

Map Reduce Analysis on Motor Vehicle Collisions in New York City

Statement and Summary

The below New York city datasets track several measurable collision statistics. The goal is to measure certain patterns/trends with this data and identify if certain factors contribute more to these crashes and formulate solutions to mitigate them and improve people's safety.

There are three datasets that track information on crashes that took place in New York city

Collisions – Crashes

Collisions – Vehicles

Collisions -- Persons

<https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95>

<https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Vehicles/bm4k-52h4/data#Filter>

<https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Person/f55k-p6yu/data>

I have filtered the dataset to include only records from 2019 onwards

Collisions – Crashes Data set

This data records the location details of a crash event – Borough, Street Name, Latitude and Longitude etc.

Table Preview

[View Data](#) [Create Visualization](#)

CRASH DATE	CRASH TIME	BOROUGH	ZIP C...	LATI...	LON...	LOCA...	ON S...	CROS...	OFF S...
04/13/2021	0:00	BROOKLYN	11222	40.726444	-73.95233	(40.72644...			745 MAN...
04/13/2021	0:00	BROOKLYN	11231	40.678524	-74.0021	(40.67852...			667 HEN...
04/13/2021	0:00	QUEENS	11420	40.677498	-73.8276	(40.67749...	111 STREET	ROCKAW...	
04/13/2021	0:00						ALEXAND...		
04/13/2021	0:00			40.877235	-73.91781	(40.87723...	JOHNSON...		
04/13/2021	0:00			40.90021	-73.8865	(40.90021...	MOSHOL...		
04/13/2021	0:07	BRONX	10456	40.82942	-73.91226	(40.82942...	EAST 166 ...	CLAY AVE...	
04/13/2021	0:15	QUEENS	11378	40.727432	-73.90713	(40.72743...	LONG ISL...	MAURICE ...	
04/13/2021	0:45			40.843956	-73.8978	(40.84395...	CROSS B...		
04/13/2021	10:00	BRONX	10463				Ft indepe...	Bailey pla...	
04/13/2021	10:00	BROOKLYN	11207	40.686226	-73.9124	(40.68622...	BUSHWIC...	COVERT S...	
04/13/2021	10:00	BROOKLYN	11209	40.634743	-74.03529	(40.63474...	NARROW...	73 STREET	
04/13/2021	10:00			40.81782	-73.915276	(40.81782...	3 AVENUE	EAST 152 ...	

Collisions – Vehicle Data set

This data set includes information on the vehicles involved in the crash.
There could be multiple vehicle records linked to the same crash id

Table Preview

[View Data](#) [Create Visualization](#)

UNIQ...	COLL...	CRAS...	CRAS...	VEHI...	STAT...	VEHI...	VEHI...	VEHI...	VEHI...	TRAV...	VEHI...	DRI...
10385780	100201	09/07/2012	9:03	1	NY	PASSENG...						
14887647	3307608	10/02/2015	17:18	2	NY	TAXI						
14889754	3308693	10/04/2015	20:34	1	NY	PASSENG...						
14400270	297666	04/25/2013	21:15	1	NY	PASSENG...						
17044639	3434155	05/02/2016	17:35	219456	NY	4 dr sedan	MERZ -CA...		2015	East	2	M
17303317	3503027	08/18/2016	12:39	672828	NY	Station W...	FORD F-150 MERZ -CAR/SUV		2005	Southwest	2	F
12254536	196425	07/16/2013	11:20	1	NY	PASSENG...						
11804847	2975897	11/26/2012	18:12	2	NY	PASSENG...						
17285715	3487936	07/22/2016	15:40	554272	NY	Convertible	VOLK -CA...		2013	South	1	M
14809587	3268858	08/01/2015	8:17	2	NJ	PASSENG...						
17307366	3499697	08/13/2016	21:05	650962	NY	Sedan	NISS -CAR...		2015	West	2	F
11912713	176016	08/11/2012	19:36	2	NY	BICYCLE						
17401424	3516125	09/08/2016	14:40	764668	NY	Station W...	LINC -CA...		2015	West	1	M

Crashes – Persons

This dataset includes information on the persons involved in the crash and what their role was – whether they were an vehicle occupant, pedestrian, cyclists etc

Table Preview

[View Data](#)[Create Visualization](#)

UNIQ...	COLL...	CRAS...	CRAS...	PERS...	PERS...	PERS...	VEHI...	PERS...	EJECT...	EMO...	BODI...	POS
6650180	3565527	11/21/2016	13:05	2782525	Occupant	Unspecifi...						
5783504	3428861	03/29/2016	16:30	204724	Occupant	Unspecifi...						
6377814	3491131	07/06/2016	1:51	1380463	Occupant	Unspecifi...						
6341088	3487108	07/19/2016	21:50	1273001	Occupant	Unspecifi...						
6618640	3577702	12/10/2016	6:01	2994887	Occupant	Unspecifi...						
6687364	3565713	11/22/2016	18:30	2792542	Occupant	Unspecifi...						
6503982	3464681	06/17/2016	14:00	1004357	Occupant	Unspecifi...						
5733678	3414732	03/14/2016	8:30	97248	Pedestrian	Injured		30	Does Not ...	Conscious	Shoulder ...	Does
6529616	3514065	08/25/2016	14:00	1849393	Pedestrian	Injured		42		Conscious	Knee-Low...	
6482113	3493179	07/29/2016	11:00	1417915	Occupant	Unspecifi...						
5830513	3409015	04/21/2016	21:40	452620	Occupant	Unspecifi...						
5975062	3439295	05/07/2016	20:00	608018	Occupant	Unspecifi...						
6748026	3549869	10/30/2016	12:00	2570219	Pedestrian	Injured		61		Conscious	Entire Body	

There could be multiple records linked to a crash id , depending on the number of persons involved in the crash.

All these data sets are loaded into Hadoop File System.

Analysis Performed

Evaluate top Boroughs in NYC with collisions and the streets within this boroughs with most number of crashes.

Evaluate major contributing factors for crashes by Sex – Male/Female

Analyse if more crashes occur during the day time or night

Analyse which streets have higher number of Pedestrian casualties

Observe monthly trend in Crash numbers

Evaluate the average number of persons affected by month due to collisions

Vehicle make (manufacturers) involved in crashes by percentage

Analyse which type of vehicles are causing accidents with Bicyclists.

Review if persons getting ejected from their vehicles were using any form of safety equipment

Filter data – Vehicle type using Bloom filter to evaluate if a vehicle is member of a set

Files loaded to HDFS

```
hadoop fs -copyFromLocal ~/downloads/MotorVehiclesSelect.csv  
/NYC_Project
```

```
hadoop fs -copyFromLocal ~/downloads/MotorCrashesSelect.csv  
/NYC_Crashes
```

```
hadoop fs -copyFromLocal ~/downloads/MotorPersonsSelect.csv  
/NYC_Persons
```

First Step - Evaluate the top Boroughs in New York City by Crash numbers

```
hadoop jar ~/Desktop/Hadoop_FinalProject/nyc_borough.jar nyc_borough.  
DriverClass /NYC_Crashes /NYC_Crash_top5_borough
```

Get the first top 5 boroughs for crashes

```
[madhav437@MacBook-Pro hadoop-3.3.1 % hadoop fs  
2021-12-11 00:29:04,350 WARN util.NativeCodeLoa  
g builtin-java classes where applicable  
STATEN ISLAND 13034  
MANHATTAN 73019  
BRONX 74881  
QUEENS 120092  
BROOKLYN 142696  
madhav437@MacBook-Pro hadoop-3.3.1 %
```

Take a subset of top 3 boroughs in NYC, use Borough as Natural Grouping Key and Secondary sort by Street within these boroughs

```
hadoop jar ~/Desktop/Hadoop_FinalProject/nyc_borough_compare2.jar  
nyc_borough_compare2.DriverClass /NYC_Crashes  
/NYC_Crash_boroughCompare2
```

Do Secondary sorting based on Streets within the Borough

```
KeyCompare.java
2+ * To change this license header, choose License Headers in Project Properties.
6 package nyc_borough_compare2;
7
8 import org.apache.hadoop.io.Text;
20
21 public class KeyCompare implements WritableComparable<KeyCompare>{
22
23
24     // private Date crashDate = new Date();
25     private String borough;
26     private String streetName;
27
28
29
30     private final static SimpleDateFormat fmt = new SimpleDateFormat("yyyy-MM-dd");
31
32     public int compareTo(KeyCompare k) {
33
34         int result = 0 ;
35         try {
36             result = this.borough.compareTo(k.getBorough());
37             if(0 == result) {
38                 result = -1*this.streetName.compareTo(k.getStreetName());
39
40             }
41         }
42         catch (Exception e) {
43
44             }
45         return result;
46     }
47
48
49
50     @Override
51     public String toString() {
52         return (new StringBuilder().append(borough).append("\t")
53                         .append(streetName)).toString();
54     }
55 }
```

QUEENS	LONG STREET	1
QUEENS	LORETTA ROAD	2
QUEENS	LOUBET STREET	1
QUEENS	LOVINGHAM PLACE	14
QUEENS	LUCAS STREET	1
QUEENS	LUDLUM AVENUE	2
QUEENS	LUTHERAN AVENUE	1
QUEENS	LYMAN STREET	1
QUEENS	Lefferts Boulevard	1
QUEENS	Liberty Avenue	1
QUEENS	Long Island expwy	2
QUEENS	MACNISH STREET	33
QUEENS	MADISON STREET	6
QUEENS	MAIN AVENUE	364
QUEENS	MAIN STREET	14
QUEENS	MANGIN AVENUE	3
QUEENS	MANSE STREET	5
QUEENS	MANTON STREET	32
QUEENS	MAPLE AVENUE	36
QUEENS	MARATHON PARKWAY	1
QUEENS	MARATHON PKWY	1
QUEENS	MARENGO STREET	1
QUEENS	MARGARET PLACE	1
QUEENS	MARINA ROAD	2
QUEENS	MARINE TERMINAL ROAD	4
QUEENS	MARKWOOD ROAD	3
QUEENS	MARNE PLACE	2
QUEENS	MARS PLACE	10
QUEENS	MARSDEN STREET	8
QUEENS	MARTENSE AVENUE	1
QUEENS	MARYLAND ROAD	29
QUEENS	MASPETH AVENUE	1
QUEENS	MATHEWSON COURT	1
QUEENS	MAURICE AVE	73
QUEENS	MAURICE AVENUE	14
QUEENS	MAYDA ROAD	1
QUEENS	MAYFAIR ROAD	2
QUEENS	MAYVILLE STREET	2

