Flight Delay Analysis

Poojith Shankar Shetty
001821661
4/26/2019

AGENDA

- Introduction
- Dataset Overview
- List of Analysis
- Representation using Tableau
- Analysis Outputs
- Technologies and Patterns used
- Appendix
- Additional Screenshots

INTRODUCTION

- Flight Cancellation Analysis
  - Based on Airlines, month, State etc.
  - Reason for the Delay.
- Flight Delay Analysis
  - Time of Delay
  - Based on Airlines, month and State
  - Reason for the Delay
- Dataset Link
  - https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time
- Flight Distance Travelled
  - Details about how much distance the different flight travelled

DATA SET OVERVIEW

Columns of Dataset:

YEAR	ORIGIN_CITY_NAME	**DEP_DELAY_NEW**	CRS_ELAPSED_TIME
QUARTER	**ORIGIN_STATE_ABR**	TAXI_OUT	ACTUAL_ELAPSED_TIME
MONTH	**ORIGIN_STATE_NM**	WHEELS_ON	AIR_TIME
DAY_OF_MONTH	DEST_AIRPORT_ID	TAXI_IN	FLIGHTS
DAY_OF_WEEK	DEST	CRS_ARR_TIME	**DISTANCE**
FL_DATE	DEST_CITY_NAME	ARR_TIME	**DISTANCE_GROUP**
OP_UNIQUE_CARRIER	**DEST_STATE_ABR**	ARR_DELAY	**CARRIER_DELAY**
TAIL_NUM	**DEST_STATE_NM**	**ARR_DELAY_NEW**	**WEATHER_DELAY**
OP_CARRIER_FL_NUM	CRS_DEP_TIME	**CANCELLED**	**NAS_DELAY**
ORIGIN_AIRPORT_ID	DEP_TIME	**CANCELLATION_CODE**	**SECURITY_DELAY**
ORIGIN	DEP_DELAY	DIVERTED	**LATE_AIRCRAFT_DELAY**

DATA SET OVERVIEW

- Cancellation Code: These are the reasons for Flight cancellation
  - A – Carrier
  - B – Weather
  - C – National Air System
  - D – Security
- All the time in the data set are in Minutes.(Delay related columns).

LIST OF ANALYSIS

1. Used PutMerge to Merge the Data.
2. Used AWS Cloud EMR instance for running MapReduce Jobs.
3. Used counting with Counters to determine Number of Flights in different Distance Groups
4. Used Memory-Conscious Median and Standard Deviation of Distance travelled based on Airliner used separate Combiner to simplify the input to Reducer.
5. Calculated the each Cancellation of Flight based on Airliner each month.
6. Calculated the each Delay of Flight based on Airliner each month.
7. Did Secondary Sorting of the States and the available Airports in that state to know the list of Airports Present in US.(Partitioner is used in this case)
8. Calculated Average Delay by the airliner each month used Custom Writable class for this technique.

LIST OF ANALYSIS

9. Calculated Total Average Flight Delay by different Airliner.
10. Used Shuffling Technique to shuffle the Data.
11. Calculated Total Cancelled Flights based on Different Reason.
12. Calculated Total Delay of Flight each month
13. Determined Top 10 flight data based on longest Distance.
14. Determined Delay Median based on Airliner
15. Determined Distinct Airline in the Dataset.
16. Separated the data based on Cancellation Reason by using Binning Technique.

LIST OF ANALYSIS

17. Used Bloom Filter to Filter the data based on the Airport input.
18. Used Filtered output using Chaining and determined the Flights from the Airport using Inverted Index Method.
19. Based on the Carrier Names from a different Carrier CSV found the Full name of the Flights present in the original Data.
20. Determined the Minimum and Maximum Distance travelled by flight based on each Airliner.
21. Calculated Flight cancelled statewise in each month to determine which state has most flight cancelled.
22. Determined Flight delayed state wise in each month.
23. Found Top 10 Airline with Most Flights using Pig.
24. Filtered the Latest Data based on Year using Pig.
25. Joined Airline Data based on Airline Name using Pig.

Analysis using Tableau

Pages

Columns: Airline_Name

Rows: SUM(Count)

Filters

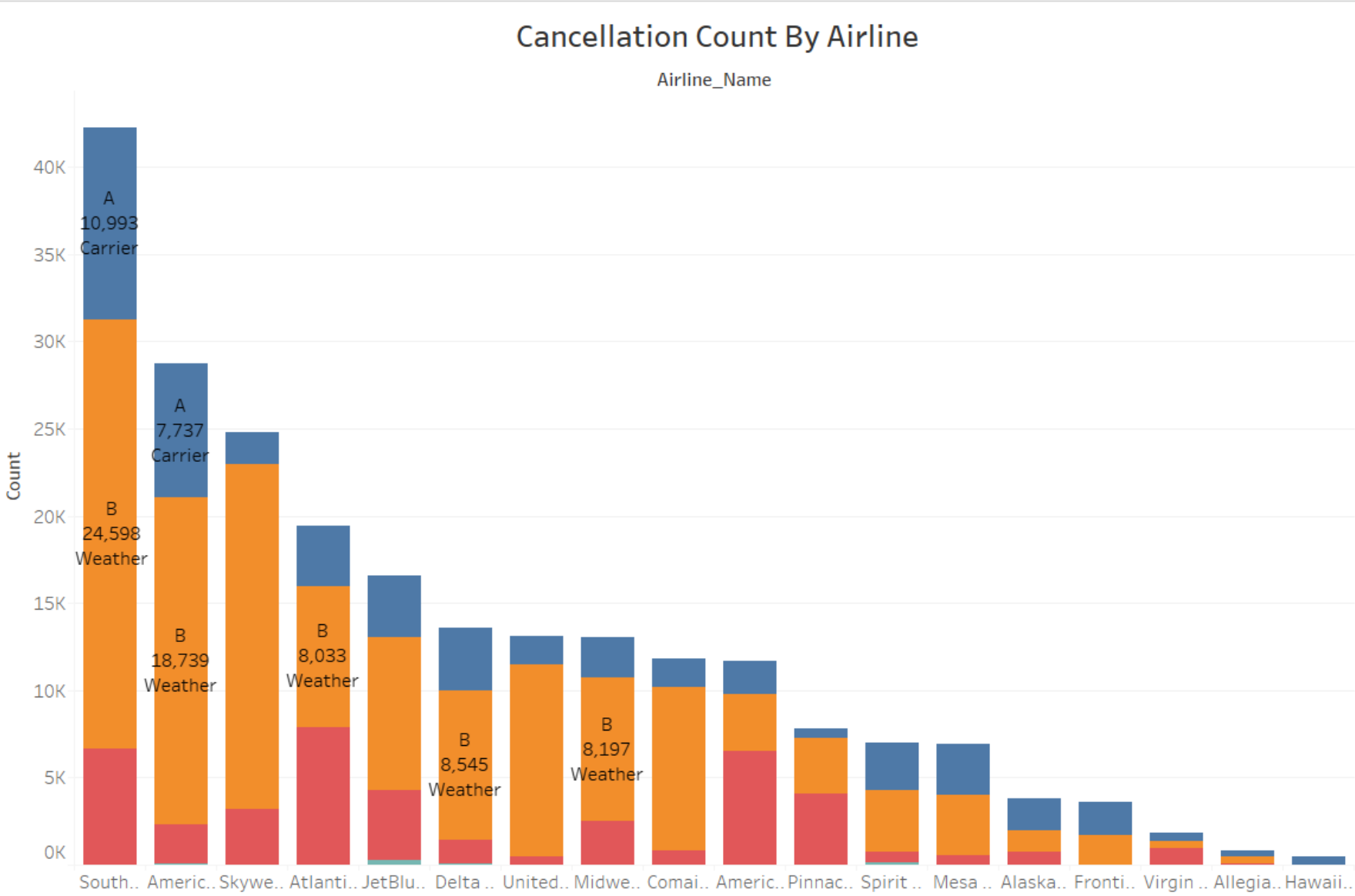
- Month
- Airline_Name

Marks

Automatic

Color | Size | Label | Detail | Tooltip

- Canc_Reason
- Canc_Reason
- SUM(Count)
- Description



Airline_Name

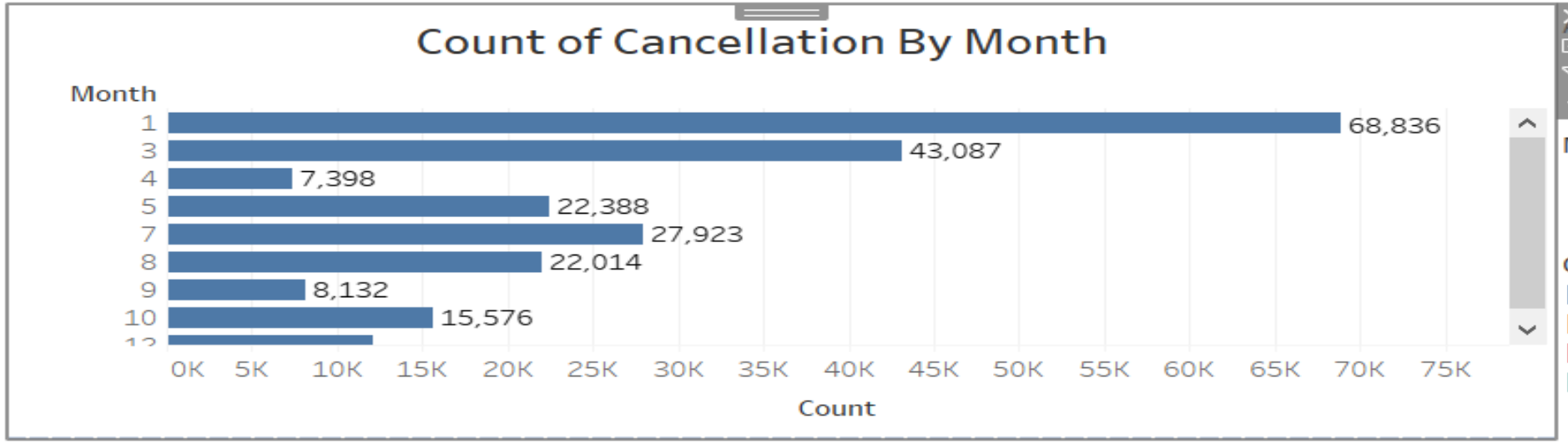
- (All)
- Alaska Airlines Inc.
- Allegiant Air
- American Airlines...
- American Eagle A...
- Atlantic Southea...
- Comair Inc.
- Delta Air Lines Inc.
- Frontier Airlines I...
- Hawaiian Airlines...
- JetBlue Airways
- Mesa Airlines Inc.
- Midwest Airline
- Pinnacle Airlines I...

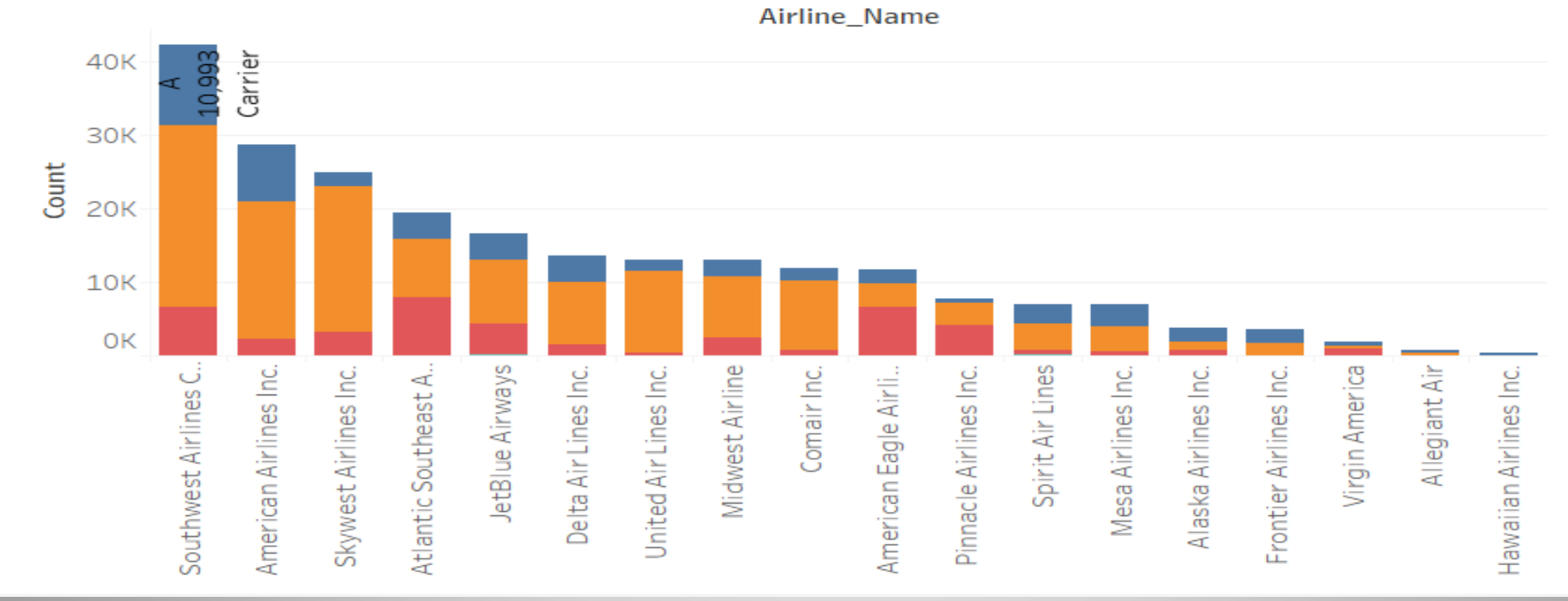
Month

- (All)
- 1
- 3
- 4
- 5
- 7
- 8
- 9
- 10
- 12

Canc_Reason

- A
- B
- C
- D


Count of Cancellation By Month
Month
1
3
4
5
7
8
9
10
68,836
43,087
7,398
22,388
27,923
22,014
8,132
15,576
0K 5K 10K 15K 20K 25K 30K 35K 40K 45K 50K 55K 60K 65K 70K 75K
Count


Cancellation Count By Airline
Airline_Name
Count
40K
30K
20K
10K
0K
A
10,993
Carrier
Southwest Airlines C...
American Airlines Inc.
Skywest Airlines Inc.
Atlantic Southeast A...
JetBlue Airways
Delta Air Lines Inc.
United Air Lines Inc.
Midwest Airline
Comair Inc.
American Eagle Airli...
Pinnacle Airlines Inc.
Spirit Air Lines
Mesa Airlines Inc.
Alaska Airlines Inc.
Frontier Airlines Inc.
Virgin America
Allegiant Air
Hawaiian Airlines Inc.

Pages

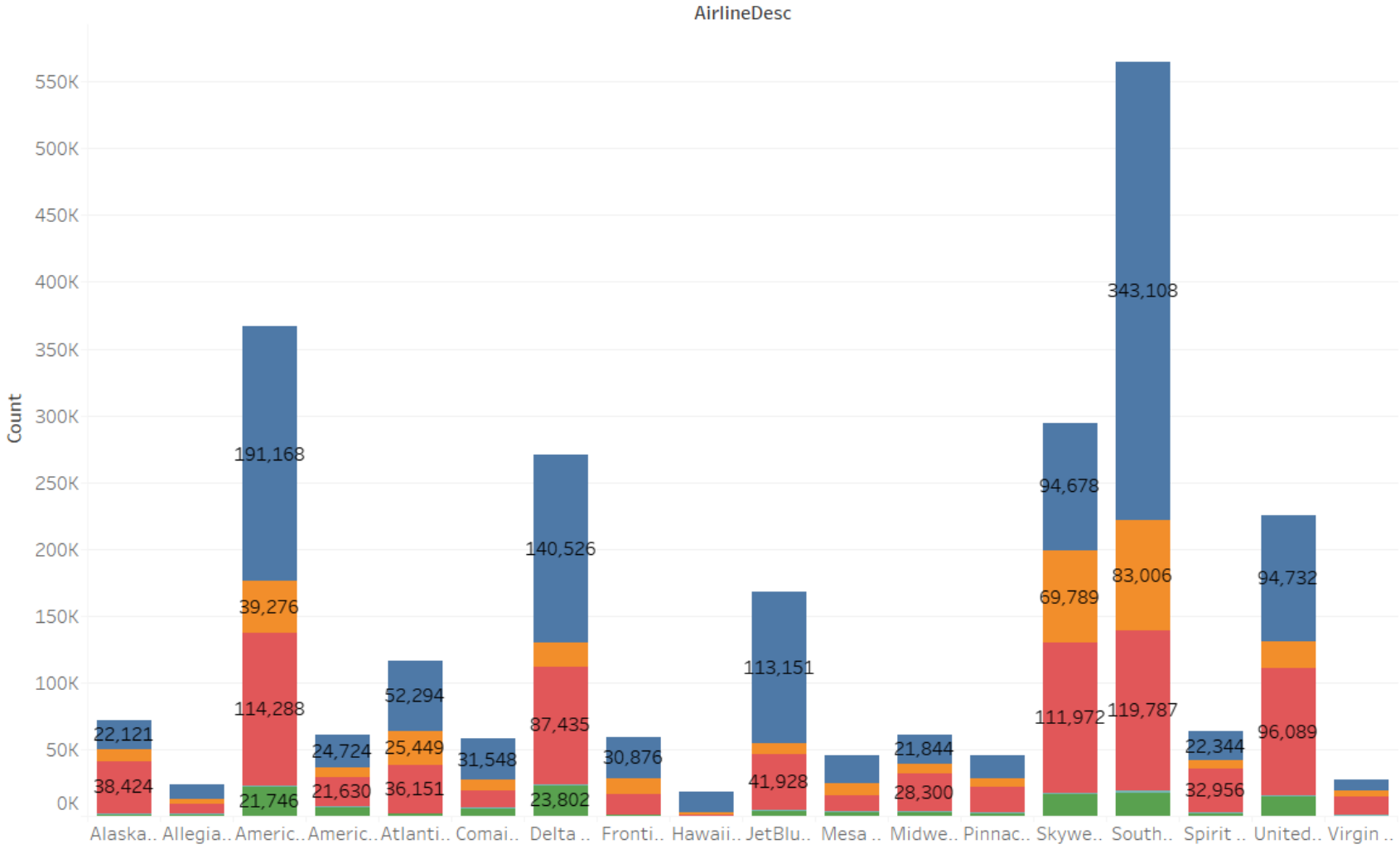
Columns: AirlineDesc

Rows: SUM(Count)

Filters
- Month
- AirlineDesc

Marks
- Automatic
- Color
- Size
- Label
- Detail
- Tooltip
- Delay_reason
- SUM(Count)

Delay Reason by airline count



AirlineDesc
- (All)
- Alaska Airlines Inc.
- Allegiant Air
- American Airlines...
- American Eagle A...
- Atlantic Southea...
- Comair Inc.
- Delta Air Lines Inc.
- Frontier Airlines I...
- Hawaiian Airlines...
- JetBlue Airways
- Mesa Airlines Inc.
- Midwest Airline

Month
- (All)
- 1
- 3
- 4
- 5
- 7
- 8
- 9
- 10
- 12

Delay_reason
- CARRIER_DELAY
- LATE_AIRCRAFT_DEL..
- NAS_DELAY
- SECURITY_DELAY
- WEATHER_DELAY

Pages

Columns: Delay_reason, AirlineDesc

Rows: SUM(Delay_month)

Filters

Month

AirlineDesc

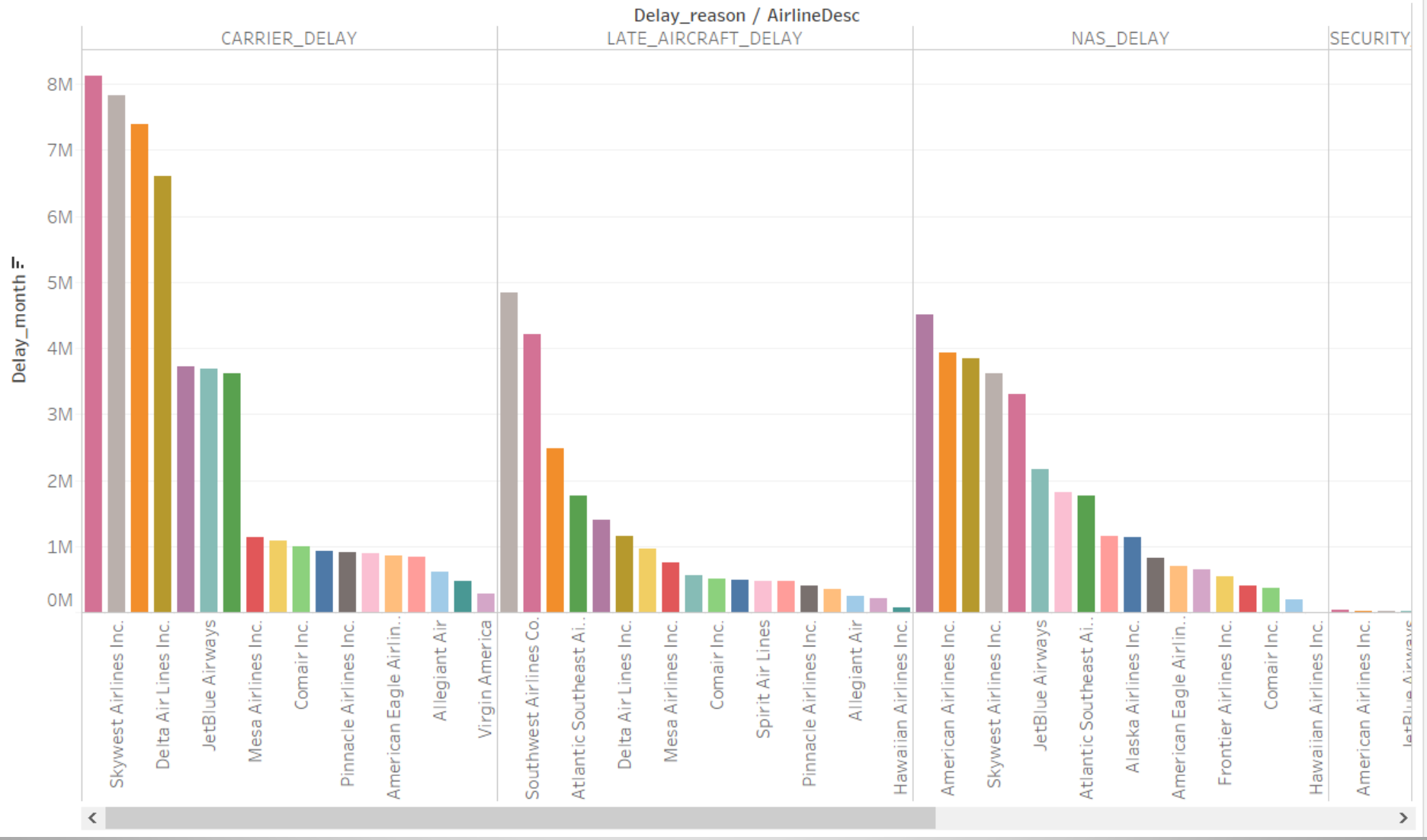
Marks

Automatic

Color | Size | Label | Detail | Tooltip

AirlineDesc

Delay By airline and reason



AirlineDesc

- (All)
- Alaska Airlines Inc.
- Allegiant Air
- American Airlines...
- American Eagle A...
- Atlantic Southea...
- Comair Inc.
- Delta Air Lines Inc.
- Frontier Airlines I...

Month

- (All)
- 1
- 3
- 4
- 5
- 7
- 8
- 9
- 10

AirlineDesc

- Alaska Airlines Inc.
- Allegiant Air
- American Airlines I..
- American Eagle Air..
- Atlantic Southeast ..
- Comair Inc.
- Delta Air Lines Inc.
- Frontier Airlines Inc.
- Hawaiian Airlines I..
- JetBlue Airways

Pages

Columns: Month, Delay_reason

Rows: SUM(Delay_month)

Filters

Marks

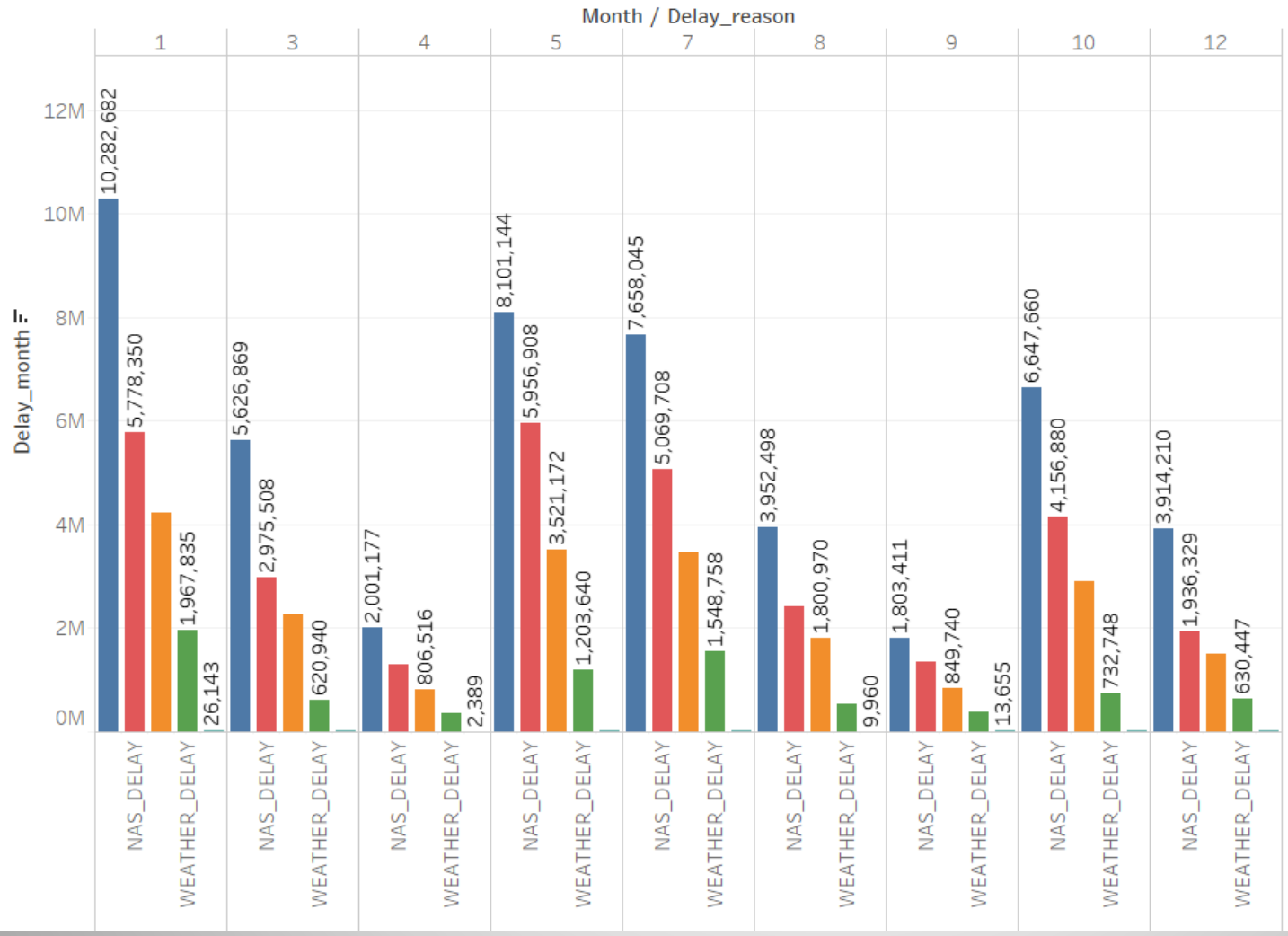
Automatic

Color | Size | Label | Detail | Tooltip

Delay_reason

SUM(Delay_mo..

