35 ~]$
```

DATA ANALYSIS USING APACHE PIG

Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

At the present time, Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject). Pig's language layer currently consists of a textual language called Pig Latin, which has the following key properties:

- **Ease of programming.** It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain.
- **Optimization opportunities.** The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.
- **Extensibility.** Users can create their own functions to do special-purpose processing.

TOP 10 WEATHER CONDITIONS DURING ACCIDENTS IN MASSACHUSETTS (MA)

```
grunt> run ~/Downloads/Projects/BigDataFinalProject/pig/q1.pig
```

Pig
Script:

```

system already initialized!
2020-08-03 15:28:49,143 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics
system already initialized!
2020-08-03 15:28:49,144 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics
system already initialized!
2020-08-03 15:28:49,146 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics
system already initialized!
2020-08-03 15:28:49,147 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics
system already initialized!
2020-08-03 15:28:49,148 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics
system already initialized!
2020-08-03 15:28:49,150 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduce
Launcher - Success!
2020-08-03 15:28:49,152 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has alread
y been initialized
2020-08-03 15:28:49,155 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input fi
les to process : 1
2020-08-03 15:28:49,155 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total
input paths to process : 1
(MA - Clear,7303)
(MA - Overcast,4990)
(MA - Fair,4857)
(MA - Mostly Cloudy,4614)
(MA - Partly Cloudy,3523)
(MA - Light Rain,2868)
(MA - Cloudy,2533)
(MA - Scattered Clouds,2135)
(MA - Light Snow,944)
(MA - Rain,522)
grunt>
grunt> 2020-08-03 15:28:49,162 [main] ERROR org.apache.pig.tools.grunt.Grunt - ERROR 2999: Unexpected inter
nal error: null
Details at logfile: /Users/christy/pig_1596482870955.log
grunt> []

```

```

DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader();

data = LOAD '/Accidents.csv' USING CSVLoader() AS (ID:CHARARRAY,
Source:CHARARRAY, TMC:DOUBLE, Severity:INT, Start_Time:CHARARRAY,
End_Time:CHARARRAY, Start_Lat:DOUBLE, Start_Lng:DOUBLE, End_Lat:DOUBLE,
End_Lng:DOUBLE, Distance:DOUBLE, Description:CHARARRAY, Number:DOUBLE,
Street:CHARARRAY, Side:CHARARRAY, City:CHARARRAY, County:CHARARRAY,
State:CHARARRAY, Zipcode:CHARARRAY, Country:CHARARRAY, Timezone:CHARARRAY,
Airport_Code:CHARARRAY, Weather_Timestamp:CHARARRAY, Temperature:DOUBLE,
Wind_Chill:DOUBLE, Humidity:DOUBLE, Pressure:DOUBLE, Visibility:DOUBLE,
Wind_Direction:CHARARRAY, Wind_Speed:DOUBLE, Precipitation:DOUBLE,
Weather_Condition:CHARARRAY, Amenity:BOOLEAN, Bump:BOOLEAN,
Crossing:BOOLEAN, Give_Way:BOOLEAN, Junction:BOOLEAN, No_Exit:BOOLEAN,
Railway:BOOLEAN, Roundabout:BOOLEAN, Station:BOOLEAN, Stop:BOOLEAN,
Traffic_Calming:BOOLEAN, Traffic_Signal:BOOLEAN, Turning_Loop:BOOLEAN,
Sunrise_Sunset:CHARARRAY, Civil_Twilight:CHARARRAY,
Nautical_Twilight:CHARARRAY, Astronomical_Twilight:CHARARRAY);

inter = FOREACH data GENERATE State, Weather_Condition;

ma = FILTER inter BY State == 'MA' AND NOT Weather_Condition == '';

grouped = GROUP ma BY CONCAT (State, ' - ', Weather_Condition);

concat_count = FOREACH grouped GENERATE group, COUNT(ma) AS cnt;

sorted = ORDER concat_count BY cnt DESC;

top = LIMIT sorted 10;

DUMP top;

```

CITIES IN MASSACHUSETTS WHERE ACCIDENTS HAPPEN DURING SNOWY DAYS

```
grunt> run ~/Downloads/Projects/BigDataFinalProject/pig/q2.pig
```

```
2020-08-03 15:36:37,735 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2020-08-03 15:36:37,735 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Lee,MA)
(Avon,MA)
(Lynn,MA)
(Stow,MA)
(Acton,MA)
(Ashby,MA)
(Athol,MA)
(Barre,MA)
(Dover,MA)
(Salem,MA)
(Waban,MA)
(Auburn,MA)
(Becket,MA)
(Berlin,MA)
(Bolton,MA)
(Boston,MA)
(Canton,MA)
(Carver,MA)
(Dedham,MA)
(Devens,MA)
(Dracut,MA)
(Dudley,MA)
(Groton,MA)
(Hadley,MA)
(Lowell,MA)
(Ludlow,MA)
(Malden,MA)
(Milton,MA)
(Monson,MA)
(Newton,MA)
```

Pig Script: (Uses Inner Join)

```
-- Cities in Massachusetts where accidents happen during snowy days
-- Find rows in State = MA where it snowed during accidents

weather_data = FOREACH data GENERATE ID, State, Weather_Condition;
ma = FILTER weather_data BY State == 'MA' AND NOT Weather_Condition == '';
snowy = FILTER ma BY Weather_Condition == 'Light Snow' OR Weather_Condition == 'Snow';

-----
-- Find All cities in US cities = FOREACH data GENERATE ID, City;