Filter Top 10 streets by crash numbers within the top 3 boroughs – Brooklyn, Bronx and Queens

Using internal tree map within each of the mentioned boroughs

```
15
16 public class MapperClass extends Mapper<LongWritable, Text, KeyCompare, IntWritable> {
17
18
19     // hadoop datatype
20     KeyCompare kc = new KeyCompare();
21     //private final static SimpleDateFormat frmt = new SimpleDateFormat("yyyy-MM-dd");
22     IntWritable one = new IntWritable(1);
23     String crsDate;
24     static final List<String> boroughs = Arrays.asList("BROOKLYN", "QUEENS", "BRONX");
25
26
27     @Override
28     protected void map(LongWritable key, Text value, Mapper.Context context) throws IOException {
29
30         String line = value.toString();
31
32         String[] logs = line.split(",");
33
34         if(logs.length>0 && boroughs.contains(logs[2])
35             && !logs[8].isBlank()) {
36
37             String borough = logs[2].trim();
38
39
40             if (!borough.equals("BOROUGH") && !borough.isEmpty()) {
41
42                 kc.setBorough(borough);
43
44                 String crsDate = logs[8].trim();
45
46                 /* String[] dateSplit = crsDt.split("/");
47
48                 try {
49                     crsDate = dateSplit[2] + "/" + dateSplit[0] + "/" + dateSplit[1];
50
51
52
53
```

```
27
28
29
30     protected void reduce(KeyCompare key, Iterable<IntWritable> values, Context context) throws IOException {
31
32         int sum=0;
33         String temp = " ";
34
35         for(IntWritable v: values) {
36             if (temp.equals(" "))
37                 temp = key.getStreetName();
38
39             if (!temp.equals(key.getStreetName())) {
40                 temp = key.getStreetName();
41                 tmapTemp.put(sum,key.toString());
42
43             // Add Top 5
44             if (tmapTemp.size() > 10)
45             {
46                 tmapTemp.remove(tmapTemp.firstKey());
47             }
48             sum = 0;}
49             sum += v.get();
50
51
52 }
```

```

File Input Format Counters
    Bytes Read=87068566
File Output Format Counters
    Bytes Written=796
madhav437@MacBook-Pro hadoop-3.3.1 % hadoop fs -cat /NYC_Crash_boroughCompare2/part-r-00000
2021-12-11 20:43:11,802 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
here applicable
BRONX    BOSCOBEL PLACE   563
BRONX    BRUCKNER BLVD   1063
BRONX    EAST 148 STREET  439
BRONX    EAST CLARKE PLACE 417
BRONX    EAST FORDHAM ROAD 486
BRONX    EAST MOUNT EDEN AVENUE 711
BRONX    GRAND AVENUE    587
BRONX    JEROME AVE      708
BRONX    WEBSTER AVE     545
BRONX    WHEELER AVENUE   499
BROOKLYN   ATKINS AVENUE  1657
BROOKLYN   BEDELL LANE    645
BROOKLYN   EAST NEW YORK AVENUE 663
BROOKLYN   FILLMORE AVENUE 989
BROOKLYN   KING STREET    700
BROOKLYN   LINDEN BLVD    939
BROOKLYN   NORWOOD AVENUE 602
BROOKLYN   OCEAN COURT    596
BROOKLYN   PENNSYLVANIA AVE 710
BROOKLYN   UNION STREET   632
QUEENS    FRAME PLACE    538
QUEENS    HILBURN AVENUE  691
QUEENS    JAMAICA AVE     606
QUEENS    LINCOLN STREET  615
QUEENS    NORMAN STREET   951
QUEENS    NORTHERN BLVD   1367
QUEENS    QUEENS BLVD     1116
QUEENS    ROCKAWAY BEACH BOULEVARD 761
QUEENS    SOUTH CONDUIT AVE 762
QUEENS    WOODBINE STREET  838
madhav437@MacBook-Pro hadoop-3.3.1 %

```

Demographics

Male/Female – Top contributing factors for crashes

Secondary Sort and Top N

Using Natural Key as Driver Sex and top contributing factors by Driver Sex

```

hadoop jar ~/Desktop/Hadoop_FinalProject/nyc_mf_compare.jar
nyc_mf_compare.DriverClass /NYC_Project /NYC_mf

```

```

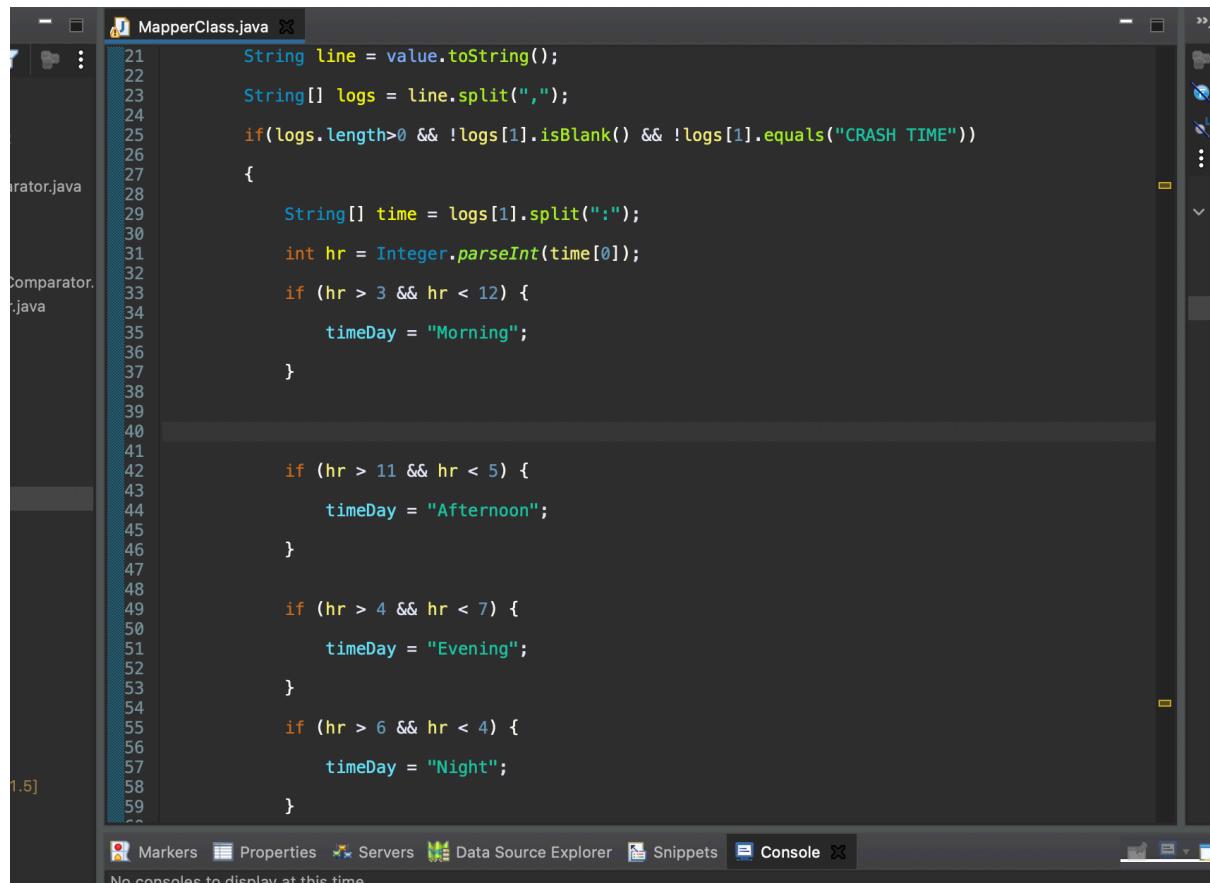
File Output Format Counters
    Bytes Written=210
madhav437@MacBook-Pro hadoop-3.3.1 % hadoop fs -cat /NYC_mf/part-r-00000
2021-12-12 01:04:45,078 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
here applicable
F      Cell Phone (hands-free) 38664
F      Fell Asleep      12801
F      Unsafe Speed     135904
M      Cell Phone (hands-free) 121208
M      Fell Asleep      43405
M      Unsafe Speed     359908
U      Brakes Defective  350
U      Fell Asleep      108
U      Unsafe Speed     1164
madhav437@MacBook-Pro hadoop-3.3.1 %

```

'Cell Phone' and 'Fell Asleep' are the top factors contributing to crashes for both Males and Females

Evaluate Crashes by time of the day – Morning/Afternoon/Evening/Night

```
hadoop jar ~/Desktop/Hadoop_FinalProject/nyc_crash_time.jar  
nyc_crash_time.DriverClass /NYC_Crashes /NYC_time
```



```
MapperClass.java
1  String line = value.toString();
2
3  String[] logs = line.split(",");
4
5  if(logs.length>0 && !logs[1].isBlank() && !logs[1].equals("CRASH TIME"))
6  {
7
8      String[] time = logs[1].split(":");
9
10     int hr = Integer.parseInt(time[0]);
11
12     if (hr > 3 && hr < 12) {
13
14         timeDay = "Morning";
15
16     }
17
18
19     if (hr > 11 && hr < 5) {
20
21         timeDay = "Afternoon";
22
23     }
24
25     if (hr > 4 && hr < 7) {
26
27         timeDay = "Evening";
28
29     }
30
31     if (hr > 6 && hr < 4) {
32
33         timeDay = "Night";
34
35     }
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
```

```
BYTES WRITTEN=27
madhav437@MacBook-Pro hadoop-3.3.1 % hadoop fs -cat /NYC_time/part-r-00000
2021-12-12 02:01:48,471 WARN util.NativeCodeLoader: Unable to load native-hadoop library for yo
here applicable
Evening 59332
Morning 364383
madhav437@MacBook-Pro hadoop-3.3.1 %
```

Evening and Morning (during rush hours) seem to be the ones when most of the crashes take place.