Delay reason by month



Delay_reason

- CARRIER_DELAY
- LATE_AIRCRAFT_DEL..
- NAS_DELAY
- SECURITY_DELAY
- WEATHER_DELAY

Pages

Columns AirlineName

Rows Measure Values

Filters

Measure Names

Marks

Automatic

Color Size Label

Detail Tooltip

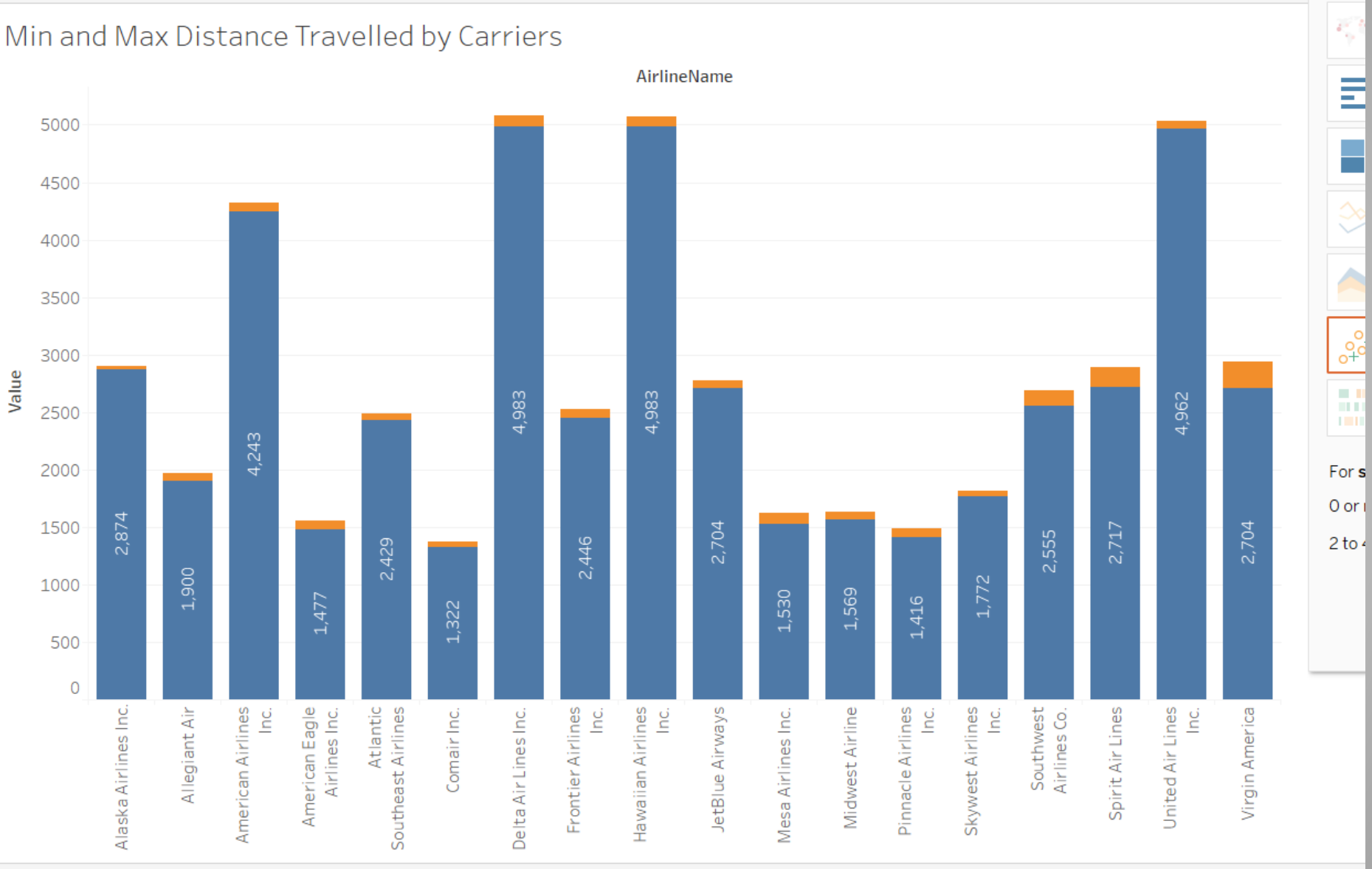
Measure Na..

Measure Values

Measure Values

SUM(Min)

SUM(Max)



For s

0 or

2 to

Pages

Columns: State

Rows: SUM(Count)

Filters

Month

Marks

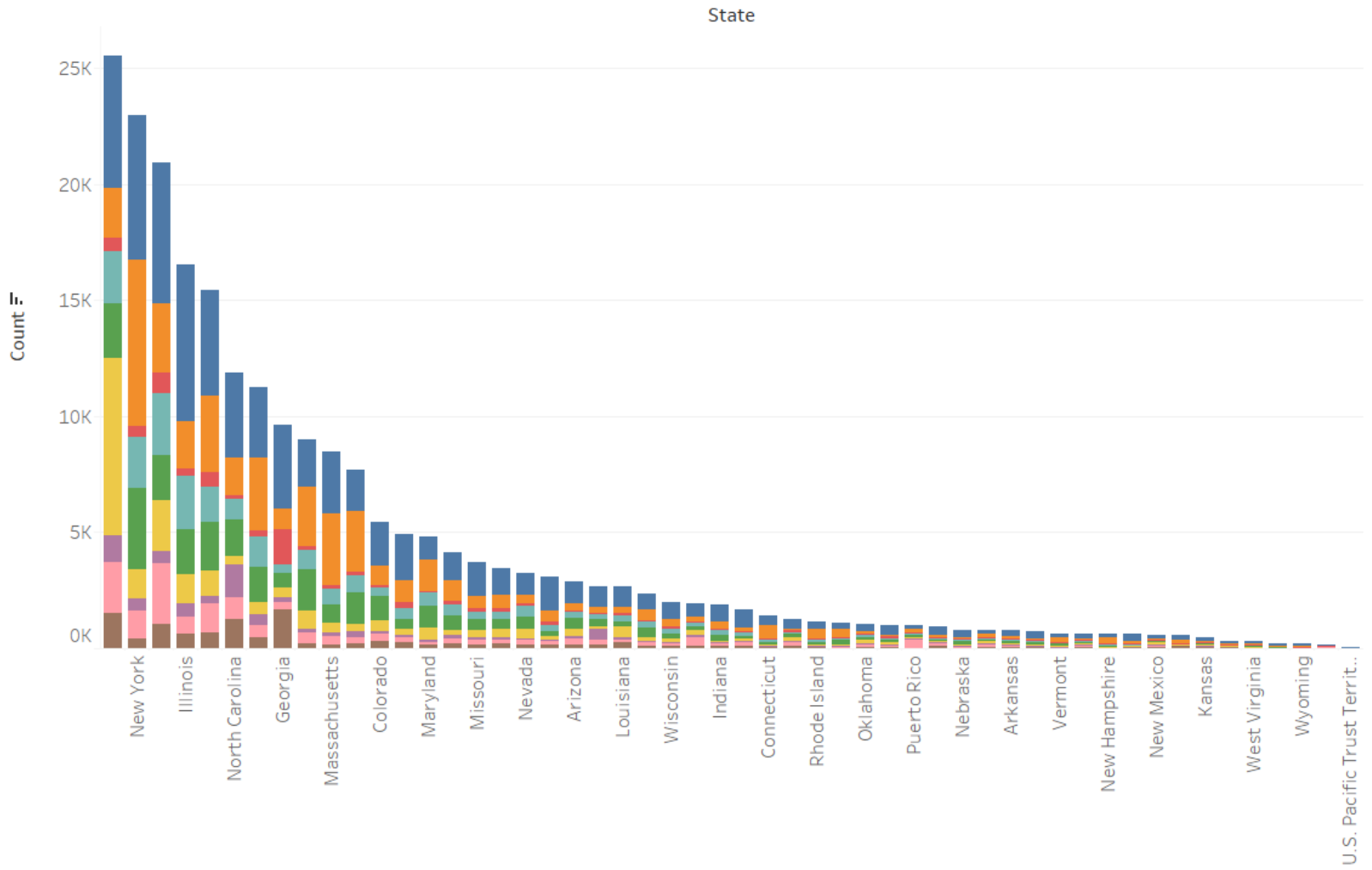
Automatic

Color | Size | Label

Detail | Tooltip

Month

Statewise Cancellation Monthly



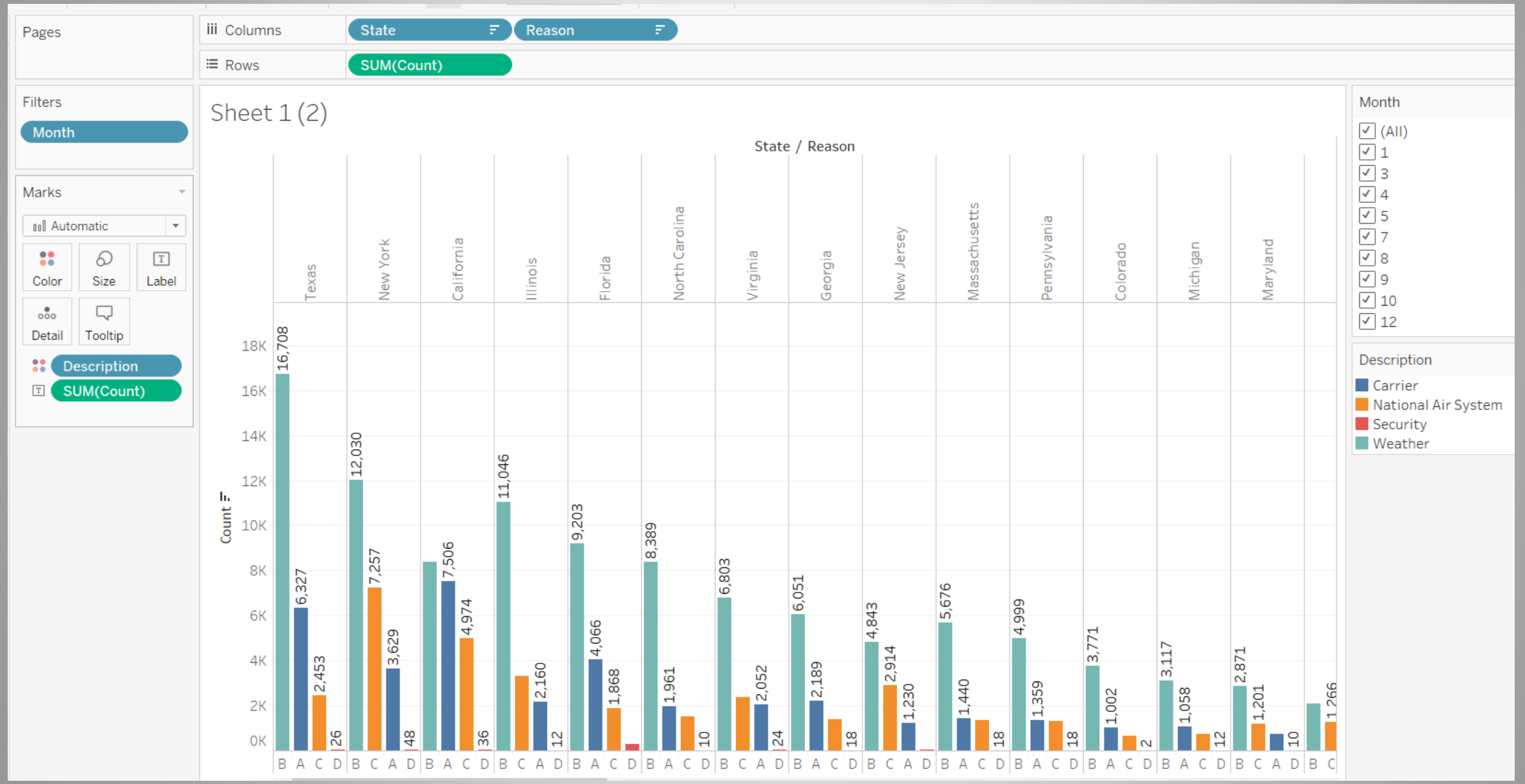
Month

- (All)
- 1
- 3
- 4
- 5
- 7
- 8
- 9
- 10
- 12

Month

- 1
- 3
- 4
- 5
- 7
- 8
- 9
- 10
- 12

STATEWISE BASED CANCELLATION COUNT



Pages

Columns: State

Rows: SUM(Count_Delay)

Filters

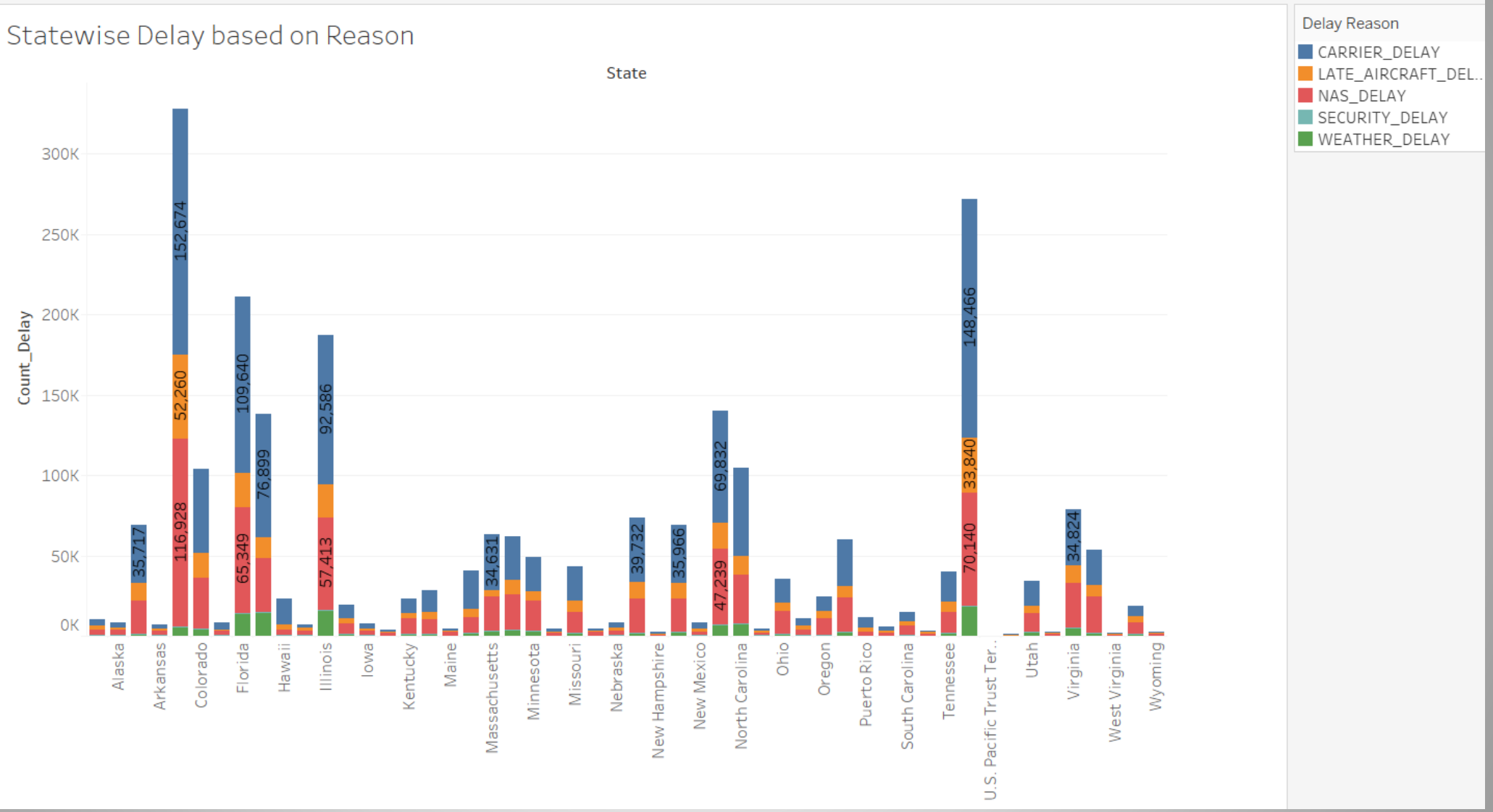
Marks

Automatic

Color | Size | Label | Detail | Tooltip

Delay Reason

SUM(Count_D..



Pages

Columns: State

Rows: SUM(Total_Delay)

Filters

Month

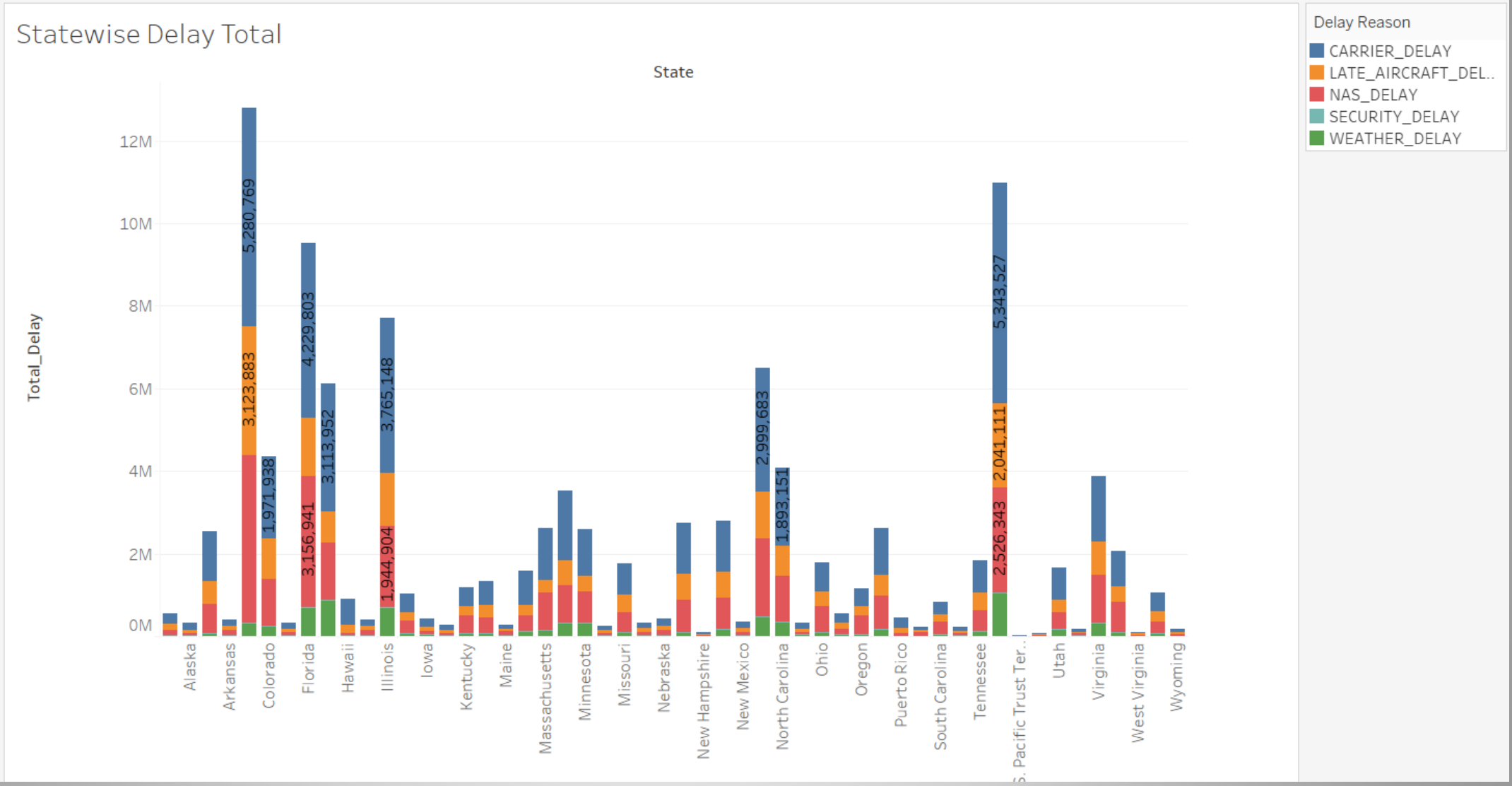
Marks

Automatic

Color | Size | Label | Detail | Tooltip

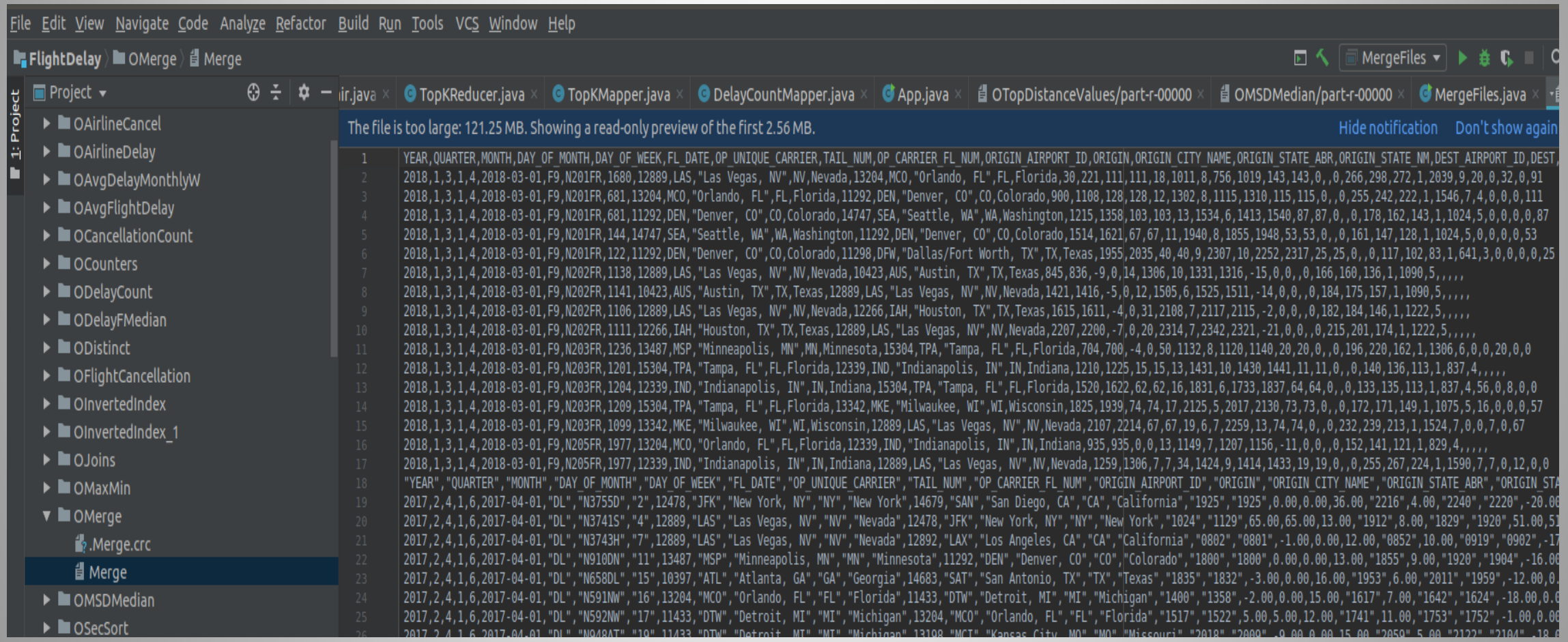
Delay Reason

SUM(Total_Del..



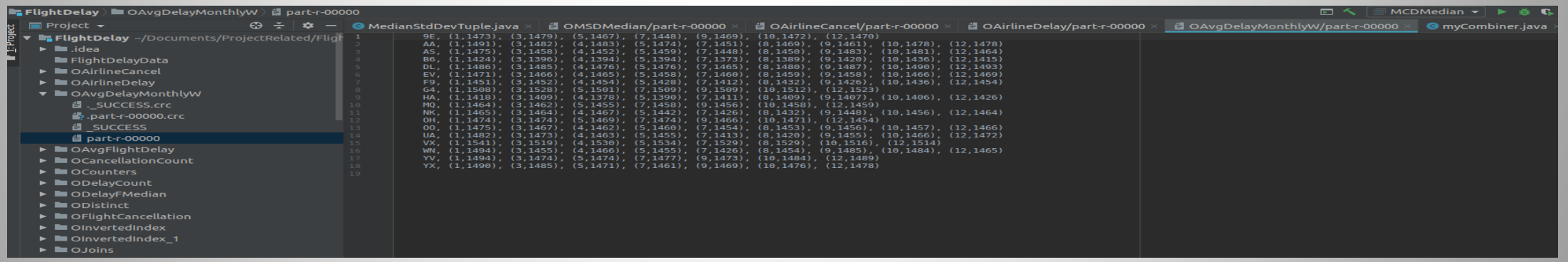
ANALYSIS OUTPUTS

PutMerge:



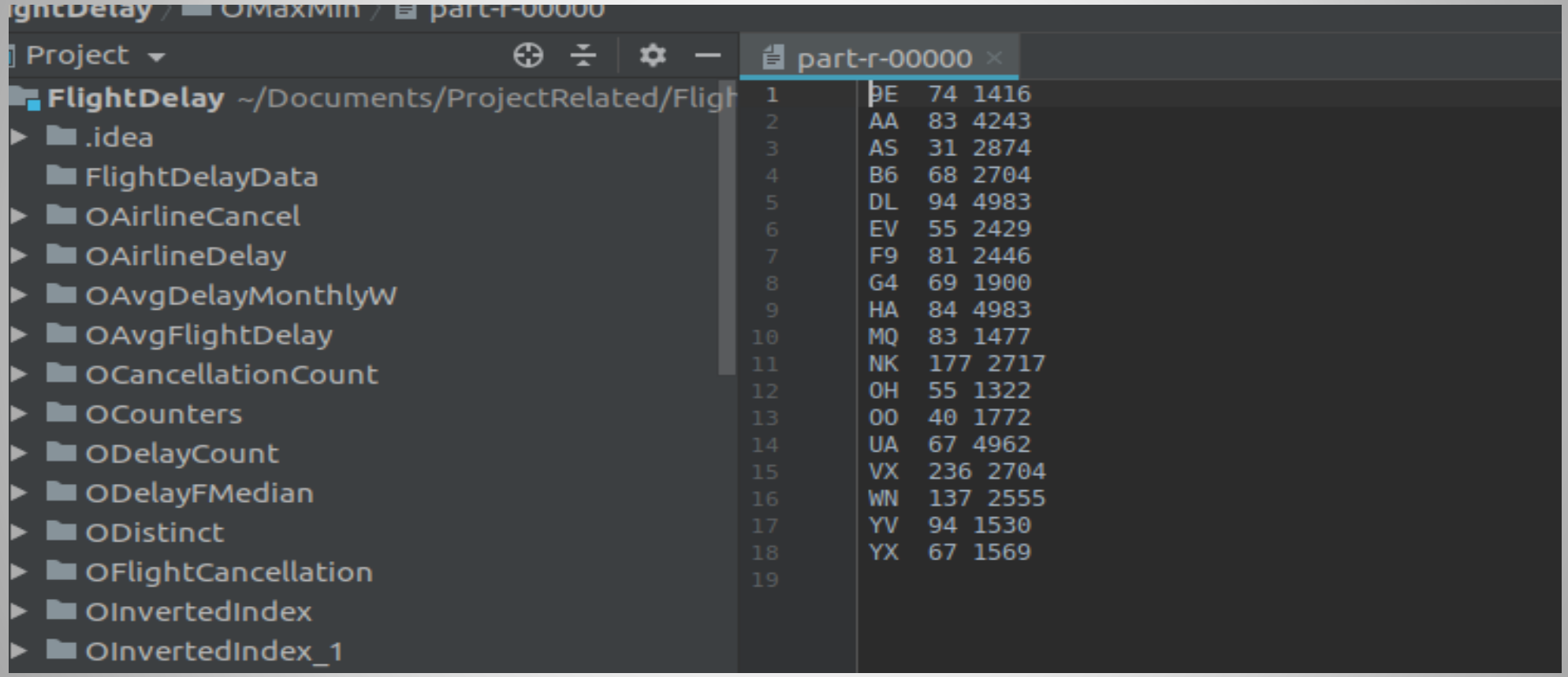
AIRLINE DELAY:

Calculated Average Delay by the airliner each month used Custom Writable class for this technique.



