NAME: KAUSHAL CHAUDHARY
NUID: 001886763

AIRLINE ON-TIME PERFORMANCE

Taking an airplane is one of the most important and efficient ways to travel. However, many travelers have experienced delayed flight. Which airline carriers delayed most often? Which airports have highest probability to make you wait for a long time? Also, which day of week and which month of year are better for your journey without severe delay?

The aim of this presentation is to produce a summary of the airline performance data. Due to the large size of the data set, the author retrieved the recent 11 years' (1998-2008) data for analysis.

Use Cases:

MapReduce

- Percentage flights that were delayed every year
- Top 10 Routes that Experience delays
- Cancellation Analysis
- Partitioning Delays based on Day of the Week Month and year
- Top 10 Best Airport every year

Pig

- Top 10 Carriers with Minimum Delay
- Which Carrier accounted maximum delay every month in the year 2008
- How many flights that were delayed due to bad weather between 2005 – 2008
- Which Routes experienced most diversions

Data Set Link: http://stat-computing.org/dataexpo/2009/the-data.html

Supplementary Data Link: http://stat-computing.org/dataexpo/2009/supplemental-data.html

The data comprises of flight performance and the delays caused due to different factors for every single business trip inside the USA, from October 2000 to April 2008. This is a large dataset, nearly 70 million records in total and takes up to 6 gigabytes when uncompressed. The dataset consists of different null values across various features and in order to do data processing, the null values needs to be handled correctly.

9 years of data in a csv format:

2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008

Other supplement data:

- *carriers.csv* – Listing of carrier codes with full names.
- *airports.csv* – Describes the locations of US airports.
- *plane-data.csv* – Individual plane specification and

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Variable descriptions

	Name	*Description*
1	Year	1987-2008
2	Month	1-12
3	DayofMonth	1-31
4	DayOfWeek	1 (Monday) - 7 (Sunday)
5	DepTime	actual departure time (local, hhmm)
6	CRSDepTime	scheduled departure time (local, hhmm)
7	ArrTime	actual arrival time (local, hhmm)
8	CRSArrTime	scheduled arrival time (local, hhmm)
9	UniqueCarrier	unique carrier code
10	FlightNum	flight number
11	TailNum	plane tail number
12	ActualElapsedTime	in minutes
13	CRSElapsedTime	in minutes
14	AirTime	in minutes
15	ArrDelay	arrival delay, in minutes
16	DepDelay	departure delay, in minutes
17	Origin	origin IATA airport code
18	Dest	destination IATA airport code
19	Distance	in miles
20	TaxiIn	taxi in time, in minutes
21	TaxiOut	taxi out time in minutes
22	Cancelled	was the flight cancelled?
23	CancellationCode	reason for cancellation (A = carrier, B = weather, C = NAS, D = security)
24	Diverted	1 = yes, 0 = no
25	CarrierDelay	in minutes
26	WeatherDelay	in minutes
27	NASDelay	in minutes
28	SecurityDelay	in minutes
29	LateAircraftDelay	in minutes

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

USE CASES:

MERGING DATA USING PUT MERGE METHOD

To make mapreduce jobs more optimized I decided to merge all the data files into one big file so that the processing time will increase. To merge this data I used put merge technique which introduced in class. It's a good technique to merge the data from local and store it into HDFS.

Screenshot:

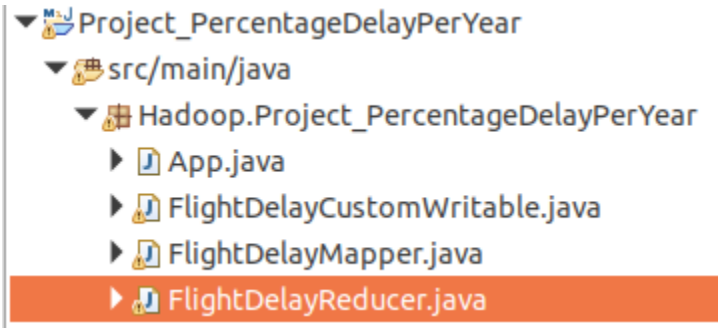
```
public static void main(String[] args) throws IOException {

  Configuration conf = new Configuration();
  FileSystem hdfs  = FileSystem.get(conf);
  FileSystem local = FileSystem.getLocal(conf);

  Path inputDir = new Path(args[0]);  // Specify input directory
  Path hdfsFile = new Path(args[1]);  // Specify output file

  try {
    // Get list of local files
    FileStatus[] inputFiles = local.listStatus(inputDir);
    // Create HDFS output stream
    FSDataOutputStream out = hdfs.create(hdfsFile);

    for (int i=0; i<inputFiles.length; i++) {
      System.out.println(inputFiles[i].getPath().getName());
      // Open local input stream
      FSDataInputStream in = local.open(inputFiles[i].getPath());

      // Copy local file to HDFS
      byte buffer[] = new byte[256];
      int bytesRead = 0;
      while( (bytesRead = in.read(buffer)) > 0) {
          out.write(buffer, 0, bytesRead);
      }
      in.close();
    }
    out.close();
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Percentage flights that were delayed every year

To see how many planes that experience delays over the years, calculating the percentage of delays every year would have been an ideal metric to give us a holistic view of the dataset. Not only it will tell us whether there was an increase or decrease in the percentage delay every year but it will also give us a good idea of how a MapReduce algorithm will process this long 6 GB of dataset.

Total Number of Classes



CustomWritableComparable:

In order to calculate the percentage, I had to pass two values (Count and Sum) from the mapper. Therefore I used CustomWritable as a value that will propagate the values to reducer phase.

```
public class FlightDelayCustomWritable implements WritableComparable {

    double totalCount;
    double totalDelay;

    public FlightDelayCustomWritable() {
        totalCount = 0.0;
        totalDelay = 0.0;
    }

    public FlightDelayCustomWritable(double totalCount, double totalDelay) {
        super();
        this.totalCount = totalCount;
        this.totalDelay = totalDelay;
    }

    public double getTotalCount() {
        return totalCount;
    }

    public void setTotalCount(double totalCount) {
        this.totalCount = totalCount;
    }

    public double getTotalDelay() {
        return totalDelay;
    }

    public void setTotalDelay(double totalDelay) {
        this.totalDelay = totalDelay;
    }

    public void readFields(DataInput in) throws IOException {
        totalCount = in.readDouble();
        totalDelay = in.readDouble();
    }

    public void write(DataOutput out) throws IOException {
        out.writeDouble(totalCount);
        out.writeDouble(totalDelay);
    }
```

Mapper:

The Mapper Function was basic mapping step. It read the TextInputFormat and mapped the count = 1 where there was delay experienced along with the line count.

```
String[] tokens = value.toString().split(",");
String year = tokens[0];
if(tokens[14].isEmpty()||tokens[14].equalsIgnoreCase("NA")) {
     delay = 0.0;
}
else {
    delay = 1.0;
}
outKey.set(year);
outValue.setTotalDelay(delay);
outValue.setTotalCount(1.0);

context.write(outKey,outValue);
```

Reducer:

The Reducer Function grabbed all the values from the mapper and Calculated the sum of all the counts (Delay and Splits).

Once the counts were aggregated, I calculated the Percentage using simple Percentage Formula that is Total Counts of Delay * 100 / Total Counts

```
public void reduce(Text key,Iterable<FlightDelayCustomWritable> value,
    FlightDelayCustomWritable fcd = new FlightDelayCustomWritable();
    DoubleWritable outValue = new DoubleWritable();
    double totalDelay = 0.0;
    double totalCount = 0.0;
    double percentage = 0.0;

    for(FlightDelayCustomWritable val:value) {
        totalDelay += val.getTotalDelay();
        totalCount += val.getTotalCount();
    }
    percentage = totalDelay*100/totalCount;
    outValue.set(percentage);
    context.write(key, outValue);
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Driver Class:

```
Configuration config = new Configuration();

try {
Job job = Job.getInstance(config,"Percentage of Delays");

job.setJarByClass(App.class);

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

job.setMapperClass(FlightDelayMapper.class);
job.setReducerClass(FlightDelayReducer.class);

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

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

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

FileSystem fs = FileSystem.get(config);
fs.delete(new Path(args[1]), true);

boolean success = job.waitForCompletion(true);
System.out.println(success);

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Output:

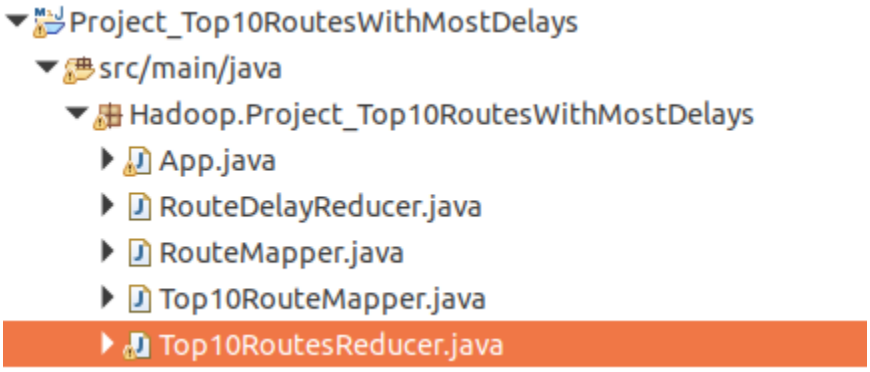
```
2000    96.4500733497365
2001    95.90958446859636
2002    98.60569162525262
2003    98.26076436301541
2004    98.01464946621464
2005    97.93073295282355
2006    98.06606680946669
2007    97.6127483240454
2008    97.79308127219772
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Top 10 Routes Experience Most Delays

In this analysis I wanted to see which routes were having most delays. Since I am calculating Top 10 routes with most delays I am using chaining method to run 2 jobs, one to fetch total delays per route and the other one to get the Top Ten values out of it.

Total Classes:



Mapper 1:

```
if(!(tokens[16].isEmpty()||tokens[16].equalsIgnoreCase("NA")) && !(tokens[17].isEmpty()||tokens[17].equalsIgnoreCase("NA"))){
    String arivalAirport = tokens[16].trim();
    String depAirport = tokens[17].trim();
    String combinedPath = arivalAirport +"-" + depAirport;
    double arrDelay;
    double depDelay;
    if(tokens[14].contains("NA")) {
        arrDelay = 0.0;
    }
    else {
        arrDelay = Double.parseDouble(tokens[14]);
    }
    if(tokens[15].contains("NA")) {
        depDelay = 0.0;
    }
    else {
        depDelay = Double.parseDouble(tokens[15]);
    }
    double totalDelay = arrDelay+depDelay;

    outKey.set(combinedPath);
    outValue.set(totalDelay);

    context.write(outKey, outValue);
```

Reducer 1:

```
DoubleWritable outValue = new DoubleWritable();
double sum = 0.0;
for(DoubleWritable val:values) {
    sum += val.get();
}
outValue.set(sum);
context.write(key, outValue);
```

Top 10 Mapper:

```
private TreeMap<Double,Text> records = new TreeMap<Double,Text>();

public void map(LongWritable key,Text value,Context context) throws IOException, Int
    String[] tokens = value.toString().split("\t");
    records.put(Double.parseDouble(tokens[1].trim()), new Text(tokens[0].trim()));

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

protected void cleanup(Context context) throws IOException, InterruptedException {
    for (Map.Entry<Double, Text> entry : records.entrySet())
    {

        double count = entry.getKey();
        Text name = entry.getValue();

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

Top 10 Reducer:

```
private TreeMap<Double,Text> outRecords = new TreeMap<Double,Text>(Collections.reverseOrder());

public void reduce(Text key, Iterable<DoubleWritable> values,Context context) throws IOException,

    double outResult = 0.0;

    for(DoubleWritable value : values) {
        outResult = value.get();

    }
    outRecords.put(outResult, new Text(key));

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

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

for (Map.Entry<Double, Text> entry : outRecords.entrySet())
{
    double count = entry.getKey();
    Text name = entry.getValue();

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

Output:

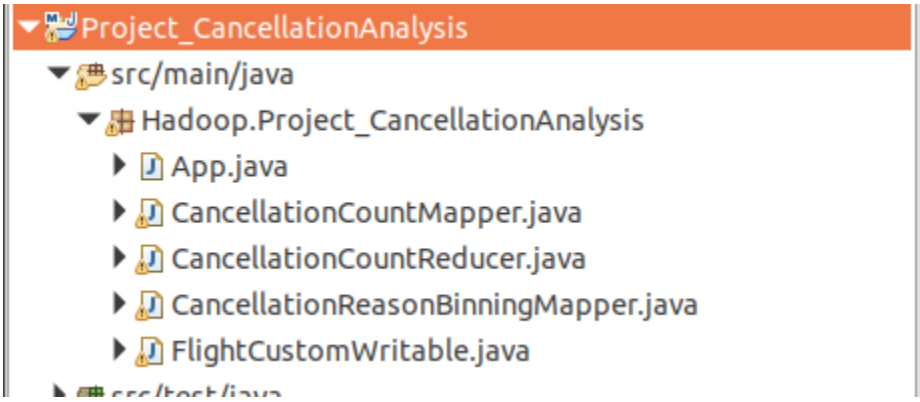
```
ORD-LGA 2925679.0
ORD-EWR 2733918.0
ATL-LGA 2427110.0
ATL-EWR 2354661.0
LGA-ORD 2351390.0
LAX-SFO 2102107.0
EWR-ORD 2099720.0
ATL-ORD 2090856.0
LAX-LAS 2068797.0
ORD-PHL 2036221.0
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Cancellation Analysis using Binning

In this analysis, Binning pattern is used to find out cancellation counts on reasons, for each airlines. It consists of two jobs, one to find the total counts of flights that experienced delays and then performed Binning to categorize the Reduced output to separate bins based on Cancellation Codes.

Total Classes:



Custom Writable Snapshot of variables used :

```
public class FlightCustomWritable implements WritableComparable{

    private String year;
    private String carrier;
    private String cancellationCode;
    public FlightCustomWritable() {
        year = "";
        carrier = "";
        cancellationCode = "";
    }

    public FlightCustomWritable(String year, String carrier,String cancellationCode) {
        super();
        this.year = year;
        this.carrier = carrier;
        this.cancellationCode = cancellationCode;
    }
```

Driver Class:

```
Job job2 = Job.getInstance(conf, "Cancellation Binning ");
job2.setJarByClass(App.class);

job2.setMapperClass(CancellationReasonBinningMapper.class);

MultipleOutputs.addNamedOutput(job2, "bins", TextOutputFormat.class, Text.class, IntWritable.class);
MultipleOutputs.setCountersEnabled(job2, true);

job2.setNumReduceTasks(0);

job2.setOutputKeyClass(Text.class);
job2.setOutputValueClass(NullWritable.class);

FileInputFormat.addInputPath(job2, new Path(args[1]));
FileOutputFormat.setOutputPath(job2, new Path(args[2]));
```

Binning Mapper:

```
private MultipleOutputs<Text,NullWritable> multipleOutputs = null;

@Override
public void setup(Context context)throws IOException, InterruptedException {
    multipleOutputs = new MultipleOutputs(context);
}

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

    String[] tokens = value.toString().split("\t");
    String cancellationCode = tokens[3];
    String selectedValue = String.join(",", Arrays.asList(tokens[0],tokens[1],tokens[2],tokens[4]));

    if(cancellationCode.equalsIgnoreCase("A"))
        multipleOutputs.write("bins", selectedValue + "," + "Carrier-cancellation", NullWritable.get(),"Carrier-cancellation");
    if(cancellationCode.equalsIgnoreCase("B"))
        multipleOutputs.write("bins", selectedValue+ "," + "Weather-cancellation", NullWritable.get(),"Weather-cancellation");
    if(cancellationCode.equalsIgnoreCase("C"))
        multipleOutputs.write("bins", selectedValue + "," + "NAS-cancellation", NullWritable.get(),"NAS-cancellation");
    if(cancellationCode.equalsIgnoreCase("D"))
        multipleOutputs.write("bins", selectedValue + "," + "Security-cancellation", NullWritable.get(),"Security-cancellation");
    else
        multipleOutputs.write("bins", selectedValue + "," + "Unknown-cancellation", NullWritable.get(),"Unknown-cancellation");
}


    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        multipleOutputs.close();
    }
```

Output:

Bins

- _SUCCESS
- Carrier-cancellation-m-00000
- NAS-cancellation-m-00000
- Security-cancellation-m-00000
- Unknown-cancellation-m-00000
- Weather-cancellation-m-00000

```
2000,AA,1,29677,Unknown-cancellation
2000,AQ,1,173,Unknown-cancellation
2000,AS,1,7506,Unknown-cancellation
2000,CO,1,7296,Unknown-cancellation
2000,DL,1,31569,Unknown-cancellation
2000,HP,1,9422,Unknown-cancellation
2000,NW,1,15340,Unknown-cancellation
2000,TW,1,5254,Unknown-cancellation
2000,UA,1,44159,Unknown-cancellation
2000,US,1,28055,Unknown-cancellation
2000,WN,1,9039,Unknown-cancellation
2001,AA,1,32230,Unknown-cancellation
2001,AQ,1,1489,Unknown-cancellation
2001,AS,1,5825,Unknown-cancellation
2001,CO,1,10996,Unknown-cancellation
2001,DL,1,32826,Unknown-cancellation
2001,HP,1,6379,Unknown-cancellation
2001,MQ,1,35382,Unknown-cancellation
2001,NW,1,17494,Unknown-cancellation
2001,TW,1,5747,Unknown-cancellation
```

```
2003,AA,1,1963,Carrier-cancellation
2003,AS,1,1432,Carrier-cancellation
2003,B6,1,25,Carrier-cancellation
2003,CO,1,209,Carrier-cancellation
2003,DH,1,1580,Carrier-cancellation
2003,DL,1,1383,Carrier-cancellation
2003,EV,1,1515,Carrier-cancellation
2003,FL,1,381,Carrier-cancellation
2003,HA,1,30,Carrier-cancellation
2003,HP,1,498,Carrier-cancellation
2003,MQ,1,2754,Carrier-cancellation
2003,NW,1,1527,Carrier-cancellation
2003,OO,1,1823,Carrier-cancellation
2003,TZ,1,191,Carrier-cancellation
2003,UA,1,1245,Carrier-cancellation
2003,US,1,1732,Carrier-cancellation
2003,WN,1,3792,Carrier-cancellation
2003,XE,1,322,Carrier-cancellation
2004,AA,1,3861,Carrier-cancellation
2004,AS,1,2535,Carrier-cancellation
```

```
2003,AA,1,2975,NAS-cancellation
2003,B6,1,11,NAS-cancellation
2003,CO,1,121,NAS-cancellation
2003,DH,1,1841,NAS-cancellation
2003,DL,1,742,NAS-cancellation
2003,EV,1,678,NAS-cancellation
2003,FL,1,117,NAS-cancellation
2003,HP,1,192,NAS-cancellation
2003,MQ,1,913,NAS-cancellation
2003,NW,1,1224,NAS-cancellation
2003,OO,1,614,NAS-cancellation
2003,TZ,1,129,NAS-cancellation
2003,UA,1,1014,NAS-cancellation
2003,US,1,919,NAS-cancellation
2003,WN,1,647,NAS-cancellation
2003,XE,1,1715,NAS-cancellation
2004,AA,1,3405,NAS-cancellation
2004,CO,1,58,NAS-cancellation
2004,DH,1,3091,NAS-cancellation
2004,DL,1,1821,NAS-cancellation
2004,EV,1,1489,NAS-cancellation
2004,FL,1,363,NAS-cancellation
```

```
2003,AA,1,2,Security-cancellation
2003,CO,1,42,Security-cancellation
2003,EV,1,3,Security-cancellation
2003,OO,1,11,Security-cancellation
2003,WN,1,2,Security-cancellation
2003,XE,1,126,Security-cancellatio
2004,CO,1,3,Security-cancellation
2004,DL,1,3,Security-cancellation
2004,EV,1,11,Security-cancellation
2004,HA,1,7,Security-cancellation
2004,MQ,1,19,Security-cancellation
2004,OH,1,2,Security-cancellation
2004,OO,1,40,Security-cancellation
2004,TZ,1,6,Security-cancellation
2004,US,1,8,Security-cancellation
2004,WN,1,16,Security-cancellation
2005,DH,1,1,Security-cancellation
2005,EV,1,16,Security-cancellation
2005,MQ,1,4,Security-cancellation
2005,OH,1,60,Security-cancellation
```

```
2003,AA,1,1480,Weather-cancellation
2003,AS,1,444,Weather-cancellation
2003,B6,1,64,Weather-cancellation
2003,CO,1,856,Weather-cancellation
2003,DH,1,1570,Weather-cancellation
2003,DL,1,1349,Weather-cancellation
2003,EV,1,911,Weather-cancellation
2003,FL,1,291,Weather-cancellation
2003,HA,1,7,Weather-cancellation
2003,HP,1,180,Weather-cancellation
2003,MQ,1,3662,Weather-cancellation
2003,NW,1,688,Weather-cancellation
2003,OO,1,1180,Weather-cancellation
2003,TZ,1,90,Weather-cancellation
2003,UA,1,721,Weather-cancellation
2003,US,1,722,Weather-cancellation
2003,WN,1,809,Weather-cancellation
2003,XE,1,1462,Weather-cancellation
2004,AA,1,5136,Weather-cancellation
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Summarization of Flight delays based on Date

In this analysis, I wanted to carry out Daily, Weekly, Monthly and Yearly analysis of Delays for each airport. In order to perform this type of Analysis I would have used 4 separate jobs but since the Mapper would have been different for each type I ran all four jobs together using chaining. I also partitioned the reduced output using Partitioner Class for each type of analysis.
This was a summarization techniques which provides different values based on different day, week, month and year

Total Classes:

- Project_BestTimeToFly
  - src/main/java
    - Hadoop.Project_BestTimeToFly
      - App.java
      - CustomWritable.java
      - DailyTrafficMapper.java
      - MonthlyTrafficMapper.java
      - TrafficPartitioner.java
      - TrafficReducer.java
      - WeeklyTrafficMapper.java
      - YearlyTrafficMapper.java
      - YearPartitioner.java

Custom Writable Snapshot to see the number of variables used as keys:

```
public class CustomWritable implements WritableComparable{
    int arrDelay = 0;
    int taxing = 0;
    int flightCount =0;
    int cancelled = 0;
    int diverted = 0;

    public CustomWritable() {

        arrDelay = 0;
        taxing = 0;
        flightCount =0;
        cancelled = 0;
        diverted = 0;

    }

    public CustomWritable(int arrDelay, int taxing, int flightCount, int cancelled, int diverted) {
        super();
        this.arrDelay = arrDelay;
        this.taxing = taxing;
        this.flightCount = flightCount;
        this.cancelled = cancelled;
        this.diverted = diverted;
    }
```

Mapper:
The mapper class for each of the use cases were similar. The only difference was in the key of the map.

```
String obj = String.valueOf(dayNum)+"\t"+airport;
outKey.set(obj);

outValue.setArrDelay(Integer.parseInt(tokens[14]));
outValue.setCancelled(Integer.parseInt(tokens[21]));
outValue.setDiverted(Integer.parseInt(tokens[23]));
outValue.setFlightCount(1);

int taxin;
int taxout;

if(tokens[19].contains("NA")||tokens[19].isEmpty()) {
    taxin = 0;
}
else {
    taxin = Integer.parseInt(tokens[19]);
}

if(tokens[20].contains("NA")||tokens[20].isEmpty()) {
    taxout = 0;
}
else {
    taxout = Integer.parseInt(tokens[20]);
}

outValue.setTaxing(taxin+taxout);
context.write(outKey, outValue);
```

Reducer:

```
CustomWritable outValue = new CustomWritable();
for(CustomWritable value :values) {
    arrDelay += value.getArrDelay();
    flightCount += value.getFlightCount();
    cancelled += value.getCancelled();
    diverted += value.getDiverted();
    totaltaxing += value.getTaxing();
}

outValue.setArrDelay(arrDelay);
outValue.setCancelled(cancelled);
outValue.setDiverted(diverted);
outValue.setFlightCount(flightCount);
outValue.setTaxing(totaltaxing);

context.write(key, outValue);
```

Partitioner:

```
@Override
public int getPartition(Text key, CustomWritable value, int numOfPartitions) {

    int val = Integer.parseInt(key.toString().split("\t")[0]);
    return (val % numOfPartitions);
}
```

Yearly Partitioner:

```
private static final String MIN_LAST_ACCESS_DATE_YEAR = "min.last.access.date.year";
private Configuration conf = null;
private int minLastAccessDateYear = 0;

public Configuration getConf() {
    // TODO Auto-generated method stub
    return conf;
}

public void setConf(Configuration conf) {
    // TODO Auto-generated method stub
    this.conf = conf;
    minLastAccessDateYear = conf.getInt(MIN_LAST_ACCESS_DATE_YEAR, 0);
}

@Override
public int getPartition(Text key, CustomWritable value, int numPartitions) {
    // TODO Auto-generated method stub
    return Integer.parseInt(key.toString().split("\t")[0]) - minLastAccessDateYear;
}

public static void setMinLastAccessDateYear(Job job,int minLastAccessDateYear) {
    job.getConfiguration().setInt(MIN_LAST_ACCESS_DATE_YEAR,minLastAccessDateYear);
}
```