Joining Two Datasets to evaluate Streets where more Pedestrians are affected with collisions

This would need inner joining of **Crashes** and **Persons** datasets using Collision id as key

File inputs would be done using Multiple Inputs

```
public class DriverClass2 extends Configured implements Tool {
    private static final String DATA_SEPARATOR = ",";
    public int run(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        // configuration.set("mapreduce.output.textoutputformat.separator", DATA_SEPARATOR);
        Job job = Job.getInstance(configuration);
        job.setJobName("New Job");
        job.setJarByClass(DriverClass2.class);
        // Map
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        // Job
        job.setOutputKeyClass(LongWritable.class);
        job.setOutputValueClass(Text.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setReducerClass(ReducerClass.class);
        MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class,
            CrashMapperClass.class);
        MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class,
            PersonMapperClass.class);
        FileOutputFormat.setOutputPath(job, new Path(args[2]));
        job.waitForCompletion(true);
        return 0;
    }
    public static void main(String[] args) throws Exception {
        if (args.length == 3) {
            int result = ToolRunner.run(new Configuration(), new DriverClass2(), args);
            if (0 == result) {
                System.out.println("");
            } else {
                System.out.println("");
            }
        } else {
            System.out.println("USAGE <InputPath1><InputPath2><OutputPath>");
        }
    }
}
```

```
hadoop jar
~/Desktop/Hadoop_FinalProject/nyc_ped_compare.jar
nyc_ped_compare.DriverClass2 /NYC_Crashes
/NYC_Persons /NYC_Out2
```

```
ReducerClass.java PersonMapperClass.java
15
16     Text values = new Text();
17     String tempVal= " ";
18     String tempVal1= " ";
19
20     @Override
21     protected void map(LongWritable key, Text value, Mapper.Context context) throws IOException,
22
23         String line = value.toString();
24
25         String[] logs = line.split(",");
26
27
28
29
30
31         if(logs.length>0
32             // && !logs2[1].isBlank()
33             // && !logs2[2].isBlank()
34             && !logs[1].isBlank()
35             && !logs[1].equals("COLLISION_ID")
36             && logs[5].equals("Pedestrian")
37             && (logs[6].equals("Injured") || logs[6].equals("Killed")))
38
39
40
41     {
42
43
44
45
46
47         String colId = logs[1];
48
49         tempVal = logs[5].trim()+ "," + logs[6];
50         // String tempVal = logs2[1].trim() + ' ' + logs2[2].trim();
51
52         tempVal = "Z" + " " + tempVal ;

```

```
ReducerClass.java
15
16
17
18     @Override
19     protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
20
21         listA.clear();
22         listB.clear();
23
24
25         for(Text v: values) {
26             if (v.charAt(0) == 'Y')
27                 listA.add(new Text(v.toString().substring(1)+ ","));
28             else if (v.charAt(0) == 'Z')
29                 listB.add(new Text(v.toString().substring(1)));
30
31         executeJoin(context);
32
33
34     }
35
36     private void executeJoin(Context context) throws IOException,
37     InterruptedException{
38
39         if(!listA.isEmpty() && !listB.isEmpty())
40         {
41             for(Text A: listA){
42
43                 for(Text B : listB)
44                 {
45                     context.write(A,B);
46                 }
47
48             }
49
50         }
51     }

```

```

Q\ killed
QUEENS,, Pedestrian,Injured
BROOKLYN,, Pedestrian,Injured
QUEENS,101 AVENUE, Pedestrian,Injured
QUEENS,, Pedestrian,Injured
QUEENS,, Pedestrian,Injured
BROOKLYN,ROCKAWAY AVENUE, Pedestrian,Injured
BRONX,MELROSE AVENUE, Pedestrian,Injured
BROOKLYN,WEST 13 STREET, Pedestrian,Injured
BRONX,EAST 169 STREET, Pedestrian,Injured
BRONX,EAST 180 STREET, Pedestrian,Injured
QUEENS,ASTORIA BOULEVARD, Pedestrian,Injured
BROOKLYN,NEW LOTS AVENUE, Pedestrian,Injured
MANHATTAN,EAST 27 STREET, Pedestrian,Injured
BRONX,, Pedestrian,Injured
QUEENS,135 AVENUE, Pedestrian,Injured
BRONX,, Pedestrian,Injured
BRONX,, Pedestrian,Injured
BROOKLYN,EAST 54 STREET, Pedestrian,Injured
QUEENS,BAISLEY BOULEVARD, Pedestrian,Injured
BRONX,WHITE PLAINS ROAD, Pedestrian,Injured
QUEENS,FOREST AVENUE, Pedestrian,Injured
MANHATTAN,WEST 64 STREET, Pedestrian,Injured
QUEENS,137 STREET, Pedestrian,Injured
QUEENS,FRANCIS LEWIS BOULEVARD, Pedestrian,Injured
BROOKLYN,ADELPHI STREET, Pedestrian,Injured
MANHATTAN,, Pedestrian,Injured
BROOKLYN,EAST 13 STREET, Pedestrian,Injured
BRONX,BATHGATE AVENUE, Pedestrian,Injured
BRONX,WATSON AVENUE, Pedestrian,Injured
BRONX,, Pedestrian,Injured
MANHATTAN,, Pedestrian,Injured
BRONX,POND PLACE, Pedestrian,Injured
BRONX,EAST 150 STREET, Pedestrian,Injured
BRONX,EAST 150 STREET, Pedestrian,Injured
BRONX,MARION AVENUE, Pedestrian,Injured
BROOKLYN,13 AVENUE, Pedestrian,Injured
BROOKLYN,60 STREET, Pedestrian,Injured
madhav437@MacBook-Pro hadoop-3.3.1 %

```

```

hadoop jar
~/Desktop/Hadoop_FinalProject/nyc_ped_compare2.jar
nyc_ped_compare2.DriverClass /NYC_Out2/part-r-00000
/NYC_Out3

```

This output shows the top 15 locations where Pedestrians are getting injured or killed

```

2021-12-15 15:57:05,245 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
here applicable
QUEENS-QUEENS BOULEVARD 66
MANHATTAN-3 AVENUE 68
MANHATTAN-1 AVENUE 74
BRONX-EAST TREMONT AVENUE 76
BROOKLYN-FLATBUSH AVENUE 83
BROOKLYN-NOSTRAND AVENUE 87
QUEENS-NORTHERN BOULEVARD 93
MANHATTAN-BROADWAY 94
BROOKLYN-ATLANTIC AVENUE 97
MANHATTAN-2 AVENUE 101
STATEN ISLAND- 124
MANHATTAN- 919
QUEENS- 1076
BRONX- 1122
BROOKLYN- 1729
madhav437@MacBook-Pro hadoop-3.3.1 %

```

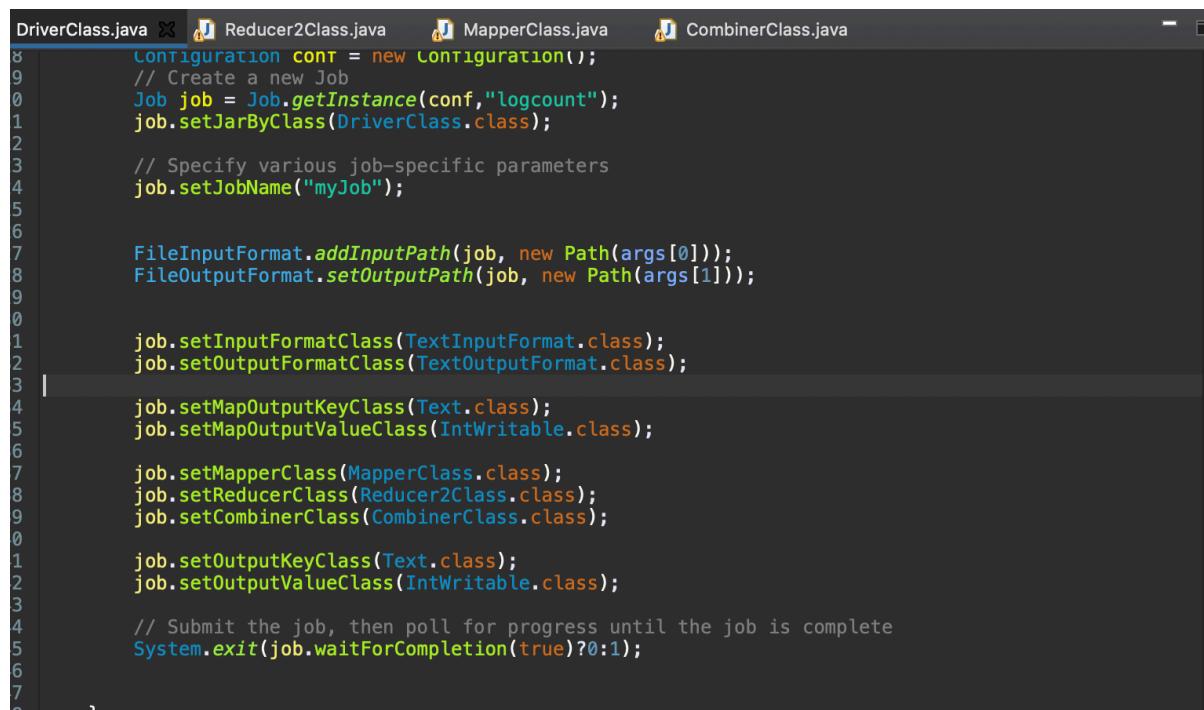
Accidents by Month – Breakup using Crashes dataset

```
hadoop jar  
~/Desktop/Hadoop_FinalProject/nyc_crash_month.jar  
nyc_crash_month.DriverClass /NYC_Crashes /NYC_Out4
```

```
Bytes Read=3,000,000  
File Output Format Counters  
Bytes Written=158  
madhav437@MacBook-Pro hadoop-3.3.1 % hadoop fs -cat /NYC_Out4/part-r-00000  
2021-12-15 16:23:34,258 WARN util.NativeCodeLoader: Unable to load native-hadoop library for  
here applicable  
December 25435  
April 29705  
November 32918  
May 36029  
February 36731  
August 36900  
September 37022  
March 37092  
October 37477  
July 37695  
June 37761  
January 38960  
madhav437@MacBook-Pro hadoop-3.3.1 %
```