MIN-MAX DELAY:

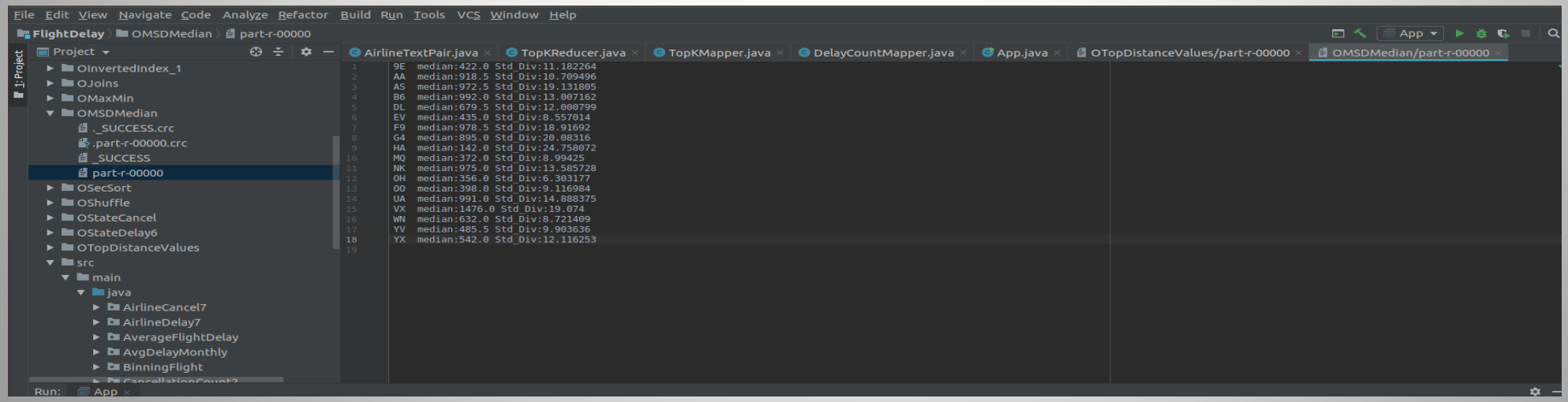
Determined the Minimum and Maximum Distance travelled by flight based on each Airliner.



MEDIAN

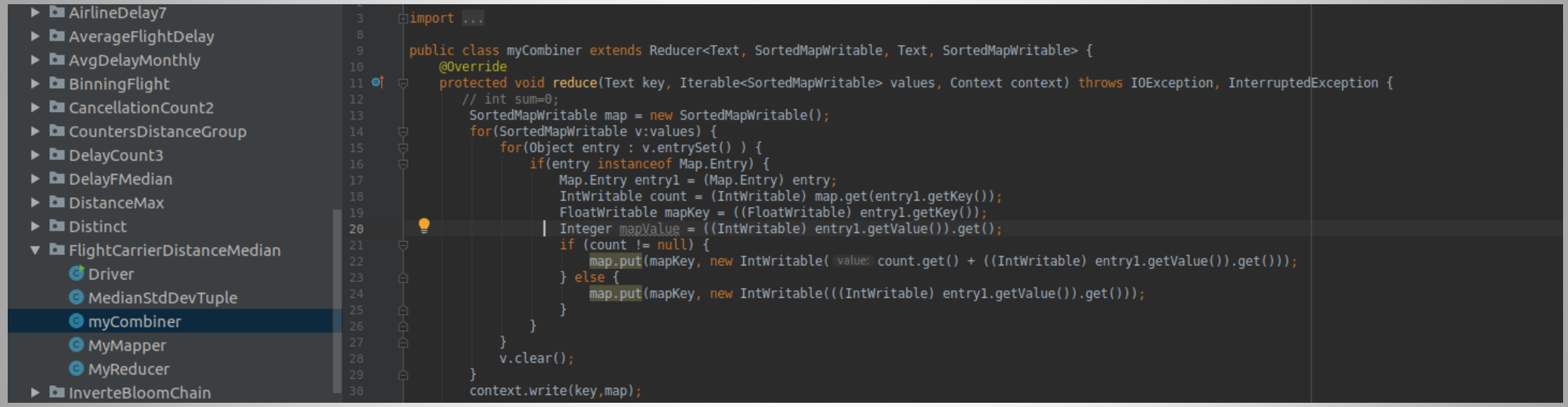
Determined Delay Median based on Airliner

```
protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
Integer median=0;
    //int sum = 0;
    //int count = 0;

/Integer temp= (int) Float.parseFloat(key.toString());
    try {
        for (Text value:values) {

            Integer temp = (int) Float.parseFloat(value.toString());
            if (temp != 0){
                array.add(temp);
            }
            //context.write(new Text(key), NullWritable.get());
        }
        Collections.sort(array);
        int len=array.size();
        if(array.size()%2==0){
           // median= array.get((int) len/2 - 1)+array.get((int) len/2);
            median= (array.get((int) len/2 - 1)+array.get((int) len/2)) /2;
        }
        else{
            median= array.get(len/2);
        }

        context.write(key,new Text( string: median+""));
```



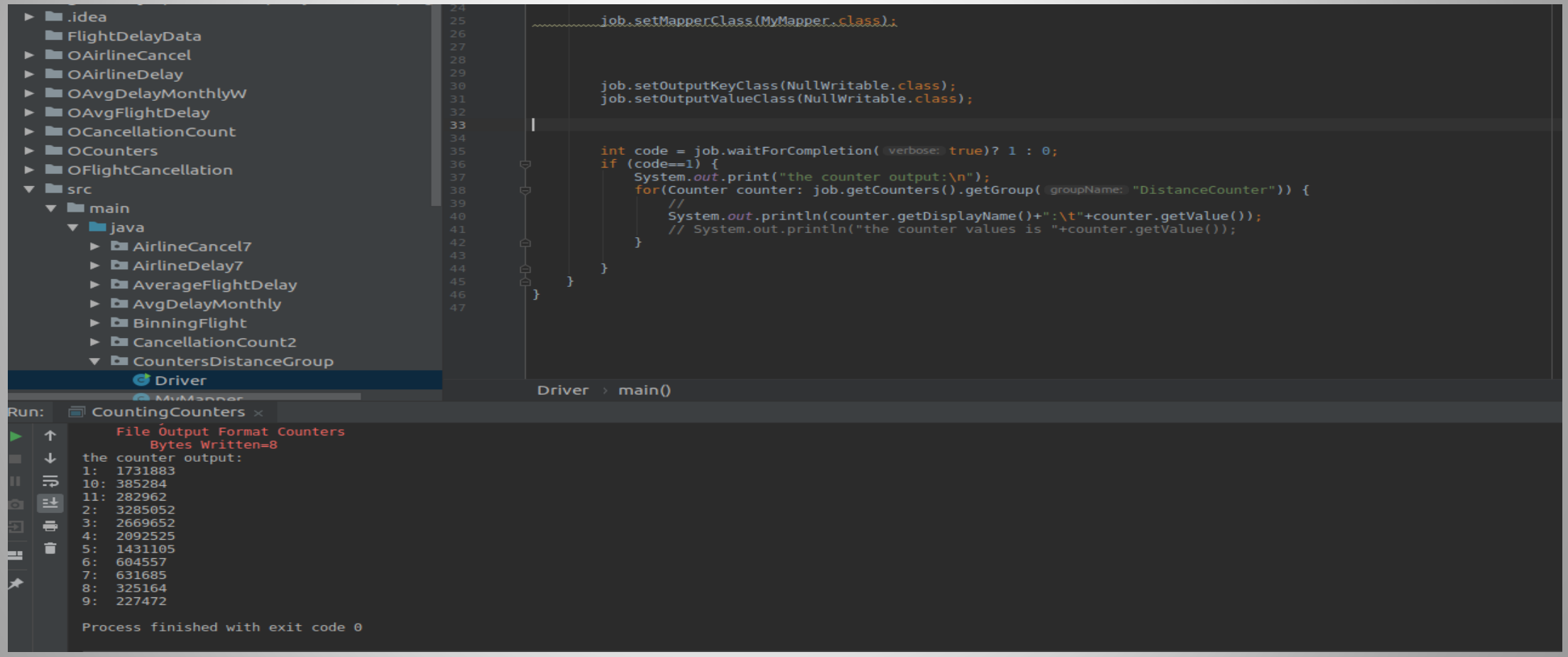
MEMORY CONSCIOUS MEDIAN

Used Memory-Conscious Median and Standard Deviation of Distance travelled based on Airliner used separate Combiner to simplify the input to Reducer.



COUNTING WITH COUNTERS

Used counting with Counters to determine Number of Flights in different Distance Groups

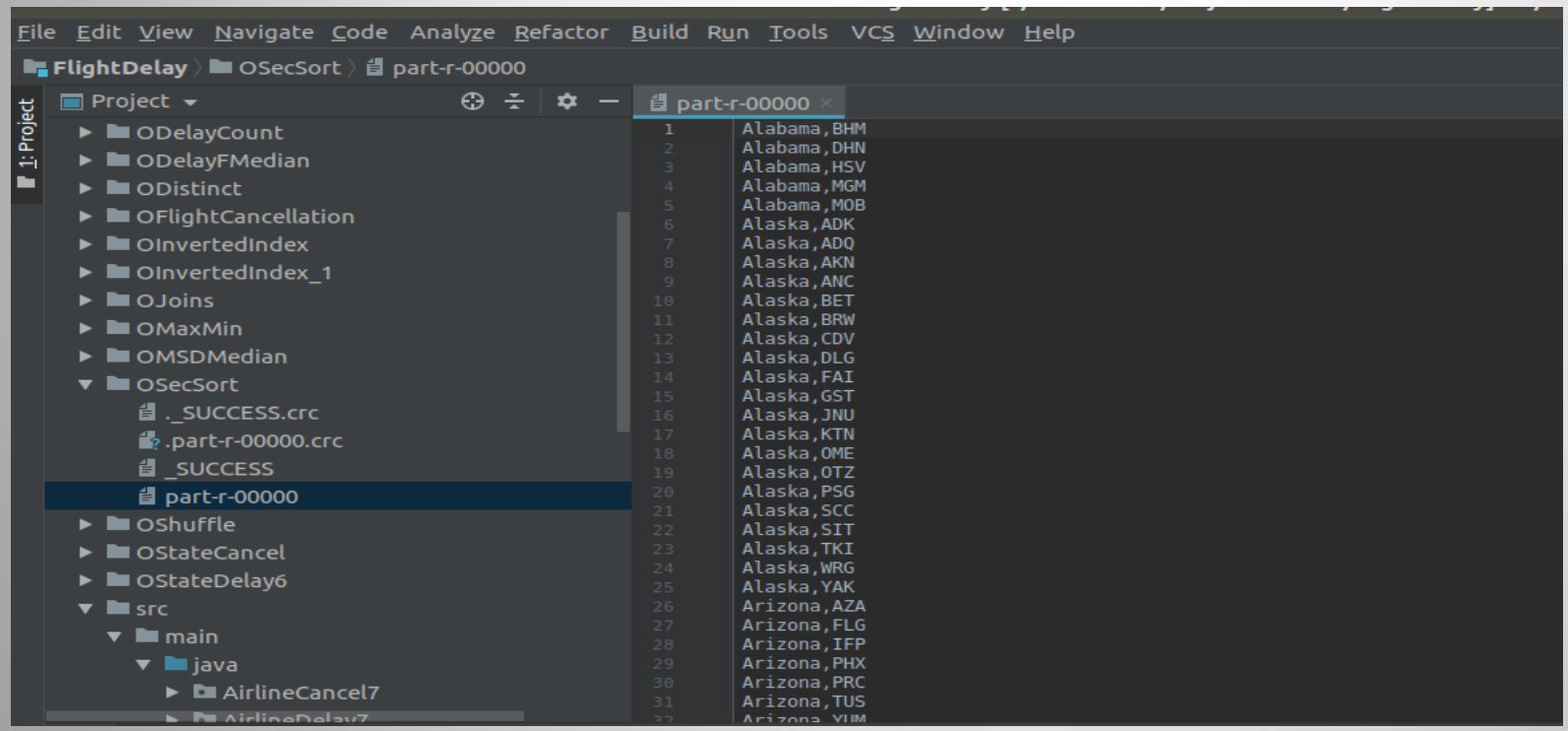


INVERTED INDEX:

Used Filtered output and determined the Flights from the Airport using Inverted Index Method.

SECONDARY SORTING:

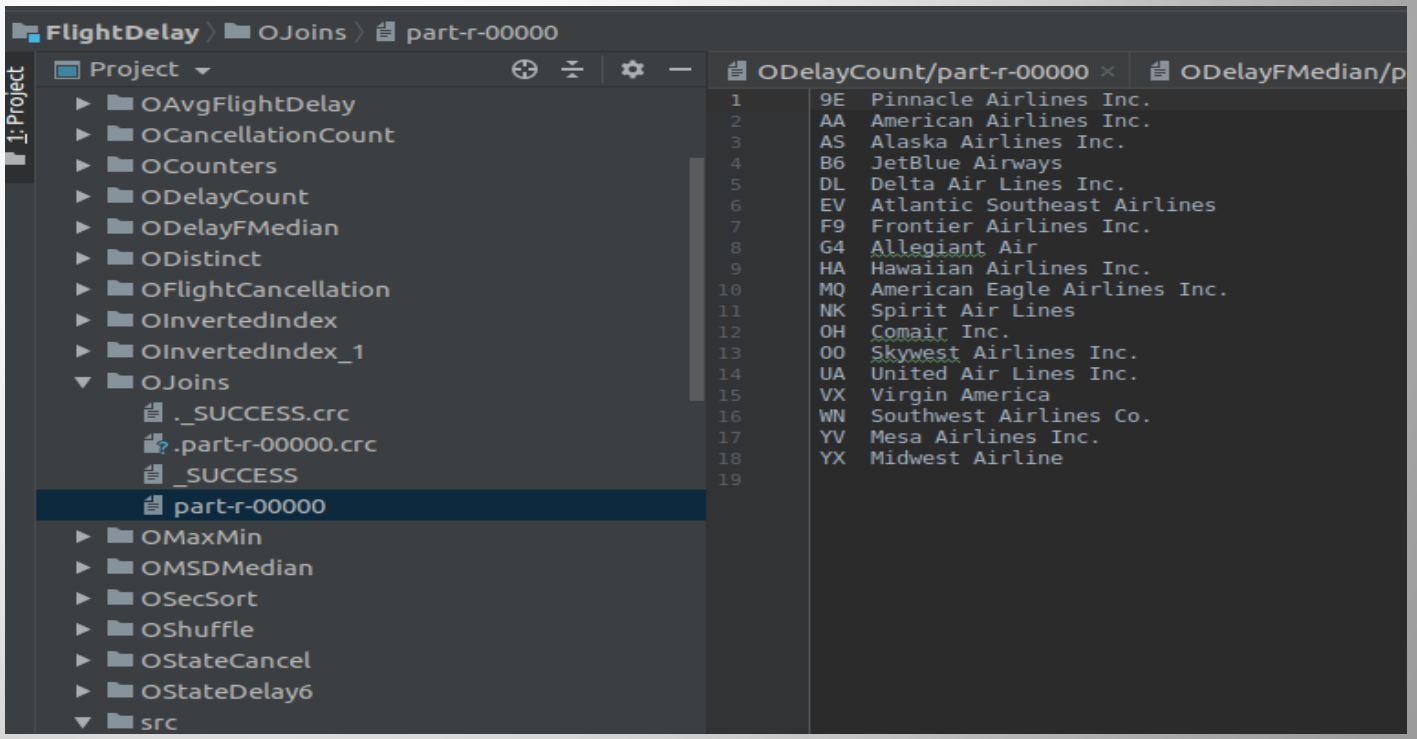
Did Secondary Sorting of the States and the available Airports in that state to know the list of Airports Present in US.(Partitioner is used in this case)



JOINS:

Based on the Carrier Names from a different Carrier CSV joined the Full name of the Flights present in the original Data.

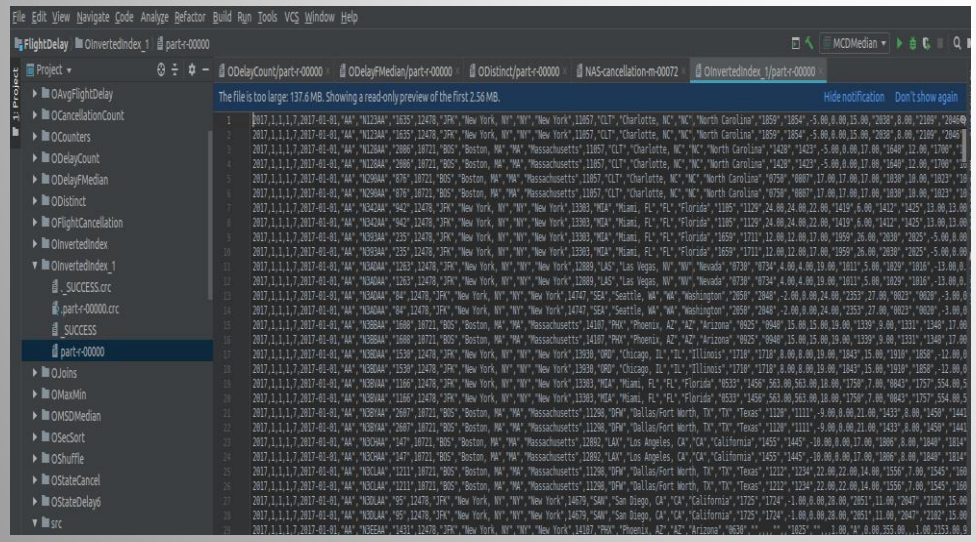
Code	Description
02Q	Titan Airways
04Q	Tradewind Aviation
05Q	Comlux Aviation, AG
06Q	Master Top Linhas Aereas Ltd.
07Q	Flair Airlines Ltd.
09Q	Swift Air, LLC
0BQ	DCA
0CQ	ACM AIR CHARTER GmbH
0FQ	Maine Aviation Aircraft Charter, LLC
0GQ	Inter Island Airways, d/b/a Inter Island Air
0HQ	Polar Airlines de Mexico d/b/a Nova Air
0J	JetClub AG
0JQ	Vision Airlines
0KQ	Mokulele Flight Services, Inc.
0LQ	Metropix UK, LLP.
0MQ	Multi-Aero, Inc. d/b/a Air Choice One
0Q	Flying Service N.V.
16	PSA Airlines Inc.
17	Piedmont Airlines
1I	Sky Trek Int'l Airlines
2E	Smokey Bay Air Inc.
2F	Frontier Flying Service
2M	Midway Express Airlines
2O	Island Air Service
2R	Regal Air
2T	Canada 3000 Airlines Ltd.
2U	Valley Air Express Inc.
37	Zeal 320
3C	Regions Air, Inc.
3F	Pacific Airways, Inc.
3M	Gulfstream Int
3Z	Tatonduk Flying Service
4B	Olson Air Service
4E	Tanana Air Service



BLOOM FILTER:

Used Bloom Filter to Filter the data based on the Airport and used Chaining

```
private BloomFilter<String> origin;

Funnel<String> p = (from, into) -> {

        into.putString(from, Charsets.UTF_8);

};

@Override
 protected void setup(Mapper<LongWritable, Text, Text, NullWritable>.Context context) throws IOException, InterruptedException {

    this.origin = BloomFilter.create(p,  expectedInsertions: 300000,  falsePositiveProbability: 0.1);

    String p1 = "BOS";
    String p2 = "JFK";
   // Person p2 = new Person("Jamie", "Scott");

    ArrayList<String> originList = new ArrayList<>();
    originList.add(p1);
    originList.add(p2);

    for (String ps : originList) {
        origin.put(ps);
    }
```

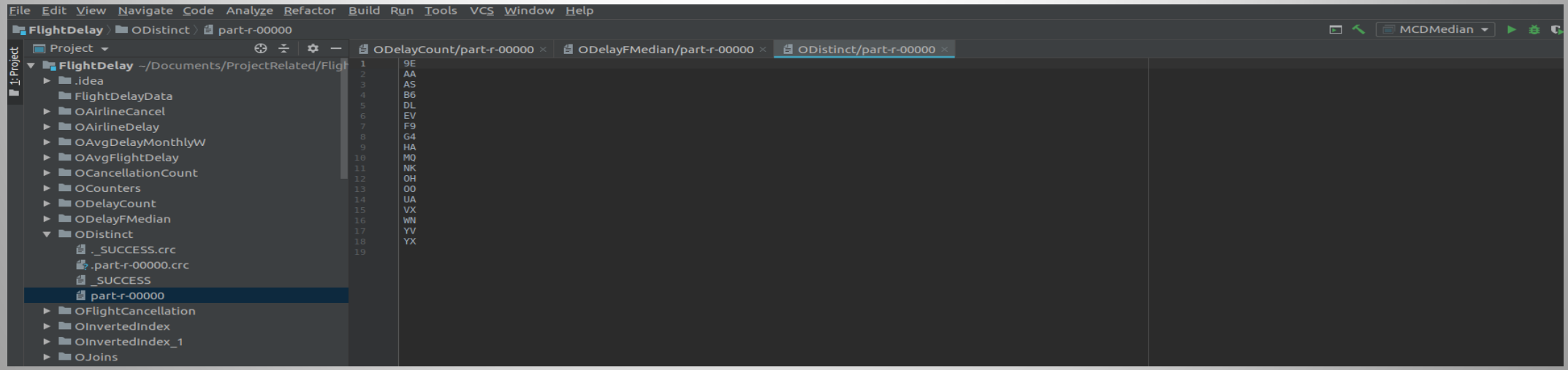



```
    job.setOutputFormatClass(TextOutputFormat.class);
    job.setMapOutputValueClass(NullWritable.class);
    job.setMapOutputKeyClass(Text.class);
    job.setInputFormatClass(TextInputFormat.class);

//  job.setNumReduceTasks(1);


    Boolean a=job.waitForCompletion( verbose: true);

    if(a==true)
    {
        Job job2 = Job.getInstance();

        job2.setJarByClass(Driver.class);

        // job2.setGroupingComparatorClass(GroupComparator.class);
        // job2.setSortComparatorClass(SecodarySortComparator.class);
        //job2.setPartitionerClass(KeyPartition.class);

        FileInputFormat.addInputPath(job2, new Path( pathString: args[1]+"_1"));
        Path outDir1 = new Path(args[1]);
```

DISTINCT:

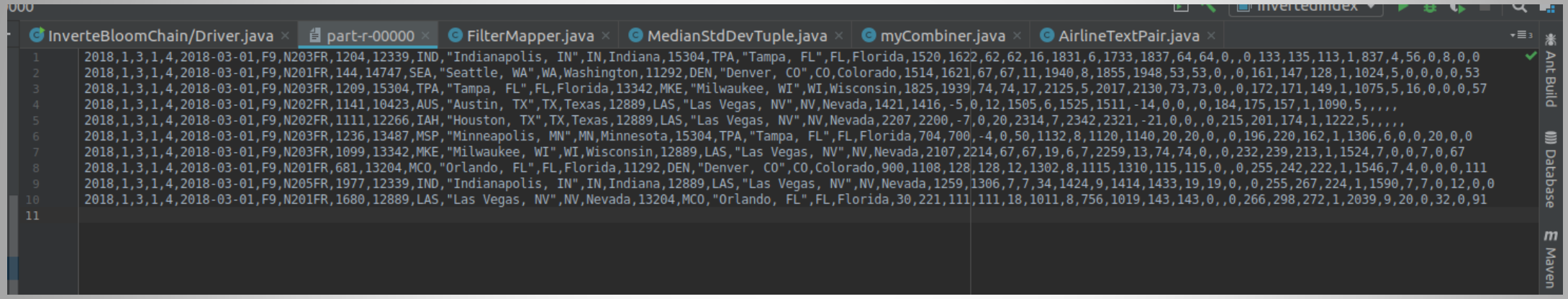
Determined Distinct Airline in the Dataset. (Used NLineInputFormat)

```
FileInputFormat.addInputPath(job2, new Path(args[0]));
Path outDir1 = new Path(args[1]);
FileOutputFormat.setOutputPath(job2, outDir1);

job2.setMapperClass(DistinctMapper.class);
job2.setReducerClass(DistinctReducer.class);
job2.setCombinerClass(DistinctReducer.class);

job2.setOutputFormatClass(TextOutputFormat.class);
job2.setMapOutputValueClass(NullWritable.class);
job2.setMapOutputKeyClass(Text.class);
job2.setInputFormatClass(NLineInputFormat.class);
NLineInputFormat.setNumLinesPerSplit(job2, numLines: 10000);
job2.setNumReduceTasks(1);
```



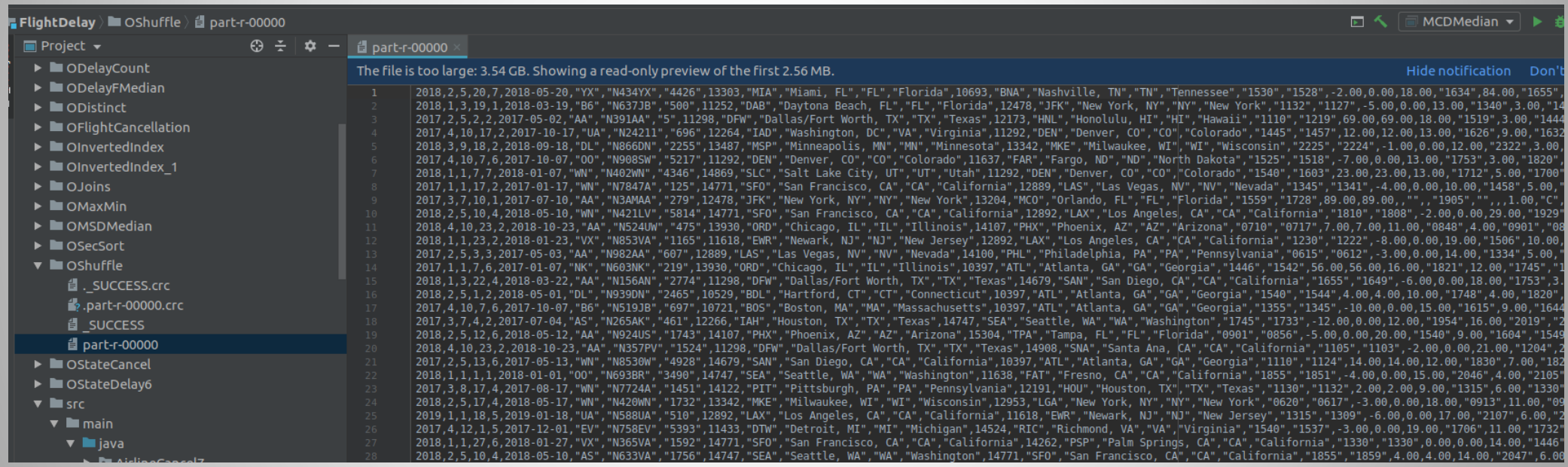
TOP K:

Determined Top 10 flight data based on longest Distance.