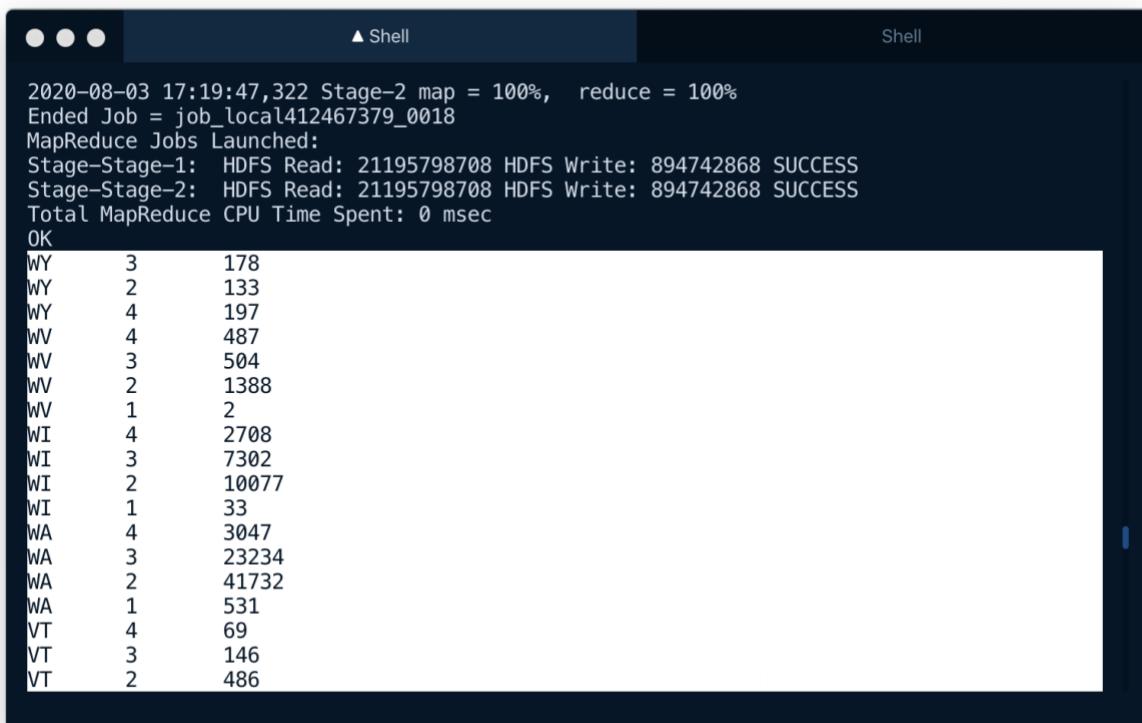
-----
joined = JOIN cities BY ID, snowy BY ID;
intermediate = FOREACH joined GENERATE City, State;
result = DISTINCT intermediate;
DUMP result;
```

DATA ANALYSIS USING APACHE HIVE

The Apache Hive™ data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL. Structure can be projected onto data already in storage. A command line tool and JDBC driver are provided to connect users to Hive.

```
$ hive -f ~/Downloads/Projects/BigDataFinalProject/hive/queries.hql
```

NUMBER OF ACCIDENTS PER SEVERITY PER STATE (PARTITIONING)



A screenshot of a terminal window titled "Shell". The window shows the output of a Hive query. The output includes job statistics and a table of accident counts by state and severity level. The table has three columns: State, Severity Level, and Count.

State	Severity Level	Count
WY	3	178
WY	2	133
WY	4	197
WV	4	487
WV	3	504
WV	2	1388
WV	1	2
WI	4	2708
WI	3	7302
WI	2	10077
WI	1	33
WA	4	3047
WA	3	23234
WA	2	41732
WA	1	531
VT	4	69
VT	3	146
VT	2	486

NUMBER OF ACCIDENTS PER TIMEZONE

```

Query ID = christy_20200803170011_5016e977-ca88-42b5-8967-07f948fb3e7
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 6
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2020-08-03 17:00:15,883 Stage-1 map = 0%,  reduce = 0%
2020-08-03 17:00:16,923 Stage-1 map = 100%,  reduce = 0%
2020-08-03 17:00:19,947 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1855470249_0001
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 11975075005 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
US/Mountain      200902
US/Central        837122
US/Pacific        986859
US/Eastern        1484854
Time taken: 8.181 seconds, Fetched: 4 row(s)
hive> 
```

TOP 10 - WEATHER CONDITION VS NUMBER OF ACCIDENT

```

set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2020-08-03 17:03:52,674 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_local1820747999_0007
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 41629707936 HDFS Write: 911106042 SUCCESS
Stage-Stage-2:  HDFS Read: 8046219592 HDFS Write: 165655644 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Clear    808202
Fair     547721
Mostly Cloudy   488094
Overcast    382485
Partly Cloudy  344815
Cloudy    212878
Scattered Clouds 204660
Light Rain    176942
Light Snow    50435
Rain       42016
Time taken: 7.037 seconds, Fetched: 10 row(s)
hive> 
```

```
-- Create Table

CREATE TABLE accidents (ID STRING, Source STRING, TMC DOUBLE, Severity INT,
Start_Time STRING, End_Time STRING, Start_Lat DOUBLE, Start_Lng DOUBLE,
End_Lat DOUBLE, End_Lng DOUBLE, Distance DOUBLE, Description STRING,
Number DOUBLE, Street STRING, Side STRING, City STRING, County STRING,
State STRING, Zipcode STRING, Country STRING, Timezone STRING,
Airport_Code STRING, Weather_Timestamp STRING, Temperature DOUBLE,
Wind_Chill DOUBLE, Humidity DOUBLE, Pressure DOUBLE, Visibility DOUBLE,
Wind_Direction STRING, Wind_Speed DOUBLE, Precipitation DOUBLE,
Weather_Condition STRING, Amenity BOOLEAN, Bump BOOLEAN, Crossing BOOLEAN,
Give_Way BOOLEAN, Junction BOOLEAN, No_Exit BOOLEAN, Railway BOOLEAN,
Roundabout BOOLEAN, Station BOOLEAN, Stop BOOLEAN, Traffic_Calming
BOOLEAN, Traffic_Signal BOOLEAN, Turning_Loop BOOLEAN, Sunrise_Sunset
STRING, Civil_Twilight STRING, Nautical_Twilight STRING,
Astronomical_Twilight STRING) row format delimited fields terminated by
',';

-- Load Data into Table from Hive LOAD DATA INPATH '/Accidents_hive.csv'
OVERWRITE INTO TABLE accidents;

-----
-- Number of Accidents per timezone SELECT timezone, count(*) from
accidents WHERE timezone is not NULL AND timezone <> '' AND
length(timezone) > 0 GROUP BY timezone;

-----
-- Number of Accidents per Severity per State (Partitioning)

CREATE TABLE IF NOT EXISTS state_part (city string, severity string)
PARTITIONED BY (state string);

SET hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE state_part PARTITION(state) SELECT
city, severity, state from accidents;

SELECT state, severity, count(*) from state_part WHERE severity is not
NULL GROUP BY state, severity ORDER BY state DESC;

-----
-- Top 10 - Weather Condition vs Number of Accident SELECT
weather_condition, count(*) as cnt from accidents GROUP BY
weather_condition HAVING weather_condition <> '' ORDER BY cnt DESC LIMIT
10;
```

REFERENCES

1. <https://www.kaggle.com/sobhanmoosavi/us-accidents>
2. <https://medium.com/@ayusoccer/us-road-accidents-data-analysis-8d88d1cf5d9b>
3. <https://www.kaggle.com/biphili/road-accidents-in-us>
4. <https://hadoop.apache.org/docs/current/api/>
5. <https://pig.apache.org/docs/r0.17.0/basic.html>
6. <https://cwiki.apache.org/confluence/display/Hive/LanguageManual>
7. <https://docs.aws.amazon.com/emr/index.html>
8. https://www.picknbuy24.com/column_331.html
9. <https://stackoverflow.com>
10. <https://docs.mongodb.com/manual/mongo/>

APENDIX

Number of Accidents Per Month