Average number of people getting affected by month using Combiner

```
hadoop-3.3.1 % hadoop jar  
~/Desktop/Hadoop_FinalProject/nyc_month_avg.jar  
nyc_month_avg.DriverClass /NYC_Crashes /NYC_Out6
```



```
DriverClass.java ✘ Reducer2Class.java ✘ MapperClass.java ✘ CombinerClass.java  
1 Configuration conf = new Configuration();  
2 // Create a new Job  
3 Job job = Job.getInstance(conf, "logcount");  
4 job.setJarByClass(DriverClass.class);  
5  
6 // Specify various job-specific parameters  
7 job.setJobName("myJob");  
8  
9  
10 FileInputFormat.addInputPath(job, new Path(args[0]));  
11 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
12  
13 job.setInputFormatClass(TextInputFormat.class);  
14 job.setOutputFormatClass(TextOutputFormat.class);  
15  
16 job.setMapOutputKeyClass(Text.class);  
17 job.setMapOutputValueClass(IntWritable.class);  
18  
19 job.setMapperClass(MapperClass.class);  
20 job.setReducerClass(Reducer2Class.class);  
21 job.setCombinerClass(CombinerClass.class);  
22  
23 job.setOutputKeyClass(Text.class);  
24 job.setOutputValueClass(IntWritable.class);  
25  
26 // Submit the job, then poll for progress until the job is complete  
27 System.exit(job.waitForCompletion(true)?0:1);  
28
```

```
DriverClass.java Reducer2Class.java MapperClass.java CombinerClass.java
2 public class CombinerClass extends Reducer<Text, Text, Text, Text> {
3
4     Text mainValue = new Text();
5     String mainVal = " ";
6
7
8     @Override
9     protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
10         int sum=0;
11         int cnt=0;
12         String num = " ";
13
14         for(Text v: values) {
15             num = v.toString();
16             sum += Integer.parseInt(num);
17             cnt += 1;}
18
19         mainVal = String.valueOf(sum) + "," + String.valueOf(cnt);
20
21     }
22
23
24     context.write(key,mainValue);
25
26
27
28
29
30
31 }
```

```
DriverClass.java Reducer2Class.java MapperClass.java CombinerClass.java
2
3
4     @Override
5     protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
6
7         int sum=0;
8         int cnt=0;
9
10
11         for(Text v: values) {
12             String nums = v.toString();
13             String num1 [] = nums.split(",");
14             sum += Integer.parseInt(num1[0]);
15             sum += Integer.parseInt(num1[1]);
16
17         }
18         float sum2 = (float)sum/cnt;
19
20
21         tmap.put(sum2,key.toString());
22
23
24
25
26         if (tmap.size() > 12)
27         {
28             tmap.remove(tmap.firstKey());
29         }
30     }
31 }
```

For every 10 crashes – 3 people are getting killed or injured.

```
February      0.2943835
January       0.3032084
March         0.30650815
December      0.34027913
April          0.34600237
May            0.3713675
November      0.3741418
June           0.3898202
October        0.3925074
July            0.39551663
September      0.40659606
August          0.4119512
madhav437@MacBook-Pro hadoop-3.3.1 %
```

Top Car manufacturers involved in crashes by percentage



The screenshot shows a code editor window with a dark theme. The code is written in Java and uses a series of if statements to map vehicle make codes from a log entry to specific car brands. The code is numbered from 34 to 70 on the left.

```
34     String [] vehMk = logs[7].split("-");
35
36     if (vehMk[0].equals("CHEV ")) {
37         vehicleMake = "CHEVROLET";
38     }
39
40     if (vehMk[0].equals("FORD ")) {
41         vehicleMake = "FORD";
42     }
43
44     if (vehMk[0].equals("NISS ")) {
45         vehicleMake = "NISSAN";
46     }
47
48     if (vehMk[0].equals("HOND ")) {
49         vehicleMake = "HONDA";
50     }
51
52     if (vehMk[0].equals("TOYT ")) {
53         vehicleMake = "TOYOTA";
54     }
55
56     if (vehMk[0].equals("KIA ")) {
57         vehicleMake = "KIA";
58     }
59
60     if (vehMk[0].equals("SUBA ")) {
61         vehicleMake = "SUBARU";
62     }
63
64     if (vehMk[0].equals("BMW ")) {
65         vehicleMake = "BMW";
66     }
67
68     if (vehMk[0].equals("AUDI ")) {
69         vehicleMake = "AUDI";
70 }
```

```
54     {
55
56         //Count Total Records
57
58         for (Map.Entry<Integer, String> entry : tmap.entrySet())
59         {
60
61             totSum += entry.getKey();
62
63
64
65         }
66
67
68         for (Map.Entry<Integer, String> entry : tmap.entrySet())
69         {
70
71             int count = entry.getKey();
72             double percentage = ((float)count/totSum)*100;
73
74             DecimalFormat df = new DecimalFormat("#,###,##0.00");
75
76             String per = String.valueOf(df.format(percentage)) + " %";
77
78             String name = entry.getValue();
79
80             context.write(new Text(name), new Text(per));
81
82         }
83
84     }
85
86 }
87
88 }
```

```
here applicable
CHEVROLET      8.87 %
FORD          15.40 %
NISSAN        19.55 %
HONDA          23.85 %
TOYOTA        32.33 %
madhav437@MacBook-Pro hadoop-3.3.1 %
```

Evaluate Bicyclists hit by vehicles

Bicyclists come from Persons dataset

Vehicles information comes from Vehicles dataset

Using Pig

```
crashes = LOAD 'hdfs://localhost:9000/NYC_Crashes' USING PigStorage(',')
as
(CRASH_DATE,CRASH_TIME,BOROUGH,ZIP_CODE,LATITUDE,LONGITUDE,LOCATION,ON_ST
REET_NAME,CROSS_STREET_NAME,OFF_STREET_NAME,NUMBER_OF_PERSONS_INJURED,N
UMBER_OF_PERSONS_KILLED,NUMBER_OF_PEDESTRIANS_INJURED,NUMBER_OF_PEDESTR
IANS_KILLED,NUMBER_OF_CYCLIST_INJURED,NUMBER_OF_CYCLIST_KILLED,NUMBER_OF_
MOTORIST_INJURED,NUMBER_OF_MOTORIST_KILLED,CONTRIBUTING_FACTOR_VEHICLE_1,
CONTRIBUTING_FACTOR_VEHICLE_2,CONTRIBUTING_FACTOR_VEHICLE_3,CONTRIBUTING_
FACTOR_VEHICLE_4,CONTRIBUTING_FACTOR_VEHICLE_5,COLLISION_ID,VEHICLE_TYPE_CO
```

```
DE_1,VEHICLE_TYPE_CODE_2,VEHICLE_TYPE_CODE_3,VEHICLE_TYPE_CODE_4,VEHICLE_TYPE_CODE_5);
```

```
persons = LOAD 'hdfs://localhost:9000/NYC_Persons' USING PigStorage(',')  
as  
(UNIQUE_ID,COLLISION_ID,CRASH_DATE,CRASH_TIME,PERSON_ID,PERSON_TYPE:chararray,  
PERSON_INJURY,VEHICLE_ID,PERSON_AGE,EJECTION,EMOTIONAL_STATUS,BODILY_INJURY,  
POSITION_IN_VEHICLE,SAFETY_EQUIPMENT,PED_LOCATION,PED_ACTION,COMPLAINT,PED  
_ROLE,CONTRIBUTING_FACTOR_1,CONTRIBUTING_FACTOR_2,PERSON_SEX);
```

```
filter_bicylists = FILTER persons BY PERSON_TYPE == 'Bicyclist';
```

```
crash_persons = JOIN crashes BY COLLISION_ID, filter_bicylists BY COLLISION_ID;
```

```
group_veh_hittingBicyclists = GROUP crash_persons by VEHICLE_TYPE_CODE_1;
```

```
count_veh = foreach group_veh_hittingBicyclists generate group, COUNT(crash_persons)as  
veh_cnt;
```

```
vehicle_order = ORDER count_veh BY veh_cnt DESC;
```

```
(Bike,530)  
(Sedan,247)  
(Station Wagon/Sport Utility Vehicle,216)  
(Taxi,33)  
(E-Bike,29)  
(E-Scooter,29)  
(Box Truck,10)  
(E-Bik,9)  
(Pick-up Truck,8)  
(Bus,5)  
(Convertible,5)  
(E-Sco,5)  
(Van,3)  
(Dump,2)  
(Moped,2)  
(Pedicab,2)  
(Flat Rack,1)  
(PK,1)  
(SCOOTER,1)  
(Motorcycle,1)  
(Horse,1)  
(Garbage or Refuse,1)  
(Motorbike,1)
```

```
STORE vehicle_order INTO 'hdfs://localhost:9000/NYC_Bicyclists';
```

Evaluate if persons ejected from their motor vehicles were using safety equipment

```
crashes = LOAD 'hdfs://localhost:9000/NYC_Crashes' USING PigStorage(',')  
as  
(CRASH_DATE,CRASH_TIME,BOROUGH,ZIP_CODE,LATITUDE,LONGITUDE,LOCATION,ON_STREET_NAME,CROSS_STREET_NAME,OFF_STREET_NAME,NUMBER_OF_PERSONS_INJURED,NUMBER_OF_PERSONS_KILLED,NUMBER_OF_PEDESTRIANS_INJURED,NUMBER_OF_PEDESTRIANS_KILLED,NUMBER_OF_CYCLIST_INJURED,NUMBER_OF_CYCLIST_KILLED,NUMBER_OF_
```