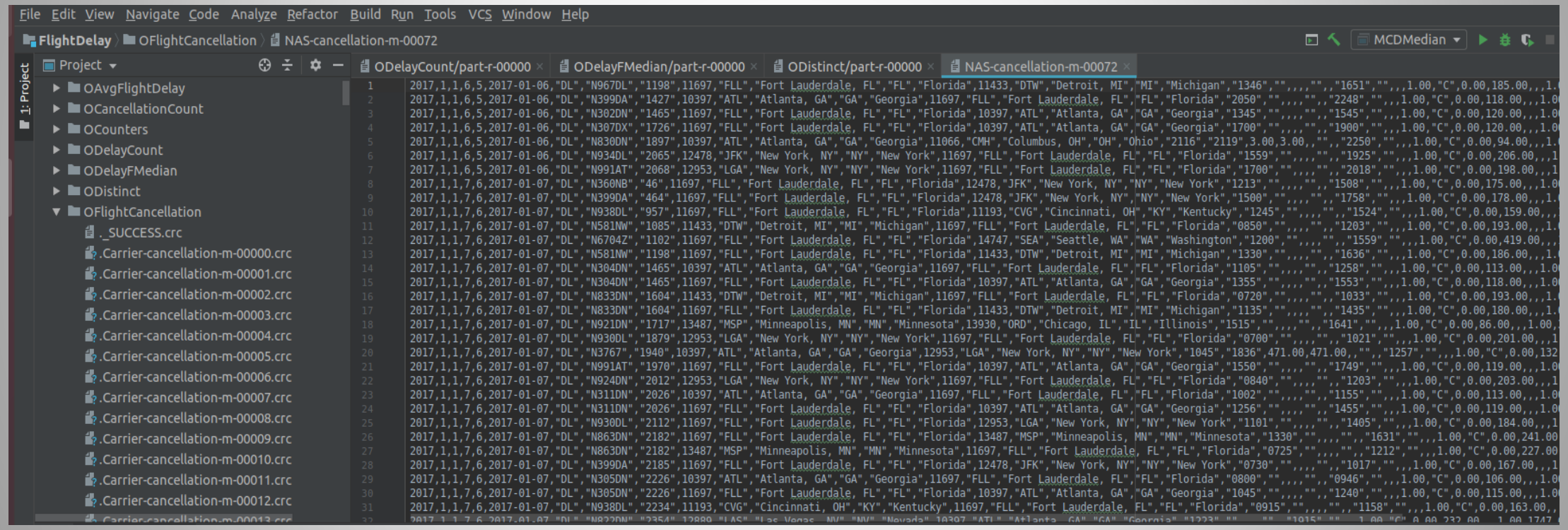
SHUFFLING:

Used Shuffling Technique to shuffle the Data.



BINNING:

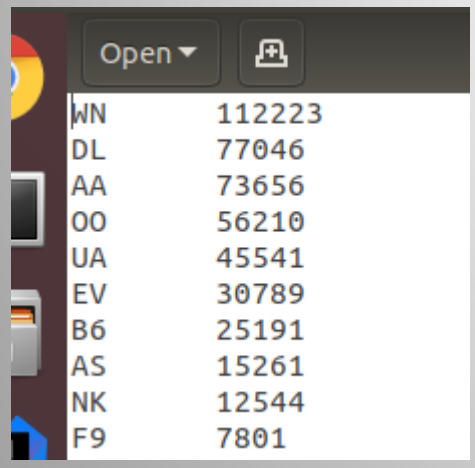
Separated the data based on Cancellation Reason by using Binning Technique.



PIG: TOP K

Found Top 10 Airline with Most Flights using Pig.

```
flights = LOAD 'apr17.csv' using PigStorage(',');
grouped = GROUP flights BY $6;
summed = FOREACH grouped GENERATE group, COUNT(flights) AS cntd;
sorted = ORDER summed BY cntd DESC;
top25 = LIMIT sorted 10;
Dump top25;
STORE top25 into 'top10Carriers';
```



PIG: JOIN

Joined Airline csv and unique Airline names using Pig.

```
uniqueid = LOAD 'distinct' using PigStorage(',') as (uid);
carriers = LOAD 'carriers.csv' using PigStorage(',') as (id,name);
joined = JOIN uniqueid BY uid, carriers BY id;
summed = FOREACH joined GENERATE $0,$2;
STORE summed into 'joinoutput';
```

```
"9E"    "Pinnacle Airlines Inc."
"AA"    "American Airlines Inc."
"AS"    "Alaska Airlines Inc."
"B6"    "JetBlue Airways"
"DL"    "Delta Air Lines Inc."
"EV"    "Atlantic Southeast Airlines"
"F9"    "Frontier Airlines Inc."
"G4"    "Allegiant Air"
"HA"    "Hawaiian Airlines Inc."
"MQ"    "American Eagle Airlines Inc."
"NK"    "Spirit Air Lines"
"OH"    "Comair Inc."
"OO"    "Skywest Airlines Inc."
"UA"    "United Air Lines Inc."
"VX"    "Virgin America"
"WN"    "Southwest Airlines Co."
"YV"    "Mesa Airlines Inc."
"YX"    "Midwest Airline
```

PIG FILTER AND MERGE:

Filtered the Latest Data based on Year using Pig.

```
flights = LOAD 'Merge.csv' using PigStorage(',');
filter1 = FILTER flights BY $0 > 2017;
filter2 = FOREACH filter1 GENERATE $0,$1,$2,$3,$4,$5,$6,$7,$8,$9,'LatestData';
STORE filter2 into 'LatestData';
```

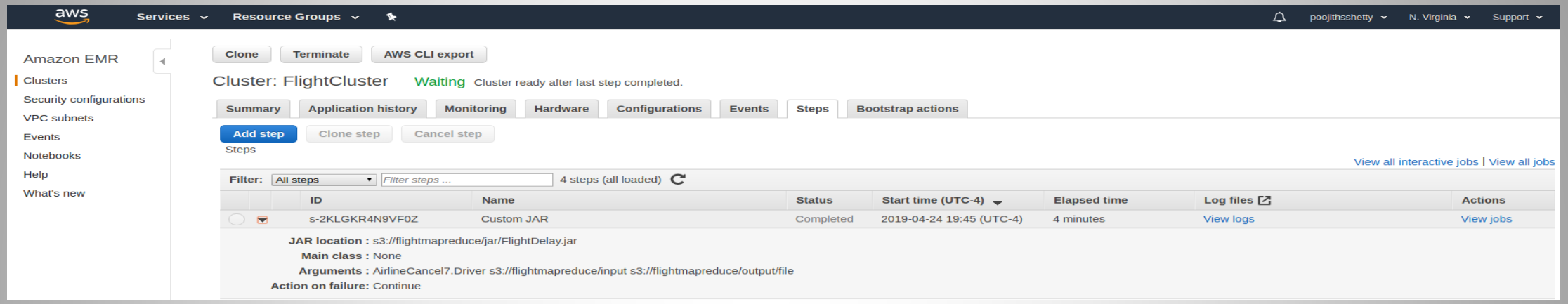
Open

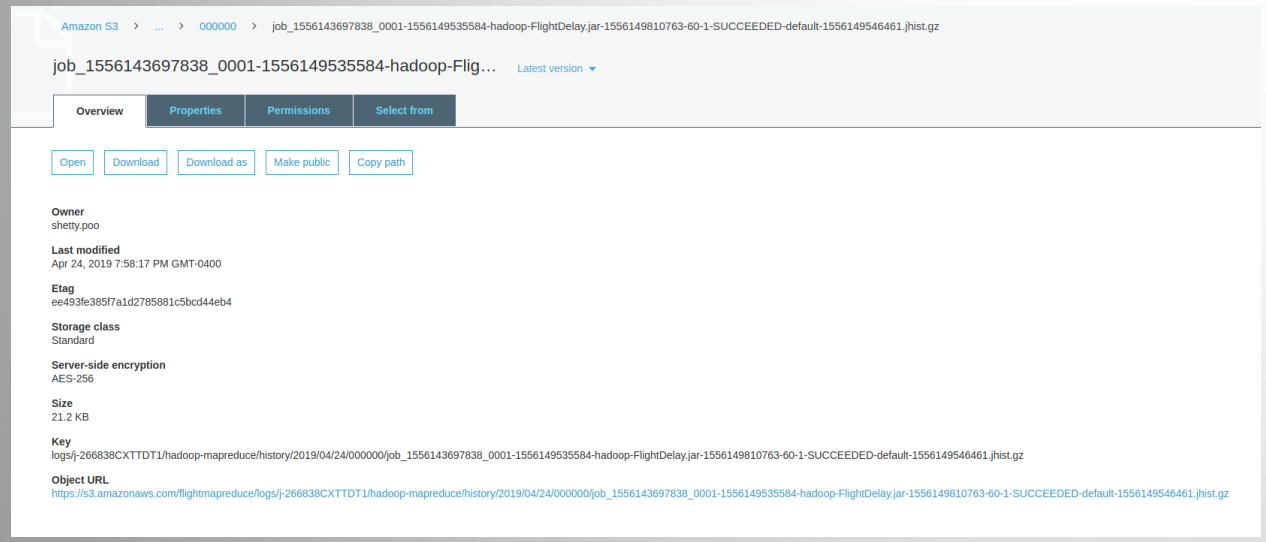
/usr/local/pig-0.17.0/bin/Late

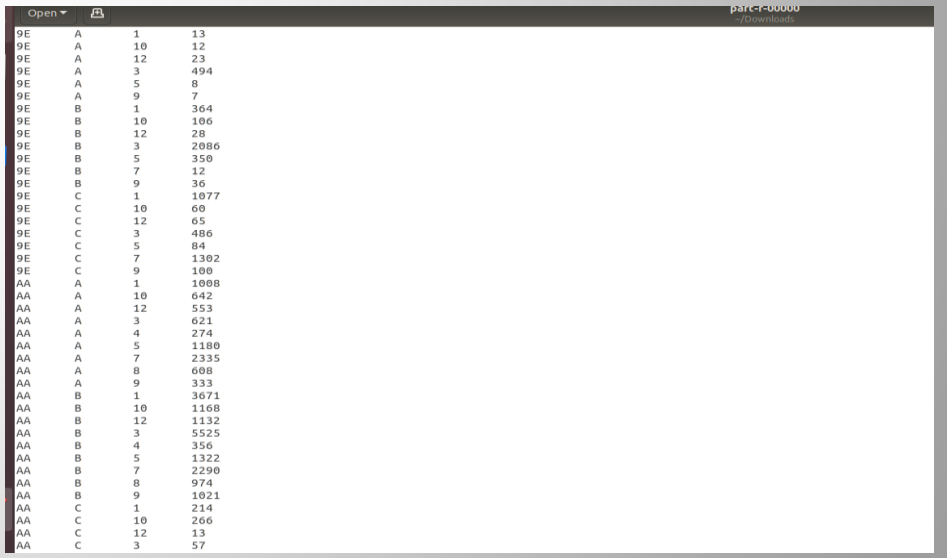
2018	1	3	1	4	2018-03-01	F9	N201FR	1680	12889	Latest Data
2018	1	3	1	4	2018-03-01	F9	N201FR	681	13204	Latest Data
2018	1	3	1	4	2018-03-01	F9	N201FR	681	11292	Latest Data
2018	1	3	1	4	2018-03-01	F9	N201FR	144	14747	Latest Data
2018	1	3	1	4	2018-03-01	F9	N201FR	122	11292	Latest Data
2018	1	3	1	4	2018-03-01	F9	N202FR	1138	12889	Latest Data
2018	1	3	1	4	2018-03-01	F9	N202FR	1141	10423	Latest Data
2018	1	3	1	4	2018-03-01	F9	N202FR	1106	12889	Latest Data
2018	1	3	1	4	2018-03-01	F9	N202FR	1111	12266	Latest Data
2018	1	3	1	4	2018-03-01	F9	N203FR	1236	13487	Latest Data
2018	1	3	1	4	2018-03-01	F9	N203FR	1201	15304	Latest Data
2018	1	3	1	4	2018-03-01	F9	N203FR	1204	12339	Latest Data
2018	1	3	1	4	2018-03-01	F9	N203FR	1209	15304	Latest Data
2018	1	3	1	4	2018-03-01	F9	N203FR	1099	13342	Latest Data

AWS CLOUD

Used AWS Cloud EMR instance for running MapReduce Jobs.







TECHNOLOGIES AND PATTERNS USED

- Max Min
- Count
- Average
- Median
- Memory Conscious Median
- Inverted Index
- Counting with Counters
- Input Format
- Custom Combiner
- Partitioner
- Bloom Filter
- Top K
- Distinct
- Joins
- Shuffling
- Binning
- Chaining
- AWS Cloud
- PIG Top K
- PIG Filter and Merge
- PIG Joins

Appendix

Code Snippet:

1. **Airline Cancellation:**

```
package AirlineCancel7;



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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


import java.io.IOException;


public class Driver {

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

      Job job = Job.getInstance();

      job.setJarByClass(Driver.class);


      job.setMapperClass(CancelCountMapper.class);

      job.setCombinerClass(CancelCountReducer.class);

      job.setReducerClass(CancelCountReducer.class);


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

```
        job.setOutputFormatClass(TextOutputFormat.class);

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

        job.setNumReduceTasks(1);

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

        FileSystem fs = FileSystem.get(job.getConfiguration());
//      Path outDir = new Path(args[1]);
//      if(fs.exists(outDir)) {
//          fs.delete(outDir, true);
//      }

        // Submit the job, then poll for progress until the job is complete
        job.waitForCompletion(true);
    }
}

package AirlineCancel7;

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

import java.io.IOException;
```

```
public class CancelCountMapper extends Mapper<LongWritable, Text, Text, Text> {
   private static final IntWritable one = new IntWritable(1);

   @Override
   protected void map(LongWritable key, Text value, Context context)
         throws IOException, InterruptedException {
      String line = value.toString();
      String t= "";
      String[] tokens = line.split(",");
     //  for(String s : tokens) {
      if(!tokens[0].contains("YEAR") && tokens.length >33 && !tokens[32].equalsIgnoreCase("0")
            && !tokens[33].equalsIgnoreCase(" ") &&
!tokens[33].replace("\"","").trim().equalsIgnoreCase("")) {

         context.write(new
Text(tokens[6].replace("\"","").trim()+"\t"+tokens[33].replace("\"","").trim()+"\t"+tokens[2]), new
Text(1+""));
      }
      //}
   }

   @Override
   protected void cleanup(Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.cleanup(context);
   }

   @Override
```

```
    public void run(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.run(context);
    }

    @Override
    protected void setup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }

}
```

2. Airline Cancellation:

```
package AirlineCancel7;

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

import java.io.IOException;

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

    @Override
    protected void cleanup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.cleanup(context);
```

```
}

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

   int sum = 0;
   int count=0;
   for (Text value : values) {
      String a[]=value.toString().split(" ");
     // sum += Float.parseFloat(a[0]);
      count+=Integer.parseInt(a[0]);
   }

   context.write(key, new Text(count+""));
}

@Override
public void run(Context arg0)
      throws IOException, InterruptedException {
   // TODO Auto-generated method stub
   super.run(arg0);
}

@Override
protected void setup(Context context)
      throws IOException, InterruptedException {
   // TODO Auto-generated method stub
   super.setup(context);
```

```
   }


}
```

3. Airline Delay:

```
package AirlineDelay7;



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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


import java.io.IOException;


public class Driver {

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

      Job job = Job.getInstance();

      job.setJarByClass(Driver.class);


      job.setMapperClass(DelayCountMapper.class);

      job.setCombinerClass(DelayCountReducer.class);

      job.setReducerClass(DelayCountReducer.class);


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

```
        job.setOutputFormatClass(TextOutputFormat.class);

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

        job.setNumReduceTasks(1);

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

        FileSystem fs = FileSystem.get(job.getConfiguration());
//      Path outDir = new Path(args[1]);
//      if(fs.exists(outDir)) {
//          fs.delete(outDir, true);
//      }

        // Submit the job, then poll for progress until the job is complete
        job.waitForCompletion(true);
    }
}

package AirlineDelay7;

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

import java.io.IOException;
```

```
public class DelayCountMapper extends Mapper<LongWritable, Text, Text, Text> {
   private static final IntWritable one = new IntWritable(1);

   @Override
   protected void map(LongWritable key, Text value, Context context)
         throws IOException, InterruptedException {
      String line = value.toString();
      String t= "";
      String[] tokens = line.split(",");
      Integer delay=0;
     //  for(String s : tokens) {
      if(!tokens[0].contains("YEAR") && tokens.length >45 && !tokens[41].equalsIgnoreCase("")
            && !tokens[41].equalsIgnoreCase(" ") && !tokens[41].equalsIgnoreCase("NA")) {
      if(Float.parseFloat(tokens[41]) >0 ){
      t="CARRIER_DELAY";
      delay=(int)Float.parseFloat(tokens[41]);
      }
      else if(Float.parseFloat(tokens[42]) >0 ){
         t="WEATHER_DELAY";
         delay=(int)Float.parseFloat(tokens[42]);
               }
      else if(Float.parseFloat(tokens[43]) >0 ){
         t="NAS_DELAY";
         delay=(int)Float.parseFloat(tokens[43]);
      }
      else if (Float.parseFloat(tokens[44]) >0 ){
         t="SECURITY_DELAY";
         delay=(int)Float.parseFloat(tokens[44]);
```

```
    }
    else{
       t="LATE_AIRCRAFT_DELAY";
       delay=(int)Float.parseFloat(tokens[45]);
    }
       context.write(new Text(tokens[6].replace("\"","")+"\t"+t+"\t"+tokens[2]), new
Text(delay.toString()+"\t"+1));
    }
    //}
  }

  @Override
  protected void cleanup(Context context)
       throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.cleanup(context);
  }

  @Override
  public void run(Context context)
       throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.run(context);
  }

  @Override
  protected void setup(Context context)
       throws IOException, InterruptedException {
    // TODO Auto-generated method stub
```

```
        super.setup(context);
    }

}
```

4. Airline delay State:

```
package AirlineDelay7;

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

import java.io.IOException;

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

    @Override
    protected void cleanup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.cleanup(context);
    }

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

        int sum = 0;
        int count=0;
        for (Text value : values) {
            String a[]=value.toString().split("\t");
```

```
            sum += Float.parseFloat(a[0]);
            count+=Integer.parseInt(a[1]);
        }

        context.write(key, new Text(sum+"\t"+count));
    }

    @Override
    public void run(Context arg0)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.run(arg0);
    }

    @Override
    protected void setup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }

}
```

5. Avg Flight Delay:

```
package AverageFlightDelay;


import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
```

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

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import java.util.Set;


public class AvgFlightDelay {

   public static class MapDelay extends Mapper<LongWritable, Text, Text, DelayAveragingPair> {

      private Text outText = new Text();

      private Map<String,DelayAveragingPair> pairMap = new HashMap<String,DelayAveragingPair>();

      private DelayAveragingPair pair = new DelayAveragingPair();


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

         String line = value.toString();

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

if(columnValues.length > 24) {

   if (!columnValues[22].contains("DEP_DELAY") && !columnValues[24].contains("TAXI_OUT") &&
!columnValues[24].equalsIgnoreCase("NA") && columnValues[24] != null &&
!columnValues[24].equalsIgnoreCase("") ) {

      outText.set(columnValues[6]);
```

```
        int delayTime = (int) Float.parseFloat(columnValues[24]);
        DelayAveragingPair pair = pairMap.get(columnValues[6]);
        if (pair == null) {
            pair = new DelayAveragingPair();
            pairMap.put(columnValues[6], pair);
        }
        int delay = pair.getDelay().get() + delayTime;
        int count = pair.getCount().get() + 1;
        pair.set(delay, count);
    }
}
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            Set<String> keys = pairMap.keySet();
            Text keyText = new Text();
            for (String key : keys) {
                keyText.set(key.replace("\"","").trim());
                context.write(keyText,pairMap.get(key));
            }
        }
    }

    public static class Reduce extends Reducer<Text, DelayAveragingPair, Text, IntWritable> {
        public void reduce(Text key, Iterable<DelayAveragingPair> values
                , Context context)
                throws IOException, InterruptedException {
            int sum = 0;
```

```
        int count = 0;
        for (DelayAveragingPair val : values) {
            sum += val.getDelay().get();
            count += val.getCount().get();
        }
        context.write(key, new IntWritable(sum/count));
    }
}
```

6. Flight Average Delay

```
public static class DelayAveragingPair implements Writable,
WritableComparable<DelayAveragingPair> {
    private IntWritable delay;
    private IntWritable count;

    public DelayAveragingPair() {
        set(new IntWritable(0), new IntWritable(0));
    }

    public DelayAveragingPair(int delay, int count) {
        set(new IntWritable(delay), new IntWritable(count));
    }

    public void set(int delay, int count){
        this.delay.set(delay);
        this.count.set(count);
    }

    public void set(IntWritable delay, IntWritable count) {
        this.delay = delay;
```

```
    this.count = count;
}


public void write(DataOutput out) throws IOException {
    delay.write(out);
    count.write(out);
}


public void readFields(DataInput in) throws IOException {
    delay.readFields(in);
    count.readFields(in);
}


public int compareTo(DelayAveragingPair other) {
    int compareVal = this.delay.compareTo(other.getDelay());
    if (compareVal != 0) {
        return compareVal;
    }
    return this.count.compareTo(other.getCount());
}

public static DelayAveragingPair read(DataInput in) throws IOException {
    DelayAveragingPair averagingPair = new DelayAveragingPair();
    averagingPair.readFields(in);
    return averagingPair;
}
```

```
@Override
public boolean equals(Object o) {
   if (this == o) return true;
   if (o == null || getClass() != o.getClass()) return false;
   DelayAveragingPair that = (DelayAveragingPair) o;

   if (!count.equals(that.count)) return false;
   if (!delay.equals(that.delay)) return false;

   return true;
}

@Override
public int hashCode() {
   int result = delay.hashCode();
   result = 163 * result + count.hashCode();
   return result;
}

@Override
public String toString() {
   return "DelayAveragingPair{" +
         "delay=" + delay +
         ", count=" + count +
         '}';
}

public IntWritable getDelay() {
```

```
        return delay;
      }

      public IntWritable getCount() {
        return count;
      }
    }

    public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();

      Job job = new Job(conf, "flight count");
      job.setJarByClass(AvgFlightDelay.class);

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

      job.setMapperClass(MapDelay.class);
      job.setReducerClass(Reduce.class);

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

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

      job.waitForCompletion(true);
    }
}
```

7. Average Delay by Month

```
package AvgDelayMonthly;

import au.com.bytecode.opencsv.CSVParser;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.IOException;

public class AverageFlightDelayByMonth {
        public static class AverageFlightDelayByMonthMapper extends
                        Mapper<Object, Text, AirlineTextPair, Text> {

                // initialize CSVParser as comma separated values
                private CSVParser csvParser = new CSVParser(',', '"');

                /**
                 * Key : Byte offset from where the line is being read
                 * value : string representing the entire line of flight data
```

```
		 */
		public void map(Object offset, Text value, Context context)
				throws IOException, InterruptedException {

			// Parse the input line
			String[] line = this.csvParser.parseLine(value.toString());
			AirlineTextPair key = new AirlineTextPair();
			Text arrDelayMinutes = new Text();

			if (line.length > 0 && isValidEntry(line) && !line[2].contains("MONTH") &&
!line[24].equalsIgnoreCase("")) {
				// set key as AirlineTextPair
				key.setAirLineName(line[6].replace("\"","").trim());
				key.setMonth(line[2]);

				// set value as arrDelayMinutes
				arrDelayMinutes.set(line[24]);

				context.write(key, arrDelayMinutes);
			}
		}

		/**
		 * Function determines the validity of the input record
		 *
		 * @param record : line record representing flight  details
		 * @return true iff entry contain all required data fields
		 *			false otherwise
		 */
```

```
		private boolean isValidEntry(String[] record) {

			if (record == null || record.length == 0) {
				return false;
			}

			// If any of required field is missing, we'll ignore the record
			if (record[0].equalsIgnoreCase("") ) {
				return false;
			}

//			if(! record[0].equals("2008")){
//				return false;
//			}

			// Whether flight was cancelled
			if (record[32].equals("1")) {
				return false;
			}

			return true;
		}
	}

	/**
	 * Reduce class
	 */
	public static class AverageFlightDelayByMonthReducer extends
			Reducer<AirlineTextPair, Text, Text, Text> {
```

```
/**
 * Reduce function will be called per Airline's Carrier Name and
 * months will be in increasing sorted order
 * Key : AirlineTextPair which implements WritableComparable
 * values : sorted delay in increasing order by month for given key airline
 */
public void reduce(AirlineTextPair key, Iterable<Text> values,
                    Context context) throws IOException, InterruptedException {
          double totalDelay = 0;
          int totalFligths = 0;
          int currMonth = 1;
          StringBuilder output = new StringBuilder();
          output.append(key.getAirLineName().toString());

          for(Text value : values){
                    // Check whether month has changed for the given flight
                    // On change append data to output string and clear the counters
                    if(currMonth != Integer.parseInt(key.getMonth().toString())){
                              int avgDelay = (int)Math.ceil(totalDelay/(double)totalFligths);
                              output.append(", (" + currMonth + "," + avgDelay + ")");
                              totalDelay = 0;
                              totalFligths = 0;
                              currMonth = Integer.parseInt(key.getMonth().toString());
                    }

                    totalDelay += Double.parseDouble(value.toString());
                    totalFligths++;
          }
```

```
                // Append last valid data of the current flight
                int avgDelay = (int)Math.ceil(totalDelay/(double)totalFligths);
                output.append(", (" + currMonth + "," + avgDelay + ")");

                // Write down remaining months as 0 delay
                while(currMonth < 12)
                {
                        currMonth++;
                        output.append(", (" + currMonth + ", 0 )");
                }