```
package com.hadoop.finalProject.Q1;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class ComparatorClass extends WritableComparator {

    SimpleDateFormat parser = new SimpleDateFormat("MMMMM");

    protected ComparatorClass() {
        super(Text.class, true);
    }

    @Override
    public int compare(Object a, Object b) {
        Text m1 = (Text) a;
        Text m2 = (Text) b;

        Date d1 = new Date();
        Date d2 = new Date();

        try {
            d1 = parser.parse(m1.toString());
            d2 = parser.parse(m2.toString());
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
}
```

```

    }

    return d1.compareTo(d2);
}

}

```

```

package com.hadoop.finalProject.Q1;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class MapperClass extends Mapper<LongWritable, Text, Text, IntWritable> {

    Text month = new Text();
    IntWritable one = new IntWritable(1);

    SimpleDateFormat parser = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    SimpleDateFormat format = new SimpleDateFormat("MMMMMM");

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String []tokens = value.toString().split(",");
        Date date = new Date();

        if(!tokens[0].equals("ID")) {
            String timeStamp = tokens[4];

            try {

```

```
        date = parser.parse(timeStamp);
    } catch (ParseException e) {
        e.printStackTrace();
    }
}

month.set(format.format(date));

context.write(month, one);

}
}
```

```
package com.hadoop.finalProject.Q1;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ReducerClass extends Reducer<Text, IntWritable, Text, LongWritable> {

    LongWritable sum = new LongWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
    InterruptedException {

        long count = 0;

        for(IntWritable v : values) {
            count += v.get();
        }
    }
}
```

```
    sum.set(count);

    context.write(key, sum);
}

}
```

```
package com.hadoop.finalProject.Q1;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.LocalFileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class DriverClass {

    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "NumberOfAccidentsPerMonth");

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.getConfiguration().set("mapreduce.output.textoutputformat.recordseparator","\t");

        // Driver Class
        job.setJarByClass(DriverClass.class);
    }
}
```

```
job.setInputFormatClass(TextInputFormat.class);

Path outDir = new Path(args[1]);
FileOutputFormat.setOutputPath(job, outDir);

// Comparator
job.setSortComparatorClass(ComparatorClass.class);

// Mapper
job.setMapperClass(MapperClass.class);

job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);

// Reducer
job.setReducerClass(ReducerClass.class);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

// Create only 1 reducer
job.setNumReduceTasks(1);

FileSystem fs = FileSystem.get(job.getConfiguration());

if(fs.exists(outDir)){
    fs.delete(outDir, true);
}

// Submit the job, then poll for progress until the job is complete
System.exit(job.waitForCompletion(true) ? 0 : 1);
}

}
```

Number of Accidents vs Hour of Day

```
package com.hadoop.finalProject.Q2;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparator;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class ComparatorClass extends WritableComparator {

    SimpleDateFormat parser = new SimpleDateFormat("HH");

    protected ComparatorClass() {
        super(Text.class, true);
    }

    @Override
    public int compare(Object a, Object b) {
        Text m1 = (Text) a;
        Text m2 = (Text) b;

        Date d1 = new Date();
        Date d2 = new Date();

        try {
            d1 = parser.parse(m1.toString());
            d2 = parser.parse(m2.toString());
        } catch (ParseException e) {
            e.printStackTrace();
        }

        return d1.compareTo(d2);
    }
}
```

```
}
```

```
package com.hadoop.finalProject.Q2;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class MapperClass extends Mapper<LongWritable, Text, Text, IntWritable> {

    Text hour = new Text();
    IntWritable one = new IntWritable(1);

    SimpleDateFormat parser = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    SimpleDateFormat format = new SimpleDateFormat("HH");

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String []tokens = value.toString().split(",");
        Date date = new Date();

        if(!tokens[0].equals("ID")) {
            String timeStamp = tokens[4];
            try {
                date = parser.parse(timeStamp);
            } catch (ParseException e) {
                e.printStackTrace();
            }
            hour.set(date.getHours());
            one.set(1);
            context.write(hour, one);
        }
    }
}
```

```
    }

}

hour.set(format.format(date));

context.write(hour, one);

}
```

```
package com.hadoop.finalProject.Q2;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ReducerClass extends Reducer<Text, IntWritable, Text, LongWritable> {

    LongWritable sum = new LongWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
    InterruptedException {

        long count = 0;

        for(IntWritable v : values) {
            count += v.get();
        }

        sum.set(count);

        context.write(key, sum);
    }
}
```

```
}
```

```
package com.hadoop.finalProject.Q2;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class DriverClass {

    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "NumberOfAccidentsPerHourOfDay");

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.getConfiguration().set("mapreduce.output.textoutputformat.recordseparator","\t");

        // Driver Class
        job.setJarByClass(DriverClass.class);

        job.setInputFormatClass(TextInputFormat.class);

        Path outDir = new Path(args[1]);
        FileOutputFormat.setOutputPath(job, outDir);
    }
}
```

```
// Comparator
job.setSortComparatorClass(ComparatorClass.class);

// Mapper
job.setMapperClass(MapperClass.class);

job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);

// Reducer
job.setReducerClass(ReducerClass.class);

// Create only 1 reducer
job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileSystem fs = FileSystem.get(job.getConfiguration());
if(fs.exists(outDir)){
    fs.delete(outDir, true);
}

// Submit the job, then poll for progress until the job is complete
System.exit(job.waitForCompletion(true) ? 0 : 1);
}

}
```

Number of Accidents Per Weekday

```
package com.hadoop.finalProject.Q3;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparator;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class ComparatorClass extends WritableComparator {

    SimpleDateFormat parser = new SimpleDateFormat("EEEE");

    protected ComparatorClass() {
        super(Text.class, true);
    }

    @Override
    public int compare(Object a, Object b) {
        Text m1 = (Text) a;
        Text m2 = (Text) b;