MOTORIST_INJURED,NUMBER_OF_MOTORIST_KILLED,CONTRIBUTING_FACTOR_VEHICLE_1,
CONTRIBUTING_FACTOR_VEHICLE_2,CONTRIBUTING_FACTOR_VEHICLE_3,CONTRIBUTING_
FACTOR_VEHICLE_4,CONTRIBUTING_FACTOR_VEHICLE_5,COLLISION_ID,VEHICLE_TYPE_CO
DE_1,VEHICLE_TYPE_CODE_2,VEHICLE_TYPE_CODE_3,VEHICLE_TYPE_CODE_4,VEHICLE_TYP
E_CODE_5);

```
persons = LOAD 'hdfs://localhost:9000/NYC_Persons' USING PigStorage(',')  
as  
(UNIQUE_ID,COLLISION_ID,CRASH_DATE,CRASH_TIME,PERSON_ID,PERSON_TYPE:chararray,  
PERSON_INJURY,VEHICLE_ID,PERSON_AGE,EJECTION,EMOTIONAL_STATUS,BODILY_INJURY,  
POSITION_IN_VEHICLE,SAFETY_EQUIPMENT,PED_LOCATION,PED_ACTION,COMPLAINT,PED  
_ROLE,CONTRIBUTING_FACTOR_1,CONTRIBUTING_FACTOR_2,PERSON_SEX);  
  
filter_ejection= FILTER persons BY EJECTION == 'Ejected';  
  
group_ejections = GROUP filter_ejection by SAFETY_EQUIPMENT;  
  
count_ejections = foreach group_ejections generate group, COUNT(filter_ejection)as  
saf_cnt;  
  
SafetyEquip = ORDER count_ejections BY saf_cnt DESC;  
  
STORE SafetyEquip INTO 'hdfs://localhost:9000/NYC_SafetyEq';
```

```
2021-12-15 22:33:12,822 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceInputFormat
(None,4004)
(Helmet (Motorcycle Only),1388)
(Unknown,1083)
(Helmet Only (In-Line Skater/Bicyclist),983)
(Helmet/Other (In-Line Skater/Bicyclist),307)
( or rear passenger on a bicycle,205)
(Lap Belt & Harness,132)
(Other,116)
( if two or more persons,60)
( or passenger lying across a seat",56)
(Lap Belt,39)
( pick-up truck,24)
(Air Bag Deployed,11)
(Air Bag Deployed/Lap Belt/Harness,6)
(Harness,4)
(Stoppers Only (In-Line Skater/Bicyclist),4)
(Pads Only (In-Line Skater/Bicyclist),2)
(Air Bag Deployed/Lap Belt,1)
(Child Restraint Only,1)
grunt>
```

Bloom filter to select Vehicle Types involved

```
String[] logs = line.split(",");
filter.add(key1);

if(logs.length>0){
    String vehicle = logs[6].trim();

    if (!vehicle.equals("VEHICLE_TYPE") && !vehicle.isEmpty()) {
        if (filter.membershipTest(new Key(vehicle.trim().getBytes())))
        {
            vehicleSelect.set(vehicle);

            context.write(vehicleSelect,one);
        }
    }
}
```

```
|madhav43@MacBook-Pro hadoop-3.3.1 % hadoop fs -cat /NYC_Out8/part-r-00000
2021-12-16 14:15:02,888 WARN util.NativeCodeLoader: Unable to load
Sedan      541318
madhav43@MacBook-Pro hadoop-3.3.1 %
```

Appendix – Source Codes

nyc_borough

```
package nyc_borough;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
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;
```

```

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,"logcount");
        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);
        job.setOutputFormatClass(TextOutputFormat.class);

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

        job.setMapperClass(MapperClass.class);
        job.setReducerClass(Reducer2Class.class);
        // job.setCombinerClass(ReducerClass.class);

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

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

package nyc_borough;

import java.io.IOException;

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

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

    //    hadoop datatype
    Text boroughs = new Text();
    IntWritable one = new IntWritable(1);

    @Override

```

```

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

        String line = value.toString();

        String[] logs = line.split(",");
        if(logs.length>0){

            String borough = logs[2];

            if (!borough.equals("BOROUGH") && !borough.isEmpty()) {
                boroughs.set(borough);
            }
            context.write(boroughs,one);
        }
    }

}

package nyc_borough;

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

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

public class Reducer2Class extends Reducer<Text, IntWritable, Text,
IntWritable> {

    private TreeMap<Integer, String> tmap;

    @Override
    public void setup(Context context) throws IOException,
                           InterruptedException
    {
        tmap = new TreeMap<Integer, String>();
    }

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

        int sum=0;
        for(IntWritable v: values)
            sum += v.get();

        tmap.put(sum, key.toString());
    }
}

```

```

        if (tmap.size() > 5)
        {
            tmap.remove(tmap.firstKey());
        }
    }

    @Override

    public void cleanup(Context context) throws IOException,
                           InterruptedException

    {

        for (Map.Entry<Integer, String> entry : tmap.entrySet())
        {

            int count = entry.getKey();
            String name = entry.getValue();

            context.write(new Text(name), new
IntWritable(count));
        }
    }

}

```

nyc_borough_compare2

```

package nyc_borough_compare2;

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

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

public class CompositeKeyComparator extends WritableComparator
{
    protected CompositeKeyComparator()
    {
        super(KeyCompare.class, true);

    @Override
    public int compare(WritableComparable w1, WritableComparable
w2) {
        KeyCompare k1 = (KeyCompare)w1;
        KeyCompare k2 = (KeyCompare)w2;
        int result = 0;

```

```

        try {
            result = k1.getBorough().compareTo(k2.getBorough());
            if(result == 0) {
                result = -
            }
            1*k1.getStreetName().compareTo(k2.getStreetName());
        }
    }
    catch (Exception e) {
        }
    return result;
}
}

package nyc_borough_compare2;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparator;
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;

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,"logcount");
        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.setPartitionerClass(NaturalKeyPartitioner.class);

        job.setGroupingComparatorClass(NaturalKeyGroupingComparator.class);
        job.setSortComparatorClass(CompositeKeyComparator.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

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

        job.setMapperClass(MapperClass.class);
    }
}

```

```

        job.setReducerClass(ReducerClass2.class);
        // job.setCombinerClass(ReducerClass.class);

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

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

    }

}

/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package nyc_borough_compare2;

import org.apache.hadoop.io.Text;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.EOFException;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableUtils;

public class KeyCompare implements WritableComparable<KeyCompare>{

    // private Date crashDate = new Date();
    private String borough;
    private String streetName;

    private final static SimpleDateFormat frmt = new
SimpleDateFormat("yyyy-MM-dd");

    public int compareTo(KeyCompare k) {

        int result = 0 ;
        try {
            result = this.borough.compareTo(k.getBorough());
            if(0 == result) {
                result = -
1*this.streetName.compareTo(k.getStreetName());
            }
        }
    }
}

```

```
        catch (Exception e) {
            }
        return result;
    }

@Override
public String toString() {
    return (new StringBuilder().append(borough).append("\t")
        .append(streetName)).toString();
}

public void readFields(DataInput dataInput) throws IOException
{
    borough = WritableUtils.readString(dataInput);
    streetName = WritableUtils.readString(dataInput);
}

public void write(DataOutput dataOutput) throws IOException {
    WritableUtils.writeString(dataOutput, borough);
    WritableUtils.writeString(dataOutput, streetName);
}

public String getStreetName() {
    return streetName;
}

public void setStreetName(String streetName) {
    this.streetName = streetName;
}

public String getBorough() {
    return borough;
}

public void setBorough(String borough) {
    this.borough = borough;
}

}

package nyc_borough_compare2;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.List;

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

```

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

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

    //    hadoop datatype
    KeyCompare kc = new KeyCompare();
    //private final static SimpleDateFormat frmt = new
SimpleDateFormat("yyyy-MM-dd");
    IntWritable one = new IntWritable(1);
    String crsDate;
    static final List<String> boroughs = Arrays.asList("BROOKLYN",
"QUEENS", "BRONX");

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

        String line = value.toString();

        String[] logs = line.split(",");
        if(logs.length>0 && boroughs.contains(logs[2])
        && !logs[8].isBlank()) {

            String borough = logs[2].trim();

            if (!borough.equals("BOROUGH") && !borough.isEmpty())  {

                kc.setBorough(borough);

                String crsDate = logs[8].trim();
                /* String[] dateSplit = crsDt.split("/");
                try {
                    crsDate =
dateSplit[2] + "/" + dateSplit[0] + "/" + dateSplit[1];
                } catch (Exception e) {} */
                kc.setStreetName(crsDate);

            }
            context.write(kc,one);
        }
    }
}

```

```

}

package nyc_borough_compare2;

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

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

public class NaturalKeyGroupingComparator extends WritableComparator
{
    protected NaturalKeyGroupingComparator()
    {
        super(KeyCompare.class, true);

        public int compare(WritableComparable w1, WritableComparable
w2) {
            KeyCompare k1 = (KeyCompare)w1;
            KeyCompare k2 = (KeyCompare)w2;
            return k1.getBorough().compareTo(k2.getBorough());}}
    }

package nyc_borough_compare2;

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

import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;

public class NaturalKeyPartitioner extends Partitioner<KeyCompare,
DoubleWritable>
{
@Override
public int getPartition(KeyCompare key, DoubleWritable val, int
numPartitions) {
int hash = key.getStreetName().hashCode();
int partition = hash % numPartitions;
return partition;
}
}

```

```

package nyc_borough_compare2;

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

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

public class ReducerClass2 extends Reducer<KeyCompare, IntWritable,
Text, IntWritable> {

    Text sortKey = new Text();
    private TreeMap<String, Integer> tmap;

    private TreeMap<Integer, String> tmapTemp;

    @Override
    public void setup(Context context) throws IOException,
                                         InterruptedException
    {
        tmap = new TreeMap<String, Integer>();
        tmapTemp = new TreeMap<Integer, String>();

    }

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

        int sum=0;
        String temp = " ";

        for(IntWritable v: values) {
            if (temp.equals(" "))
                temp = key.getStreetName();

            if (!temp.equals(key.getStreetName())) {
                temp = key.getStreetName();
                tmapTemp.put(sum, key.toString());

                // Add Top 5
                if (tmapTemp.size() > 10)
                {
                    tmapTemp.remove(tmapTemp.firstKey());
                }
                sum = 0;}
            sum += v.get();
        }

        //Add from Temp tmapTemp to Main tmap
    }
}

```

```

        for (Map.Entry<Integer, String> entry : tmapTemp.entrySet())
        {
            int count = entry.getKey();
            String name = entry.getValue();

            tmap.put(name, count);
            ;
        }

        tmapTemp.clear();
    }

    // context.write(key, new IntWritable(sum));

    @Override

    public void cleanup(Context context) throws IOException,
                           InterruptedException
    {

        for (Map.Entry<String, Integer> entry : tmap.entrySet())
        {
            int count = entry.getValue();
            String name = entry.getKey();

            context.write(new Text(name), new IntWritable(count));
        }
    }
}

```

nyc_crash_compare

```

package nyc_crash_compare;

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

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

```

```

public class CompositeKeyComparator extends WritableComparator {
    protected CompositeKeyComparator() {
        super(KeyCompare.class, true);}