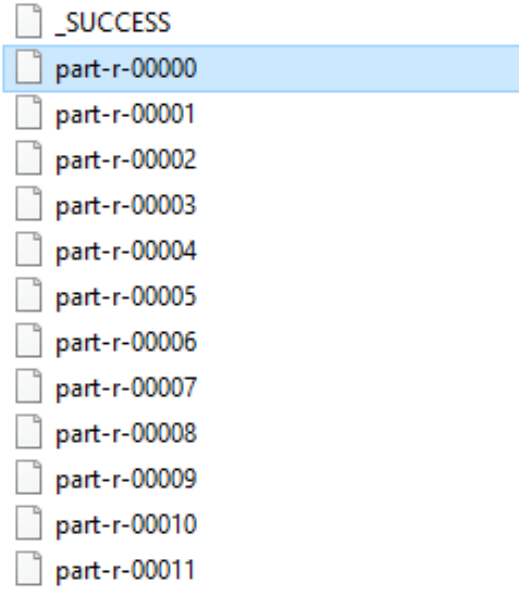
Driver Class:

```
Job job2 = Job.getInstance(conf, "Monthly Partitioning");
job2.setJarByClass(App.class);

job2.setMapperClass(MonthlyTrafficMapper.class);
job2.setMapOutputKeyClass(Text.class);
job2.setMapOutputValueClass(CustomWritable.class);

job2.setPartitionerClass(TrafficPartitioner.class);
job2.setNumReduceTasks(12);

job2.setCombinerClass(TrafficReducer.class);
job2.setReducerClass(TrafficReducer.class);

job2.setOutputKeyClass(Text.class);
job2.setOutputValueClass(CustomWritable.class);

FileInputFormat.addInputPath(job2, new Path(args[0]));
FileOutputFormat.setOutputPath(job2, new Path(args[2]));

fs.delete(new Path(args[2]), true);

boolean success_2 = job2.waitForCompletion(true);
System.out.println(success_2);
Job job3 = Job.getInstance(conf, "Daily Partitioning");
job3.setJarByClass(App.class);
```

Output:

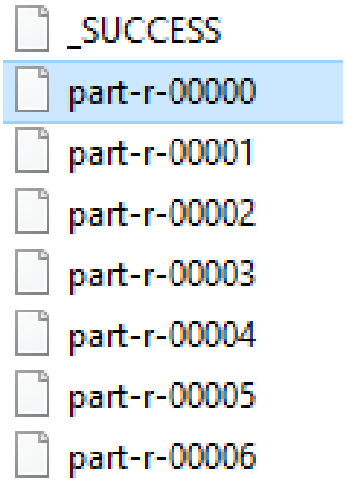
Monthly Analysis Snapshot

12	ABE	December	34451	80307	2987	0	0
12	ABI	December	7748	35072	1782	0	0
12	ABQ	December	215758	460014	27403	0	0
12	ABY	December	6016	11585	629 0	0	
12	ACT	December	-914	34142	1646	0	0
12	ACV	December	38235	25924	1797	0	0
12	ACY	December	2658	5721	257 0	0	
12	ADK	December	652 410	39 0	0		
12	ADQ	December	4676	4683	445 0	0	
12	AEX	December	16214	32708	1368	0	0
12	AGS	December	16999	29167	1311	0	0
12	AKN	December	1280	886 97	0 0		
12	ALB	December	126360	234444	10734	0	0
12	ALO	December	2734	1107	33 0	0	
12	AMA	December	41400	84500	5329	0	0
12	ANC	December	140093	253389	12755	0	0
12	APF	December	2083	5178	239 0	0	
12	ASE	December	36697	30383	1236	0	0
12	ATL	December	3187530	8163332	255224	0	0
12	ATW	December	33197	44294	1551	0	0



Weekly Analysis Snapshot

7	ABE	Sunday	28275	134757	5530	0	0
7	ABI	Sunday	7182	57752	3039	0	0
7	ABQ	Sunday	191211	768229	47082	0	0
7	ABY	Sunday	11529	21540	1120	0	0
7	ACK	Sunday	7097	6859	255 0	0	
7	ACT	Sunday	966 61628	2908	0	0	
7	ACV	Sunday	35750	40503	2987	0	0
7	ACY	Sunday	3135	13493	588 0	0	
7	ADK	Sunday	3252	2671	258 0	0	
7	ADQ	Sunday	8096	8420	840 0	0	
7	AEX	Sunday	19063	52304	2329	0	0
7	AGS	Sunday	27632	59822	2656	0	0
7	AKN	Sunday	8325	5511	598 0	0	
7	ALB	Sunday	91498	373734	18984	0	0
7	ALO	Sunday	1 607	25 0	0		
7	AMA	Sunday	58795	137511	9050	0	0
7	ANC	Sunday	152300	451679	25105	0	0
7	APF	Sunday	4543	8907	388 0	0	
7	ASE	Sunday	31671	42130	2008	0	0

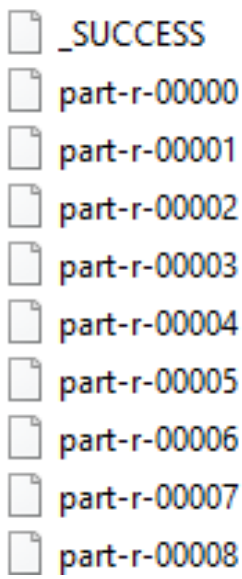


Daily Analysis Snapshot

6	ABE	4316	117331	5363	0	0
6	ABI	6619	49791	2476	0	0
6	ABQ	64364	678391	41862	0	0
6	ABY	3135	19069	934	0	0
6	ACK	4165	6041	234	0	0
6	ACT	-1019	50934	2408	0	0
6	ACV	27297	36958	2745	0	0
6	ACY	-1969	10643	520	0	0
6	ADQ	10128	8685	844	0	0
6	AEX	4375	41665	2010	0	0
6	AGS	10271	44142	2165	0	0
6	AKN	3431	2468	246	0	0
6	ALB	39330	340327	17701	0	0
6	ALO	1980	2089	85	0	0
6	AMA	27761	112229	7082	0	0
6	ANC	179651	462689	25688	0	0
6	APF	1981	9564	384	0	0
6	ASE	34387	41608	2077	0	0
6	ATL	2207678	11637173	408161	0	
6	ATW	3319	59411	2532	0	0

Yearly Analysis Snapshot

2007	ABE	45225	124878	5486	0	0
2007	ABI	30131	52800	2724	0	0
2007	ABQ	210497	703307	40688	0	0
2007	ABY	15014	25139	1283	0	0
2007	ACK	12774	8381	277	0	0
2007	ACT	15501	47475	2229	0	0
2007	ACV	54114	57666	3734	0	0
2007	ACY	12445	17773	698	0	0
2007	ADK	-658	988	89	0	0
2007	ADQ	5860	6678	634	0	0
2007	AEX	30143	68813	2894	0	0
2007	AGS	33641	49205	2216	0	0
2007	AKN	4299	1964	219	0	0
2007	ALB	154513	321223	14576	0	0
2007	ALO	3832	6999	266	0	0
2007	AMA	64202	114529	7314	0	0
2007	ANC	170789	372279	19303	0	0
2007	APF	9154	13301	534	0	0
2007	ASE	52593	95850	4748	0	0
2007	ATL	5008096	10519063	406136	0	



NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Top 10 Best Airport Partitioned by Year

In this analysis, I partitioned Top 10 Best Airports that experienced minimum delay which is then partitioned by year. Also, I wanted to carry out Reducer join to get the full name of the Airport. Therefore I am running 3 jobs in which first job will be used to Carry out basic aggregations of delays for each airport, the second job will be used to perform Reducer Joins and then the final job will be used to get the Top 10 partitioned by Year.

Mapper:

```
private TreeMap<Float,Text> outRecords = new TreeMap<Float,Text>();

public void reduce(Text key, Iterable<FloatWritable> values,Context context) throws IO

    float outResult = 0;

    for(FloatWritable value : values) {
        outResult = value.get();

    }
    outRecords.put(outResult,new Text(key));

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

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

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

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

Reducer:

```
private TreeMap<Float,Text> outRecords = new TreeMap<Float,Text>();

public void reduce(Text key, Iterable<FloatWritable> values,Context context) throws IO

    float outResult = 0;

    for(FloatWritable value : values) {
        outResult = value.get();

    }
    outRecords.put(outResult,new Text(key));

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

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

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

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

Join Reducer:

```
public static final Text EMPTY_TEXT = new Text("");
private Text temp = new Text();
private ArrayList<Text> listA = new ArrayList<Text>();
private ArrayList<Text> listB = new ArrayList<Text>();
private String joinType = null;

@Override
protected void setup(Context context) throws IOException, InterruptedException {
    joinType = context.getConfiguration().get("join.type");
}

@Override
public void reduce(Text key,Iterable<Text> value,Context context)throws IOException, InterruptedExceptic

    listA.clear();
    listB.clear();

    for(Text val:value) {
        temp = val;

        if(temp.charAt(0)=='A') {
            listA.add(new Text(temp.toString().substring(1)));
        }
        else if(temp.charAt(0)=='B') {
            listB.add(new Text(temp.toString().substring(1)));
        }

    }
    executeJoin(context);

}

private void executeJoin(Context context)throws IOException, InterruptedException {
    if(joinType.equalsIgnoreCase("inner")) {
        if(!listA.isEmpty() && !listB.isEmpty()){
            for(Text A: listA){
                for(Text B: listB){
                    //System.out.println("ListAB contains : "+ A + " " + B);
                    context.write(A, B);
                }
            }
        }
```

Driver:

```
Job job2 = Job.getInstance(config,"for Each Airport : TotalDelay/Total Trips Partitioned by Year");

job2.setJarByClass(App.class);

//job2.setMapOutputKeyClass(Text.class);
//job2.setMapOutputValueClass(FlightAnalysisCustomWritable.class);

job2.setMapperClass(BestMetricMapper.class);
job2.setMapperClass(AirportMapper.class);
job2.setReducerClass(JoinReducer.class);
//job2.setNumReduceTasks(0);

MultipleInputs.addInputPath(job2, new Path(args[1]), TextInputFormat.class, BestMetricMapper.class);
MultipleInputs.addInputPath(job2, new Path(args[2]), TextInputFormat.class, AirportMapper.class);
job2.getConfiguration().set("join.type", "inner");

job2.setOutputKeyClass(Text.class);
job2.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job2,new Path(args[1]));
FileOutputFormat.setOutputPath(job2,new Path(args[3]));
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Partitioner Driver:

```
Job job3 = Job.getInstance(config,"Top 10 Airports Partitioned by Year");

job3.setJarByClass(App.class);

job3.setMapOutputKeyClass(Text.class);
job3.setMapOutputValueClass(FloatWritable.class);

job3.setMapperClass(Top10AirportsPerYearMapper.class);
job3.setReducerClass(Top10AirportsReducer.class);

job3.setPartitionerClass(YearPartitioner.class);
YearPartitioner.setMinLastAccessDateYear(job3, 2000);
job3.setNumReduceTasks(9);
```

Output:

Reduced Join Output

```
ABE 2007    31.0        ABE Lehigh Valley International
ABE 2006    30.0        ABE Lehigh Valley International
ABE 2002    0.0     ABE Lehigh Valley International
ABE 2004    18.0        ABE Lehigh Valley International
ABE 2008    78.0        ABE Lehigh Valley International
ABE 2003    10.0        ABE Lehigh Valley International
ABE 2000    16.0        ABE Lehigh Valley International
ABE 2005    14.0        ABE Lehigh Valley International
ABE 2001    7.0     ABE Lehigh Valley International
ABI 2003    0.0     ABI Abilene Regional
ABI 2008    94.0        ABI Abilene Regional
ABI 2004    9.0     ABI Abilene Regional
ABI 2007    34.0        ABI Abilene Regional
ABI 2005    13.0        ABI Abilene Regional
ABI 2002    0.0     ABI Abilene Regional
ABI 2006    14.0        ABI Abilene Regional
ABI 2001    2.0     ABI Abilene Regional
ABQ 2002    6.0     ABQ Albuquerque International
```

Partition Output files:

- _SUCCESS
- part-r-00000
- part-r-00001
- part-r-00002
- part-r-00003
- part-r-00004
- part-r-00005
- part-r-00006
- part-r-00007
- part-r-00008

Output

```
2000    JNU Juneau International    27.0
2000    PSG James C. Johnson Petersburg 28.0
2000    YAK Yakutat 29.0
2000    LGA LaGuardia   30.0
2000    SFO San Francisco International 31.0
2000    SBA Santa Barbara Municipal 33.0
2000    OTZ Ralph Wien Memorial 35.0
2000    ORD Chicago O'Hare International    36.0
2000    EUG Mahlon Sweet    37.0
2000    MFR Rogue Valley International  40.0
```

```
2001    ORD Chicago O'Hare International    21.0
2001    DLG Dillingham  23.0
2001    OTZ Ralph Wien Memorial 24.0
2001    JNU Juneau International    26.0
2001    WRG Wrangell    27.0
2001    DRO Durango-La Plata County 28.0
2001    HDN Yampa Valley    29.0
2001    PSG James C. Johnson Petersburg 34.0
2001    DUT Unalaska    35.0
2001    ACY Atlantic City International 204.0
```