                // Output the formatted string in the file
                context.write(new Text(), new Text(output.toString()));
        }
}

/**
 * AirlinePartitioner will partition data based on the hashcode of the
 * Airline's UniqueCarrier
 */
public static class AirlinePartitioner extends
                Partitioner<AirlineTextPair, Text> {
        /**
         * Based on the configured number of reducer, this will
         * partition the data approximately evenly based on number of
         * unique carrier names
         */
        @Override
```

```
        public int getPartition(AirlineTextPair key, Text value,
                        int numPartitions) {
                // multiply by 127 to perform some mixing
                return Math.abs(key.hashCode() * 127) % numPartitions;
        }
}

/**
 * KeyComparator first sorts based on the unique carrier name and
 * then sorts months in increasing order using compareTo method of the
 * AirlineTextPair class
 */
public static class KeyComparator extends WritableComparator {
        protected KeyComparator() {
                super(AirlineTextPair.class, true);
        }

        @Override
        public int compare(WritableComparable w1, WritableComparable w2) {
                AirlineTextPair airline1 = (AirlineTextPair) w1;
                AirlineTextPair airline2 = (AirlineTextPair) w2;
                return airline1.compareTo(airline2);
        }
}

/**
 * GroupComparator ignores the month files of the key so that
 * single reduce function call receives all flight data for given
 * unique carrier name
```

```
 */
public static class GroupComparator extends WritableComparator {
	protected GroupComparator() {
		super(AirlineTextPair.class, true);
	}

	@Override
	public int compare(WritableComparable w1, WritableComparable w2) {
		AirlineTextPair airline1 = (AirlineTextPair) w1;
		AirlineTextPair airline2 = (AirlineTextPair) w2;
		return airline1.compare(airline2);
	}
}

/**
 * @param args
 */
public static void main(String[] args) throws Exception {

	Configuration conf = new Configuration();
	String[] otherArgs = new GenericOptionsParser(conf, args)
			.getRemainingArgs();
	if (otherArgs.length != 2) {
		System.err.println("Usage: AverageFlighDelayByMonth <in> <out>");
		System.exit(2);
	}
	Job job = new Job(conf, "Average monthly flight dealy");
	job.setJarByClass(AverageFlightDelayByMonth.class);
	job.setMapperClass(AverageFlightDelayByMonthMapper.class);
```

```
                job.setReducerClass(AverageFlightDelayByMonthReducer.class);
                job.setOutputKeyClass(AirlineTextPair.class);
                job.setOutputValueClass(Text.class);
                job.setPartitionerClass(AirlinePartitioner.class);
                job.setSortComparatorClass(KeyComparator.class);
            job.setGroupingComparatorClass(GroupComparator.class);
                FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
                FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
                System.exit(job.waitForCompletion(true) ? 0 : 1);
        }

}

package AvgDelayMonthly;

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

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

/**
 * AirlineTextPair implements the WritableComparable
 * This class contains useful functions used for keyComparator and
 * Grouping comparator
 */
public class AirlineTextPair implements WritableComparable {
        // Airline's UniqueCarrier
```

```
Text airLineName;
// Airlines fly Month
Text month;

/**
 * constructor
 */
public AirlineTextPair() {
	this.airLineName = new Text();
	this.month = new Text();
}

/**
 * constructor
 * @param airLineName
 * @param month
 */
public AirlineTextPair(Text airLineName, Text month) {
	this.airLineName = airLineName;
	this.month = month;
}

/**
 * set airline's unique carrier name
 * @param airLineName
 */
public void setAirLineName(String airLineName) {
	this.airLineName.set(airLineName.getBytes());
}
```

```
/**
 * set the month of the flight fly
 * @param month
 */
public void setMonth(String month) {
	this.month.set(month.getBytes());
}

/**
 * get airline's unique carrier name
 */
public Text getAirLineName() {
	return this.airLineName;
}

/**
 * get the month of the flight fly
 */
public Text getMonth() {
	return this.month;
}

/**
 * overrider the write method to support write operation
 */
public void write(DataOutput out) throws IOException {
	 this.airLineName.write(out);
	 this.month.write(out);
```

```
}

/**
 * overrider readFiled method to support reading fields
 */
public void readFields(DataInput in) throws IOException {
      if (this.airLineName == null)
        this.airLineName = new Text();

      if (this.month == null)
        this.month = new Text();

      this.airLineName.readFields(in);
      this.month.readFields(in);
}

/**
 * Sort first by airline name and then by month in increasing order
 */
public int compareTo(Object object) {
      AirlineTextPair ip2 = (AirlineTextPair) object;
      int cmp = getAirLineName().compareTo(ip2.getAirLineName());
      if (cmp != 0) {
        return cmp;
      }
      int m1 = Integer.parseInt(getMonth().toString());
      int m2 = Integer.parseInt(ip2.getMonth().toString());
      return m1 == m2 ? 0 : (m1 < m2 ? -1 : 1);
}
```

```
    /**
     * provide comparator for airline name for grouping comparator
     */
    public int compare(Object object){
            AirlineTextPair airline2 = (AirlineTextPair) object;
            return getAirLineName().compareTo(airline2.getAirLineName());
    }

    /**
     * use this hashcode for partitioning
     */
    public int hashCode() {
          return this.airLineName.hashCode();
    }

    public boolean equals(Object o) {
          AirlineTextPair p = (AirlineTextPair) o;
          return this.airLineName.equals(p.getAirLineName());
    }

    public Text toText(){
            return new Text("(" + this.airLineName.toString() + "," + this.month.toString() +")");
    }
}
```

8. Binning

```
package BinningFlight;
```

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

import java.io.IOException;

public class FlightCancellation
{

public static class TMapper extends Mapper<Object, Text, Text, NullWritable>
{

private MultipleOutputs<Text, NullWritable> mos=null;

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

 @Override
```

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

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

      if(!row[32].equalsIgnoreCase("0") && !row[33].equalsIgnoreCase("\"\"") ){
                String cancellationCode = row[33];

               // System.out.println(cancellationCode);
                if(cancellationCode.contains("A"))
                   mos.write("bins", value, NullWritable.get(),"Carrier-cancellation");
                if(cancellationCode.contains("B"))
                   mos.write("bins", value, NullWritable.get(),"Weather-cancellation");
                if(cancellationCode.contains("C"))
                   mos.write("bins", value, NullWritable.get(),"NAS-cancellation");
                if(cancellationCode.contains("D"))
                   mos.write("bins", value, NullWritable.get(),"Security-cancellation");

   }

}

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

```
}

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException
{
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Cancellation Binning ");
    job.setJarByClass(FlightCancellation.class);
    job.setMapperClass(TMapper.class);

    MultipleOutputs.addNamedOutput(job, "bins", TextOutputFormat.class, Text.class,
IntWritable.class);
    MultipleOutputs.setCountersEnabled(job, true);
    job.setNumReduceTasks(0);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(NullWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
 }

}
```

9. Flight Cancellation Count

```
package CancellationCount2;



import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


import java.io.IOException;


public class Driver {

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

      Job job = Job.getInstance();

      job.setJarByClass(Driver.class);


      job.setMapperClass(CancellationCountMapper.class);

      job.setCombinerClass(CancellationCountReducer.class);

      job.setReducerClass(CancellationCountReducer.class);


      job.setInputFormatClass(TextInputFormat.class);

      job.setOutputFormatClass(TextOutputFormat.class);
```

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

        job.setNumReduceTasks(1);

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

        FileSystem fs = FileSystem.get(job.getConfiguration());
//      Path outDir = new Path(args[1]);
//      if(fs.exists(outDir)) {
//          fs.delete(outDir, true);
//      }

        // Submit the job, then poll for progress until the job is complete
        job.waitForCompletion(true);
    }
}
```

```
package CancellationCount2;

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

import java.io.IOException;

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

```
   private static final IntWritable one = new IntWritable(1);


   @Override
   protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
IntWritable>.Context context)
         throws IOException, InterruptedException {
      String line = value.toString();


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


     //  for(String s : tokens) {
      if(!tokens[0].equalsIgnoreCase("YEAR") && !tokens[32].equalsIgnoreCase("0") &&
!tokens[33].equalsIgnoreCase(""))
         context.write(new Text(tokens[33].replace("\"","").trim()), one);
      //}
   }


   @Override
   protected void cleanup(Mapper<LongWritable, Text, Text, IntWritable>.Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.cleanup(context);
   }


   @Override
   public void run(Mapper<LongWritable, Text, Text, IntWritable>.Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.run(context);
```

```
    }

    @Override
    protected void setup(Mapper<LongWritable, Text, Text, IntWritable>.Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }

}

package CancellationCount2;

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

import java.io.IOException;

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

    @Override
    protected void cleanup(Reducer<Text, IntWritable, Text, IntWritable>.Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.cleanup(context);
    }

    @Override
```

```
    protected void reduce(Text key, Iterable<IntWritable> values,
                        Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException,
InterruptedException {


        int sum = 0;


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


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


    @Override
    public void run(Reducer<Text, IntWritable, Text, IntWritable>.Context arg0)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.run(arg0);
    }


    @Override
    protected void setup(Reducer<Text, IntWritable, Text, IntWritable>.Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }


}
```

10. Counting with Counters

```
package CountersDistanceGroup;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Counter;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class Driver {
   public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException
   {
      Job job = Job.getInstance();

      job.setJarByClass(Driver.class);



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

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

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



        int code = job.waitForCompletion(true)? 1 : 0;
        if (code==1) {
           System.out.print("the counter output:\n");
           for(Counter counter: job.getCounters().getGroup("DistanceCounter")) {
              //
              System.out.println(counter.getDisplayName()+":\t"+counter.getValue());
              // System.out.println("the counter values is "+counter.getValue());
           }

        }
     }
}
```

```
package CountersDistanceGroup;

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

```
import java.util.HashSet;


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

   @Override

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


      String distances[]= {"1","2","3","4","5","6","7", "8","9","10","11","12"};

      HashSet<String> distanceHashSet= new HashSet<String>(Arrays.asList(distances));


      String input = value.toString();

      String splitString[] = input.split(",");

      if(splitString.length > 39 && !value.toString().contains("YEAR")) {

         String distanceGroup = splitString[40].replace("\"","").trim();

         //  System.out.print(month);


         if (distanceHashSet.contains(distanceGroup) && splitString[40] != "grade") {

            // System.out.print(month);

            context.getCounter("DistanceCounter", distanceGroup).increment(1);

         } else {

            context.getCounter("DistanceCounter", "unknown").increment(1);


         }

      }



   }

}
```

11. Delay Count based on reason

```
package DelayCount3;



import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


import java.io.IOException;


public class Driver {

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

      Job job = Job.getInstance();

      job.setJarByClass(Driver.class);


      job.setMapperClass(DelayCountMapper.class);

      job.setCombinerClass(DelayCountReducer.class);

      job.setReducerClass(DelayCountReducer.class);


      job.setInputFormatClass(TextInputFormat.class);

      job.setOutputFormatClass(TextOutputFormat.class);
```

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

        job.setNumReduceTasks(1);

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

        FileSystem fs = FileSystem.get(job.getConfiguration());
//      Path outDir = new Path(args[1]);
//      if(fs.exists(outDir)) {
//          fs.delete(outDir, true);
//      }

        // Submit the job, then poll for progress until the job is complete
        job.waitForCompletion(true);
    }
}

package DelayCount3;

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

import java.io.IOException;

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

```
    private static final IntWritable one = new IntWritable(1);


    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String t= "";
        String[] tokens = line.split(",");
        Integer delay=0;
       //  for(String s : tokens) {
        if(!tokens[0].contains("YEAR") && tokens.length >45 && !tokens[41].equalsIgnoreCase("")
                && !tokens[41].equalsIgnoreCase(" ") && !tokens[41].equalsIgnoreCase("NA")) {
if(Float.parseFloat(tokens[41]) >0 ){
t="CARRIER_DELAY";
delay=(int)Float.parseFloat(tokens[41]);
}
else if(Float.parseFloat(tokens[42]) >0 ){
    t="WEATHER_DELAY";
    delay=(int)Float.parseFloat(tokens[42]);
            }
else if(Float.parseFloat(tokens[43]) >0 ){
    t="NAS_DELAY";
    delay=(int)Float.parseFloat(tokens[43]);
}
else if (Float.parseFloat(tokens[44]) >0 ){
    t="SECURITY_DELAY";
    delay=(int)Float.parseFloat(tokens[44]);
}
else{
```

```
    t="LATE_AIRCRAFT_DELAY";
    delay=(int)Float.parseFloat(tokens[45]);
}
        context.write(new Text(t+" "+tokens[2]), new Text(delay.toString()+" "+1));
      }
      //}
    }

    @Override
    protected void cleanup(Context context)
        throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.cleanup(context);
    }

    @Override
    public void run(Context context)
        throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.run(context);
    }

    @Override
    protected void setup(Context context)
        throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.setup(context);
    }
```

```
}

package DelayCount3;

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

import java.io.IOException;

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

   @Override
   protected void cleanup(Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.cleanup(context);
   }

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

      int sum = 0;
      int count=0;
      for (Text value : values) {
         String a[]=value.toString().split(" ");
         sum += Float.parseFloat(a[0]);
         count+=Integer.parseInt(a[1]);
```

```
        }

        context.write(key, new Text(sum+" "+count));
    }

    @Override
    public void run(Context arg0)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.run(arg0);
    }

    @Override
    protected void setup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }

}
```

12. Delay Median using Arraylist

```
package DelayFMedian;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.SortedMapWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.lib.IdentityReducer;
```

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

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import DelayFMedian.FilterMapper;



import java.io.IOException;


public class Driver {

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

   {

      Job job = Job.getInstance();


      job.setJarByClass(Driver.class);


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

      Path outDir = new Path(args[1]);

      FileOutputFormat.setOutputPath(job, outDir);


      job.setMapperClass(FilterMapper.class);

      job.setReducerClass(FilterReducer.class);

     //  job.setCombinerClass(myCombiner.class);


      //job.setOutputFormatClass(TextOutputFormat.class);

//       job.setMapOutputValueClass(SortedMapWritable.class);

//       job.setMapOutputKeyClass(Text.class);
```

```
//       job.setInputFormatClass(TextInputFormat.class);
//       job.setOutputKeyClass(Text.class);
//       job.setOutputValueClass(MedianStdDevTuple.class);

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

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

        job.setNumReduceTasks(1);

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

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

        job.waitForCompletion(true);
    }
}
```

```
package DelayFMedian;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

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

```
   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      try {
         String split[]=value.toString().split(",");
        // IntWritable one = new IntWritable(1);
         String  symbol = split[0];
         if(!symbol.contains("YEAR") && split.length>20 && !split[31].equalsIgnoreCase("0") &&
!split[31].equalsIgnoreCase("") && !split[31].equalsIgnoreCase("")) {
            context.write(new Text(split[6]),new Text(split[31]));
         }

      } catch(Exception ex) {
         System.out.println("Exception in mapper"+ex);
      }
   }
}

package DelayFMedian;

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

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

public class FilterReducer extends Reducer<Text, Text, Text,Text> {
   ArrayList<Integer> array= new ArrayList<Integer>();
```

```
   @Override
   protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
   Integer median=0;
      //int sum = 0;
      //int count = 0;

//Integer temp= (int) Float.parseFloat(key.toString());
      try {
         for (Text value:values) {

            Integer temp = (int) Float.parseFloat(value.toString());
            if (temp != 0){
               array.add(temp);
            }
            //context.write(new Text(key), NullWritable.get());
         }
         Collections.sort(array);
         int len=array.size();
         if(array.size()%2==0){
           // median= array.get((int) len/2 - 1)+array.get((int) len/2);
            median= (array.get((int) len/2 - 1)+array.get((int) len/2)) /2;
         }
         else{
            median= array.get(len/2);
         }

         context.write(key,new Text(median+""));
```

```
      }
      catch (Exception e){
         System.out.println(key+e.getMessage());
      }
   }
}
```

13. Find Max and Min distance

```
package DistanceMax;

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

import java.io.IOException;

public class Driver {
   public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException
   {
      Job job = Job.getInstance();

      job.setJarByClass(Driver.class);



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

```
        FileOutputFormat.setOutputPath(job, outDir);

        job.setMapperClass(MyMapper.class);
        job.setReducerClass(MyReducer.class);
        job.setCombinerClass(MyReducer.class);

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

        job.setNumReduceTasks(1);

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



        job.waitForCompletion(true);
    }
}

package DistanceMax;

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

import java.io.IOException;

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

```
    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        Text symbol;
        String line = value.toString();
        String[] fields = line.split(",");
        if (fields.length > 20) {
            if (!line.contains("YEAR")) {
                symbol = new Text(fields[6].replace("\"", "").trim());
                context.write(symbol, new Text(fields[39]+"\t"+fields[39]));
            }

        }
    }
}

package DistanceMax;

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

import java.io.IOException;

public class MyReducer extends Reducer<Text,Text,Text,Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
        Integer currentMaxRateValue;
        Integer currentMinRateValue;
```

```
        Integer maxRateValue =  0;
        Integer minRateValue = Integer.MAX_VALUE;

        for (Text value : values) {
           String a[]=value.toString().split("\t");
           currentMaxRateValue = (int) Float.parseFloat(a[1].trim().replace("\"", "").trim());
           currentMinRateValue =  (int) Float.parseFloat(a[0].trim().replace("\"", "").trim());

           if (currentMaxRateValue > maxRateValue) {
              maxRateValue = currentMaxRateValue;
           }
           if (currentMinRateValue < minRateValue) {
              minRateValue = currentMinRateValue;
           }
        }
        context.write(key, new Text(minRateValue+"\t"+maxRateValue));
     }
}
```

14. **Distinct**:

```
package Distinct;

import SecSortPartitionerWritable.*;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
```

```
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.NLineInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


import java.io.IOException;


public class Driver

{

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

   {

//       Job job = Job.getInstance();

//

//       job.setJarByClass(Driver.class);

//

//       job.setGroupingComparatorClass(GroupComparator.class);

//       job.setSortComparatorClass(SecodarySortComparator.class);

//       job.setPartitionerClass(KeyPartition.class);

//

//       FileInputFormat.addInputPath(job, new Path(args[0]));

//       Path outDir = new Path(args[1]);

//       FileOutputFormat.setOutputPath(job, outDir);

//

//       job.setMapperClass(MyMapper.class);

//       job.setReducerClass(MyReducer.class);

//

//       job.setNumReduceTasks(1);
```

```
//
//      job.setOutputKeyClass(CompositeKeyWritable.class);
//      job.setOutputValueClass(NullWritable.class);
//
//      FileSystem fs = FileSystem.get(job.getConfiguration());
//      if(fs.exists(outDir)) {
//        fs.delete(outDir, true);
//      }
//
//      Boolean a=job.waitForCompletion(true);
//      if(a==true)
//      {
        Job job2 = Job.getInstance();

        job2.setJarByClass(InverteBloomChain.Driver.class);



        FileInputFormat.addInputPath(job2, new Path(args[0]));
        Path outDir1 = new Path(args[1]);
        FileOutputFormat.setOutputPath(job2, outDir1);

        job2.setMapperClass(DistinctMapper.class);
        job2.setReducerClass(DistinctReducer.class);
        job2.setCombinerClass(DistinctReducer.class);

        job2.setOutputFormatClass(TextOutputFormat.class);
        job2.setMapOutputValueClass(NullWritable.class);
        job2.setMapOutputKeyClass(Text.class);
        job2.setInputFormatClass(NLineInputFormat.class);
```

```
        NLineInputFormat.setNumLinesPerSplit(job2, 10000);
        job2.setNumReduceTasks(1);

        job2.waitForCompletion(true);
  //    }
   }
}

package Distinct;

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 java.io.IOException;

public class DistinctMapper extends Mapper<LongWritable, Text, Text, NullWritable> {
   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      try {
         String input = value.toString();

         String splitString[] = input.split(",");
         if(splitString.length>10 && !input.contains("YEAR")) {
            String origin = splitString[6].replace("\"","").trim();

            context.write(new Text(origin), NullWritable.get());
```

```
        }
      } catch(Exception e) {
        System.out.println(e);
      }
    }
}
```

```
package Distinct;

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

import java.io.IOException;

public class DistinctReducer extends Reducer<Text, NullWritable, Text, NullWritable> {
   @Override
   protected void reduce(Text key, Iterable<NullWritable> values, Context context) throws IOException,
InterruptedException {

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

15. Flight Carrier Distance Memory Conscious Median:

```
package FlightCarrierDistanceMedian;

import org.apache.hadoop.fs.Path;
```

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

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


import java.io.IOException;


public class Driver {

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

   {

      Job job = Job.getInstance();


      job.setJarByClass(Driver.class);


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

      Path outDir = new Path(args[1]);

      FileOutputFormat.setOutputPath(job, outDir);


      job.setMapperClass(MyMapper.class);

      job.setReducerClass(MyReducer.class);

      job.setCombinerClass(myCombiner.class);


      //job.setOutputFormatClass(TextOutputFormat.class);

      job.setMapOutputValueClass(SortedMapWritable.class);

      job.setMapOutputKeyClass(Text.class);

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

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

        job.setNumReduceTasks(1);

        job.waitForCompletion(true);
    }
}

package FlightCarrierDistanceMedian;

import org.apache.hadoop.io.Writable;

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

public class MedianStdDevTuple implements Writable {
    private  float median;
    private float stdDev;

    public MedianStdDevTuple(float median, float stdDev) {
        this.median = median;
        this.stdDev = stdDev;
    }

    public float getMedian() {
        return median;
    }
```

```
public void setMedian(float median) {
   this.median = median;
}

public float getStdDev() {
   return stdDev;
}

public void setStdDev(float stdDev) {
   this.stdDev = stdDev;
}

public void write(DataOutput out) throws IOException {
   out.writeFloat(this.median);
   out.writeFloat(this.stdDev);
}

@Override
public String toString() {

   return "median:"+ this.median +" Std_Div:"+this.stdDev;
}

public void readFields(DataInput in) throws IOException {
   this.median=in.readFloat();
   this.stdDev=in.readFloat();
}
```

```
    public MedianStdDevTuple() {
    }
}

package FlightCarrierDistanceMedian;

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

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

public class myCombiner extends Reducer<Text, SortedMapWritable, Text, SortedMapWritable> {
    @Override
    protected void reduce(Text key, Iterable<SortedMapWritable> values, Context context) throws
IOException, InterruptedException {
      // int sum=0;
       SortedMapWritable map = new SortedMapWritable();
       for(SortedMapWritable v:values) {
          for(Object entry : v.entrySet() ) {
             if(entry instanceof Map.Entry) {
                Map.Entry entry1 = (Map.Entry) entry;
                IntWritable count = (IntWritable) map.get(entry1.getKey());
                FloatWritable mapKey = ((FloatWritable) entry1.getKey());
                Integer mapValue = ((IntWritable) entry1.getValue()).get();
                if (count != null) {
                   map.put(mapKey, new IntWritable(count.get() + ((IntWritable) entry1.getValue()).get()));
                } else {
```

```
                map.put(mapKey, new IntWritable(((IntWritable) entry1.getValue()).get()));
              }
            }
          }
          v.clear();
        }
        context.write(key,map);

    }
}
```

```
package FlightCarrierDistanceMedian;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MyMapper extends Mapper<LongWritable, Text, Text, SortedMapWritable> {
    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      try {
          String split[]=value.toString().split(",");
          IntWritable one = new IntWritable(1);
          String  symbol = split[6];
          if(!split[0].contains("YEAR") && split.length>30 && !split[31].equalsIgnoreCase("0") &&
!split[31].equalsIgnoreCase("\"0\"")) {
```

```
            Float avg = Float.parseFloat(split[39]);
            SortedMapWritable map = new SortedMapWritable();
            map.put(new FloatWritable(avg),one);
            context.write(new Text(symbol.replace("\"","")),map);
          }

      } catch(Exception ex) {
         System.out.println("Exception in mapper"+ex);
      }
   }
}

package FlightCarrierDistanceMedian;

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

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

public class MyReducer extends Reducer<Text, SortedMapWritable, Text,MedianStdDevTuple> {
   @Override
   protected void reduce(Text key, Iterable<SortedMapWritable> values, Context context) throws
IOException, InterruptedException {

       //int sum = 0;
       //int count = 0;
       MedianStdDevTuple result = new MedianStdDevTuple();
```

```
TreeMap<Float, Integer> finalMap= new TreeMap<Float, Integer>();


try{
   Float sum = 0f;
   Integer totalCount = 0;
   for(SortedMapWritable v:values) {
      for(Object entry: v.entrySet() ) {
         if(entry instanceof Map.Entry) {
            Map.Entry entry1 = (Map.Entry) entry;
            Float mapKey = ((FloatWritable) entry1.getKey()).get();
            Integer mapValue = ((IntWritable) entry1.getValue()).get();
            Integer count = finalMap.get(mapKey);


            sum = sum + mapValue * mapKey;
            totalCount = totalCount + mapValue;



            if (count != null) {
               finalMap.put(mapKey, mapValue + count);
            } else {
               finalMap.put(mapKey, mapValue);
            }
         }


      }
      v.clear();
   }
   Integer middleIndex = totalCount/2;
   int prev=0;
```

```
int current = 0;
Float prevKey = 0f;
Float median=0f;



for(Map.Entry<Float,Integer> entry: finalMap.entrySet()) {
   current = entry.getValue();
   if(middleIndex > prev && middleIndex<=(current+prev)) {
      if(totalCount%2==0) {
         median = (entry.getKey()+prevKey)/2;

      } else {
         median = entry.getKey();
      }
     break;
   }
   prev= current+prev;
   prevKey = entry.getKey();

}
result.setMedian(median);

Float mean = sum/totalCount;

Float sumOfSquares=0.f;
for(Map.Entry<Float,Integer> entry:finalMap.entrySet()) {
   sumOfSquares = sumOfSquares + (entry.getKey()-mean)*(entry.getKey()-mean);
}
Float stdDev = (float) Math.sqrt(sumOfSquares/(totalCount-1));
```

```
            result.setStdDev(stdDev);

            context.write(key,result);

        } catch(Exception e) {
            System.out.println("exception in reducer "+ e);
        }
    }
}
```

16. Bloom Filter

Inverted Bloom Chain:

```
package InverteBloomChain;

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

import java.io.IOException;

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

```
{
   Job job = Job.getInstance();

   job.setJarByClass(Driver.class);

   FileInputFormat.addInputPath(job, new Path(args[0]));
   Path outDir = new Path(args[1]+"_1");
   FileOutputFormat.setOutputPath(job, outDir);

   job.setMapperClass(FilterMapper.class);
  //  job.setReducerClass(MyReducer.class);
  // job.setCombinerClass(MyReducer.class);

   job.setOutputFormatClass(TextOutputFormat.class);
   job.setMapOutputValueClass(NullWritable.class);
   job.setMapOutputKeyClass(Text.class);
   job.setInputFormatClass(TextInputFormat.class);

  //  job.setNumReduceTasks(1);


   Boolean a=job.waitForCompletion(true);

   if(a==true)
   {
      Job job2 = Job.getInstance();

      job2.setJarByClass(Driver.class);
```

```
        // job2.setGroupingComparatorClass(GroupComparator.class);
        // job2.setSortComparatorClass(SecodarySortComparator.class);
        //job2.setPartitionerClass(KeyPartition.class);

        FileInputFormat.addInputPath(job2, new Path(args[1]+"_1"));
        Path outDir1 = new Path(args[1]);
        FileOutputFormat.setOutputPath(job2, outDir1);

        job2.setMapperClass(MyMapper.class);
        job2.setReducerClass(MyReducer.class);
        job2.setCombinerClass(MyReducer.class);

        job2.setOutputFormatClass(TextOutputFormat.class);
        job2.setMapOutputValueClass(Text.class);
        job2.setMapOutputKeyClass(Text.class);
        job2.setInputFormatClass(TextInputFormat.class);

        job2.setNumReduceTasks(1);

        // job2.setOutputKeyClass(CompositeKeyWritable.class);
        // job2.setOutputValueClass(NullWritable.class);


        job2.waitForCompletion(true);
      }
   }
}

package InverteBloomChain;
```