        Date d1 = new Date();
        Date d2 = new Date();

        try {
            d1 = parser.parse(m1.toString());
            d2 = parser.parse(m2.toString());
        } catch (ParseException e) {
            e.printStackTrace();
        }

        return d1.compareTo(d2);
    }
}
```

```
}
```

```
package com.hadoop.finalProject.Q3;
```

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
import java.io.IOException;
```

```
public class DriverClass {
```

```
    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();
```

```
        Job job = Job.getInstance(conf, "NumberOfAccidentsPerWeekday");
```

```
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
        //job.getConfiguration().set("mapreduce.output.textoutputformat.recordseparator","\t");
```

```
        // Driver Class
```

```
        job.setJarByClass(DriverClass.class);
```

```
        job.setInputFormatClass(TextInputFormat.class);
```

```
        Path outDir = new Path(args[1]);
        FileOutputFormat.setOutputPath(job, outDir);
```

```
// Comparator
job.setSortComparatorClass(ComparatorClass.class);

// Mapper
job.setMapperClass(MapperClass.class);

job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);

// Reducer
job.setReducerClass(ReducerClass.class);

// Create only 1 reducer
job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileSystem fs = FileSystem.get(job.getConfiguration());
if(fs.exists(outDir)){
    fs.delete(outDir, true);
}

// Submit the job, then poll for progress until the job is complete
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

```
package com.hadoop.finalProject.Q3;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class MapperClass extends Mapper<LongWritable, Text, Text, IntWritable> {

    Text hour = new Text();
    IntWritable one = new IntWritable(1);

    SimpleDateFormat parser = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    SimpleDateFormat format = new SimpleDateFormat("EEEEEE");

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String []tokens = value.toString().split(",");
        Date date = new Date();

        if(!tokens[0].equals("ID")) {
            String timeStamp = tokens[4];

            try {
                date = parser.parse(timeStamp);
            } catch (ParseException e) {
                e.printStackTrace();
            }
        }

        hour.set(format.format(date));

        context.write(hour, one);

    }
}
```

```
package com.hadoop.finalProject.Q3;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ReducerClass extends Reducer<Text, IntWritable, Text, LongWritable> {

    LongWritable sum = new LongWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
    InterruptedException {

        long count = 0;

        for(IntWritable v : values) {
            count += v.get();
        }

        sum.set(count);

        context.write(key, sum);
    }
}
```

Percentage of Accidents per side of the road

```
package com.hadoop.finalProject.Q4;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class DriverClass {

    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "PercentagePerSideOfRoad");

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        //job.getConfiguration().set("mapreduce.output.textoutputformat.recordseparator","\t");

        // Driver Class
        job.setJarByClass(DriverClass.class);

        job.setInputFormatClass(TextInputFormat.class);

        Path outDir = new Path(args[1]);
        FileOutputFormat.setOutputPath(job, outDir);
    }
}
```

```

// Mapper
job.setMapperClass(MapperClass.class);

job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);

// Reducer
job.setReducerClass(ReducerClass.class);

// Create only 1 reducer
job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileSystem fs = FileSystem.get(job.getConfiguration());
if(fs.exists(outDir)){
    fs.delete(outDir, true);
}

// Submit the job, then poll for progress until the job is complete
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

```

package com.hadoop.finalProject.Q4;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

```

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class MapperClass extends Mapper<LongWritable, Text, Text, IntWritable> {

    Text sideOfRoad = new Text();
    IntWritable one = new IntWritable(1);

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String []tokens = value.toString().split(",");
        if(!tokens[0].equals("ID")) {
            String side = tokens[14];
            if(!side.equals(" ")) {
                if(side.equals("L")) {
                    side = "Left";
                } else {
                    side = "Right";
                }

                sideOfRoad.set(side);
                context.write(sideOfRoad, one);
            }
        }
    }

    return;
}

}
```

```
package com.hadoop.finalProject.Q4;
```

```

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ReducerClass extends Reducer<Text, IntWritable, Text, Text> {

    Text percentage = new Text();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
    InterruptedException {

        int count = 0;

        for(IntWritable v : values) {
            count += v.get();
        }

        int total = 3513617; // total number of records

        double perc = ((double) count / total) * 100;
        percentage.set(String.format("%.2f", perc) + "%");

        context.write(key, percentage);
    }
}

```

Accidents Per State

```

package com.hadoop.finalProject.Q5;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;

```

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class DriverClass {

    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "NumberOfAccidentsPerState");

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        //job.getConfiguration().set("mapreduce.output.textoutputformat.recordseparator","\t");

        // Driver Class
        job.setJarByClass(DriverClass.class);

        job.setInputFormatClass(TextInputFormat.class);

        Path outDir = new Path(args[1]);
        FileOutputFormat.setOutputPath(job, outDir);

        // Mapper
        job.setMapperClass(MapperClass.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        // Reducer
    }
}
```

```

job.setReducerClass(ReducerClass.class);
job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileSystem fs = FileSystem.get(job.getConfiguration());
if(fs.exists(outDir)){
    fs.delete(outDir, true);
}

// Submit the job, then poll for progress until the job is complete
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

```

package com.hadoop.finalProject.Q5;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.Date;

public class MapperClass extends Mapper<LongWritable, Text, Text, IntWritable> {

    Text state = new Text();
    IntWritable one = new IntWritable(1);

    StateMap statesMap = new StateMap();

    @Override

```

```

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

    String []tokens = value.toString().split(",");
    String stateString = "";

    if(!tokens[0].equals("ID")) {
        stateString = statesMap.getStateFromAbbr(tokens[1]);
    }

    state.set(stateString);

    context.write(state, one);

}
}

```

```

package com.hadoop.finalProject.Q5;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ReducerClass extends Reducer<Text, IntWritable, Text, LongWritable> {

    LongWritable sum = new LongWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
    InterruptedException {
        long count = 0;

```

```
for(IntWritable v : values) {  
    count += v.get();  
}  
  
sum.set(count);  
  
context.write(key, sum);  
}  
}
```

```
package com.hadoop.finalProject.Q5;  
  
import java.util.HashMap;  
import java.util.Map;  
  
public class StateMap {  
  
    public static final Map<String, String> STATE_MAP;  
  
    static {  
        STATE_MAP = new HashMap<String, String>();  
        STATE_MAP.put("AL", "Alabama");  
        STATE_MAP.put("AK", "Alaska");  
        STATE_MAP.put("AB", "Alberta");  
        STATE_MAP.put("AZ", "Arizona");  
        STATE_MAP.put("AR", "Arkansas");  
        STATE_MAP.put("BC", "British Columbia");  
        STATE_MAP.put("CA", "California");  
        STATE_MAP.put("CO", "Colorado");  
        STATE_MAP.put("CT", "Connecticut");  
        STATE_MAP.put("DE", "Delaware");  
        STATE_MAP.put("DC", "District Of Columbia");  
        STATE_MAP.put("FL", "Florida");  
        STATE_MAP.put("GA", "Georgia");  
        STATE_MAP.put("GU", "Guam");  
    }  
}
```