```
2002    HRL Valley International    17.0
2002    HOU William P Hobby 18.0
2002    OME Nome    19.0
2002    YAK Yakutat 20.0
2002    OTZ Ralph Wien Memorial 22.0
2002    JNU Juneau International    23.0
2002    DUT Unalaska    24.0
2002    CAK Akron-Canton Regional   25.0
2002    JAC Jackson Hole    27.0
2002    PSG James C. Johnson Petersburg 28.0
```

```
2003    PSG James C. Johnson Petersburg 32.0
2003    DLG Dillingham  33.0
2003    DUT Unalaska    37.0
2003    EYW Key West International  42.0
2003    OGD Ogden-Hinckley  43.0
2003    ADK Adak    48.0
2003    BFF Scotts Bluff County 317.0
2003    FMN Four Corners Regional   335.0
2003    CYS Cheyenne    585.0
2003    SUX Sioux Gateway   738.0
```

```
2004    AKN King Salmon 45.0
2004    OME Nome    46.0
2004    EYW Key West International  54.0
2004    ACK Nantucket Memorial  55.0
2004    DLG Dillingham  58.0
2004    DUT Unalaska    80.0
2004    ADK Adak    97.0
2004    PUB Pueblo Memorial 204.0
2004    OGD Ogden-Hinckley  516.0
2004    FMN Four Corners Regional   626.0
```

```
2005    ADK Adak    51.0
2005    OME Nome    53.0
2005    EYW Key West International  55.0
2005    BRW Wiley Post Will Rogers Memorial 61.0
2005    CWA Central Wisconsin   64.0
2005    DLG Dillingham  66.0
2005    ACK Nantucket Memorial  79.0
2005    CYS Cheyenne    193.0
2005    OGD Ogden-Hinckley  282.0
2005    CKB Benedum 390.0
```

```
2006    EYW Key West International  51.0
2006    CEC Jack McNamara   53.0
2006    ORD Chicago O'Hare International    54.0
2006    PFN Panama City-Bay County International    55.0
2006    LWB Greenbrier Valley   57.0
2006    SPI Capital 58.0
2006    CWA Central Wisconsin   59.0
2006    ILG New Castle County   60.0
2006    ADK Adak    83.0
2006    ACK Nantucket Memorial  102.0
```

```
2007    ACY Atlantic City International 61.0
2007    GNV Gainesville Regional    65.0
2007    AKN King Salmon 67.0
2007    HHH Hilton Head 69.0
2007    CEC Jack McNamara   70.0
2007    MCN Middle Georgia Regional 72.0
2007    SPI Capital 75.0
2007    SOP Moore County    116.0
2007    PIR Pierre Regional 117.0
2007    ACK Nantucket Memorial  135.0
```

```
2008    OTH North Bend Muni 103.0
2008    LMT Klamath Falls International 106.0
2008    CEC Jack McNamara   107.0
2008    SPI Capital 110.0
2008    EGE Eagle County Regional   116.0
2008    CMX Houghton County Memorial    128.0
2008    ACK Nantucket Memorial  130.0
2008    TEX Telluride Regional  131.0
2008    ACY Atlantic City International 132.0
2008    PUB Pueblo Memorial 175.0
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Top 10 Carriers with Minimum Delay

I carried out this analysis using PIG to carry out top 10 Carriers that experienced minimum delay. In order to get the name of the Carriers, I have joined the Airline Data set with Carriers dataset using carrier code. This was a simple pig scripting job which didn't reduce too much to code, just the basic understanding of pig. To carry out the same task in mapreduce, it would have taken multiple joins and lots of lines of code which can be easily done in pig and also the processing speed was better.

```
airlineRawData = LOAD '/AirArrivalDataSet/'
USING PigStorage(',') AS (Year:chararray, Month:chararray, DayofMonth:chararray,
DayOfWeek:chararray, DepTime:chararray, CRSDepTime:chararray, ArrTime:chararray,
CRSArrTime:chararray, UniqueCarrier:chararray, FlightNum:chararray,
TailNum:chararray, ActualElapsedTime:chararray, CRSElapsedTime:chararray,
AirTime:chararray, ArrDelay:int, DepDelay:int, Origin:chararray, Dest:chararray,
Distance:chararray, TaxiIn:chararray, TaxiOut:chararray, Cancelled:chararray,
CancellationCode:chararray, Diverted:chararray, CarrierDelay:chararray,
WeatherDelay:int, NASDelay:chararray, SecurityDelay:chararray,
LateAircraftDelay:chararray);

filtered_data = FILTER airlineRawData BY Year == '2008' AND (DepDelay is not null OR ArrDelay is not null );

carrier = LOAD '/Carrier/'
USING PigStorage(',') AS (iata:chararray,CarrierName:chararray);

carrierData = FOREACH carrier GENERATE REPLACE(iata,'\\"','') as iata,REPLACE(CarrierName,'\\"','') as CarrierName;

filter_carrierData = FILTER carrierData by (iata is not null) AND (CarrierName is not null) ;

mergedData = JOIN filtered_data by UniqueCarrier, filter_carrierData by iata;

records = FOREACH mergedData GENERATE CarrierName, DepDelay;
grpd = GROUP records BY CarrierName;
counts = FOREACH grpd GENERATE group AS CarrierName,
                               COUNT(records) AS DelayCount;
sorted = ORDER counts BY DelayCount DESC;
q3_1 = LIMIT sorted 10;

STORE q3_1 INTO '/Project/pig/Top10AirportsWithMinimumDelay';
```

Output:

```
Southwest Airlines Co.  1189396
American Airlines Inc.  587533
Skywest Airlines Inc.   554902
American Eagle Airlines Inc.    472532
US Airways Inc.     447081
Delta Air Lines Inc.    445170
United Air Lines Inc.   439061
Expressjet Airlines Inc.    364580
Northwest Airlines Inc. 344777
Continental Air Lines Inc.  294796
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Which Carrier accounted maximum delay every month in the year 2008

In this analysis, I wanted to find which Carrier had maximum delay every month in the year 2008 using Pig.

```
airlineRawData = LOAD '/AirArrivalDataSet/'
USING PigStorage(',') AS (Year:chararray, Month:chararray, DayofMonth:chararray,
DayOfWeek:chararray, DepTime:chararray, CRSDepTime:chararray, ArrTime:chararray,
CRSArrTime:chararray, UniqueCarrier:chararray, FlightNum:chararray,
TailNum:chararray, ActualElapsedTime:chararray, CRSElapsedTime:chararray,
AirTime:chararray, ArrDelay:int, DepDelay:int, Origin:chararray, Dest:chararray,
Distance:chararray, TaxiIn:chararray, TaxiOut:chararray, Cancelled:chararray,
CancellationCode:chararray, Diverted:chararray, CarrierDelay:chararray,
WeatherDelay:int, NASDelay:chararray, SecurityDelay:chararray,
LateAircraftDelay:chararray);

filtered_data = FILTER airlineRawData BY Year == '2008';

carrier = LOAD '/Carrier/'
USING PigStorage(',') AS (iata:chararray,CarrierName:chararray);

carrierData = FOREACH carrier GENERATE REPLACE(iata,'\\"','') as iata,REPLACE(CarrierName,'\\"','') as CarrierName;

filter_carrierData = FILTER carrierData by (iata is not null) AND (CarrierName is not null) ;

mergedData = JOIN filtered_data by UniqueCarrier, filter_carrierData by iata;

records = FOREACH mergedData GENERATE Month,CarrierName,FlightNum,Origin,
            (ArrDelay + DepDelay) AS SumDelay;

grpd = GROUP records BY Month;

top_delays = FOREACH grpd {
 sum_delays = ORDER records BY SumDelay DESC;
 sum_delays_top1 = LIMIT sum_delays 1;
 GENERATE group AS Month,FLATTEN(sum_delays_top1.CarrierName) AS Carrier,FLATTEN(sum_delays_top1.Origin) AS Airport , FLATTEN(sum_delays_top1.SumDelay) AS SumDelay;};

STORE top_delays INTO '/Project/pig/top10_delay' USING PigStorage(',');
```

Output:

```
1,American Airlines Inc.,EGE,2800
2,Northwest Airlines Inc.,HNL,4918
3,Northwest Airlines Inc.,BNA,2980
4,Northwest Airlines Inc.,CLT,4920
5,Northwest Airlines Inc.,RSW,3903
6,American Eagle Airlines Inc.,LIT,3417
7,Northwest Airlines Inc.,SEA,3028
8,Northwest Airlines Inc.,ABQ,2726
9,Northwest Airlines Inc.,OMA,3135
10,Northwest Airlines Inc.,PHL,2761
11,American Airlines Inc.,LAS,2594
12,Northwest Airlines Inc.,BOS,3252
```

Based on the Pig Job I found the following findings

1. the **Maximum delay** happens in February and April.
2. the **smallest delay** occur in November.
3. the maximal delays in summer and autumn is smaller than the maximal delays in winter and spring.
4. Northwest airlines recorded maximum delay for almost most of the months in the year 2008

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

How many flights that were delayed due to bad weather

To answer this question, I retreive flight records needed and group them by the Year of the flights. For each group, I count the number of flight records whose Weather Delay is more than 0 minute(s) and show the calculation results.

```
airlineRawData = LOAD '/AirArrivalDataSet/'
USING PigStorage(',') AS (Year:chararray, Month:chararray, DayofMonth:chararray,
DayOfWeek:chararray, DepTime:chararray, CRSDepTime:chararray, ArrTime:chararray,
CRSArrTime:chararray, UniqueCarrier:chararray, FlightNum:chararray,
TailNum:chararray, ActualElapsedTime:chararray, CRSElapsedTime:chararray,
AirTime:chararray, ArrDelay:int, DepDelay:int, Origin:chararray, Dest:chararray,
Distance:chararray, TaxiIn:chararray, TaxiOut:chararray, Cancelled:chararray,
CancellationCode:chararray, Diverted:chararray, CarrierDelay:chararray,
WeatherDelay:int, NASDelay:chararray, SecurityDelay:chararray,
LateAircraftDelay:chararray);

filtered_data = FILTER airlineRawData BY Year in ('2005,2006','2007','2008') ;

records = FOREACH filtered_data GENERATE Year, WeatherDelay;
grpd = GROUP records BY Year;
q2 = FOREACH grpd {
    weather_delays = FILTER records BY WeatherDelay > 0;
    GENERATE group, COUNT(weather_delays);
};

STORE q2 INTO '/Project/pig/DelayByWeather';
```

Output:

```
2007    127849

2008    99985
```

From the table and the figure above, we can find that

There are no weather delays from 2005 and 2006 in flight records. However, I think this may due to the missing of the data or the fact that they did not record if a delay was caused in the flights before.

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Top 10 Routes that were Diverted

Using Pig, I wanted to Analyze which route experienced most delays.