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

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

import org.apache.hadoop.io.NullWritable;

import com.google.common.base.Charsets;
import com.google.common.hash.BloomFilter;
import com.google.common.hash.Funnel;
import com.google.common.hash.Sink;

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

   private BloomFilter<String> origin;

   Funnel<String> p = new Funnel<String>() {

      public void funnel(String from, Sink into) {

         into.putString(from, Charsets.UTF_8);

      }
   };

   @Override
```

```
    protected void setup(Mapper<LongWritable, Text, Text, NullWritable>.Context context) throws
IOException, InterruptedException {


        this.origin = BloomFilter.create(p, 300000, 0.1);


        String p1 = "BOS";

        String p2 = "JFK";

       // Person p2 = new Person("Jamie", "Scott");


        ArrayList<String> originList = new ArrayList<String>();

        originList.add(p1);

        originList.add(p2);


        for (String ps : originList) {

           origin.put(ps);

        }


    }


    @Override

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

       try {

          String input = value.toString();


          String splitString[] = input.split(",");

          String origins = splitString[10].replace("\"","").trim();

          //System.out.println(origins);

          //String values[] = value.toString().split(",");
```

```
        //Person p = new Person(values[1], values[2]);


        if (origin.mightContain(origins) && !splitString[0].contains("YEAR")) {
           context.write(value,NullWritable.get());
        }


        //context.write(new Text(origin.replace("\"","").trim()), new
Text(splitString[6].replace("\"","").trim()));
     } catch(Exception e) {
        System.out.println(e);
     }
  }


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

17. **Inverted Index**

```
package InverteBloomChain;


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


import java.io.IOException;


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

```
   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      try {
         String input = value.toString();

         String splitString[] = input.split(",");
         String origin = splitString[10];

         context.write(new Text(origin.replace("\"","").trim()), new
Text(splitString[6].replace("\"","").trim()));
      } catch(Exception e) {
         System.out.println(e);
      }
   }
}

package InverteBloomChain;

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

import java.io.IOException;

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

      String concatString = "";
```

```
      for(Text s:values) {
         if(!concatString.contains(s.toString())) {
            concatString = s.toString() + "," + concatString;
         }

      }
      context.write(key,new Text(concatString));
   }
}
```

18. **Join:**

```
package Join;

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

import java.io.IOException;

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

   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      String[] fields = value.toString().split(",");
      Text t1 = new Text(fields[0].replace("\"","").trim());
      String val = "M"+fields[1].replace("\"","").trim();
      Text t2 = new Text(val);
```

```
        context.write(t1,t2);
    }
}

package Join;

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

import java.io.IOException;

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

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

        String[] fields = value.toString().split(" ");
        Text t1 = new Text(fields[0].replace("\"","").trim());
        String val = "R"+fields[0].replace("\"","").trim();
        Text t2 = new Text(val);
        context.write(t1,t2);
    }
}

package Join;

import org.apache.hadoop.fs.FileSystem;
```

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

public class JoinPattern {

   public static void main(String[] args){

      try {
         Job job = Job.getInstance();
         job.setJarByClass(JoinPattern.class);

         job.getConfiguration().set("type",args[3]);

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

         //Reducer
         job.setReducerClass(JoinReducer.class);
         job.setOutputFormatClass(TextOutputFormat.class);

         //Number of Reducers
         job.setNumReduceTasks(1);

         //Specify Key value
```

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

        //Specify Output Path
        Path output = new Path(args[2]);
        FileOutputFormat.setOutputPath(job, output);

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

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

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

package Join;

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

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

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

```
    String joinType=null;
     Text emptyText = new Text("");
     Text temp = new Text();
     ArrayList<Text> r = new ArrayList<Text>();
     ArrayList<Text> m = new ArrayList<Text>();

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

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

       for(Text val: values){
          temp=val;
          if(val.toString().charAt(0)=='R') r.add(new Text(temp.toString().substring(1)));
          else if(val.toString().charAt(0)=='M') m.add(new Text(temp.toString().substring(1)));
       }
       executeJoin(context);
    }

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

       if(joinType.equalsIgnoreCase("inner")){
```

```
    if(!r.isEmpty() && !m.isEmpty()){
        for(Text ra: r){
            for(Text mo: m){
                context.write(ra,mo);
            }
        }
    }
}

else if(joinType.equalsIgnoreCase("leftouter")){
    for(Text ra: r){
        if(!m.isEmpty()){
            for(Text mo : m){
                context.write(ra,mo);
            }
        }
        else context.write(ra,emptyText);
    }
}

else if(joinType.equalsIgnoreCase("rightouter")){
    for(Text mo: m){
        if(!r.isEmpty()){
            for(Text ra: r){
                context.write(ra,mo);
            }
        }
        else context.write(emptyText,mo);
    }
```

```
}

else if(joinType.equalsIgnoreCase("fullouter")){
   if(!r.isEmpty()){
      for(Text ra:r){
         if(!m.isEmpty()){
            for(Text mo: m){
               context.write(ra,mo);
            }
         }else context.write(ra,emptyText);
      }
   }
   else {
      for(Text mo: m){
         context.write(emptyText,mo);
      }
   }
}

else if(joinType.equalsIgnoreCase("antijoin")){
   if(r.isEmpty()^m.isEmpty()){
      for(Text ra: r){
         context.write(ra,emptyText);
      }
      for(Text mo: m){
         context.write(emptyText,mo);
      }
   }
}
```

```
   }
}
```

19. PutMerge:

```
package PutMerge;

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

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class MergeFiles {

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



//Hadoop File System instance
      Configuration conf = new Configuration();
      FileSystem hdfs = FileSystem.get(conf);

      //local File System instance
      FileSystem local = FileSystem.getLocal(conf);

      //Path to the directory in local file System
      Path inputDir = new Path(args[0]);
      Path hdfsFile = new Path(args[1]);
```

```
        try {
           //List of Files Names from FileStatus
           FileStatus[] inputFiles = local.listStatus(inputDir);
           //Writing to hdfs using OutputStream
           FSDataOutputStream out = hdfs.create(hdfsFile);

           //Reading the input files using FSDataInputStream
           for (int i = 0; i < inputFiles.length; i++) {
              FSDataInputStream in = local.open(inputFiles[i].getPath());
             // BufferedReader br = new BufferedReader(new InputStreamReader(in, UTF8));
              //int lineCount = 0;
              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();
        }
     }
}
```

20. Secondary Sort:

```
package SecSortPartitionerWritable;


import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


import java.io.IOException;


public class Driver

{

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

   {

      Job job = Job.getInstance();


      job.setJarByClass(Driver.class);


      job.setGroupingComparatorClass(GroupComparator.class);

      job.setSortComparatorClass(SecodarySortComparator.class);

      job.setPartitionerClass(KeyPartition.class);


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

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

        job.setMapperClass(MyMapper.class);
        job.setReducerClass(MyReducer.class);

        job.setNumReduceTasks(1);

        job.setOutputKeyClass(CompositeKeyWritable.class);
        job.setOutputValueClass(NullWritable.class);

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

        Boolean a=job.waitForCompletion(true);

    }
}

package SecSortPartitionerWritable;

import org.apache.hadoop.io.WritableComparable;

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

```
public class CompositeKeyWritable implements WritableComparable<CompositeKeyWritable> {

	private String state;
	private String airportId;

	public CompositeKeyWritable() {
		super();
	}

	public CompositeKeyWritable(String state, String id) {
		super();
		this.state = state;
		this.airportId = id;
	}

	public void write(DataOutput out) throws IOException {

		out.writeUTF(state);
		out.writeUTF(airportId);
	}

	public void readFields(DataInput in) throws IOException {

		state = in.readUTF();
		airportId = in.readUTF();
	}

	public int compareTo(CompositeKeyWritable o) {
		int result = this.state.compareTo(o.state);
```

```
            if (result == 0) {
                return this.airportId.compareTo(o.airportId);
            }
            return result;
        }

        public String getState() {
            return state;
        }

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

        public String getAirportId() {
            return airportId;
        }

        public void setAirportId(String airportId) {
            this.airportId = airportId;
        }

        @Override
        public String toString() {

            return state + "," + airportId;
        }

}
```

```
package SecSortPartitionerWritable;

import org.apache.hadoop.io.WritableComparator;

public class GroupComparator extends WritableComparator {

	protected GroupComparator() {
		super(CompositeKeyWritable.class, true);
	}

	@Override
	public int compare(Object a, Object b) {

		CompositeKeyWritable ckw1 = (CompositeKeyWritable)a;
		CompositeKeyWritable ckw2 = (CompositeKeyWritable)b;

		return ckw1.getState().compareTo(ckw2.getState());
	}

}

package SecSortPartitionerWritable;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Partitioner;

public class KeyPartition extends Partitioner<CompositeKeyWritable, NullWritable> {
```

```
	@Override
	public int getPartition(CompositeKeyWritable key, NullWritable value, int numPartitions) {

		return key.getState().hashCode()%numPartitions;

	}

}

package SecSortPartitionerWritable;

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 java.io.IOException;

public class MyMapper extends Mapper<LongWritable, Text, CompositeKeyWritable, NullWritable> {

	@Override
	protected void map(LongWritable key, Text value,
	      Mapper<LongWritable, Text, CompositeKeyWritable, NullWritable>.Context context)
			throws IOException, InterruptedException {
		String line = value.toString();

		String[] tokens = line.split(",");
		String state = null;
		String airportId = null;
```

```
        if(!value.toString().contains("YEAR")) {
        try {

                state = tokens[14].replace("\"", "").trim();
                airportId = tokens[10].replace("\"", "").trim();

        } catch (Exception e) {

        }

        if (state != null && airportId != null) {

                CompositeKeyWritable outKey = new CompositeKeyWritable(state, airportId);

                context.write(outKey, NullWritable.get());

        }
        }
        }

}

package SecSortPartitionerWritable;

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

import java.io.IOException;
```

```
public class MyReducer extends Reducer<CompositeKeyWritable, NullWritable, CompositeKeyWritable,
NullWritable> {

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

		//for (NullWritable v : values) {
			context.write(key, NullWritable.get());
		//}

	}

}
```

```
package SecSortPartitionerWritable;

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

public class SecodarySortComparator extends WritableComparator {

	protected SecodarySortComparator() {
		super(CompositeKeyWritable.class, true);
	}

	@Override
```

```
	public int compare(WritableComparable a, WritableComparable b) {

		CompositeKeyWritable ckw1 = (CompositeKeyWritable) a;
		CompositeKeyWritable ckw2 = (CompositeKeyWritable) b;

		int result = ckw1.getState().compareTo(ckw2.getState());

		if (result == 0) {
			return ckw1.getAirportId().compareTo(ckw2.getAirportId());
		}

		return result;
	}



}
```

21. Shuffle:

```
package ShuffleData;

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

```
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


import java.io.IOException;


public class Driver {

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

      Job job = Job.getInstance();

      job.setJarByClass(Driver.class);


      job.setMapperClass(ShuffleMapper.class);

      job.setReducerClass(ShuffleReducer.class);


      job.setInputFormatClass(TextInputFormat.class);

      job.setOutputFormatClass(TextOutputFormat.class);


      job.setOutputKeyClass(IntWritable.class);

      job.setOutputValueClass(Text.class);


      job.setNumReduceTasks(1);


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

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


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



      job.waitForCompletion(true);

   }
```

```
}

package ShuffleData;

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

import java.io.IOException;
import java.util.Random;

public class ShuffleMapper extends Mapper<LongWritable, Text,IntWritable, Text> {
   private static final IntWritable one = new IntWritable(1);
   private Random random= new Random();
   @Override
   protected void map(LongWritable key, Text value, Context context)
         throws IOException, InterruptedException {
      String line = value.toString();

      String[] tokens = line.split(",");
      if(tokens.length > 15) {
         context.write(new IntWritable(random.nextInt()), value);
      }

   }

   @Override
   protected void cleanup(Context context)
```

```
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.cleanup(context);
  }

  @Override
  public void run(Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.run(context);
  }

  @Override
  protected void setup(Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.setup(context);
  }


}

package ShuffleData;

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

```
import java.io.IOException;

public class ShuffleReducer extends Reducer<IntWritable, Text,Text, NullWritable> {

   @Override
   protected void cleanup(Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.cleanup(context);
   }

   @Override
   protected void reduce(IntWritable key, Iterable<Text> values,
                  Context context) throws IOException, InterruptedException {
      for (Text val: values) {
        // context.write(new IntWritable(),val);
         context.write(val,NullWritable.get());
      }
   }

   @Override
   public void run(Context arg0)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.run(arg0);
   }

   @Override
   protected void setup(Context context)
```

```
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.setup(context);
  }

}
```

22. State Cancellation:

```
package StateCancellationCount6;



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

import java.io.IOException;


public class Driver {
  public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
    Job job = Job.getInstance();
    job.setJarByClass(Driver.class);


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

```
        job.setCombinerClass(CancelCountReducer.class);
        job.setReducerClass(CancelCountReducer.class);

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

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

        job.setNumReduceTasks(1);

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

        FileSystem fs = FileSystem.get(job.getConfiguration());
//      Path outDir = new Path(args[1]);
//      if(fs.exists(outDir)) {
//          fs.delete(outDir, true);
//      }

        // Submit the job, then poll for progress until the job is complete
        job.waitForCompletion(true);
    }
}

package StateCancellationCount6;

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

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

import java.io.IOException;

public class CancelCountMapper extends Mapper<LongWritable, Text, Text, Text> {
   private static final IntWritable one = new IntWritable(1);

   @Override
   protected void map(LongWritable key, Text value, Context context)
         throws IOException, InterruptedException {
      String line = value.toString();
      String t= "";
      String[] tokens = line.split(",");
     //  for(String s : tokens) {
      if(!tokens[0].contains("YEAR") && tokens.length >33 && !tokens[32].equalsIgnoreCase("0")
            && !tokens[33].equalsIgnoreCase(" ") &&
!tokens[33].replace("\"","").trim().equalsIgnoreCase("")) {

         context.write(new
Text(tokens[14].replace("\"","").trim()+"\t"+tokens[33].replace("\"","").trim()+"\t"+tokens[2]), new
Text(1+""));
      }
      //}
   }

   @Override
   protected void cleanup(Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
```

```
        super.cleanup(context);
    }

    @Override
    public void run(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.run(context);
    }

    @Override
    protected void setup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }

}

package StateCancellationCount6;

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

import java.io.IOException;

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

    @Override
```

```
protected void cleanup(Context context)
      throws IOException, InterruptedException {
   // TODO Auto-generated method stub
   super.cleanup(context);
}

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

   int sum = 0;
   int count=0;
   for (Text value : values) {
      String a[]=value.toString().split(" ");
     // sum += Float.parseFloat(a[0]);
      count+=Integer.parseInt(a[0]);
   }

   context.write(key, new Text(count+""));
}

@Override
public void run(Context arg0)
      throws IOException, InterruptedException {
   // TODO Auto-generated method stub
   super.run(arg0);
}

@Override
```

```
   protected void setup(Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.setup(context);
   }

}
```

23 State Delay Count:

```
package StateDelayCount6;



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

import java.io.IOException;

public class Driver {
   public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
      Job job = Job.getInstance();
      job.setJarByClass(Driver.class);
```

```
        job.setMapperClass(DelayCountMapper.class);
        job.setCombinerClass(DelayCountReducer.class);
        job.setReducerClass(DelayCountReducer.class);

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

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

        job.setNumReduceTasks(1);

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

        FileSystem fs = FileSystem.get(job.getConfiguration());
//      Path outDir = new Path(args[1]);
//      if(fs.exists(outDir)) {
//          fs.delete(outDir, true);
//      }

        // Submit the job, then poll for progress until the job is complete
        job.waitForCompletion(true);
    }
}

package StateDelayCount6;

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

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

import java.io.IOException;

public class DelayCountMapper extends Mapper<LongWritable, Text, Text, Text> {
   private static final IntWritable one = new IntWritable(1);

   @Override
   protected void map(LongWritable key, Text value, Context context)
         throws IOException, InterruptedException {
      String line = value.toString();
      String t= "";
      String[] tokens = line.split(",");
      Integer delay=0;
     //  for(String s : tokens) {
      if(!tokens[0].contains("YEAR") && tokens.length >45 && !tokens[41].equalsIgnoreCase("")
            && !tokens[41].equalsIgnoreCase(" ") && !tokens[41].equalsIgnoreCase("NA")) {
      if(Float.parseFloat(tokens[41]) >0 ){
      t="CARRIER_DELAY";
      delay=(int)Float.parseFloat(tokens[41]);
      }
      else if(Float.parseFloat(tokens[42]) >0 ){
         t="WEATHER_DELAY";
         delay=(int)Float.parseFloat(tokens[42]);
               }
      else if(Float.parseFloat(tokens[43]) >0 ){
         t="NAS_DELAY";
```

```
            delay=(int)Float.parseFloat(tokens[43]);
        }
        else if (Float.parseFloat(tokens[44]) >0 ){
            t="SECURITY_DELAY";
            delay=(int)Float.parseFloat(tokens[44]);
        }
        else{
            t="LATE_AIRCRAFT_DELAY";
            delay=(int)Float.parseFloat(tokens[45]);
        }
            context.write(new Text(tokens[14].replace("\"","")+"\t"+t+"\t"+tokens[2]), new
Text(delay.toString()+"\t"+1));
        }
        //}
    }

    @Override
    protected void cleanup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.cleanup(context);
    }

    @Override
    public void run(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.run(context);
    }
```

```
    @Override
    protected void setup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }

}

package StateDelayCount6;

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

import java.io.IOException;

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

    @Override
    protected void cleanup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.cleanup(context);
    }

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

```
        int sum = 0;
        int count=0;
        for (Text value : values) {
            String a[]=value.toString().split("\t");
            sum += Float.parseFloat(a[0]);
            count+=Integer.parseInt(a[1]);
        }

        context.write(key, new Text(sum+"\t"+count));
    }

    @Override
    public void run(Context arg0)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.run(arg0);
    }

    @Override
    protected void setup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }

}
```

24. Top 10 flight Distance:

```
package Top10FlightDistance;

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

import java.io.IOException;



/**
 * Hello world!
 *
 */
public class App
{
   public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException
   {
      Job job = Job.getInstance();
      job.setJarByClass(App.class);


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

```
        Path input = new Path(args[0]);
        Path output = new Path(args[1]);

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

        FileInputFormat.addInputPath(job, input);
        FileOutputFormat.setOutputPath(job, output);

        job.setNumReduceTasks(1);

        job.setMapperClass(TopKMapper.class);
        job.setReducerClass(TopKReducer.class);

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

        job.waitForCompletion(true);
    }
}

package Top10FlightDistance;

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

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

	private static final int K = 10;
	private TreeMap<Integer, Text> tMap;

	@Override
	protected void setup(Mapper<LongWritable, Text, NullWritable, Text>.Context context)
			throws IOException, InterruptedException {
		this.tMap = new TreeMap<Integer, Text>();
	}

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

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

		try {

			String score = tokens[39].replace("\"","").trim();
			tMap.put(Integer.parseInt(score), new Text(value));

		} catch (Exception ex) {
```

```
                }

                if (tMap.size() > K) {
                        tMap.remove(tMap.firstKey());
                }
        }

        @Override
        protected void cleanup(Mapper<LongWritable, Text, NullWritable, Text>.Context context)
                        throws IOException, InterruptedException {

                for (Text t : tMap.values()) {
                        context.write(NullWritable.get(), t);
                }

        }

}

package Top10FlightDistance;

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

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

```
public class TopKReducer extends Reducer<NullWritable, Text, NullWritable, Text> {

	private static final int K = 10;
	private TreeMap<Integer, Text> tMap;

	@Override
	protected void setup(Reducer<NullWritable, Text, NullWritable, Text>.Context context)
			throws IOException, InterruptedException {

		this.tMap = new TreeMap<Integer, Text>();

	}

	@Override
	protected void reduce(NullWritable arg0, Iterable<Text> values,
		Reducer<NullWritable, Text, NullWritable, Text>.Context arg2) throws IOException,
InterruptedException {

		for (Text t : values) {
			String[] tokens = t.toString().split(",");
			tMap.put(Integer.parseInt(tokens[39].replace("\"","").trim()), new Text(t));

			if (tMap.size() > K) {
				tMap.remove(tMap.firstKey());
			}
		}
	}

	@Override
```

```
        protected void cleanup(Reducer<NullWritable, Text, NullWritable, Text>.Context context)
                        throws IOException, InterruptedException {

                for(Text t : tMap.values()) {
                        context.write(NullWritable.get(), t);
                }
        }

}
```

Pig:

25. Join:

```
uniqueid = LOAD 'distinct' using PigStorage(',') as (uid);
carriers = LOAD 'carriers.csv' using PigStorage(',') as (id,name);
joined = JOIN uniqueid BY uid, carriers BY id;
summed = FOREACH joined GENERATE $0,$2;
STORE summed into 'joinoutput';
```

26. Filter Merge:

```
flights = LOAD 'Merge.csv' using PigStorage(',');
filter1 = FILTER flights BY $0 > 2017;
filter2 = FOREACH filter1 GENERATE $0,$1,$2,$3,$4,$5,$6,$7,$8,$9,'LatestData';
STORE filter2 into 'LatestData';
```

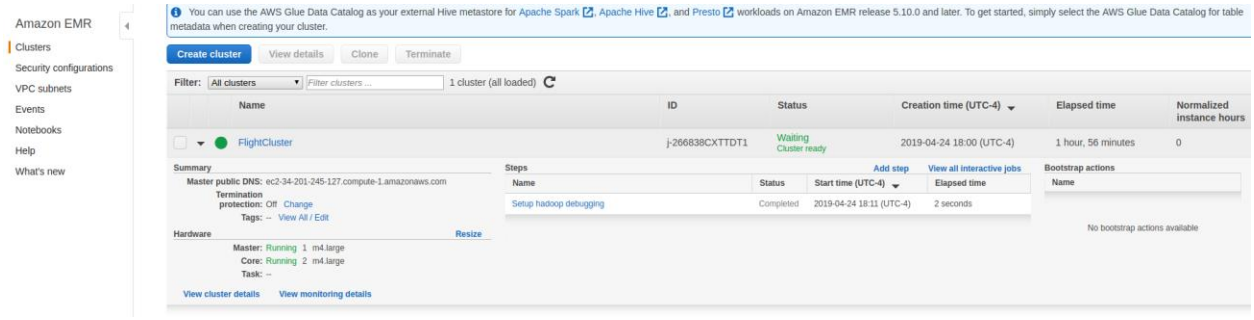
27. Topk:

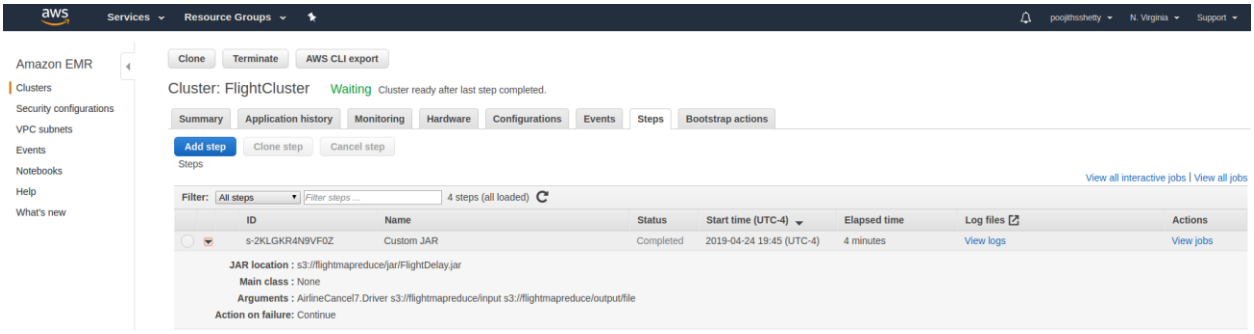
```
flights = LOAD 'apr17.csv' using PigStorage(',');
grouped = GROUP flights BY $6;
summed = FOREACH grouped GENERATE group, COUNT(flights) AS cntd;
```

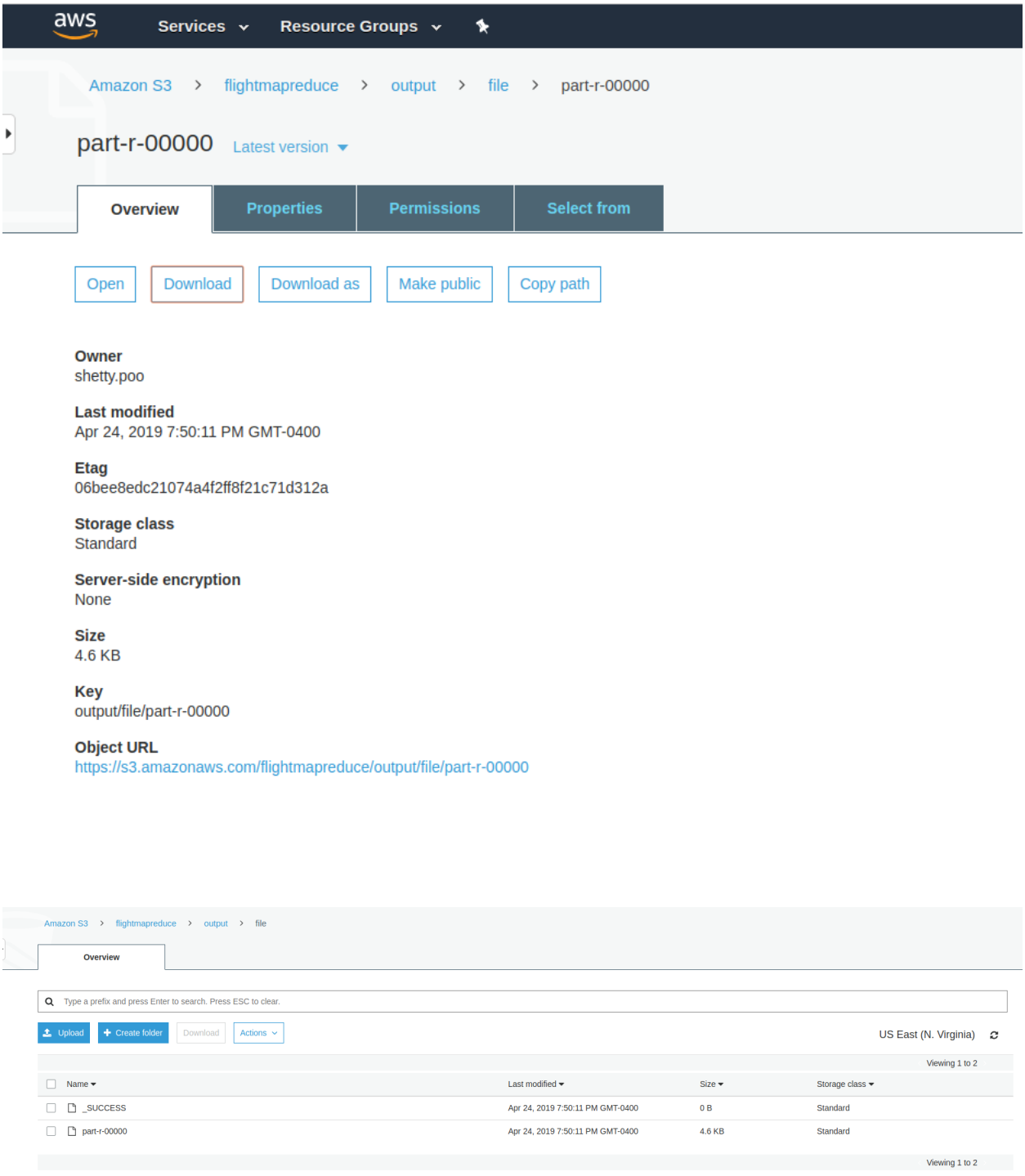
```
sorted = ORDER summed BY cntd DESC;

top25 = LIMIT sorted 10;