```
STATE_MAP.put("HI", "Hawaii");
STATE_MAP.put("ID", "Idaho");
STATE_MAP.put("IL", "Illinois");
STATE_MAP.put("IN", "Indiana");
STATE_MAP.put("IA", "Iowa");
STATE_MAP.put("KS", "Kansas");
STATE_MAP.put("KY", "Kentucky");
STATE_MAP.put("LA", "Louisiana");
STATE_MAP.put("ME", "Maine");
STATE_MAP.put("MB", "Manitoba");
STATE_MAP.put("MD", "Maryland");
STATE_MAP.put("MA", "Massachusetts");
STATE_MAP.put("MI", "Michigan");
STATE_MAP.put("MN", "Minnesota");
STATE_MAP.put("MS", "Mississippi");
STATE_MAP.put("MO", "Missouri");
STATE_MAP.put("MT", "Montana");
STATE_MAP.put("NE", "Nebraska");
STATE_MAP.put("NV", "Nevada");
STATE_MAP.put("NB", "New Brunswick");
STATE_MAP.put("NH", "New Hampshire");
STATE_MAP.put("NJ", "New Jersey");
STATE_MAP.put("NM", "New Mexico");
STATE_MAP.put("NY", "New York");
STATE_MAP.put("NF", "Newfoundland");
STATE_MAP.put("NC", "North Carolina");
STATE_MAP.put("ND", "North Dakota");
STATE_MAP.put("NT", "Northwest Territories");
STATE_MAP.put("NS", "Nova Scotia");
STATE_MAP.put("NU", "Nunavut");
STATE_MAP.put("OH", "Ohio");
STATE_MAP.put("OK", "Oklahoma");
STATE_MAP.put("ON", "Ontario");
STATE_MAP.put("OR", "Oregon");
STATE_MAP.put("PA", "Pennsylvania");
STATE_MAP.put("PE", "Prince Edward Island");
STATE_MAP.put("PR", "Puerto Rico");
```

```

STATE_MAP.put("QC", "Quebec");
STATE_MAP.put("RI", "Rhode Island");
STATE_MAP.put("SK", "Saskatchewan");
STATE_MAP.put("SC", "South Carolina");
STATE_MAP.put("SD", "South Dakota");
STATE_MAP.put("TN", "Tennessee");
STATE_MAP.put("TX", "Texas");
STATE_MAP.put("UT", "Utah");
STATE_MAP.put("VT", "Vermont");
STATE_MAP.put("VI", "Virgin Islands");
STATE_MAP.put("VA", "Virginia");
STATE_MAP.put("WA", "Washington");
STATE_MAP.put("WV", "West Virginia");
STATE_MAP.put("WI", "Wisconsin");
STATE_MAP.put("WY", "Wyoming");
STATE_MAP.put("YT", "Yukon Territory");
}

public String getStateFromAbbr(String abbr) {
    return STATE_MAP.get(abbr);
}

}

```

Top 10 Accident Prone States

```

package com.hadoop.finalProject.Q6;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

```

```
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class DriverClass {

    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException{

        Configuration conf = new Configuration();
        // Create a new Job
        Job job = Job.getInstance(conf,"wordcount");
        job.setJarByClass(DriverClass.class);

        // Specify various job-specific parameters
        job.setJobName("myjob");

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setInputFormatClass(TextInputFormat.class);

        Path outDir = new Path(args[1]);
        FileOutputFormat.setOutputPath(job, outDir);

        job.setMapOutputKeyClass(NullWritable.class);
        job.setMapOutputValueClass(Text.class);

        job.setMapperClass(MapperClass.class);
        job.setReducerClass(ReducerClass.class);

        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);
    }
}
```

```
// Create only 1 reducer
job.setNumReduceTasks(1);

FileSystem fs = FileSystem.get(job.getConfiguration());
if(fs.exists(outDir)){
    fs.delete(outDir, true);
}

// Submit the job, then poll for progress until the job is complete
System.exit(job.waitForCompletion(true)?0:1);

}
```

```
package com.hadoop.finalProject.Q6;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MapperClass extends Mapper<Object, Text, NullWritable, Text> {

    @Override
    protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {

        context.write(NullWritable.get(),value);
    }
}
```

```
package com.hadoop.finalProject.Q6;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.Comparator;
import java.util.TreeMap;

public class ReducerClass extends Reducer<NullWritable, Text, NullWritable, Text> {

    static class DescOrder implements Comparator<Integer> {

        @Override
        public int compare(Integer o1, Integer o2) {
            return o2.compareTo(o1);
        }
    }

    private TreeMap<Integer, Text> countMap = new TreeMap<>(new DescOrder());

    @Override
    protected void reduce(NullWritable key, Iterable<Text> values, Context context) throws IOException,
    InterruptedException {
        for (Text value : values) {

            String[] tokens = value.toString().split("\t");
            int countOfStates = Integer.parseInt(tokens[1]);

            countMap.put(countOfStates, new Text(value));

            if (countMap.size() > 10)
                countMap.remove(countMap.lastKey());
        }

        for (Text t : countMap.values())
            context.write(NullWritable.get(), t);
    }
}
```

State - Cities = Inverted Index

```
package com.hadoop.finalProject.Q7;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class DriverClass {

    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "NumberOfAccidentsPerState");

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Driver Class
        job.setJarByClass(DriverClass.class);

        job.setInputFormatClass(TextInputFormat.class);

        Path outDir = new Path(args[1]);
        FileOutputFormat.setOutputPath(job, outDir);

        // Mapper
```

```
job.setMapperClass(MapperClass.class);

job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(Text.class);

// Reducer
job.setReducerClass(ReducerClass.class);

// Create only 1 reducer
job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileSystem fs = FileSystem.get(job.getConfiguration());
if(fs.exists(outDir)){
    fs.delete(outDir, true);
}

// Submit the job, then poll for progress until the job is complete
System.exit(job.waitForCompletion(true) ? 0 : 1);
}

}

package com.hadoop.finalProject.Q7;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MapperClass extends Mapper<LongWritable, Text, Text, Text> {
```

```

Text state = new Text();
Text cities = new Text();

@Override
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

    String []tokens = value.toString().split(",");
    String cityString = "";
    String stateString = "";

    if(!tokens[0].equals("ID")) {
        cityString = tokens[15];
        stateString = tokens[17];
    }

    state.set(stateString);
    cities.set(cityString);

    context.write(state, cities);
}

}

```

```

package com.hadoop.finalProject.Q7;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.HashSet;

public class ReducerClass extends Reducer<Text, Text, Text, Text> {

```

```

HashSet<String> hs = new HashSet<>();