```
airlineRawData = LOAD '/AirArrivalDataSet/'
USING PigStorage(',') AS (Year:chararray, Month:chararray, DayofMonth:chararray,
DayOfWeek:chararray, DepTime:chararray, CRSDepTime:chararray, ArrTime:chararray,
CRSArrTime:chararray, UniqueCarrier:chararray, FlightNum:chararray,
TailNum:chararray, ActualElapsedTime:chararray, CRSElapsedTime:chararray,
AirTime:chararray, ArrDelay:int, DepDelay:int, Origin:chararray, Dest:chararray,
Distance:chararray, TaxiIn:chararray, TaxiOut:chararray, Cancelled:chararray,
CancellationCode:chararray, Diverted:chararray, CarrierDelay:chararray,
WeatherDelay:int, NASDelay:chararray, SecurityDelay:chararray,
LateAircraftDelay:chararray);

filtered_data = FILTER airlineRawData BY Year in ('2005,2006','2007','2008') and Diverted == '1';

records = FOREACH filtered_data GENERATE Origin,Dest,Diverted;

grpd = GROUP records by (Origin,Dest);

counted = FOREACH grpd GENERATE $0.Origin as Origin , $1.Dest as Dest , COUNT(records) as diverted_count;

Ordered = ORDER counted by diverted_count Desc;

top15 = LIMIT Ordered  10;

STORE top15 INTO '/Project/pig/TopRoutesWithDiversion' using PigStorage(',');
```

Output:

```
BUR,JFK,194
ORD,LGA,141
BUR,DFW,136
ATL,DFW,118
ATL,LGA,113
LGA,DFW,112
SFO,JFK,101
DFW,LGA,101
LAX,JFK,95
DAL,HOU,83
```

From the result we can conclude that the Route BOU-JFK experienced maximum diversions,

Appendix: Source Code

Flight Summarization based on Day, Month, Week and Year

Driver Class:

```
package Hadoop.Project_BestTimeToFly;

import java.io.IOException;

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

public class App
{
   public static void main( String[] args ) throws IOException, ClassNotFoundException, InterruptedException
   {
         Configuration conf = new Configuration();
      Job job1 = Job.getInstance(conf, "Weekly Partitioning");
      job1.setJarByClass(App.class);

      job1.setMapperClass(WeeklyTrafficMapper.class);
      job1.setMapOutputKeyClass(Text.class);
      job1.setMapOutputValueClass(CustomWritable.class);

      job1.setPartitionerClass(TrafficPartitioner.class);
      job1.setNumReduceTasks(7);

      job1.setCombinerClass(TrafficReducer.class);
      job1.setReducerClass(TrafficReducer.class);

      job1.setOutputKeyClass(Text.class);
      job1.setOutputValueClass(CustomWritable.class);

      FileInputFormat.addInputPath(job1, new Path(args[0]));
      FileOutputFormat.setOutputPath(job1, new Path(args[1]));

      FileSystem fs = FileSystem.get(conf);
                  fs.delete(new Path(args[1]), true);

                  boolean success_1 = job1.waitForCompletion(true);
                  System.out.println(success_1);

                  Job job2 = Job.getInstance(conf, "Monthly Partitioning");
                  job2.setJarByClass(App.class);

                  job2.setMapperClass(MonthlyTrafficMapper.class);
                  job2.setMapOutputKeyClass(Text.class);
```

```
        job2.setMapOutputValueClass(CustomWritable.class);

        job2.setPartitionerClass(TrafficPartitioner.class);
        job2.setNumReduceTasks(12);

        job2.setCombinerClass(TrafficReducer.class);
        job2.setReducerClass(TrafficReducer.class);

        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(CustomWritable.class);

FileInputFormat.addInputPath(job2, new Path(args[0]));
FileOutputFormat.setOutputPath(job2, new Path(args[2]));

        fs.delete(new Path(args[2]), true);

        boolean success_2 = job2.waitForCompletion(true);
        System.out.println(success_2);
        Job job3 = Job.getInstance(conf, "Daily Partitioning");
        job3.setJarByClass(App.class);

        job3.setMapperClass(DailyTrafficMapper.class);
        job3.setMapOutputKeyClass(Text.class);
        job3.setMapOutputValueClass(CustomWritable.class);

        job3.setPartitionerClass(TrafficPartitioner.class);
        job3.setNumReduceTasks(31);

        job3.setCombinerClass(TrafficReducer.class);
        job3.setReducerClass(TrafficReducer.class);

        job3.setOutputKeyClass(Text.class);
        job3.setOutputValueClass(CustomWritable.class);

FileInputFormat.addInputPath(job3, new Path(args[0]));
FileOutputFormat.setOutputPath(job3, new Path(args[3]));

        fs.delete(new Path(args[3]), true);

        boolean success_3 = job3.waitForCompletion(true);
        System.out.println(success_3);

        Job job4 = Job.getInstance(conf, "Yearly Partitioning");
        job4.setJarByClass(App.class);

        job4.setMapperClass(YearlyTrafficMapper.class);
        job4.setMapOutputKeyClass(Text.class);
        job4.setMapOutputValueClass(CustomWritable.class);

        job4.setPartitionerClass(TrafficPartitioner.class);
        YearPartitioner.setMinLastAccessDateYear(job4, 2000);
```

```
                job4.setNumReduceTasks(9);

                job4.setCombinerClass(TrafficReducer.class);
                job4.setReducerClass(TrafficReducer.class);

                job4.setOutputKeyClass(Text.class);
                job4.setOutputValueClass(CustomWritable.class);

    FileInputFormat.addInputPath(job4, new Path(args[0]));
    FileOutputFormat.setOutputPath(job4, new Path(args[4]));

                fs.delete(new Path(args[4]), true);

                boolean success_4 = job4.waitForCompletion(true);
                System.out.println(success_4);

  }
}
```

CustomWritable:

```
package Hadoop.Project_BestTimeToFly;

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

import org.apache.hadoop.io.WritableComparable;

public class CustomWritable implements WritableComparable{
        int arrDelay = 0;
        int taxing = 0;
        int flightCount =0;
        int cancelled = 0;
        int diverted = 0;

        public CustomWritable() {

                arrDelay = 0;
                taxing = 0;
                flightCount =0;
                cancelled = 0;
                diverted = 0;

        }

        public CustomWritable(int arrDelay, int taxing, int flightCount, int cancelled, int diverted) {
                super();
                this.arrDelay = arrDelay;
                this.taxing = taxing;
```

```
        this.flightCount = flightCount;
        this.cancelled = cancelled;
        this.diverted = diverted;
    }


    public int getArrDelay() {
        return arrDelay;
    }

    public void setArrDelay(int arrDelay) {
        this.arrDelay = arrDelay;
    }

    public int getTaxing() {
        return taxing;
    }

    public void setTaxing(int taxing) {
        this.taxing = taxing;
    }

    public int getFlightCount() {
        return flightCount;
    }

    public void setFlightCount(int flightCount) {
        this.flightCount = flightCount;
    }

    public int getCancelled() {
        return cancelled;
    }

    public void setCancelled(int cancelled) {
        this.cancelled = cancelled;
    }

    public int getDiverted() {
        return diverted;
    }

    public void setDiverted(int diverted) {
        this.diverted = diverted;
    }

    public void readFields(DataInput in) throws IOException {
        arrDelay = in.readInt();
        taxing = in.readInt();
        flightCount = in.readInt();
        cancelled = in.readInt();
```

```
		diverted = in.readInt();


	}

	public void write(DataOutput out) throws IOException {
		out.writeInt(arrDelay);
		out.writeInt(taxing);
		out.writeInt(flightCount);
		out.writeInt(cancelled);
		out.writeInt(diverted);
	}

	public int compareTo(Object o) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public String toString() {
		return String.join("\t",
Arrays.asList(String.valueOf(arrDelay),String.valueOf(taxing),String.valueOf(flightCount),String.valueOf(cancelled),String.
valueOf(diverted)));
	}



}
```

Daily Traffic Mapper:

```
package Hadoop.Project_BestTimeToFly;

import java.io.IOException;

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

public class DailyTrafficMapper extends Mapper<LongWritable,Text,Text,CustomWritable> {

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

	public void map(LongWritable key,Text value,Context context) throws IOException, InterruptedException {
		String[] tokens = value.toString().split(",");
		if (!value.toString().contains("UniqueCarrier")) {
	if(!tokens[14].equalsIgnoreCase("NA")){
		String airport = tokens[16];
		int dayNum = Integer.parseInt(tokens[3]);
```

```
            String obj = String.valueOf(dayNum)+"\t"+airport;
            outKey.set(obj);

            outValue.setArrDelay(Integer.parseInt(tokens[14]));
            outValue.setCancelled(Integer.parseInt(tokens[21]));
            outValue.setDiverted(Integer.parseInt(tokens[23]));
            outValue.setFlightCount(1);

            int taxin;
            int taxout;

            if(tokens[19].contains("NA")||tokens[19].isEmpty()) {
                taxin = 0;
            }
            else {
                taxin = Integer.parseInt(tokens[19]);
            }

            if(tokens[20].contains("NA")||tokens[20].isEmpty()) {
                taxout = 0;
            }
            else {
                taxout = Integer.parseInt(tokens[20]);
            }

            outValue.setTaxing(taxin+taxout);
            context.write(outKey, outValue);
        }
        }
        }
        }
```

Monthly Traffice Mapper:

```
package Hadoop.Project_BestTimeToFly;

import java.io.IOException;

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

public class MonthlyTrafficMapper extends Mapper<LongWritable,Text,Text,CustomWritable> {

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

        public void map(LongWritable key,Text value,Context context) throws IOException, InterruptedException {
            String[] tokens = value.toString().split(",");
            if (!value.toString().contains("UniqueCarrier")) {
```

```
if(!tokens[14].equalsIgnoreCase("NA")){
        String airport = tokens[16];
        int monthNum = Integer.parseInt(tokens[1]);
        String monthName;
        switch(monthNum) {
        case 1:
                monthName = "January";
                break;
        case 2:
                monthName = "February";
                break;
        case 3:
                monthName = "March";
                break;
        case 4:
                monthName = "April";
                break;
        case 5:
                monthName = "May";
                break;
        case 6:
                monthName = "June";
                break;
        case 7:
                monthName = "July";
                break;
        case 8:
                monthName = "August";
                break;
        case 9:
                monthName = "September";
                break;
        case 10:
                monthName = "October";
                break;
        case 11:
                monthName = "November";
                break;
        case 12:
                monthName = "December";
                break;

        default:
                monthName ="Unknown";
                break;
        }


        String obj = String.valueOf(monthNum)+"\t"+airport+"\t"+monthName;
        outKey.set(obj);
```

```
                outValue.setArrDelay(Integer.parseInt(tokens[14]));
                outValue.setCancelled(Integer.parseInt(tokens[21]));
                outValue.setDiverted(Integer.parseInt(tokens[23]));
                outValue.setFlightCount(1);

                int taxin;
                int taxout;

                if(tokens[19].contains("NA")||tokens[19].isEmpty()) {
                        taxin = 0;
                }
                else {
                        taxin = Integer.parseInt(tokens[19]);
                }

                if(tokens[20].contains("NA")||tokens[20].isEmpty()) {
                        taxout = 0;
                }
                else {
                        taxout = Integer.parseInt(tokens[20]);
                }

                outValue.setTaxing(taxin+taxout);
                context.write(outKey, outValue);
        }
        }
        }

}
```

Weekly Traffic Mapper:

```
package Hadoop.Project_BestTimeToFly;

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 WeeklyTrafficMapper extends Mapper<LongWritable,Text,Text,CustomWritable> {

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

        public void map(LongWritable key,Text value,Context context) throws IOException, InterruptedException {
                String[] tokens = value.toString().split(",");
                if (!value.toString().contains("UniqueCarrier")) {
        if(!tokens[14].equalsIgnoreCase("NA")){
```

```
String airport = tokens[16];
int weekNum = Integer.parseInt(tokens[3]);
String dayOfTheWeek;
switch(weekNum) {
case 1:
        dayOfTheWeek = "Monday";
        break;
case 2:
        dayOfTheWeek = "Tuesday";
        break;
case 3:
        dayOfTheWeek = "Wednesday";
        break;
case 4:
        dayOfTheWeek = "Thursday";
        break;
case 5:
        dayOfTheWeek = "Friday";
        break;
case 6:
        dayOfTheWeek = "Saturday";
        break;
case 7:
        dayOfTheWeek = "Sunday";
        break;
default:
        dayOfTheWeek ="Unknown";
        break;
}


String obj = String.valueOf(weekNum)+"\t"+airport+"\t"+dayOfTheWeek;
outKey.set(obj);

outValue.setArrDelay(Integer.parseInt(tokens[14]));
outValue.setCancelled(Integer.parseInt(tokens[21]));
outValue.setDiverted(Integer.parseInt(tokens[23]));
outValue.setFlightCount(1);

int taxin;
int taxout;

if(tokens[19].contains("NA")||tokens[19].isEmpty()) {
        taxin = 0;
}
else {
        taxin = Integer.parseInt(tokens[19]);
}

if(tokens[20].contains("NA")||tokens[20].isEmpty()) {
        taxout = 0;
```

```
                }
                else {
                        taxout = Integer.parseInt(tokens[20]);
                }

                outValue.setTaxing(taxin+taxout);
                context.write(outKey, outValue);
        }
        }
        }

}
```

Yearly Analysis Mapper:

```
package Hadoop.Project_BestTimeToFly;

import java.io.IOException;

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

public class YearlyTrafficMapper extends Mapper<LongWritable,Text,Text,CustomWritable> {

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

        public void map(LongWritable key,Text value,Context context) throws IOException, InterruptedException {
                String[] tokens = value.toString().split(",");
                if (!value.toString().contains("UniqueCarrier")) {
        if(!tokens[14].equalsIgnoreCase("NA")){
                String airport = tokens[16];
                int year = Integer.parseInt(tokens[0]);

                String obj = String.valueOf(year)+"\t"+airport;
                outKey.set(obj);

                outValue.setArrDelay(Integer.parseInt(tokens[14]));
                outValue.setCancelled(Integer.parseInt(tokens[21]));
                outValue.setDiverted(Integer.parseInt(tokens[23]));
                outValue.setFlightCount(1);

                int taxin;
                int taxout;

                if(tokens[19].contains("NA")||tokens[19].isEmpty()) {
                        taxin = 0;
                }
                else {
```

```
                taxin = Integer.parseInt(tokens[19]);
            }

            if(tokens[20].contains("NA")||tokens[20].isEmpty()) {
                taxout = 0;
            }
            else {
                taxout = Integer.parseInt(tokens[20]);
            }

            outValue.setTaxing(taxin+taxout);
            context.write(outKey, outValue);
        }
        }
        }

}
```

Reducer:

```
package Hadoop.Project_BestTimeToFly;

import java.io.IOException;

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

public class TrafficReducer extends Reducer<Text,CustomWritable,Text,CustomWritable> {

        public void reduce(Text key,Iterable<CustomWritable> values,Context context) throws IOException,
InterruptedException {
            int arrDelay = 0;
            int flightCount =0;
            int cancelled = 0;
            int diverted = 0;
            int totaltaxing = 0;

            CustomWritable outValue = new CustomWritable();
            for(CustomWritable value :values) {
                arrDelay += value.getArrDelay();
                flightCount += value.getFlightCount();
                cancelled += value.getCancelled();
                diverted += value.getDiverted();
                totaltaxing += value.getTaxing();
            }

            outValue.setArrDelay(arrDelay);
            outValue.setCancelled(cancelled);
            outValue.setDiverted(diverted);
```

```
		outValue.setFlightCount(flightCount);
		outValue.setTaxing(totaltaxing);

		context.write(key, outValue);
	}

}
```

Partitioner:

```
package Hadoop.Project_BestTimeToFly;

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

public class TrafficPartitioner extends Partitioner<Text, CustomWritable>{

	@Override
	public int getPartition(Text key, CustomWritable value, int numOfPartitions) {

		int val = Integer.parseInt(key.toString().split("\t")[0]);
		return (val % numOfPartitions);
	}

}
```

Year Partitioner:

```
package Hadoop.Project_BestTimeToFly;

import org.apache.hadoop.conf.Configurable;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Partitioner;

public class YearPartitioner extends Partitioner<Text, CustomWritable> implements Configurable{

	private static final String MIN_LAST_ACCESS_DATE_YEAR = "min.last.access.date.year";
	private Configuration conf = null;
	private int minLastAccessDateYear = 0;

	public Configuration getConf() {
		// TODO Auto-generated method stub
		return conf;
	}

	public void setConf(Configuration conf) {
		// TODO Auto-generated method stub
		this.conf = conf;
		minLastAccessDateYear = conf.getInt(MIN_LAST_ACCESS_DATE_YEAR, 0);
```

```
	}

	@Override
	public int getPartition(Text key, CustomWritable value, int numPartitions) {
		// TODO Auto-generated method stub
		return Integer.parseInt(key.toString().split("\t")[0]) - minLastAccessDateYear;
	}

	public static void setMinLastAccessDateYear(Job job,int minLastAccessDateYear) {
		job.getConfiguration().setInt(MIN_LAST_ACCESS_DATE_YEAR,minLastAccessDateYear);
	}
}
```

Percentage flights that were delayed every year

Driver Code:

```
package Hadoop.Project_PercentageDelayPerYear;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
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;

/**
 * Hello world!
 *
 */
public class App
{
   public static void main( String[] args ) throws ClassNotFoundException, InterruptedException
   {
          Configuration config = new Configuration();

      try {
                    Job job = Job.getInstance(config,"Percentage of Delays");

                    job.setJarByClass(App.class);

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

                    job.setMapperClass(FlightDelayMapper.class);
                    job.setReducerClass(FlightDelayReducer.class);

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

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

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

                    FileSystem fs = FileSystem.get(config);
```

```
                fs.delete(new Path(args[1]), true);

                boolean success = job.waitForCompletion(true);
                System.out.println(success);

        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

    }
}
```

CustomWritable:

```
package Hadoop.Project_PercentageDelayPerYear;

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

import org.apache.hadoop.io.WritableComparable;

public class FlightDelayCustomWritable implements WritableComparable {

        double totalCount;
        double totalDelay;

        public FlightDelayCustomWritable() {
                totalCount = 0.0;
                totalDelay = 0.0;
        }

        public FlightDelayCustomWritable(double totalCount, double totalDelay) {
                super();
                this.totalCount = totalCount;
                this.totalDelay = totalDelay;
        }

        public double getTotalCount() {
                return totalCount;
        }

        public void setTotalCount(double totalCount) {
                this.totalCount = totalCount;
        }

        public double getTotalDelay() {
                return totalDelay;
        }
```

```
	public void setTotalDelay(double totalDelay) {
		this.totalDelay = totalDelay;
	}

	public void readFields(DataInput in) throws IOException {
		totalCount = in.readDouble();
		totalDelay = in.readDouble();
	}

	public void write(DataOutput out) throws IOException {
		out.writeDouble(totalCount);
		out.writeDouble(totalDelay);
	}

	public int compareTo(Object o) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public String toString() {
		return totalDelay + "\t" + totalCount;
	}


}
```

Mapper:

```
package Hadoop.Project_PercentageDelayPerYear;

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


public class FlightDelayMapper extends Mapper<LongWritable,Text,Text,FlightDelayCustomWritable>{
	private FlightDelayCustomWritable outValue = new FlightDelayCustomWritable();
	private Text outKey = new Text();
	private double delay;

	public void map(LongWritable key,Text value,Context context) {
		try {
			if(key.get() == 0 && value.toString().contains("header")) {
				return;
			}
			else {
				String[] tokens = value.toString().split(",");
				String year = tokens[0];
				if(tokens[14].isEmpty()||tokens[14].equalsIgnoreCase("NA")) {
```

```
                                        delay = 0.0;
                                }
                                else {
                                        delay = 1.0;
                                }
                                outKey.set(year);
                                outValue.setTotalDelay(delay);
                                outValue.setTotalCount(1.0);

                                context.write(outKey,outValue);
                        }
                }
                catch (Exception e) {
            e.printStackTrace();
        }

    }

    }
```

Reducer:

```
package Hadoop.Project_PercentageDelayPerYear;

import java.io.IOException;

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

public class FlightDelayReducer extends Reducer<Text,FlightDelayCustomWritable,Text,DoubleWritable>{

    public void reduce(Text key,Iterable<FlightDelayCustomWritable> value,Context context) throws IOException,
InterruptedException {
        FlightDelayCustomWritable fcd = new FlightDelayCustomWritable();
        DoubleWritable outValue = new DoubleWritable();
        double totalDelay = 0.0;
        double totalCount = 0.0;
        double percentage = 0.0;

        for(FlightDelayCustomWritable val:value) {
            totalDelay += val.getTotalDelay();
            totalCount += val.getTotalCount();
        }
        percentage = totalDelay*100/totalCount;
        outValue.set(percentage);
        context.write(key, outValue);

    }

}
```

Cancellation Analysis

Driver Code:

```
package Hadoop.Project_CancellationAnalysis;

import java.io.IOException;

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


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

          Job job1 = Job.getInstance(conf, "Total Counts for Flights that were Cancelled");
      job1.setJarByClass(App.class);

      job1.setInputFormatClass(TextInputFormat.class);
                    job1.setOutputFormatClass(TextOutputFormat.class);

      job1.setMapperClass(CancellationCountMapper.class);
      job1.setReducerClass(CancellationCountReducer.class);

      job1.setMapOutputKeyClass(Text.class);
      job1.setMapOutputValueClass(LongWritable.class);

      job1.setOutputKeyClass(Text.class);
      job1.setOutputValueClass(LongWritable.class);

      FileInputFormat.addInputPath(job1, new Path(args[0]));
      FileOutputFormat.setOutputPath(job1, new Path(args[1]));

      FileSystem fs = FileSystem.get(conf);
          fs.delete(new Path(args[1]), true);

                    boolean success_1 = job1.waitForCompletion(true);
```

```
                System.out.println(success_1);

                Job job2 = Job.getInstance(conf, "Cancellation Binning ");
    job2.setJarByClass(App.class);

    job2.setMapperClass(CancellationReasonBinningMapper.class);

    MultipleOutputs.addNamedOutput(job2, "bins", TextOutputFormat.class, Text.class, IntWritable.class);
    MultipleOutputs.setCountersEnabled(job2, true);

    job2.setNumReduceTasks(0);

    job2.setOutputKeyClass(Text.class);
    job2.setOutputValueClass(NullWritable.class);

    FileInputFormat.addInputPath(job2, new Path(args[1]));
    FileOutputFormat.setOutputPath(job2, new Path(args[2]));

    fs.delete(new Path(args[2]), true);
    boolean success_2 = job2.waitForCompletion(true);
                System.out.println(success_2);
                }
  }
```

Flight Custom Writable:

```
package Hadoop.Project_CancellationAnalysis;

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

import org.apache.hadoop.io.WritableComparable;


public class FlightCustomWritable implements WritableComparable{

        private String year;
        private String carrier;
        private String cancellationCode;
        public FlightCustomWritable() {
                year = "";
                carrier = "";
                cancellationCode = "";
        }

        public FlightCustomWritable(String year, String carrier,String cancellationCode) {
                super();
                this.year = year;
```

```
        this.carrier = carrier;
        this.cancellationCode = cancellationCode;
}


public String getCancellationCode() {
        return cancellationCode;
}

public void setCancellationCode(String cancellationCode) {
        this.cancellationCode = cancellationCode;
}

public String getYear() {
        return year;
}

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

public String getCarrier() {
        return carrier;
}

public void setCarrier(String carrier) {
        this.carrier = carrier;
}

public void readFields(DataInput in) throws IOException {
        year = in.readUTF();
        carrier = in.readUTF();
        cancellationCode = in.readUTF();
}

public void write(DataOutput out) throws IOException {
        out.writeUTF(year);
        out.writeUTF(carrier);
        out.writeUTF(cancellationCode);
}

public int compareTo(Object o) {
        FlightCustomWritable ck = (FlightCustomWritable)o;
        String thisValue = this.getYear();
        String otherValue = ck.getYear();
        int result = thisValue.compareTo(otherValue);
        return (result < 0 ? -1 :(result == 0 ? 0 : 1));
}

@Override
public String toString() {
```

```
                return String.join("\t", Arrays.asList(year,carrier,cancellationCode));
        }

}
```

Count Mapper:

```
package Hadoop.Project_CancellationAnalysis;

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

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

public class CancellationCountMapper extends Mapper<LongWritable,Text,Text,LongWritable>{
        private LongWritable outValue = new LongWritable(1);
        private Text outKey = new Text();

        public void map(LongWritable key,Text value,Context context) throws IOException, InterruptedException {
                String[] tokens = value.toString().split(",");
          if (!value.toString().contains("UniqueCarrier")) {
            if(tokens[21].equalsIgnoreCase("1")){
                        String selectedValue = String.join("\t", Arrays.asList(tokens[0],tokens[8],tokens[21],tokens[22]));
                        outKey.set(selectedValue);

                        //outKey.setYear(tokens[0]);
                        //outKey.setCarrier(tokens[1]);
                        //outKey.setCancellationCode(tokens[3]);

                            context.write(outKey, outValue);
        }

}
}
        }
```

Count Reducer:

```
package Hadoop.Project_CancellationAnalysis;

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

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

```
public class CancellationCountReducer extends Reducer<Text,LongWritable,Text,LongWritable> {

	public void reduce(Text key,Iterable<LongWritable> value,Context context) throws IOException,
InterruptedException {
		long sum = 0;

		Text outKey = new Text();
		LongWritable outValue = new LongWritable(0);
		for(LongWritable val :value) {
			sum += val.get();
		}

		outValue.set(sum);
		outKey.set(key);
		//outKey.set(new Text (String.join("\t",
Arrays.asList(key.getYear(),key.getCarrier(),key.getCancellationCode()))));

		context.write(outKey, outValue);
	}

}
```

Binning Mapper:

```
package Hadoop.Project_CancellationAnalysis;

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

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;

public class CancellationReasonBinningMapper extends Mapper<LongWritable,Text,Text,NullWritable>{

	private MultipleOutputs<Text,NullWritable> multipleOutputs = null;

	@Override
	public void setup(Context context)throws IOException, InterruptedException {
		multipleOutputs = new MultipleOutputs(context);
	}

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

	   String[] tokens = value.toString().split("\t");
	String cancellationCode = tokens[3];
	String selectedValue = String.join(",", Arrays.asList(tokens[0],tokens[1],tokens[2],tokens[4]));
```

```
    if(cancellationCode.equalsIgnoreCase("A"))
        multipleOutputs.write("bins", selectedValue + "," + "Carrier-cancellation", NullWritable.get(),"Carrier-
cancellation");
    if(cancellationCode.equalsIgnoreCase("B"))
        multipleOutputs.write("bins", selectedValue+ "," + "Weather-cancellation", NullWritable.get(),"Weather-
cancellation");
    if(cancellationCode.equalsIgnoreCase("C"))
        multipleOutputs.write("bins", selectedValue + "," + "NAS-cancellation", NullWritable.get(),"NAS-cancellation");
    if(cancellationCode.equalsIgnoreCase("D"))
        multipleOutputs.write("bins", selectedValue + "," + "Security-cancellation", NullWritable.get(),"Security-
cancellation");
    else
        multipleOutputs.write("bins", selectedValue + "," + "Unknown-cancellation", NullWritable.get(),"Unknown-
cancellation");
  }


        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            multipleOutputs.close();
        }

      }
```