Dump top25;

STORE top25 into 'top10Carriers';
```

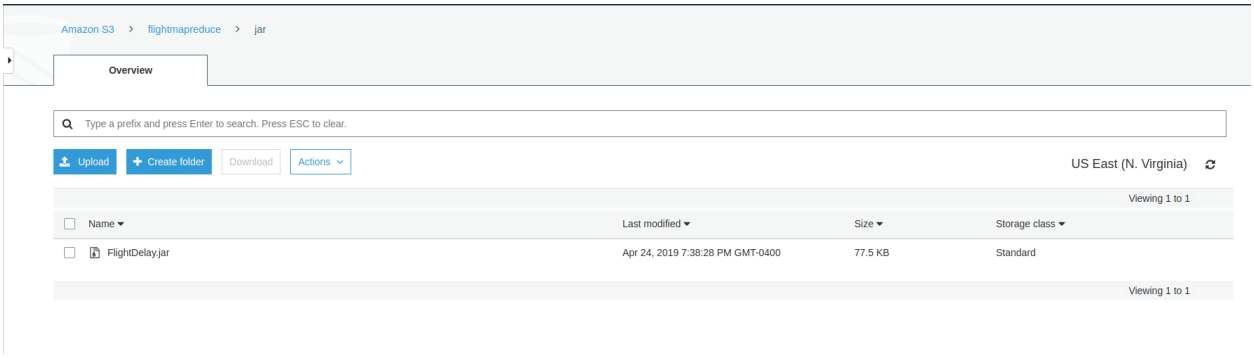
Additional Screenshots for reference:

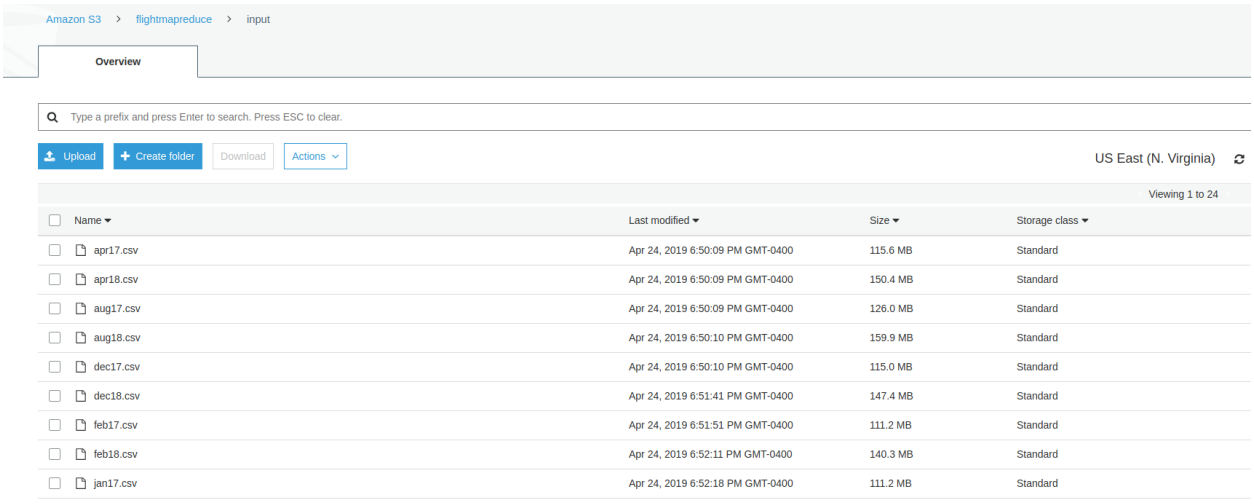
AWS:






aws
Services
Resource Groups
Amazon S3 > flightmapreduce > output > file > part-r-00000
part-r-00000
Latest version
Overview
Properties
Permissions
Select from
Open
Download
Download as
Make public
Copy path
Owner
shetty.poo
Last modified
Apr 24, 2019 7:50:11 PM GMT-0400
Etag
06bee8edc21074a4f2ff8f21c71d312a
Storage class
Standard
Server-side encryption
None
Size
4.6 KB
Key
output/file/part-r-00000
Object URL
https://s3.amazonaws.com/flightmapreduce/output/file/part-r-00000
Amazon S3 > flightmapreduce > output > file
Overview
Type a prefix and press Enter to search. Press ESC to clear.
Upload
Create folder
Download
Actions
US East (N. Virginia)
Viewing 1 to 2
Name
Last modified
Size
Storage class
_SUCCESS
Apr 24, 2019 7:50:11 PM GMT-0400
0 B
Standard
part-r-00000
Apr 24, 2019 7:50:11 PM GMT-0400
4.6 KB
Standard
Viewing 1 to 2





Owner
shetty.poo

Last modified
Apr 24, 2019 7:58:17 PM GMT-0400

Etag
ee493fe385f7a1d2785881c5bcd44eb4

Storage class
Standard

Server-side encryption
AES-256

Size
21.2 KB

Key
logs/j-266838CXTTDT1/hadoop-mapreduce/history/2019/04/24/000000/job_1556143697838_0001-1556149535584-hadoop-FlightDelay.jar-1556149810763-60-1-SUCCEEDED-default-1556149546461.jhist.gz

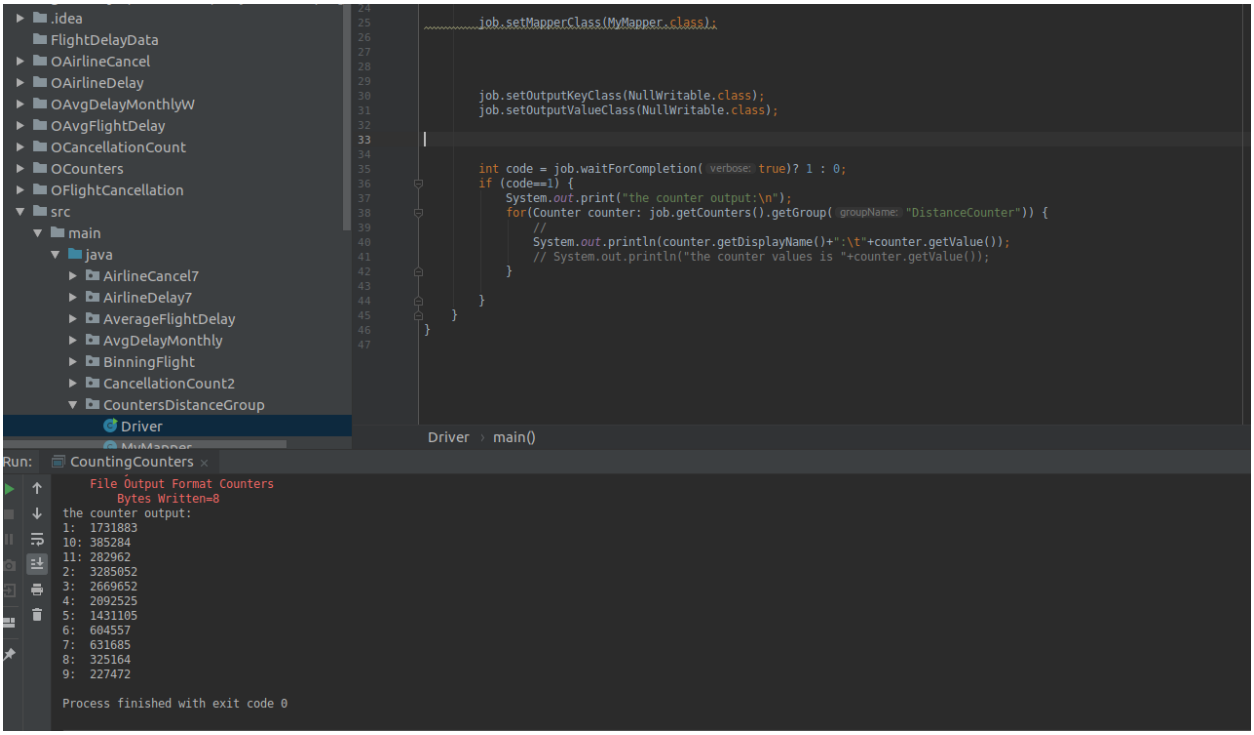
Object URL
https://s3.amazonaws.com/flightmapreduce/logs/j-266838CXTTDT1/hadoop-mapreduce/history/2019/04/24/000000/job_1556143697838_0001-1556149535584-hadoop-FlightDelay.jar-1556149810763-60-1-SUCCEEDED-default-1556149546461.jhist.gz

Open

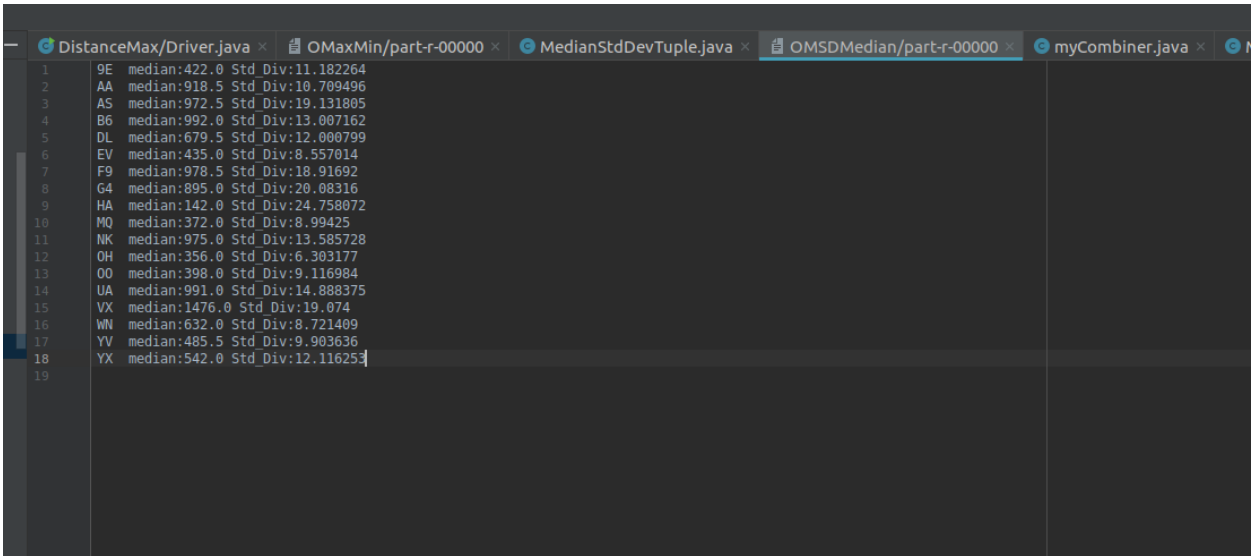
part-r-00000
~/Downloads