@Override
protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException
{

    hs.clear();
    StringBuffer sb = new StringBuffer("");

    for(Text v: values){
        hs.add(v.toString());
    }

    for(String v: hs) {
        sb.append(v);
        sb.append(" | ");
    }

    context.write(key, new Text(sb.toString()));
}
}

```

Average, Min and Max Temperature per Severity

```

package com.hadoop.finalProject.Q8;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

```

```
import java.io.IOException;

public class DriverClass {

    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "MaxAvePressurePerSeverity");

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Driver Class
        job.setJarByClass(DriverClass.class);

        job.setInputFormatClass(TextInputFormat.class);

        Path outDir = new Path(args[1]);
        FileOutputFormat.setOutputPath(job, outDir);

        // Mapper
        job.setMapperClass(MapperClass.class);

        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(TempWritable.class);

        // Reducer
        job.setReducerClass(ReducerClass.class);

        // Create only 1 reducer
        job.setNumReduceTasks(1);

        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(TempWritable.class);

        FileSystem fs = FileSystem.get(job.getConfiguration());
```

```

if(fs.exists(outDir)){
    fs.delete(outDir, true);
}

// Submit the job, then poll for progress until the job is complete
System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

```

```

package com.hadoop.finalProject.Q8;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MapperClass extends Mapper<Object, Text, IntWritable, TempWritable> {

    IntWritable sev = new IntWritable();
    TempWritable tempWritable = new TempWritable();

    @Override
    protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {

        String[] tokens = value.toString().split(",");
        if (!tokens[0].equals("ID")) {
            if(tokens[3].isEmpty() || tokens[23].isEmpty() || tokens[3].equals(" ") || tokens[26].equals(" "))
                return;
        }

        int severity = Integer.parseInt(tokens[3]);

```

```
    double temperature = Double.parseDouble(tokens[23]);

    tempWritable.setAverageTemp(temperature);
    tempWritable.setMaxTemp(temperature);
    tempWritable.setMinTemp(temperature);
    tempWritable.setCount(1);

    sev.set(severity);

    context.write(sev, tempWritable);
}

}

}
```

```
package com.hadoop.finalProject.Q8;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ReducerClass extends Reducer<IntWritable, TempWritable, IntWritable, TempWritable> {

    TempWritable newTuple = new TempWritable();

    @Override
    protected void reduce(IntWritable key, Iterable<TempWritable> values, Context context) throws IOException,
    InterruptedException {

        double maxTemp = Double.MIN_VALUE;
        double minTemp = Double.MAX_VALUE;
        long count = 0;
        double sumOfTemps = 0.0;

        for (TempWritable tuple : values) {
```

```

sumOfTemps += tuple.getAverageTemp() * tuple.getCount();
count += tuple.getCount();

if(tuple.getMinTemp() < minTemp) {
    minTemp = tuple.getMinTemp();
}

if(tuple.getMaxTemp() > maxTemp) {
    maxTemp = tuple.getMaxTemp();
}

newTuple.setAverageTemp(sumOfTemps / count);
newTuple.setMaxTemp(maxTemp);
newTuple.setMinTemp(minTemp);
newTuple.setCount(count);

context.write(key, newTuple);
}
}

```

```

package com.hadoop.finalProject.Q8;

import org.apache.hadoop.io.Writable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class TempWritable implements Writable {

    private double maxTemp;
    private double averageTemp;
    private double minTemp;

```

```
public double getMinTemp() {
    return minTemp;
}

public void setMinTemp(double minTemp) {
    this.minTemp = minTemp;
}

private long count;

public long getCount() {
    return count;
}

public void setCount(long count) {
    this.count = count;
}

public double getMaxTemp() {
    return maxTemp;
}

public void setMaxTemp(double maxTemp) {
    this.maxTemp = maxTemp;
}

public double getAverageTemp() {
    return averageTemp;
}

public void setAverageTemp(double averageTemp) {
    this.averageTemp = averageTemp;
}

@Override
public void write(DataOutput dataOutput) throws IOException {
```

```

        dataOutput.writeDouble(averageTemp);
        dataOutput.writeDouble(maxTemp);
        dataOutput.writeDouble(minTemp);
        dataOutput.writeLong(count);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        averageTemp = dataInput.readDouble();
        maxTemp = dataInput.readDouble();
        minTemp = dataInput.readDouble();
        count = dataInput.readLong();
    }

    @Override
    public String toString() {
        return "Max Temperature = " + maxTemp +
            " | Min Temperature = " + minTemp +
            " | Average Temperature = " + averageTemp;
    }
}

```

Count of Accidents Per State Per Year (SecondarySorted with 5 Partitions - Per Year)

```

package com.hadoop.finalProject.Q9;

import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

```

```
public class CompositeKeyComparator extends WritableComparator {  
  
    protected CompositeKeyComparator() {  
        super(CompositeKeyWritable.class, true);  
    }  
  
    @Override  
    public int compare(WritableComparable a, WritableComparable b) {  
  
        CompositeKeyWritable w1 = (CompositeKeyWritable) a;  
        CompositeKeyWritable w2 = (CompositeKeyWritable) b;  
  
        int result = w1.getYear().compareTo(w2.getYear());  
  
        if(result == 0) {  
            return w1.getState().compareTo(w2.getState());  
        }  
  
        return result;  
    }  
}
```

```
package com.hadoop.finalProject.Q9;  
  
import org.apache.hadoop.io.WritableComparable;  
  
import java.io.DataInput;  
import java.io.DataOutput;  
import java.io.IOException;  
  
public class CompositeKeyWritable implements WritableComparable<CompositeKeyWritable> {  
  
    private String state;
```

```
private String year;

@Override
public String toString() {
    return "CompositeKeyWritable{" +
        "state=\"" + state + '\"' +
        ", year=\"" + year + '\"' +
        '}';
}

public String getState() {
    return state;
}

public void setState(String state) {
    this.state = state;
}

public String getYear() {
    return year;
}

public void setYear(String year) {
    this.year = year;
}

public CompositeKeyWritable(String state, String weatherCondition) {
    super();
    this.state = state;
    this.year = weatherCondition;
}

public CompositeKeyWritable() {
    super();
}

@Override
```

```
public int compareTo(CompositeKeyWritable o) {  
    int result = this.state.compareTo(o.state);  
    if(result == 0) {  
        return this.year.compareTo(o.year);  
    }  
    return result;  
}  
  
@Override  
public void write(DataOutput dataOutput) throws IOException {  
    dataOutput.writeUTF(state);  
    dataOutput.writeUTF(year);  
}  
  
@Override  
public void readFields(DataInput dataInput) throws IOException {  
    state = dataInput.readUTF();  
    year = dataInput.readUTF();  
}  
}
```