Top 10 Best Airport every year

Driver Code:

```
package Hadoop.Project_BestAirportPerYear;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.FloatWritable;
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.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;

/**
 * Hello world!
 *
 */
public class App
{
   public static void main( String[] args ) throws ClassNotFoundException, InterruptedException
   {
Configuration config = new Configuration();

      try {
                  Job job1 = Job.getInstance(config,"for Each Airport : TotalDelay/Total Trips Partitioned by Year");

                  job1.setJarByClass(App.class);

                  job1.setMapOutputKeyClass(Text.class);
                  job1.setMapOutputValueClass(FlightAnalysisCustomWritable.class);

                  job1.setMapperClass(CountMapper.class);
                  job1.setReducerClass(BestMetricReducer.class);

                  job1.setInputFormatClass(TextInputFormat.class);
                  job1.setOutputFormatClass(TextOutputFormat.class);

                  job1.setOutputKeyClass(Text.class);
                  job1.setOutputValueClass(FloatWritable.class);

                  FileInputFormat.addInputPath(job1,new Path(args[0]));
                  FileOutputFormat.setOutputPath(job1,new Path(args[1]));
```

```
FileSystem fs = FileSystem.get(config);
fs.delete(new Path(args[1]), true);

boolean success1 = job1.waitForCompletion(true);
System.out.println(success1);

Job job2 = Job.getInstance(config,"for Each Airport : TotalDelay/Total Trips Partitioned by Year");

job2.setJarByClass(App.class);

//job2.setMapOutputKeyClass(Text.class);
//job2.setMapOutputValueClass(FlightAnalysisCustomWritable.class);

job2.setMapperClass(BestMetricMapper.class);
job2.setMapperClass(AirportMapper.class);
job2.setReducerClass(JoinReducer.class);
//job2.setNumReduceTasks(0);

MultipleInputs.addInputPath(job2, new Path(args[1]), TextInputFormat.class, BestMetricMapper.class);
MultipleInputs.addInputPath(job2, new Path(args[2]), TextInputFormat.class, AirportMapper.class);
job2.getConfiguration().set("join.type", "inner");

job2.setOutputKeyClass(Text.class);
job2.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job2,new Path(args[1]));
FileOutputFormat.setOutputPath(job2,new Path(args[3]));


//fs.delete(new Path(args[2]), true);
fs.delete(new Path(args[3]), true);

boolean success2 = job2.waitForCompletion(true);
System.out.println(success2);


Job job3 = Job.getInstance(config,"Top 10 Airports Partitioned by Year");

job3.setJarByClass(App.class);

job3.setMapOutputKeyClass(Text.class);
job3.setMapOutputValueClass(FloatWritable.class);

job3.setMapperClass(Top10AirportsPerYearMapper.class);
job3.setReducerClass(Top10AirportsReducer.class);

job3.setPartitionerClass(YearPartitioner.class);
YearPartitioner.setMinLastAccessDateYear(job3, 2000);
job3.setNumReduceTasks(9);
```

```
                //job3.setPartitionerClass(NaturalKeyPartitioner.class);;
                //job3.setGroupingComparatorClass(NaturalKeyGroupComparator.class);
                //job3.setSortComparatorClass(SecondaryGroupComparator.class);


                job3.setInputFormatClass(TextInputFormat.class);
                job3.setOutputFormatClass(TextOutputFormat.class);

                job3.setOutputKeyClass(Text.class);
                job3.setOutputValueClass(FloatWritable.class);

                FileInputFormat.addInputPath(job3,new Path(args[3]));
                FileOutputFormat.setOutputPath(job3,new Path(args[4]));

                fs.delete(new Path(args[4]), true);

                boolean success3 = job3.waitForCompletion(true);
                System.out.println(success3);



        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

    }
}
```

Custom Writable:

```
package Hadoop.Project_BestAirportPerYear;

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

import org.apache.hadoop.io.WritableComparable;

public class FlightAnalysisCustomWritable implements WritableComparable{

            long totalDelay;
            long totalTrips;

            public FlightAnalysisCustomWritable() {
                    totalDelay = 0;
                    totalTrips = 0;
            }

            public FlightAnalysisCustomWritable(long totalDelay, long totalTrips) {
                    super();
```

```
        this.totalDelay = totalDelay;
        this.totalTrips = totalTrips;
}



public long getTotalDelay() {
        return totalDelay;
}



public void setTotalDelay(long totalDelay) {
        this.totalDelay = totalDelay;
}



public long getTotalTrips() {
        return totalTrips;
}



public void setTotalTrips(long totalTrips) {
        this.totalTrips = totalTrips;
}



public void readFields(DataInput in) throws IOException {
        totalDelay = in.readLong();
        totalTrips = in.readLong();

}

public void write(DataOutput out) throws IOException {
        out.writeLong(totalDelay);
        out.writeLong(totalTrips);

}

public int compareTo(Object o) {
        // TODO Auto-generated method stub
        return 0;
}

@Override
public String toString() {
        return totalDelay + "\t"+ totalTrips ;
}
```

```
}
```

Count Mapper:

```
package Hadoop.Project_BestAirportPerYear;

import java.io.IOException;

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

public class CountMapper extends Mapper<LongWritable,Text,Text,FlightAnalysisCustomWritable> {

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

	private int arrDelay = 0;
	private int depDelay = 0;
	private int carrierDelay = 0;
	private int weatherDelay = 0;
	private int nasDelay = 0;
	private int securityDelay = 0;
	private int lateDelay = 0;
	private int totalDelay = 0;

	public void map(LongWritable key,Text value,Context context) throws IOException, InterruptedException {
		if (!value.toString().contains("UniqueCarrier")) {

			String[] tokens = value.toString().split(",");
			String year = tokens[0];
			String airport = tokens[16];

			if(!(tokens[14].contains("NA")||tokens[14].isEmpty())) {
				arrDelay = Integer.parseInt(tokens[14]);
			}
			if(!(tokens[15].contains("NA")||tokens[15].isEmpty())) {
				depDelay = Integer.parseInt(tokens[15]);
			}
			if(!(tokens[24].contains("NA")||tokens[24].isEmpty())) {
				carrierDelay = Integer.parseInt(tokens[24]);
			}
			if(!(tokens[25].contains("NA")||tokens[25].isEmpty())) {
				weatherDelay = Integer.parseInt(tokens[25]);
			}
			if(!(tokens[26].contains("NA")||tokens[26].isEmpty())) {
				nasDelay = Integer.parseInt(tokens[26]);
			}
			if(!(tokens[27].contains("NA")||tokens[27].isEmpty())) {
```

```
                        securityDelay = Integer.parseInt(tokens[27]);
                    }
                    if(!(tokens[28].contains("NA")||tokens[28].isEmpty())) {
                        lateDelay = Integer.parseInt(tokens[28]);
                    }

                    totalDelay = arrDelay + depDelay + carrierDelay + weatherDelay + nasDelay + securityDelay +
lateDelay;

                    outKey.set(year + "\t" + airport);
                    outValue.setTotalDelay(totalDelay);
                    outValue.setTotalTrips(1);

                    context.write(outKey, outValue);
            }
        }

}
```

Best Metric Reducer:

```
package Hadoop.Project_BestAirportPerYear;

import java.io.IOException;

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

public class BestMetricReducer extends Reducer<Text,FlightAnalysisCustomWritable,Text,FloatWritable> {

        public void reduce(Text key,Iterable<FlightAnalysisCustomWritable> value,Context context) throws IOException,
InterruptedException {
            Text outKey = new Text();
            FloatWritable outValue = new FloatWritable();
            int sum = 0;
            int count = 0;
            float metric = 0;

            for (FlightAnalysisCustomWritable val : value) {
                sum += val.getTotalDelay();
                count+= val.getTotalTrips();
            }

            metric = sum/count;

            outKey.set(key);
            outValue.set(metric);

            context.write(outKey, outValue);
        }
```

```
}
```

Best Metric Mapper:

```
package Hadoop.Project_BestAirportPerYear;

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

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

public class BestMetricMapper extends Mapper<LongWritable,Text,Text,Text>{
	private Text outKey = new Text();
	private Text outValue = new Text();

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

		String[] tokens = value.toString().split("\t");
		String year = tokens[0];
		String airport = tokens[1];
		String metric = tokens[2];

		outKey.set(airport);
		outValue.set(String.join("\t",Arrays.asList("A",airport,year,metric)));
		context.write(outKey, outValue);
	}

}
```

Airport Mapper

```
package Hadoop.Project_BestAirportPerYear;

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

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

public class AirportMapper extends Mapper<LongWritable,Text,Text,Text> {
	private Text outKey = new Text();
	private Text outValue = new Text();

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


	if(key.get() > 0) {
```

```
                String[] tokens = value.toString().replace("\"", "").split(",");
                if(!(tokens[0].contains("NA")||tokens[0].isEmpty())) {

                        outKey.set(tokens[0]);
                        outValue.set(String.join("\t",Arrays.asList("B",tokens[0],tokens[1])));
                        context.write(outKey, outValue);
                }
        }
        }

}
```

Join Reducer:

```
package Hadoop.Project_BestAirportPerYear;

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

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

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

        public static final Text EMPTY_TEXT = new Text("");
        private Text temp = new Text();
        private ArrayList<Text> listA = new ArrayList<Text>();
        private ArrayList<Text> listB = new ArrayList<Text>();
        private String joinType = null;

        @Override
        protected void setup(Context context) throws IOException, InterruptedException {
                joinType = context.getConfiguration().get("join.type");
        }

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

                listA.clear();
                listB.clear();

                for(Text val:value) {
                        temp = val;

                        if(temp.charAt(0)=='A') {
                                listA.add(new Text(temp.toString().substring(1)));
                        }
                        else if(temp.charAt(0)=='B') {
                                listB.add(new Text(temp.toString().substring(1)));
                        }
```

```
                }
                executeJoin(context);

        }

        private void executeJoin(Context context)throws IOException, InterruptedException {
                if(joinType.equalsIgnoreCase("inner")) {
                        if(!listA.isEmpty() && !listB.isEmpty()){
         for(Text A: listA){
            for(Text B: listB){
               //System.out.println("ListAB contains : "+ A + " " + B);
               context.write(A, B);
            }
         }
                        }
                }
        }
}
```

Top 10 Mapper:

```
package Hadoop.Project_BestAirportPerYear;

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

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

public class Top10AirportsPerYearMapper extends Mapper<LongWritable,Text,Text,FloatWritable> {
        private TreeMap<Text,Float> records = new TreeMap<Text,Float>();
        private Text outKey = new Text();
        public void map(LongWritable key,Text value,Context context) throws IOException, InterruptedException {
                String[] tokens = value.toString().split("\t");
                String airport = tokens[6].trim();
                String airportToken = tokens[5].trim();
                String year = tokens[2].trim();
                float metric = Float.parseFloat(tokens[3].trim());

                String outKey = String.join("\t", Arrays.asList(year,airportToken,airport));
                //outKey.setYear(year);
                //outKey.setAirport(airport+"\t"+airportToken);
                records.put(new Text(outKey),metric);

        }

        protected void cleanup(Context context) throws IOException, InterruptedException {
```

```
                for (Map.Entry<Text, Float> entry : records.entrySet())
    {

      float count = entry.getValue();
      Text name = entry.getKey();

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

        }
}
```

Top 10 Reducer:

```
package Hadoop.Project_BestAirportPerYear;

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

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

public class Top10AirportsReducer extends Reducer<Text,FloatWritable,Text,FloatWritable> {

        private TreeMap<Float,Text> outRecords = new TreeMap<Float,Text>();

        public void reduce(Text key, Iterable<FloatWritable> values,Context context) throws IOException,
InterruptedException {

                float outResult = 0;

                for(FloatWritable value : values) {
                        outResult = value.get();

                }
                outRecords.put(outResult,new Text(key));

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

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

        for (Map.Entry<Float, Text> entry : outRecords.entrySet())
  {
    float count = entry.getKey();
    Text name = entry.getValue();
```

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

            }
}
```

Year Partitioner:

```
package Hadoop.Project_BestAirportPerYear;

import org.apache.hadoop.conf.Configurable;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Partitioner;

public class YearPartitioner extends Partitioner<Text, FloatWritable> implements Configurable{

        private static final String MIN_LAST_ACCESS_DATE_YEAR = "min.last.access.date.year";
        private Configuration conf = null;
        private int minLastAccessDateYear = 0;

        public Configuration getConf() {
                // TODO Auto-generated method stub
                return conf;
        }

        public void setConf(Configuration conf) {
                // TODO Auto-generated method stub
                this.conf = conf;
                minLastAccessDateYear = conf.getInt(MIN_LAST_ACCESS_DATE_YEAR, 0);
        }