    @Override
    public int compare(WritableComparable w1, WritableComparable w2) {
        KeyCompare k1 = (KeyCompare)w1;
        KeyCompare k2 = (KeyCompare)w2;
        int result = 0;
        try {
            result = k1.getStreetName().compareTo(k2.getStreetName());
            if(result == 0) {
                result = -1*k1.getCrashDate().compareTo(k2.getCrashDate());
            }
        } catch (Exception e) {
            }
        return result;
    }
}

```

```

package nyc_crash_compare;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparator;
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;

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,"logcount");
        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.setPartitionerClass(NaturalKeyPartitioner.class);

job.setGroupingComparatorClass(NaturalKeyGroupingComparator.class);
    job.setSortComparatorClass(CompositeKeyComparator.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

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

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

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

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

}

package nyc_crash_compare;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparator;
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;

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,"logcount");
        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.setPartitionerClass(NaturalKeyPartitioner.class);
    }
}

```

```

        job.setGroupingComparatorClass(NaturalKeyGroupingComparator.class);
        job.setSortComparatorClass(CompositeKeyComparator.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

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

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

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

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

    }

}

```

```

package nyc_crash_compare;

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

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

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

    // hadoop datatype
    KeyCompare kc = new KeyCompare();
    //private final static SimpleDateFormat frmt = new
SimpleDateFormat("yyyy-MM-dd");
    IntWritable one = new IntWritable(1);
    String crsDate;

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

        String line = value.toString();
        String[] logs = line.split(",");
        if(logs.length>0){


```

```

        String streetName = logs[8];

        if (!streetName.equals("ON STREET NAME")) {

            kc.setStreetName(streetName);

            String crsDt = logs[0];

            String[] dateSplit = crsDt.split("/");

            try {

                crsDate =
dateSplit[2] + "/" + dateSplit[0] + "/" + dateSplit[1];

            } catch (Exception e) {};

            kc.setCrashDate(crsDate);

        }

        context.write(kc,one);
    }
}

package nyc_crash_compare;

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

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

public class NaturalKeyGroupingComparator extends WritableComparator
{
    protected NaturalKeyGroupingComparator()
    {
        super(KeyCompare.class, true);

        public int compare(WritableComparable w1, WritableComparable
w2) {
            KeyCompare k1 = (KeyCompare)w1;
            KeyCompare k2 = (KeyCompare)w2;
            return k1.getStreetName().compareTo(k2.getStreetName());}}

```

```

package nyc_crash_compare;

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

import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;

public class NaturalKeyPartitioner extends Partitioner<KeyCompare,
DoubleWritable>
{
@Override
public int getPartition(KeyCompare key, DoubleWritable val, int
numPartitions) {
int hash = key.getStreetName().hashCode();
int partition = hash % numPartitions;
return partition;
}
}

package nyc_crash_compare;

import java.io.IOException;

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

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

    Text sortKey = new Text();

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

        int sum=0;
        for(IntWritable v: values) {
            sum += v.get();}

        context.write(key, new IntWritable(sum));
    }
}

nyc_crash_compare2;

package nyc_crash_compare2;

```

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
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;

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,"logcount");
        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);
        job.setOutputFormatClass(TextOutputFormat.class);

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

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

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

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

    }
}

package nyc_crash_compare2;

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

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

```

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

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

    private final static SimpleDateFormat frmt = new
SimpleDateFormat("yyyy-MM-dd");

    //    hadoop datatype
    Text street = new Text();
    IntWritable one = new IntWritable(1);

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

        String line = value.toString();

        String[] logs = line.split(",");
        if(logs.length>0){

            //String streetName = logs[8];

            String crsDate = logs[0];
            if (!crsDate.equals("CRASH DATE")) {

                String[] dateSplit = crsDate.split("/");

                street.set(dateSplit[2] + "/" + dateSplit[0] + "/" + dateSplit[1]);
            }

            /* try {
                crsDate = frmt.parse(logs[0]).toString();
            } catch (ParseException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            if (!crsDate.equals("CRASH DATE")) {
                street.set(crsDate);
            }*/
            context.write(street,one);
        }
    }

}

package nyc_crash_compare2;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;

```

```

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

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

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

        int sum=0;
        for(IntWritable v: values)
            sum += v.get();
        context.write(key, new IntWritable(sum));
    }

}

```

nyc_crash_month

```

package nyc_crash_month;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
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;

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,"logcount");
        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);
        job.setOutputFormatClass(TextOutputFormat.class);

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

        job.setMapperClass(MapperClass.class);
        job.setReducerClass(Reducer2Class.class);
    }
}

```

```

// job.setCombinerClass(ReducerClass.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

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

}

package nyc_crash_month;

import java.io.IOException;

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

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

    //    hadoop datatype
    Text monthSet = new Text();
    IntWritable one = new IntWritable(1);
    String monthString = "";

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

        String line = value.toString();

        String[] logs = line.split(",");
        if(logs.length>0 && !logs[1].isBlank() &&
!logs[0].equals("CRASH DATE"))

        {

            String[] months = logs[0].split("/");
            int month = Integer.parseInt(months[0]);

            switch (month) {
                case 1: monthString = "January";
                break;
                case 2: monthString = "February";
                break;
                case 3: monthString = "March";
                break;
                case 4: monthString = "April";
                break;
                case 5: monthString = "May";
            }
        }
    }
}

```

```

        break;
    case 6: monthString = "June";
        break;
    case 7: monthString = "July";
        break;
    case 8: monthString = "August";
        break;
    case 9: monthString = "September";
        break;
    case 10: monthString = "October";
        break;
    case 11: monthString = "November";
        break;
    case 12: monthString = "December";
        break;
    default: monthString = "Invalid month";
        break;
    }

    monthSet.set(monthString);

    context.write(monthSet,one);

}

}

package nyc_crash_month;

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

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

public class Reducer2Class extends Reducer<Text, IntWritable, Text,
IntWritable> {

    private TreeMap<Integer, String> tmap;

    @Override
    public void setup(Context context) throws IOException,
                                         InterruptedException
    {
        tmap = new TreeMap<Integer, String>();
    }

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

        int sum=0;
        for(IntWritable v: values)
            sum += v.get();
    }
}

```

```

        tmap.put(sum, key.toString());

    if (tmap.size() > 12)
    {
        tmap.remove(tmap.firstKey());
    }
}

@Override

public void cleanup(Context context) throws IOException,
    InterruptedException
{

    for (Map.Entry<Integer, String> entry : tmap.entrySet())
    {

        int count = entry.getKey();
        String name = entry.getValue();

        context.write(new Text(name), new
IntWritable(count));
    }
}

package nyc_crash_time;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
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;

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,"logcount");

```

```

        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);
        job.setOutputFormatClass(TextOutputFormat.class);

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

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

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

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

    }

}

package nyc_crash_time;

import java.io.IOException;

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

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

    //    hadoop datatype
    Text timeOfDay = new Text();
    IntWritable one = new IntWritable(1);
    String timeDay = " ";

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

        String line = value.toString();
        String[] logs = line.split(",");

```

```
if(logs.length>0 && !logs[1].isBlank() &&
!logs[1].equals("CRASH TIME"))

{

    String[] time = logs[1].split(":");
    int hr = Integer.parseInt(time[0]);
    if (hr > 3 && hr < 12) {

        timeDay = "Morning";
    }

    if (hr > 11 && hr < 5) {

        timeDay = "Afternoon";
    }

    if (hr > 4 && hr < 7) {

        timeDay = "Evening";
    }

    if (hr > 6 && hr < 4) {

        timeDay = "Night";
    }

    if (!timeDay.isBlank()){

        timeOfDay.set(timeDay);
        context.write(timeOfDay,one);
    }
}

}

package nyc_crash_time;

import java.io.IOException;
import java.util.Map;
import java.util.TreeMap;
```

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

public class Reducer2Class extends Reducer<Text, IntWritable, Text,
IntWritable> {

    private TreeMap<Integer, String> tmap;

    @Override
    public void setup(Context context) throws IOException,
                                         InterruptedException
    {
        tmap = new TreeMap<Integer, String>();
    }

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

        int sum=0;
        for(IntWritable v: values)
            sum += v.get();

        tmap.put(sum, key.toString());

        if (tmap.size() > 5)
        {
            tmap.remove(tmap.firstKey());
        }
    }

    @Override

    public void cleanup(Context context) throws IOException,
                                         InterruptedException
    {

        for (Map.Entry<Integer, String> entry : tmap.entrySet())
        {
            int count = entry.getKey();
            String name = entry.getValue();

            context.write(new Text(name), new
IntWritable(count));
        }
    }
}
```

```
}
```

nyc_factors

```
package nyc_factors;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
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;

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,"logcount");
        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);
        job.setOutputFormatClass(TextOutputFormat.class);

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

        job.setMapperClass(MapperClass.class);
        job.setReducerClass(Reducer2Class.class);
        // job.setCombinerClass(ReducerClass.class);

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

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

```

}

package nyc_factors;

import java.io.IOException;

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

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

    //    hadoop datatype
    Text vehicleType = new Text();
    IntWritable one = new IntWritable(1);

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

        String line = value.toString();

        String[] logs = line.split(",");
        if(logs.length>0){

            String veh_type = logs[6];

            if (!veh_type.equals("VEHICLE_TYPE")) {
                vehicleType.set(veh_type);
            }
            context.write(vehicleType,one);
        }
    }

}

package nyc_factors;

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

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

public class Reducer2Class extends Reducer<Text, IntWritable, Text,
IntWritable> {

```

```

private TreeMap<Integer, String> tmap;

@Override
public void setup(Context context) throws IOException,
                           InterruptedException
{
    tmap = new TreeMap<Integer, String>();
}

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

    int sum=0;
    for(IntWritable v: values)
        sum += v.get();

    tmap.put(sum, key.toString());
}

if (tmap.size() > 10)
{
    tmap.remove(tmap.firstKey());
}
}

@Override

public void cleanup(Context context) throws IOException,
                           InterruptedException
{

    for (Map.Entry<Integer, String> entry : tmap.entrySet())
    {

        int count = entry.getKey();
        String name = entry.getValue();

        context.write(new Text(name), new
IntWritable(count));
    }
}

nyc_mf_compare

package nyc_mf_compare;

```

```

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

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

public class CompositeKeyComparator extends WritableComparator
{
    protected CompositeKeyComparator()
    {
        super(KeyCompare.class, true);}