```
package com.hadoop.finalProject.Q9;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.FileSystem;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
  
import java.io.IOException;
```

```
public class DriverClass {

    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
        Configuration config = new Configuration();

        Job job = Job.getInstance(config, "");

        job.setJarByClass(DriverClass.class);

        job.setGroupingComparatorClass(NaturalKeyComparator.class);
        job.setSortComparatorClass(CompositeKeyComparator.class);
        job.setPartitionerClass(NaturalKeyPartitioner.class);

        job.setMapOutputKeyClass(CompositeKeyWritable.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(MapperClass.class);
        job.setReducerClass(ReducerClass.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        Path outDir = new Path(args[1]);
        FileOutputFormat.setOutputPath(job, outDir);

        // Set Number of Reduce Tasks
        job.setNumReduceTasks(5);

        FileSystem fs = FileSystem.get(job.getConfiguration());
        if (fs.exists(outDir)) {
            fs.delete(outDir, true);
        }
    }
}
```

```
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

}
```

```
package com.hadoop.finalProject.Q9;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;

public class MapperClass extends Mapper<Object, Text, CompositeKeyWritable, IntWritable> {

    SimpleDateFormat parser = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    SimpleDateFormat formater = new SimpleDateFormat("yyyy");

    @Override
    protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {

        String []tokens = value.toString().split(",");
        if(tokens[4].isEmpty() || tokens[17].isEmpty() || tokens[4].equals(" ") || tokens[17].equals(" ") ||
tokens[0].equals("ID")) {
            return;
        }

        String stateString = tokens[17];
        String year = null;
        try {
            year = formater.format(parser.parse(tokens[4]));
        } catch (ParseException e) {
```

```
    e.printStackTrace();
}

CompositeKeyWritable cKW = new CompositeKeyWritable(stateString, year);

context.write(cKW, new IntWritable(1));
}
}
```

```
package com.hadoop.finalProject.Q9;

import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class NaturalKeyComparator extends WritableComparator {

    public NaturalKeyComparator() {
        super(CompositeKeyWritable.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        CompositeKeyWritable w1 = (CompositeKeyWritable) a;
        CompositeKeyWritable w2 = (CompositeKeyWritable) b;

        return w1.getState().compareTo(w2.getState());
    }
}
```

```
package com.hadoop.finalProject.Q9;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Partitioner;
```

```
public class NaturalKeyPartitioner extends Partitioner<CompositeKeyWritable, IntWritable> {

    @Override
    public int getPartition(CompositeKeyWritable compositeKeyWritable, IntWritable intWritable, int i) {
        return Integer.parseInt(compositeKeyWritable.getYear()) % i;
    }
}
```

```
package com.hadoop.finalProject.Q9;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ReducerClass extends Reducer<CompositeKeyWritable, IntWritable, Text, IntWritable> {

    Text text = new Text();
    IntWritable count = new IntWritable();
    StateMap stateMap = new StateMap();

    @Override
    protected void reduce(CompositeKeyWritable key, Iterable<IntWritable> values, Context context) throws
    IOException, InterruptedException {

        int sum = 0;

        for(IntWritable v : values) {
            sum += v.get();
        }

        text.set(stateMap.getStateFromAbbr(key.getState())+ " - in Year - " + key.getYear() + " : ");
        count.set(sum);

        context.write(text, count);
    }
}
```

```
    }  
}
```

```
package com.hadoop.finalProject.Q9;  
  
import java.util.HashMap;  
import java.util.Map;  
  
public class StateMap {  
  
    public static final Map<String, String> STATE_MAP;  
  
    static {  
        STATE_MAP = new HashMap<String, String>();  
        STATE_MAP.put("AL", "Alabama");  
        STATE_MAP.put("AK", "Alaska");  
        STATE_MAP.put("AB", "Alberta");  
        STATE_MAP.put("AZ", "Arizona");  
        STATE_MAP.put("AR", "Arkansas");  
        STATE_MAP.put("BC", "British Columbia");  
        STATE_MAP.put("CA", "California");  
        STATE_MAP.put("CO", "Colorado");  
        STATE_MAP.put("CT", "Connecticut");  
        STATE_MAP.put("DE", "Delaware");  
        STATE_MAP.put("DC", "District Of Columbia");  
        STATE_MAP.put("FL", "Florida");  
        STATE_MAP.put("GA", "Georgia");  
        STATE_MAP.put("GU", "Guam");  
        STATE_MAP.put("HI", "Hawaii");  
        STATE_MAP.put("ID", "Idaho");  
        STATE_MAP.put("IL", "Illinois");  
        STATE_MAP.put("IN", "Indiana");  
        STATE_MAP.put("IA", "Iowa");  
        STATE_MAP.put("KS", "Kansas");  
        STATE_MAP.put("KY", "Kentucky");  
        STATE_MAP.put("LA", "Louisiana");  
    }  
}
```

```
STATE_MAP.put("ME", "Maine");
STATE_MAP.put("MB", "Manitoba");
STATE_MAP.put("MD", "Maryland");
STATE_MAP.put("MA", "Massachusetts");
STATE_MAP.put("MI", "Michigan");
STATE_MAP.put("MN", "Minnesota");
STATE_MAP.put("MS", "Mississippi");
STATE_MAP.put("MO", "Missouri");
STATE_MAP.put("MT", "Montana");
STATE_MAP.put("NE", "Nebraska");
STATE_MAP.put("NV", "Nevada");
STATE_MAP.put("NB", "New Brunswick");
STATE_MAP.put("NH", "New Hampshire");
STATE_MAP.put("NJ", "New Jersey");
STATE_MAP.put("NM", "New Mexico");
STATE_MAP.put("NY", "New York");
STATE_MAP.put("NF", "Newfoundland");
STATE_MAP.put("NC", "North Carolina");
STATE_MAP.put("ND", "North Dakota");
STATE_MAP.put("NT", "Northwest Territories");
STATE_MAP.put("NS", "Nova Scotia");
STATE_MAP.put("NU", "Nunavut");
STATE_MAP.put("OH", "Ohio");
STATE_MAP.put("OK", "Oklahoma");
STATE_MAP.put("ON", "Ontario");
STATE_MAP.put("OR", "Oregon");
STATE_MAP.put("PA", "Pennsylvania");
STATE_MAP.put("PE", "Prince Edward Island");
STATE_MAP.put("PR", "Puerto Rico");
STATE_MAP.put("QC", "Quebec");
STATE_MAP.put("RI", "Rhode Island");
STATE_MAP.put("SK", "Saskatchewan");
STATE_MAP.put("SC", "South Carolina");
STATE_MAP.put("SD", "South Dakota");
STATE_MAP.put("TN", "Tennessee");
STATE_MAP.put("TX", "Texas");
STATE_MAP.put("UT", "Utah");
```