        @Override
        public int getPartition(Text key, FloatWritable value, int numPartitions) {
                // TODO Auto-generated method stub
                return Integer.parseInt(key.toString().split("\t")[0]) - minLastAccessDateYear;
        }

        public static void setMinLastAccessDateYear(Job job,int minLastAccessDateYear) {
                job.getConfiguration().setInt(MIN_LAST_ACCESS_DATE_YEAR,minLastAccessDateYear);
        }
}
```

Top 10 Routes Experiencing Delay

Driver:

```
package Hadoop.Project_Top10RoutesWithMostDelays;

import java.io.IOException;

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

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

      try {
                    Job job1 = Job.getInstance(config,"Routes with Most Delays");

                    job1.setJarByClass(App.class);

                    job1.setMapOutputKeyClass(Text.class);
                    job1.setMapOutputValueClass(DoubleWritable.class);

                    job1.setMapperClass(RouteMapper.class);
                    job1.setCombinerClass(RouteDelayReducer.class);
                    job1.setReducerClass(RouteDelayReducer.class);

                    job1.setInputFormatClass(TextInputFormat.class);
                    job1.setOutputFormatClass(TextOutputFormat.class);

                    job1.setOutputKeyClass(Text.class);
                    job1.setOutputValueClass(DoubleWritable.class);

                    FileInputFormat.addInputPath(job1,new Path(args[0]));
                    FileOutputFormat.setOutputPath(job1,new Path(args[1]));

                    FileSystem fs = FileSystem.get(config);
                    fs.delete(new Path(args[1]), true);

                    boolean success_1 = job1.waitForCompletion(true);
```

```
                System.out.println(success_1);

                Job job2 = Job.getInstance(config,"Routes with Most Delays");

                job2.setJarByClass(App.class);

                job2.setMapOutputKeyClass(Text.class);
                job2.setMapOutputValueClass(DoubleWritable.class);

                job2.setMapperClass(Top10RouteMapper.class);
                job2.setReducerClass(Top10RoutesReducer.class);
                //job2.setNumReduceTasks(1);

                job2.setInputFormatClass(TextInputFormat.class);
                job2.setOutputFormatClass(TextOutputFormat.class);

                job2.setOutputKeyClass(Text.class);
                job2.setOutputValueClass(DoubleWritable.class);

                FileInputFormat.addInputPath(job2,new Path(args[1]));
                FileOutputFormat.setOutputPath(job2,new Path(args[2]));

                fs.delete(new Path(args[2]), true);

                boolean success_2 = job2.waitForCompletion(true);
                System.out.println(success_2);

        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

    }
}
```

Route Delay Mapper:

```
package Hadoop.Project_Top10RoutesWithMostDelays;

import java.io.IOException;

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

public class RouteMapper extends Mapper<LongWritable,Text,Text,DoubleWritable> {
        private Text outKey = new Text();
        private DoubleWritable outValue = new DoubleWritable(0.0);

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

```
                try {
                        if(key.get() == 0 && value.toString().contains("header")) {
                                return;
                        }
                        else {
                        String[] tokens = value.toString().split(",");
                        if(!(tokens[16].isEmpty()||tokens[16].equalsIgnoreCase("NA")) &&
!(tokens[17].isEmpty()||tokens[17].equalsIgnoreCase("NA"))){
                                String arivalAirport = tokens[16].trim();
                                String depAirport = tokens[17].trim();
                                String combinedPath = arivalAirport +"-" + depAirport;
                                double arrDelay;
                                double depDelay;
                                if(tokens[14].contains("NA")) {
                                        arrDelay = 0.0;
                                }
                                else {
                                        arrDelay = Double.parseDouble(tokens[14]);
                                }
                                if(tokens[15].contains("NA")) {
                                        depDelay = 0.0;
                                }
                                else {
                                        depDelay = Double.parseDouble(tokens[15]);
                                }
                                double totalDelay = arrDelay+depDelay;

                                outKey.set(combinedPath);
                                outValue.set(totalDelay);

                                context.write(outKey, outValue);
                        }
                        }
                }

                        catch (Exception e) {
                    e.printStackTrace();
                  }

        }

}
```

Route Delay Reducer:

```
package Hadoop.Project_Top10RoutesWithMostDelays;

import java.io.IOException;

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

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

public class RouteDelayReducer extends Reducer<Text,DoubleWritable,Text,DoubleWritable>{
	@Override
	public void reduce(Text key,Iterable<DoubleWritable> values,Context context) throws IOException,
InterruptedException {
		DoubleWritable outValue = new DoubleWritable();
		double sum = 0.0;
		for(DoubleWritable val:values) {
			sum += val.get();
		}
		outValue.set(sum);
		context.write(key, outValue);

	}

}
```

Top 10 Route Mapper:

```
package Hadoop.Project_Top10RoutesWithMostDelays;

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

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

public class Top10RouteMapper extends Mapper<LongWritable,Text,Text,DoubleWritable> {
	private TreeMap<Double,Text> records = new TreeMap<Double,Text>();

	public void map(LongWritable key,Text value,Context context) throws IOException, InterruptedException {
		String[] tokens = value.toString().split("\t");
		records.put(Double.parseDouble(tokens[1].trim()), new Text(tokens[0].trim()));

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

	protected void cleanup(Context context) throws IOException, InterruptedException {
		for (Map.Entry<Double, Text> entry : records.entrySet())
	{

	  double count = entry.getKey();
	  Text name = entry.getValue();

	  context.write(name, new DoubleWritable(count));
```

```
    }

        }

}
```

Top 10 Route Reducer:

```
package Hadoop.Project_Top10RoutesWithMostDelays;

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

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

public class Top10RoutesReducer extends Reducer<Text,DoubleWritable,Text,DoubleWritable> {

        private TreeMap<Double,Text> outRecords = new TreeMap<Double,Text>(Collections.reverseOrder());

        public void reduce(Text key, Iterable<DoubleWritable> values,Context context) throws IOException,
InterruptedException {

                double outResult = 0.0;

                for(DoubleWritable value : values) {
                        outResult = value.get();

                }
                outRecords.put(outResult, new Text(key));

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

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

        for (Map.Entry<Double, Text> entry : outRecords.entrySet())
  {
    double count = entry.getKey();
    Text name = entry.getValue();

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

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Total Flights Delayed by Weather

```
airlineRawData = LOAD '/AirArrivalDataSet/'
USING PigStorage(',') AS (Year:chararray, Month:chararray, DayofMonth:chararray,
DayOfWeek:chararray, DepTime:chararray, CRSDepTime:chararray, ArrTime:chararray,
CRSArrTime:chararray, UniqueCarrier:chararray, FlightNum:chararray,
TailNum:chararray, ActualElapsedTime:chararray, CRSElapsedTime:chararray,
AirTime:chararray, ArrDelay:int, DepDelay:int, Origin:chararray, Dest:chararray,
Distance:chararray, TaxiIn:chararray, TaxiOut:chararray, Cancelled:chararray,
CancellationCode:chararray, Diverted:chararray, CarrierDelay:chararray,
WeatherDelay:int, NASDelay:chararray, SecurityDelay:chararray,
LateAircraftDelay:chararray);

filtered_data = FILTER airlineRawData BY Year in ('2005,2006','2007','2008') ;

records = FOREACH filtered_data GENERATE Year, WeatherDelay;
grpd = GROUP records BY Year;
final_df = FOREACH grpd {
   weather_delays = FILTER records BY WeatherDelay > 0;
   GENERATE group, COUNT(weather_delays);
};

STORE final_df INTO '/Project/pig/DelayByWeather';
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Top 10 Carriers with Minimum Delay

```
airlineRawData = LOAD '/AirArrivalDataSet/'
USING PigStorage(',') AS (Year:chararray, Month:chararray, DayofMonth:chararray,
DayOfWeek:chararray, DepTime:chararray, CRSDepTime:chararray, ArrTime:chararray,
CRSArrTime:chararray, UniqueCarrier:chararray, FlightNum:chararray,
TailNum:chararray, ActualElapsedTime:chararray, CRSElapsedTime:chararray,
AirTime:chararray, ArrDelay:int, DepDelay:int, Origin:chararray, Dest:chararray,
Distance:chararray, TaxiIn:chararray, TaxiOut:chararray, Cancelled:chararray,
CancellationCode:chararray, Diverted:chararray, CarrierDelay:chararray,
WeatherDelay:int, NASDelay:chararray, SecurityDelay:chararray,
LateAircraftDelay:chararray);

filtered_data = FILTER airlineRawData BY Year == '2008' AND (DepDelay is not null OR ArrDelay is not null );

carrier = LOAD '/Carrier/'
USING PigStorage(',') AS (iata:chararray,CarrierName:chararray);

carrierData = FOREACH carrier GENERATE REPLACE(iata,'\\"','') as iata,REPLACE(CarrierName,'\\"','') as CarrierName;

filter_carrierData = FILTER carrierData by (iata is not null) AND (CarrierName is not null) ;

mergedData = JOIN filtered_data by UniqueCarrier, filter_carrierData by iata;

records = FOREACH mergedData GENERATE CarrierName, DepDelay;
grpd = GROUP records BY CarrierName;
counts = FOREACH grpd GENERATE group AS CarrierName,
                    COUNT(records) AS DelayCount;
sorted = ORDER counts BY DelayCount DESC;
top10 = LIMIT sorted 10;

STORE top10 INTO '/Project/pig/Top10CarriersWithMinimumDelay';
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

How many flights that were delayed due to bad weather

```
airlineRawData = LOAD '/AirArrivalDataSet/'
USING PigStorage(',') AS (Year:chararray, Month:chararray, DayofMonth:chararray,
DayOfWeek:chararray, DepTime:chararray, CRSDepTime:chararray, ArrTime:chararray,
CRSArrTime:chararray, UniqueCarrier:chararray, FlightNum:chararray,
TailNum:chararray, ActualElapsedTime:chararray, CRSElapsedTime:chararray,
AirTime:chararray, ArrDelay:int, DepDelay:int, Origin:chararray, Dest:chararray,
Distance:chararray, TaxiIn:chararray, TaxiOut:chararray, Cancelled:chararray,
CancellationCode:chararray, Diverted:chararray, CarrierDelay:chararray,
WeatherDelay:int, NASDelay:chararray, SecurityDelay:chararray,
LateAircraftDelay:chararray);

filtered_data = FILTER airlineRawData BY Year == '2008';

carrier = LOAD '/Carrier/'
USING PigStorage(',') AS (iata:chararray,CarrierName:chararray);

carrierData = FOREACH carrier GENERATE REPLACE(iata,'\\"','') as iata,REPLACE(CarrierName,'\\"','') as CarrierName;

filter_carrierData = FILTER carrierData by (iata is not null) AND (CarrierName is not null) ;

mergedData = JOIN filtered_data by UniqueCarrier, filter_carrierData by iata;

records = FOREACH mergedData GENERATE Month,CarrierName,FlightNum,Origin,
        (ArrDelay + DepDelay) AS SumDelay;

grpd = GROUP records BY Month;

top_delays = FOREACH grpd {
 sum_delays = ORDER records BY SumDelay DESC;
 sum_delays_top1 = LIMIT sum_delays 1;
 GENERATE group AS Month,FLATTEN(sum_delays_top1.CarrierName) AS Carrier,FLATTEN(sum_delays_top1.Origin) AS
Airport , FLATTEN(sum_delays_top1.SumDelay) AS SumDelay;};

STORE top_delays INTO '/Project/pig/top10_delay' USING PigStorage(',');
```

NAME: KAUSHAL CHAUDHARY
NUID: 001886763

Top 10 Routes that were Diverted

```
airlineRawData = LOAD '/AirArrivalDataSet/'
USING PigStorage(',') AS (Year:chararray, Month:chararray, DayofMonth:chararray,
DayOfWeek:chararray, DepTime:chararray, CRSDepTime:chararray, ArrTime:chararray,
CRSArrTime:chararray, UniqueCarrier:chararray, FlightNum:chararray,
TailNum:chararray, ActualElapsedTime:chararray, CRSElapsedTime:chararray,
AirTime:chararray, ArrDelay:int, DepDelay:int, Origin:chararray, Dest:chararray,
Distance:chararray, TaxiIn:chararray, TaxiOut:chararray, Cancelled:chararray,
CancellationCode:chararray, Diverted:chararray, CarrierDelay:chararray,
WeatherDelay:int, NASDelay:chararray, SecurityDelay:chararray,
LateAircraftDelay:chararray);

filtered_data = FILTER airlineRawData BY Year in ('2005,2006','2007','2008') and Diverted == '1';

records = FOREACH filtered_data GENERATE Origin,Dest,Diverted;

grpd = GROUP records by (Origin,Dest);

counted = FOREACH grpd GENERATE  FLATTEN(group) as (Origin , Dest) , COUNT(records) as diverted_count;

Ordered = ORDER counted by diverted_count Desc;

top15 = LIMIT Ordered  10;

STORE top15 INTO '/Project/pig/TopRoutesWithDiversion' using PigStorage(',');
```