```
9E	A	1	13
9E	A	10	12
9E	A	12	23
9E	A	3	494
9E	A	5	8
9E	A	9	7
9E	B	1	364
9E	B	10	106
9E	B	12	28
9E	B	3	2086
9E	B	5	350
9E	B	7	12
9E	B	9	36
9E	C	1	1077
9E	C	10	60
9E	C	12	65
9E	C	3	486
9E	C	5	84
9E	C	7	1302
9E	C	9	100
AA	A	1	1008
AA	A	10	642
AA	A	12	553
AA	A	3	621
AA	A	4	274
AA	A	5	1180
AA	A	7	2335
AA	A	8	608
AA	A	9	333
AA	B	1	3671
AA	B	10	1168
AA	B	12	1132
AA	B	3	5525
AA	B	4	356
AA	B	5	1322
AA	B	7	2290
AA	B	8	974
AA	B	9	1021
AA	C	1	214
AA	C	10	266
AA	C	12	13
AA	C	3	57
```

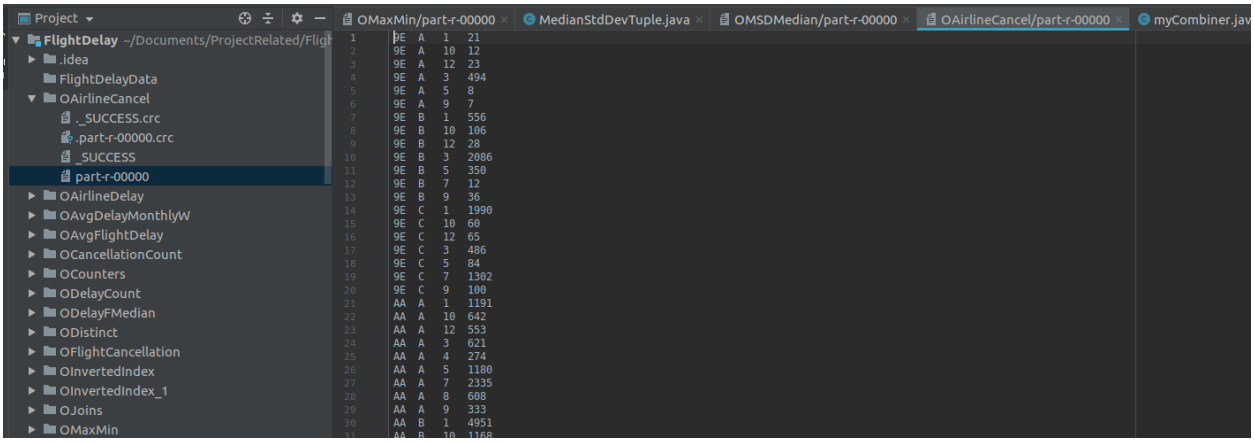
Counting with Counters:



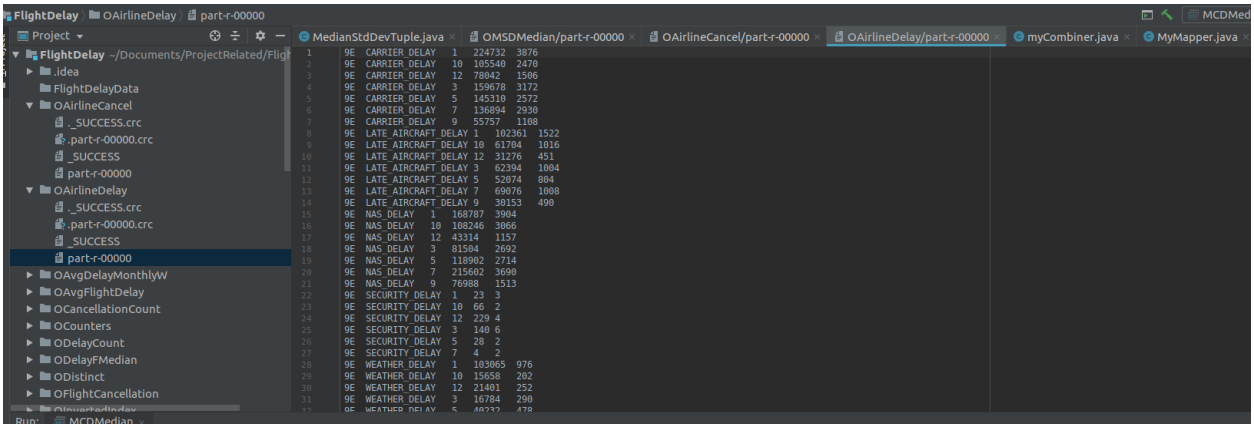
Median Based on Flight Carrier SD: MCD Combiner used



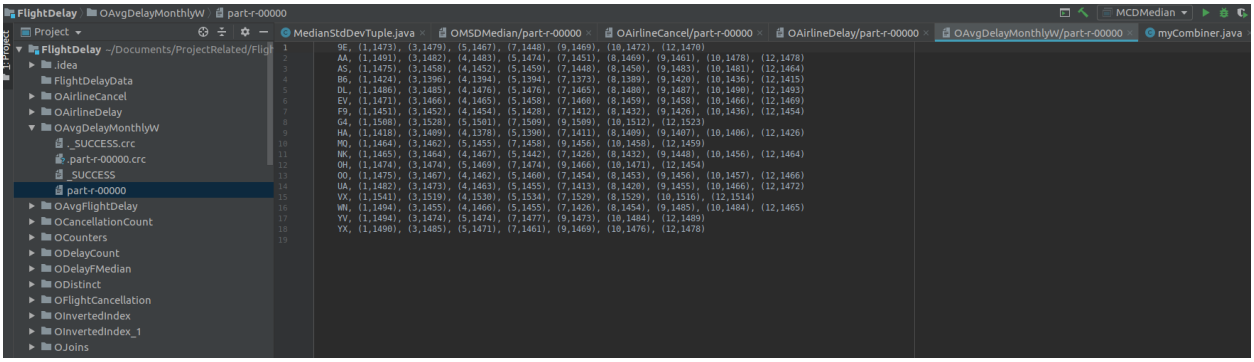
Airline Cancelled each month coint:



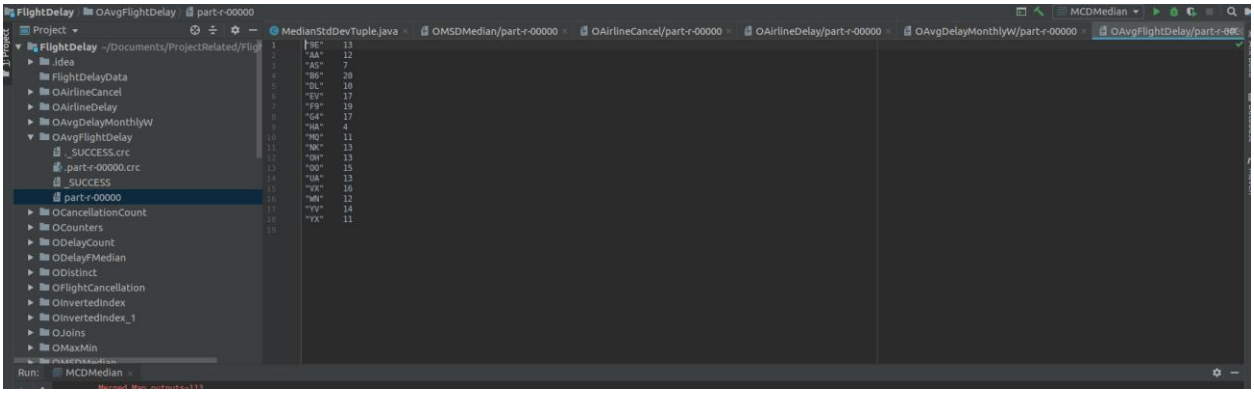
Airline Delay each month count:



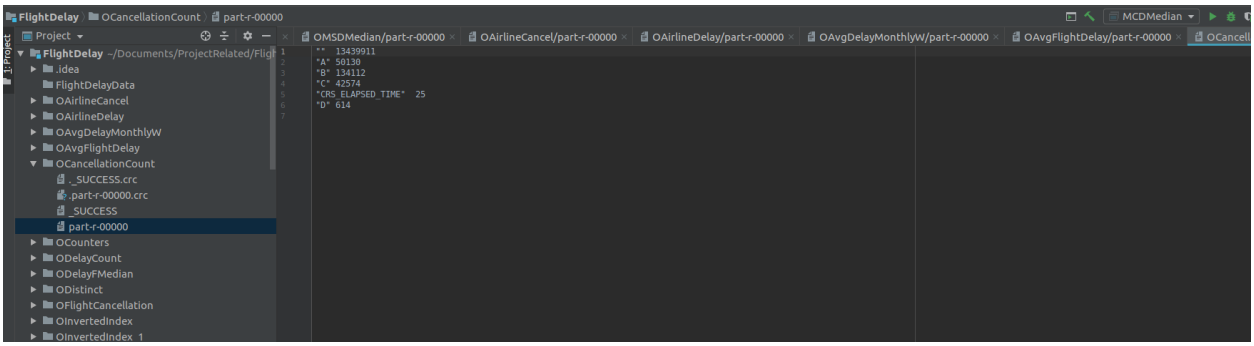
Avg Delay each month:



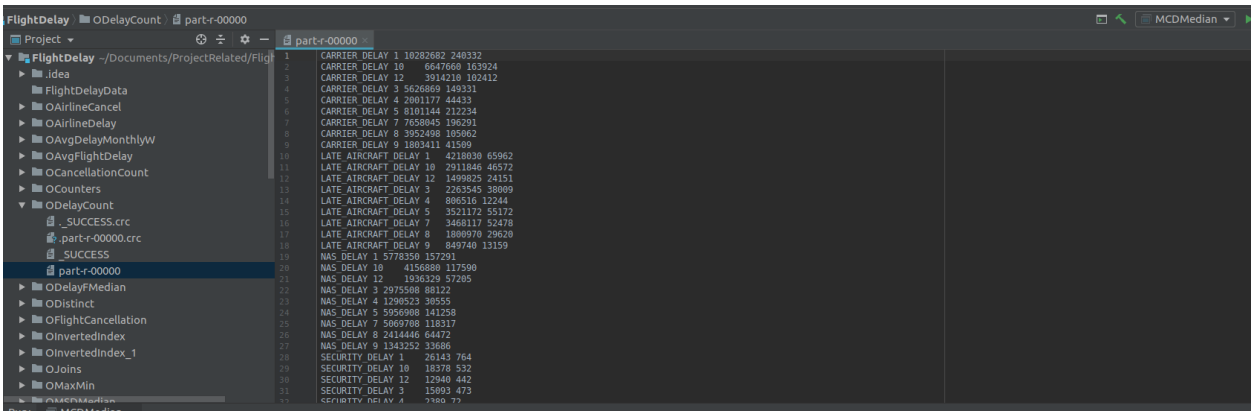
Avg Flight Delay:



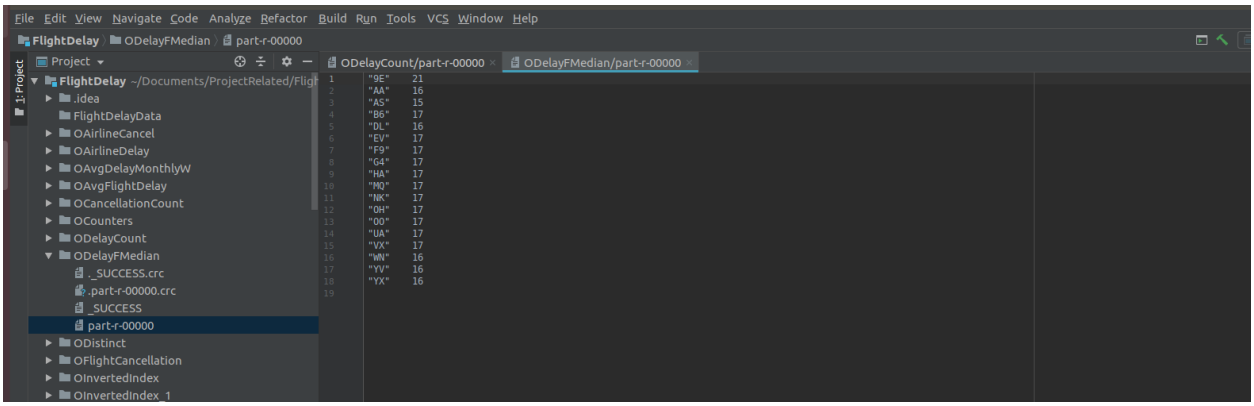
Total Cancellation Count by reason:



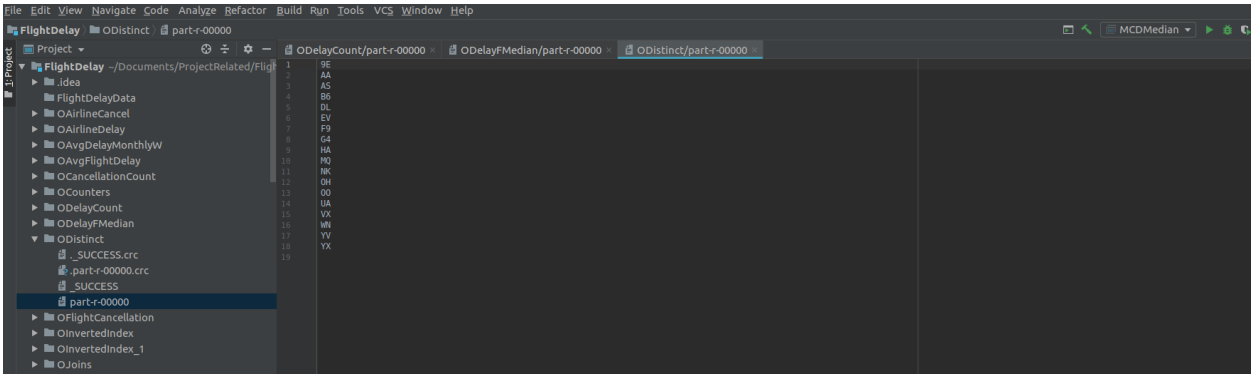
Delay Count Total each month:



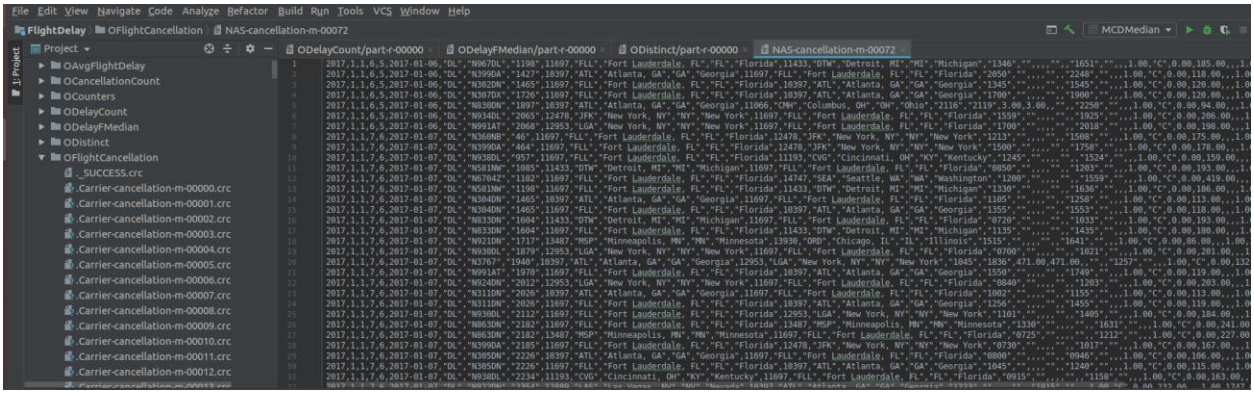
Delay Median based on Airline:



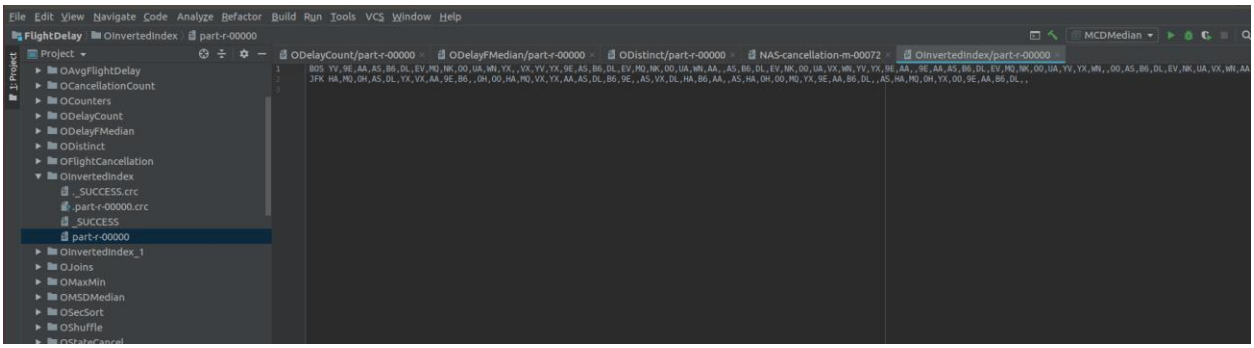
Distinct Airlines:



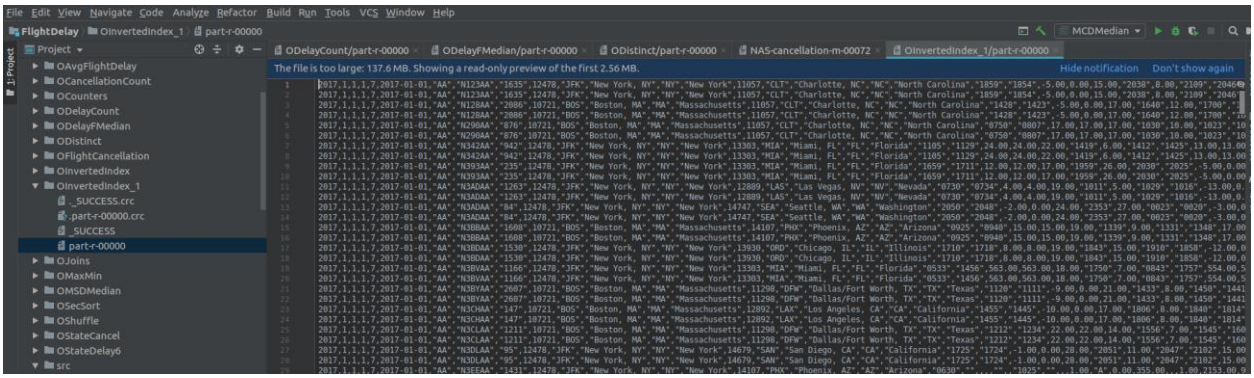
Binning based on cancellation reason separate files:



Bloom Filter Inverted Index and Cjhaining: Based on BOS and JKF flights

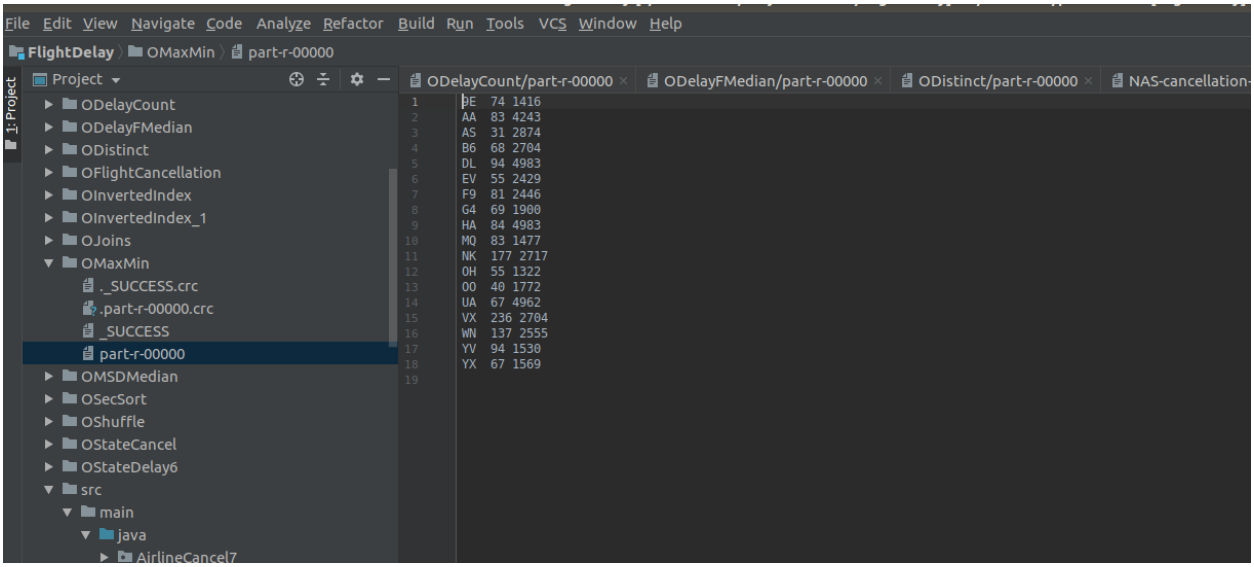


Temporary Bloom Filter Data:

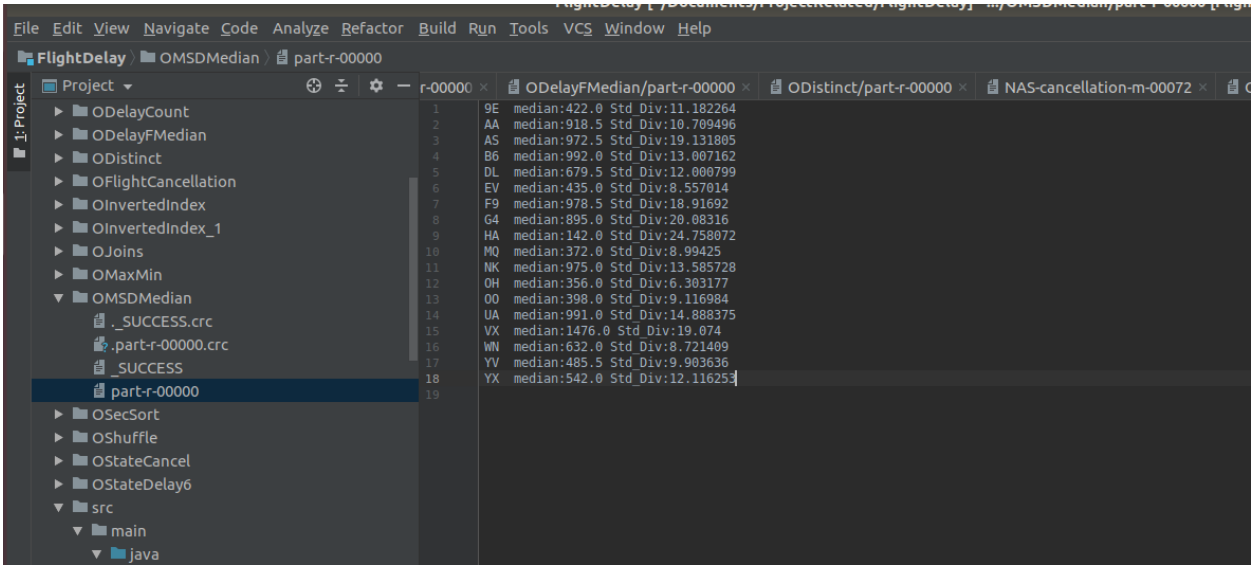


Joins:

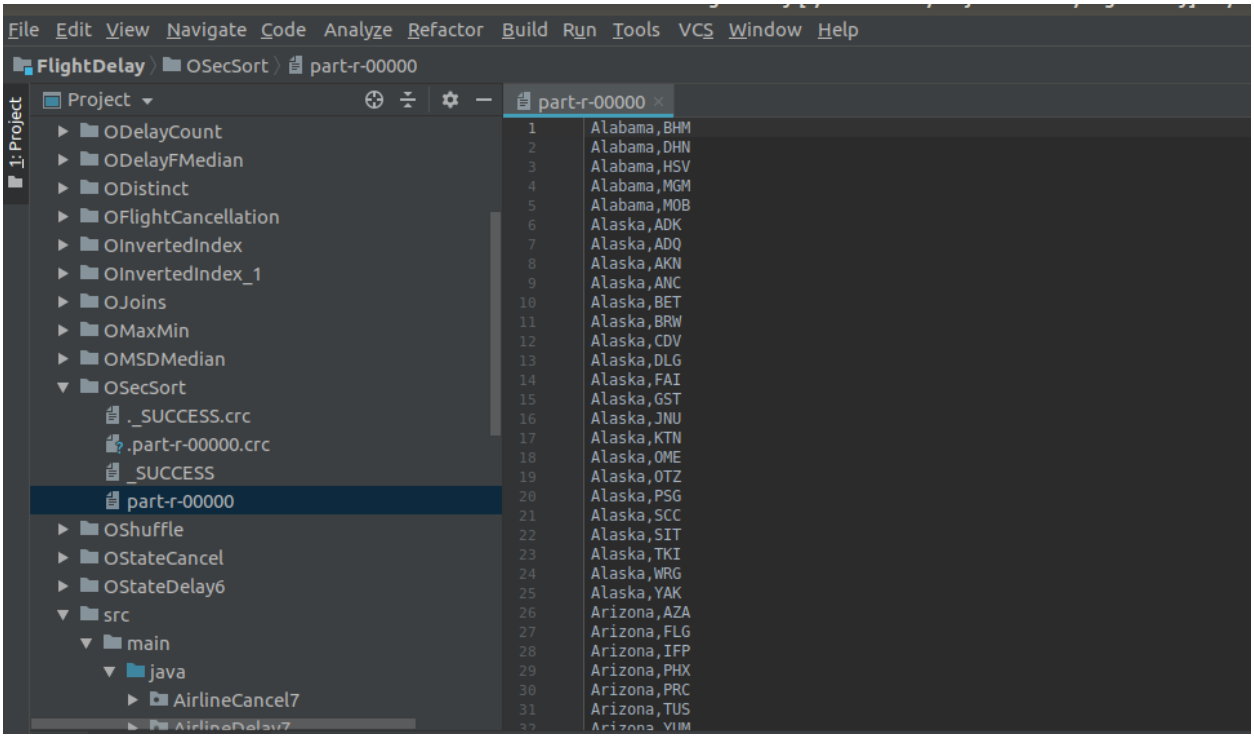
Based on Airline Min and Max distance travelled:



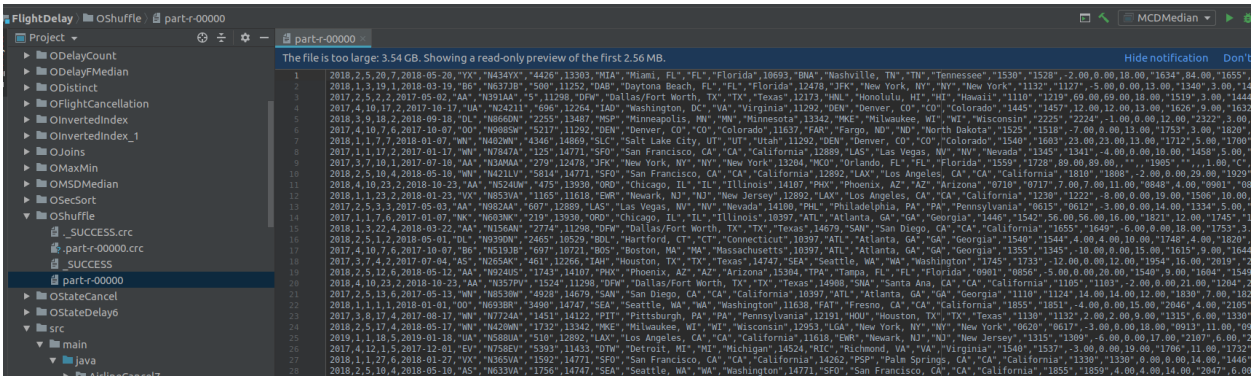
MCD Median of Distance:



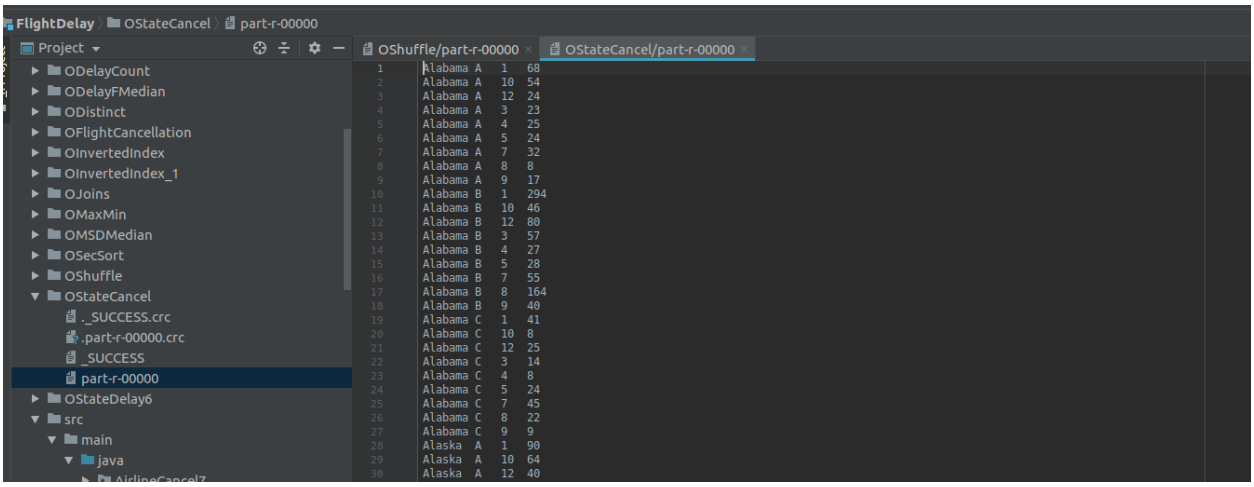
Sec sort of all State and Airports in state:



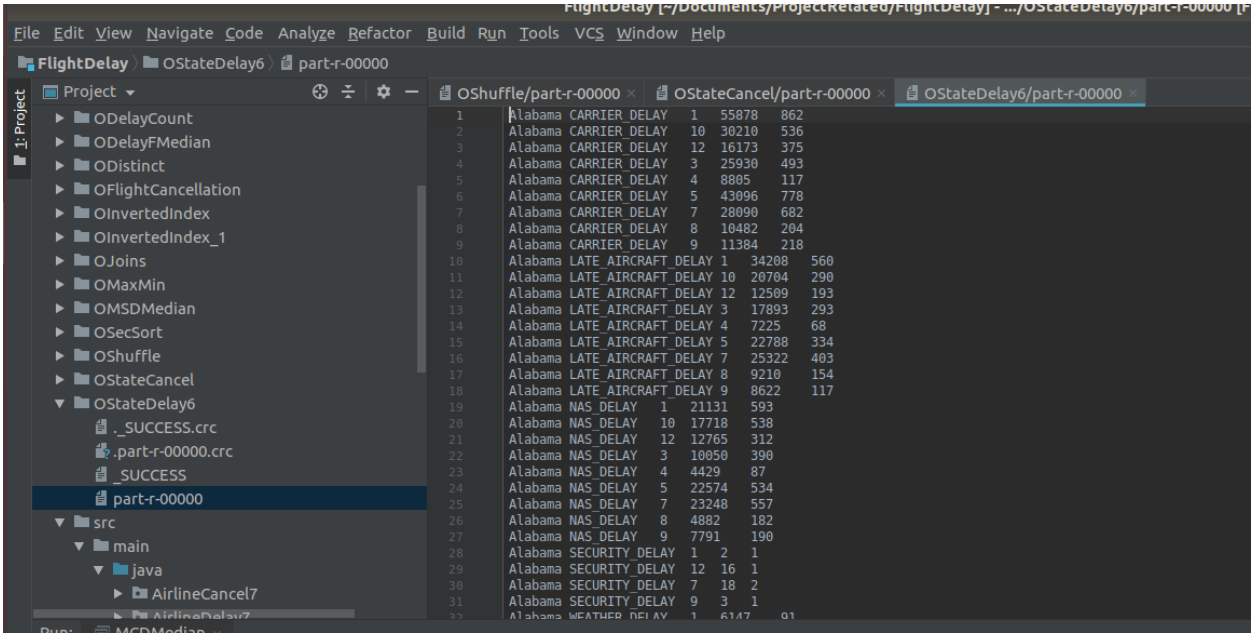
Shuffle Data:



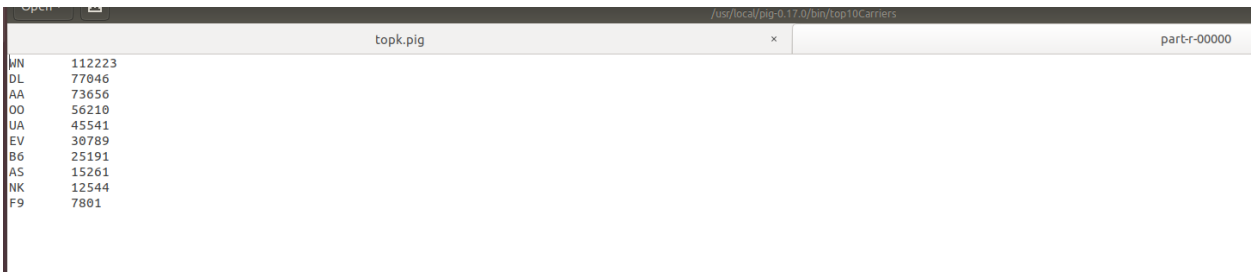
Statewise Cancelled Flights Monthly count:



Statewise delayed flights:



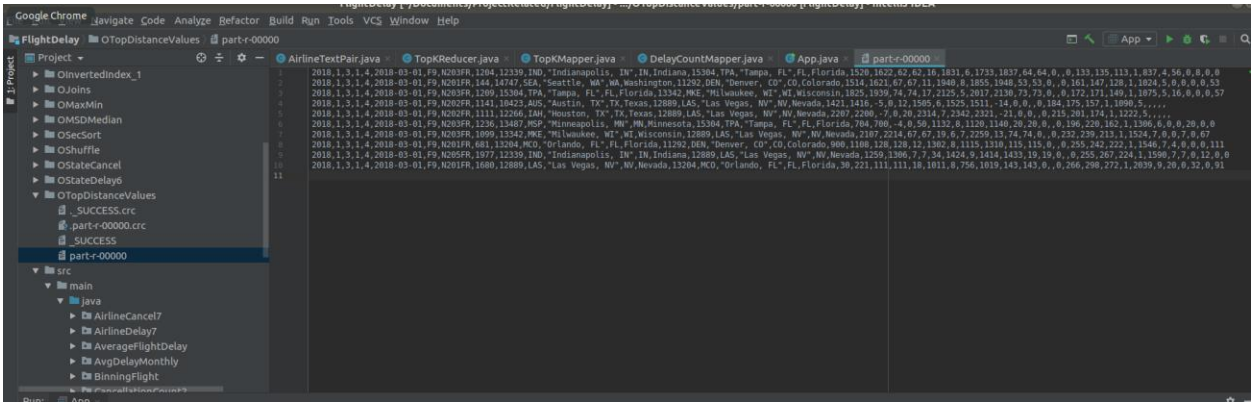
Top10 airline with most flights Pig:



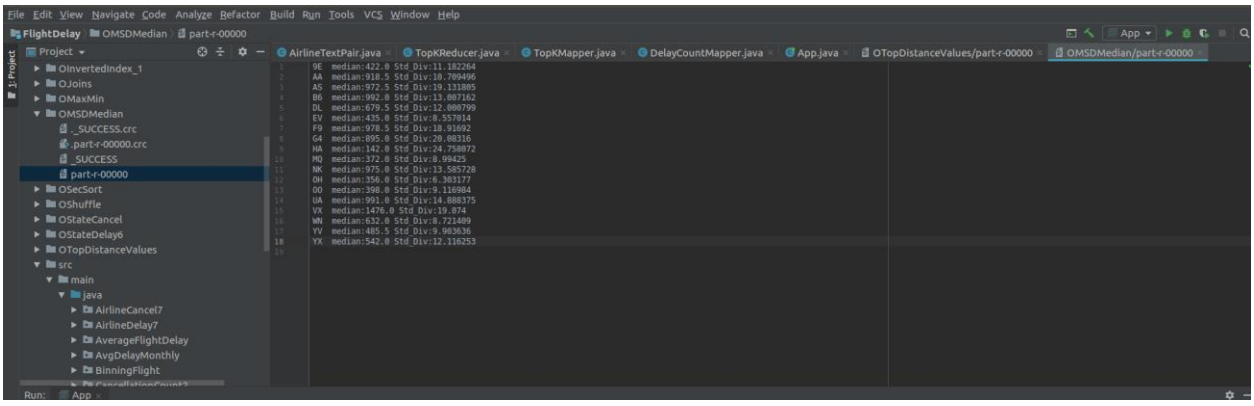
topk.pig

```
flights = LOAD 'apr17.csv' using PigStorage(',');
grouped = GROUP flights BY $6;
summed = FOREACH grouped GENERATE group, COUNT(flights) AS cntd;
sorted = ORDER summed BY cntd DESC;
top25 = LIMIT sorted 10;
Dump top25;
STORE top25 into 'top10Carriers';
```

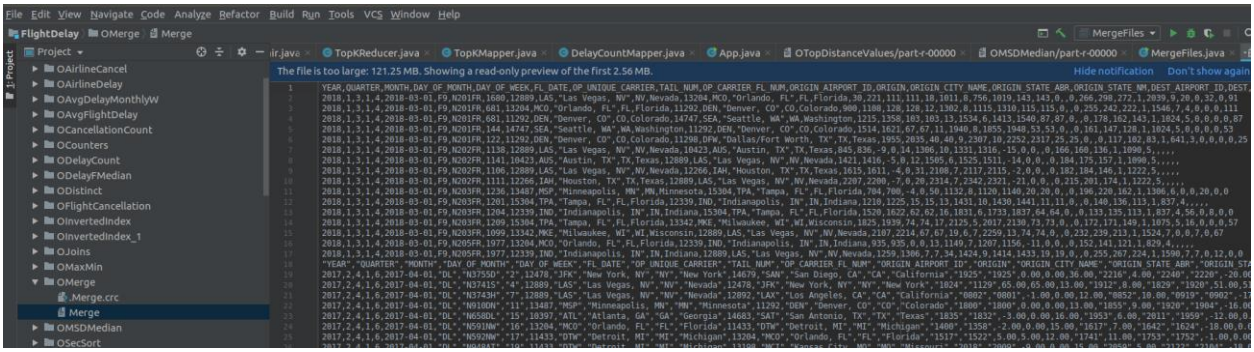
Top Distance values:



MC Median of Airliner distance:

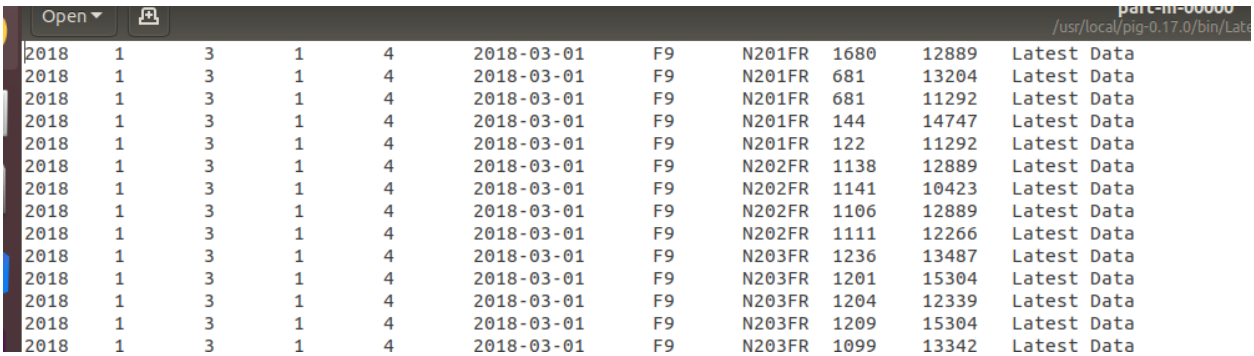


Merge: Example with small file



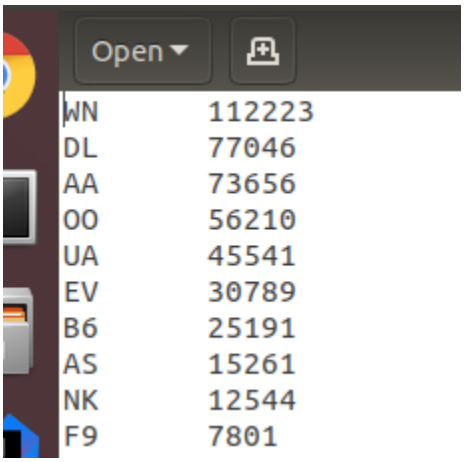
Pig:

Latest Data:



2018	1	3	1	4	2018-03-01	F9	N201FR	1680	12889	Latest Data
2018	1	3	1	4	2018-03-01	F9	N201FR	681	13204	Latest Data
2018	1	3	1	4	2018-03-01	F9	N201FR	681	11292	Latest Data
2018	1	3	1	4	2018-03-01	F9	N201FR	144	14747	Latest Data
2018	1	3	1	4	2018-03-01	F9	N201FR	122	11292	Latest Data
2018	1	3	1	4	2018-03-01	F9	N202FR	1138	12889	Latest Data
2018	1	3	1	4	2018-03-01	F9	N202FR	1141	10423	Latest Data
2018	1	3	1	4	2018-03-01	F9	N202FR	1106	12889	Latest Data
2018	1	3	1	4	2018-03-01	F9	N202FR	1111	12266	Latest Data
2018	1	3	1	4	2018-03-01	F9	N203FR	1236	13487	Latest Data
2018	1	3	1	4	2018-03-01	F9	N203FR	1201	15304	Latest Data
2018	1	3	1	4	2018-03-01	F9	N203FR	1204	12339	Latest Data
2018	1	3	1	4	2018-03-01	F9	N203FR	1209	15304	Latest Data
2018	1	3	1	4	2018-03-01	F9	N203FR	1099	13342	Latest Data

Pig Top 10:



WN	112223
DL	77046
AA	73656
OO	56210
UA	45541
EV	30789
B6	25191
AS	15261
NK	12544
F9	7801