    @Override
    public int compare(WritableComparable w1, WritableComparable
w2) {
        KeyCompare k1 = (KeyCompare)w1;
        KeyCompare k2 = (KeyCompare)w2;
        int result = 0;
        try {
            result = k1.getMf().compareTo(k2.getMf());
            if(result == 0) {
                result = -1*k1.getConFactor().compareTo(k2.getConFactor());
            }
        }
        catch (Exception e) {
            }
        return result;
    }
}

package nyc_mf_compare;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparator;
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;

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,"logcount");
        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.setPartitionerClass(NaturalKeyPartitioner.class);

job.setGroupingComparatorClass(NaturalKeyGroupingComparator.class);
job.setSortComparatorClass(CompositeKeyComparator.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

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

job.setMapperClass(MapperClass.class);
job.setReducerClass(ReducerClass2.class);
// job.setCombinerClass(ReducerClass.class);

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

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

    }

}

/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package nyc_mf_compare;

import org.apache.hadoop.io.Text;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.EOFException;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableUtils;

public class KeyCompare implements WritableComparable<KeyCompare>{

```

```
// private Date crashDate = new Date();
// private String mf;
private String conFactor;

private final static SimpleDateFormat frmt = new
SimpleDateFormat("yyyy-MM-dd");

public int compareTo(KeyCompare k) {

    int result = 0 ;
    try {
        result = this.mf.compareTo(k.getMf());
        if(0 == result) {
            result = -
1*this.conFactor.compareTo(k.getConFactor());
        }
    } catch (Exception e) {
        }
    return result;
}

@Override
public String toString() {
    return (new StringBuilder().append(mf).append("\t")
        .append(conFactor)).toString();
}

public void readFields(DataInput dataInput) throws IOException
{
    mf = WritableUtils.readString(dataInput);
    conFactor = WritableUtils.readString(dataInput);
}

public void write(DataOutput dataOutput) throws IOException {
    WritableUtils.writeString(dataOutput, mf);
    WritableUtils.writeString(dataOutput, conFactor);
}

public String getConFactor() {
    return conFactor;
}

public void setConFactor(String conFactor) {
    this.conFactor = conFactor;
}

public String getMf() {
```

```

        return mf;
    }

    public void setMf(String mf) {
        this.mf = mf;
    }

}

package nyc_mf_compare;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.List;

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

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

    //    hadoop datatype
    KeyCompare kc = new KeyCompare();
    //private final static SimpleDateFormat frmt = new
SimpleDateFormat("yyyy-MM-dd");
    IntWritable one = new IntWritable(1);
    String crsDate;

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

        String line = value.toString();
        String[] logs = line.split(",");
        String mf = logs[12].trim();

        if(logs.length>23
            && !logs[12].isBlank()
            && !mf.equals("DRIVER_SEX")

```

```

        && !logs[23].isBlank()
        //  && !mf.equals("U"))

    {

        kc.setMf(mf);

        String crsDate = logs[23].trim();

        /* String[] dateSplit = crsDt.split("/");
        try {

            crsDate =
dateSplit[2] + "/" + dateSplit[0] + "/" + dateSplit[1];

        } catch (Exception e) {} */

        kc.setConFactor(crsDate);

        context.write(kc,one);
    }

}

package nyc_mf_compare;

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

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

public class NaturalKeyGroupingComparator extends WritableComparator
{
    protected NaturalKeyGroupingComparator()
    {
        super(KeyCompare.class, true);}

    public int compare(WritableComparable w1, WritableComparable w2) {
        KeyCompare k1 = (KeyCompare)w1;
        KeyCompare k2 = (KeyCompare)w2;
        return k1.getMf().compareTo(k2.getMf());}}

package nyc_mf_compare;

```

```

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

import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;

public class NaturalKeyPartitioner extends Partitioner<KeyCompare,
DoubleWritable>
{
@Override
public int getPartition(KeyCompare key, DoubleWritable val, int
numPartitions) {
int hash = key.getMf().hashCode();
int partition = hash % numPartitions;
return partition;
}
}

package nyc_mf_compare;

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

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

public class ReducerClass2 extends Reducer<KeyCompare, IntWritable,
Text, IntWritable> {

    Text sortKey = new Text();
    private TreeMap<String, Integer> tmap;

    private TreeMap<Integer, String> tmapTemp;

    @Override
        public void setup(Context context) throws IOException,
InterruptedException
    {
        tmap = new TreeMap<String, Integer>();
        tmapTemp = new TreeMap<Integer, String>();
    }

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

```

```

int sum=0;
String temp = " ";

for(IntWritable v: values) {
    if (temp.equals(" "))
        temp = key.getConFactor();

    if (!temp.equals(key.getConFactor())) {
        temp = key.getConFactor();
        tmapTemp.put(sum,key.toString());

        // Add Top 5
        if (tmapTemp.size() > 3)
        {
            tmapTemp.remove(tmapTemp.firstKey());
        }
        sum = 0;
    }

    sum += v.get();
}

//Add from Temp tmapTemp to Main tmap

for (Map.Entry<Integer,String> entry : tmapTemp.entrySet())
{
    int count = entry.getKey();
    String name = entry.getValue();

    tmap.put(name,count);
    ;
}

tmapTemp.clear();
}

// context.write(key, new IntWritable(sum));

@Override

public void cleanup(Context context) throws IOException,
InterruptedException
{

    for (Map.Entry<String,Integer> entry : tmap.entrySet())
{

```

```
        int count = entry.getValue();
        String name = entry.getKey();

        context.write(new Text(name), new IntWritable(count));
    }
}


```

nyc_month_avg

```
package nyc_month_avg;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
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;

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,"logcount");
        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);
        job.setOutputFormatClass(TextOutputFormat.class);

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

        job.setMapperClass(MapperClass.class);
        job.setReducerClass(Reducer2Class.class);
        job.setCombinerClass(ReducerClass.class);

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

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

```

    }

}

package nyc_month_avg;

import java.io.IOException;

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

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

    //    hadoop datatype
    Text monthSet = new Text();
    Text mainValue = new Text();
    // IntWritable one = new IntWritable(1);
    String monthString = "";
    String num1 = " ";
    String num2 = " ";
    int persons = 0;

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

        String line = value.toString();

        String[] logs = line.split(",");

        if(logs.length>0 && !logs[1].isBlank() &&
!logs[0].equals("CRASH DATE"))

    {

        String[] months = logs[0].split("/");

        int month = Integer.parseInt(months[0]);
        int index = line.indexOf('"'');

        if (index>0) {
            num1 = logs[11].trim();
            num2 = logs[12].trim();

        }
        else {
            num1 = logs[10].trim();
            num2 = logs[11].trim();
        }
    }
}

```

```

        if (num1.isBlank())
            num1 = "0";

        if (num2.isBlank())
            num2 = "0";

switch (month) {
    case 1: monthString = "January";
    break;
    case 2: monthString = "February";
    break;
    case 3: monthString = "March";
    break;
    case 4: monthString = "April";
    break;
    case 5: monthString = "May";
    break;
    case 6: monthString = "June";
    break;
    case 7: monthString = "July";
    break;
    case 8: monthString = "August";
    break;
    case 9: monthString = "September";
    break;
    case 10: monthString = "October";
    break;
    case 11: monthString = "November";
    break;
    case 12: monthString = "December";
    break;
    default: monthString = "Invalid month";
    break;
}
persons = Integer.parseInt(num1) +
Integer.parseInt(num2);

monthSet.set(monthString);
mainValue.set(String.valueOf(persons));

context.write(monthSet,mainValue);

}

}

}

package nyc_month_avg;

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

import org.apache.hadoop.io.FloatWritable;

```

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

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

    Text mainValue = new Text();
    String mainVal = " ";

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

        int sum=0;
        int cnt=0;
        String num = " ";

        for(Text v: values) {
            num = v.toString();
            sum += Integer.parseInt(num);
            cnt += 1;
        }

        mainVal = String.valueOf(sum) + "," + String.valueOf(cnt);

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

```
package nyc_month_avg;

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

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

```

public class Reducer2Class extends Reducer<Text, Text, Text,
FloatWritable> {

    private TreeMap<Float, String> tmap;

    @Override
    public void setup(Context context) throws IOException,
                           InterruptedException
    {
        tmap = new TreeMap<Float, String>();
    }

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

        int sum=0;
        int cnt=0;

        for(Text v: values) {
            String nums = v.toString();
            String num1 [] = nums.split(",");
            sum += Integer.parseInt(num1[0]);
            sum += Integer.parseInt(num1[1]);

        }
        float sum2 = (float)sum/cnt;

        tmap.put(sum2,key.toString());

        if (tmap.size() > 12)
        {
            tmap.remove(tmap.firstKey());
        }
    }

    @Override

    public void cleanup(Context context) throws IOException,
                           InterruptedException
    {

        for (Map.Entry<Float, String> entry : tmap.entrySet())
        {
            float count = entry.getKey();
            String name = entry.getValue();
        }
    }
}

```

```

        context.write(new Text(name), new
FloatWritable(count));
    }
}

package nyc_ped_compare;

import java.io.IOException;

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

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

    //    hadoop datatype
    Text collisionId = new Text();
    Text values = new Text();
    String tempVal = " ";
    String colId = " ";

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

        String line = value.toString();

        String[] logs = line.split(",");
        int index = line.indexOf('\'');

        if (index>0) {
            tempVal = logs[2].trim() + "," + logs[8].trim();
            colId = logs[24];
        }else {
            if(logs.length>23 && !logs[1].equals("CRASH DATE")){
                tempVal = logs[2].trim() + ","+ logs[7].trim();
                colId = logs[23];
            }
        }

        if(logs.length>23
        && !logs[2].isBlank()
        // && !colId.isBlank()
        && !logs[2].equals("BOROUGH"))
        // && colId.equals("4230968"))

```

```

        collisionId.set(colId);
        tempVal = "Y" + " " + tempVal ;
        values.set(tempVal);

        context.write(collisionId,values);
    }

}

```

nyc_ped_compare

```

package nyc_ped_compare;

import java.io.IOException;

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

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

    //    hadoop datatype
    Text collisionId = new Text();
    Text values = new Text();
    String tempVal = " ";
    String colId = " ";

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

        String line = value.toString();