```

STATE_MAP.put("VT", "Vermont");
STATE_MAP.put("VI", "Virgin Islands");
STATE_MAP.put("VA", "Virginia");
STATE_MAP.put("WA", "Washington");
STATE_MAP.put("WV", "West Virginia");
STATE_MAP.put("WI", "Wisconsin");
STATE_MAP.put("WY", "Wyoming");
STATE_MAP.put("YT", "Yukon Territory");

}

public String getStateFromAbbr(String abbr) {
    return STATE_MAP.get(abbr);
}

}

```

Divide file into partitions divided by state

```

package com.hadoop.finalProject.Q10;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class DriverClass {

```

```

public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {
    Configuration configuration = new Configuration();
    Job job = Job.getInstance(configuration, "ReduceSideJoin");
    job.setJarByClass(DriverClass.class);

    MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, MainMapperClass.class);
    MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, StateMapperClass.class);

    job.getConfiguration().set("join.type", "inner");
    job.setReducerClass(ReducerClass.class);

    // Partitioner Class
    job.setPartitionerClass(PartitionerClass.class);

    job.setOutputFormatClass(TextOutputFormat.class);

    Path outDir = new Path(args[2]);
    FileOutputFormat.setOutputPath(job, outDir);

    // US States
    job.setNumReduceTasks(49);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    FileSystem fs = FileSystem.get(job.getConfiguration());
    if(fs.exists(outDir)){
        fs.delete(outDir, true);
    }

    System.exit(job.waitForCompletion(true) ? 0 : 2);
}
}

```

```
package com.hadoop.finalProject.Q10;
```

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.Arrays;

public class MainMapperClass extends Mapper<LongWritable, Text, Text, Text> {

    Text outKey = new Text();
    Text outValue = new Text();
    boolean first = true;

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String tokens[] = value.toString().split(",");
        if(tokens[17].equals(" ") || tokens[17].isEmpty() || tokens[0].equals("ID")) {
            return;
        }

        String stateAbbr = tokens[17];
        outKey.set(stateAbbr);
        outValue.set("@" + value.toString());
        context.write(outKey, outValue);
    }
}

package com.hadoop.finalProject.Q10;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;
```

```
public class PartitionerClass extends Partitioner<Text, Text> {

    @Override
    public int getPartition(Text text, Text text2, int i) {
        return text.hashCode() % i;
    }
}
```

```
package com.hadoop.finalProject.Q10;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.ArrayList;

public class ReducerClass extends Reducer<Text, Text, Text, Text> {

    private ArrayList<Text> mainList = new ArrayList<Text>();
    private ArrayList<Text> stateList = new ArrayList<Text>();

    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException
    {
        mainList.clear();
        stateList.clear();
        for (Text text : values) {
            if (text.charAt(0) == '@') {
                mainList.add(new Text(text.toString().substring(1)));
            } else if (text.charAt(0) == '#') {
                stateList.add(new Text(text.toString().substring(1)));
            }
        }
        executeJoinLogic(context);
    }
}
```

```

public void executeJoinLogic(Context context) throws IOException, InterruptedException {
    String joinType = context.getConfiguration().get("join.type");

    //INNER JOIN OPERATION
    if (joinType.equalsIgnoreCase("inner")) {
        if (!mainList.isEmpty() && !stateList.isEmpty()) {
            for (Text stateTuple : stateList) {
                for (Text mainTuple : mainList) {
                    context.write(stateTuple, mainTuple);
                }
            }
        }
    }
}

```

```

package com.hadoop.finalProject.Q10;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class StateMapperClass extends Mapper<LongWritable, Text, Text, Text> {

    Text outKey = new Text();
    Text outValue = new Text();

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

```

```

String tokens[] = value.toString().split(",");
String state = tokens[1];
String stateAbbr = tokens[0];

String prefix = "#";

outKey.set(stateAbbr);
outValue.set(prefix + state);

context.write(outKey, outValue);
}
}

```

Proximity to Traffic Object (Percentage / per all traffic)

```

package com.hadoop.finalProject.Q11;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class DriverClass {

    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();

```

```
Job job = Job.getInstance(conf, "ProximityToTrafficObject");

FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

// Driver Class
job.setJarByClass(DriverClass.class);

job.setInputFormatClass(TextInputFormat.class);

Path outDir = new Path(args[1]);
FileOutputFormat.setOutputPath(job, outDir);

// Mapper
job.setMapperClass(MapperClass.class);

job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);

// Reducer
job.setReducerClass(ReducerClass.class);

// Create only 1 reducer
job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileSystem fs = FileSystem.get(job.getConfiguration());
if(fs.exists(outDir)){
    fs.delete(outDir, true);
}

// Submit the job, then poll for progress until the job is complete
System.exit(job.waitForCompletion(true) ? 0 : 1);
```

```
}

}
```

```
package com.hadoop.finalProject.Q11;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MapperClass extends Mapper<LongWritable, Text, Text, IntWritable> {

    Text proximity = new Text();
    IntWritable one = new IntWritable(1);

    RowHeaderClass rhc = new RowHeaderClass();

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String[] tokens = value.toString().split(",");
        if (!tokens[0].equals("ID")) {

            for(int i = 32; i < 45; i++) {
                if(tokens[i].equals("True")) {
                    proximity.set(rhc.getRowName(i));
                    context.write(proximity, one);
                }
            }
        }
    }
}
```

```
package com.hadoop.finalProject.Q11;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ReducerClass extends Reducer<Text, IntWritable, Text, Text> {

    Text percentage = new Text();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
    InterruptedException {

        int count = 0;

        for(IntWritable v : values) {
            count += v.get();
        }

        int total = 3513617; // total number of records

        double perc = ((double) count / total) * 100;
        percentage.set("\t" + String.format("%.2f", perc) + "%");

        context.write(key, percentage);
    }
}



---


```

package com.hadoop.finalProject.Q11;

```
import java.util.HashMap;
import java.util.Map;

public class RowHeaderClass {

    Map<Integer, String> rowNames = new HashMap<>();
    String []headerNames =
    {"Amenity","Bump","Crossing","Give_Way","Junction","No_Exit","Railway","Roundabout","Station","Stop","Traffic_Calming","Traffic_Signal","Turning_Loop"};

    public RowHeaderClass() {

        for(int i = 32; i < 45; i++) {
            rowNames.put(i, headerNames[i-32]);
        }

    }

    public String getRowName(int rowNum) {
        return rowNames.get(rowNum);
    }
}
```