        String[] logs = line.split(",");
        int index = line.indexOf('\"');

        if (index>0) {
            tempVal = logs[2].trim() + "," + logs[8].trim();
            colId = logs[24];
        }else {
            if(logs.length>23 && !logs[1].equals("CRASH DATE")){
                tempVal = logs[2].trim() + ","+ logs[7].trim();
                colId = logs[23];
            }
        }
    }
}

```

```

        if(logs.length>23
            && !logs[2].isBlank()
            // && !colId.isBlank()
            && !logs[2].equals("BOROUGH"))
            // && colId.equals("4230968"))

    {

        collisionId.set(colId);
        tempVal = "Y" + " " + tempVal ;
        values.set(tempVal);

        context.write(collisionId,values);
    }

}

package nyc_ped_compare;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
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 org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
}

public class DriverClass2 extends Configured implements Tool {
    private static final String DATA_SEPARATOR = ",";
    public int run(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        //
        configuration.set("mapreduce.output.textoutputformat.separator",
        DATA_SEPARATOR);
        Job job = Job.getInstance(configuration);
        job.setJobName("New Job");
        job.setJarByClass(DriverClass2.class);
        // Map
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        // Job
        job.setOutputKeyClass(LongWritable.class);
        job.setOutputValueClass(Text.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setReducerClass(ReducerClass.class);
    }
}

```

```

        MultipleInputs.addInputPath(job, new Path(args[0]),
TextInputFormat.class,
CrashMapperClass.class);
        MultipleInputs.addInputPath(job, new Path(args[1]),
TextInputFormat.class,
PersonMapperClass.class);
        FileOutputFormat.setOutputPath(job, new Path(args[2]));
        job.waitForCompletion(true);
        return 0;
    }
    public static void main(String[] args) throws Exception {
        if (args.length == 3) {
            int result = ToolRunner.run(new Configuration(), new
DriverClass2(), args);
            if (0 == result) {
                System.out.println("");
            } else {
                System.out.println("");
            }
        } else {
            System.out.println("USAGE
<InputPath1><InputPath2><OutputPath>");
        }
    }
}

package nyc_ped_compare;

import java.io.IOException;

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

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

    //    hadoop datatype
    Text collisionId = new Text();
    Text values = new Text();
    String tempVal= " ";
    String tempVal1= " ";

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

        String line = value.toString();

        String[] logs = line.split(",");
        if(logs.length>0

```



```

        tmap = new TreeMap<Integer, String>();
    }

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

        int sum=0;
        for(IntWritable v: values)
            sum += v.get();

        tmap.put(sum,key.toString());

        if (tmap.size() > 5)
        {
            tmap.remove(tmap.firstKey());
        }
    }

    @Override

    public void cleanup(Context context) throws IOException,
                           InterruptedException
    {

        for (Map.Entry<Integer, String> entry : tmap.entrySet())
        {
            int count = entry.getKey();
            String name = entry.getValue();

            context.write(new Text(name), new
IntWritable(count));
        }
    }

}

package nyc_ped_compare;

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

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

```

```

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

    private ArrayList<Text> listA = new ArrayList<Text>();
    private ArrayList<Text> listB = new ArrayList<Text>();

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

        for(Text v: values) {
            if (v.charAt(0) == 'Y')
                listA.add(new Text(v.toString().substring(1)+ ","));
            else if (v.charAt(0) == 'Z')
                listB.add(new Text(v.toString().substring(1)));
        }

        executeJoin(context);
    }
}

private void executeJoin(Context context) throws IOException,
InterruptedException{
    if(!listA.isEmpty() && !listB.isEmpty())
    {
        for(Text A: listA){
            for(Text B : listB)
            {
                context.write(A,B);
            }
        }
    }
}

```

```

nyc_ped_compare2
package nyc_ped_compare2;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
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;

```

```

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,"logcount");
        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);
        job.setOutputFormatClass(TextOutputFormat.class);

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

        job.setMapperClass(MapperClass.class);
        job.setReducerClass(Reducer2Class.class);
        // job.setCombinerClass(ReducerClass.class);

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

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

package nyc_ped_compare2;

import java.io.IOException;

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

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

    //    hadoop datatype
    Text location = new Text();
    IntWritable one = new IntWritable(1);

    @Override

```

```

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

    String line = value.toString();

    String[] logs = line.split(",");
    if(logs.length>0){

        String loc = logs[0] + "-" + logs[1];

        location.set(loc);

        context.write(location,one);
    }
}

package nyc_ped_compare2;

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

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

public class Reducer2Class extends Reducer<Text, IntWritable, Text,
IntWritable> {

    private TreeMap<Integer, String> tmap;

    @Override
    public void setup(Context context) throws IOException,
                           InterruptedException
    {
        tmap = new TreeMap<Integer, String>();
    }

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

        int sum=0;
        for(IntWritable v: values)
            sum += v.get();

        tmap.put(sum,key.toString());
    }
}

```

```

        if (tmap.size() > 15)
        {
            tmap.remove(tmap.firstKey());
        }
    }

    @Override

    public void cleanup(Context context) throws IOException,
                           InterruptedException
    {

        for (Map.Entry<Integer, String> entry : tmap.entrySet())
        {

            int count = entry.getKey();
            String name = entry.getValue();

            context.write(new Text(name), new
IntWritable(count));
        }
    }
}

```

```

package nyc_ped_compare2;

import java.io.IOException;

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

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

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

        int sum=0;
        for(IntWritable v: values)
            sum += v.get();
        context.write(key, new IntWritable(sum));
    }
}

nyc_vehicles

package nyc_vehicles;

```

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
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;

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,"logcount");
        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);
        job.setOutputFormatClass(TextOutputFormat.class);

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

        job.setMapperClass(MapperClass.class);
        job.setReducerClass(Reducer2Class.class);
        // job.setCombinerClass(ReducerClass.class);

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

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

    }
}

package nyc_vehicles;

import java.io.IOException;

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

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

    //    hadoop datatype
    Text vehicle = new Text();
    IntWritable one = new IntWritable(1);
    String vehicleMake = " ";

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

        String line = value.toString();
        String[] logs = line.split(",");

        if(logs.length>0){

            if (!logs[7].equals("VEHICLE_TYPE") &&
            !logs[7].isEmpty()) {
                //String vehicleMake = logs[7].substring(0,4);

                String [] vehMk = logs[7].split("-");
                if (vehMk[0].equals("CHEV ")){
                    vehicleMake = "CHEVROLET";
                }
                if (vehMk[0].equals("FORD ")){
                    vehicleMake = "FORD";
                }
                if (vehMk[0].equals("NISS ")){
                    vehicleMake = "NISSAN";
                }
                if (vehMk[0].equals("HOND ")){
                    vehicleMake = "HONDA";
                }
                if (vehMk[0].equals("TOYT ")){
                    vehicleMake = "TOYOTA";
                }
            }
        }
    }
}
```

```

        vehicle.set(vehicleMake);
    }
    context.write(vehicle,one);
}
}

package nyc_vehicles;

import java.io.IOException;
import java.text.DecimalFormat;
import java.util.Map;
import java.util.TreeMap;

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

public class Reducer2Class extends Reducer<Text, IntWritable, Text,
Text> {

    private TreeMap<Integer,String> tmap;
    long totSum = 0;

    @Override
    public void setup(Context context) throws IOException,
                                         InterruptedException
    {
        tmap = new TreeMap<Integer,String>();
    }

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

        int sum=0;
        for(IntWritable v: values)
            sum += v.get();

        tmap.put(sum,key.toString());

        if (tmap.size() > 5)
        {
            tmap.remove(tmap.firstKey());
        }
    }
}
```

```

    }

    @Override

    public void cleanup(Context context) throws IOException,
                           InterruptedException

    {

        //Count Total Records

        for (Map.Entry<Integer, String> entry : tmap.entrySet())
        {

            totSum += entry.getKey();

        }

        for (Map.Entry<Integer, String> entry : tmap.entrySet())
        {

            int count = entry.getKey();
            double percentage = ((float)count/totSum)*100;

            DecimalFormat df = new
DecimalFormat("#,###,##0.00");

            String per =  String.valueOf(df.format(percentage))
+ " %" ;

            String name = entry.getValue();
            context.write(new Text(name), new Text(per));
        }
    }
}

```

nyc_bloom

```

package nyc_bloom;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
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;

public class DriverClass {
    public static void main(String[] args) throws IOException,
InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();
        conf.set("bloom_filter_file_location",args[2]);
        // Create a new Job
        Job job = Job.getInstance(conf,"logcount");
        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);
        job.setOutputFormatClass(TextOutputFormat.class);

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

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

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

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

package nyc_bloom;

import java.io.DataInputStream;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
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 org.apache.hadoop.util.bloom.BloomFilter;
import org.apache.hadoop.util.bloom.Key;
import org.apache.hadoop.util.hash.Hash;

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

    //    hadoop datatype
    Text vehicleSelect = new Text();
    IntWritable one = new IntWritable(1);
    DataInputStream dataInputStream;

    private BloomFilter filter = new
BloomFilter(40,10,Hash.MURMUR_HASH);

    static String test = "Sedan";

    static Key key1 = new Key(test.getBytes());

    protected void setup(Context context) throws IOException,
InterruptedException, EOFException{

        // Path[] localFiles =
DistributedCache.getLocalCacheFiles(context.getConfiguration().get("bloom_filter_file_location"));

        /*try {
            dataInputStream = new DataInputStream(
new
FileInputStream(context.getConfiguration().get("bloom_filter_file_lo
cation")));
            filter.readFields(dataInputStream);
            //dataInputStream.close();
        }catch (EOFException e) {
            dataInputStream.close();
        }*/
    }

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

        String line = value.toString();

        String[] logs = line.split(",");

        filter.add(key1);

        if(logs.length>0){
```

```

String vehicle = logs[6].trim();

        if (!vehicle.equals("VEHICLE_TYPE") &&
!vehicle.isEmpty()) {

            if (filter.membershipTest(new
Key(vehicle.trim().getBytes())))
{
                vehicleSelect.set(vehicle);

                    context.write(vehicleSelect,one);}
}
}

}

```

```

package nyc_bloom;

import java.io.IOException;

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

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

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

        int sum=0;
        for(IntWritable v: values)
            sum += v.get();
        context.write(key, new IntWritable(sum));
    }
}

```

