## INFO 7275 Advance Database Management Systems

**Airline On-Time statistics and arrival data analysis**

**Introduction:**

According to the U.S Bureau of Transportation Statistics, more than 1 million flights were delayed between June 2015 to June 2016. This translates to nearly 64 million minutes’ worth of delay.  
Considering all the delays and its types, it becomes necessary to consider the efficient way of travel.

The aim of this project will be to use various Map Reduce design patterns like Summarization design pattern, Data Organization Patterns, Join Patterns and Filtering Patterns to graphically represent the following questions

* Best time to fly to minimize delays
* Main factors of delays
* Best airlines to avoid delays
* Top N busiest airports

**Dataset:**

The data set consists of on-time arrival for domestic flights by major air carriers and provides additional items as departure and arrival delays, origin and destination airports, flight numbers, schedule and actual departure and arrival times, cancelled and diverted flights, taxi-out and taxi-in times.

Other supplement data:

carriers.csv   
airports.csv  
plane-data.csv

Links to the data set:

<https://www.transtats.bts.gov/Fields.asp?Table_ID=236>

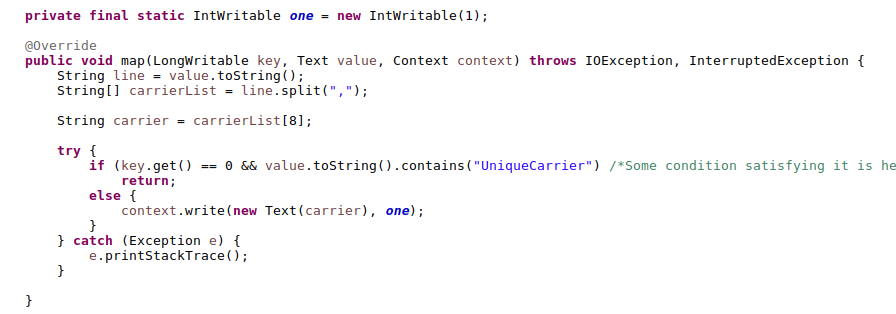
[http://stat-computing.org/dataexpo/2009/the-data.html](https://www.transtats.bts.gov/Fields.asp?Table_ID=236)

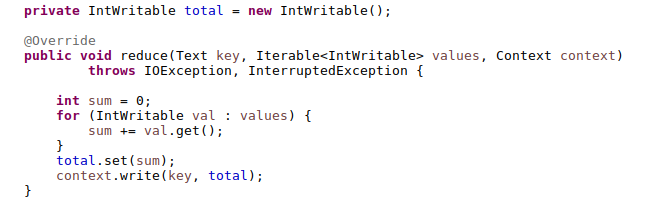
[https://www.transtats.bts.gov/TableInfo.asp](https://www.transtats.bts.gov/Fields.asp?Table_ID=236)

**Analysis performed:**

1. Total number of flights in a year

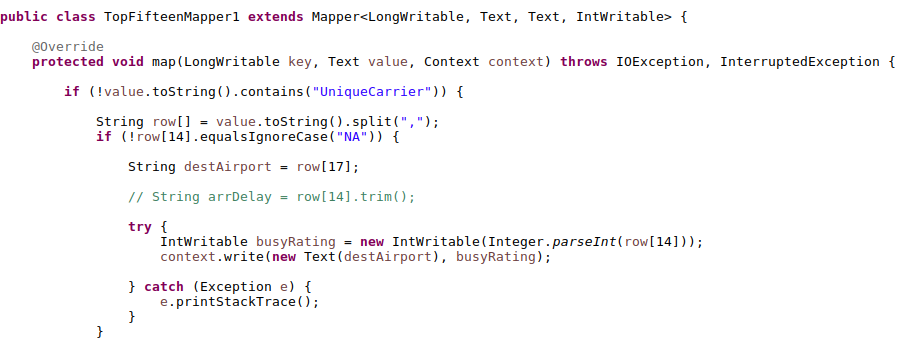
Using Summarization Pattern and calculating the total occurrence of each Unique Carrier, we can find out the total number of flights in year.

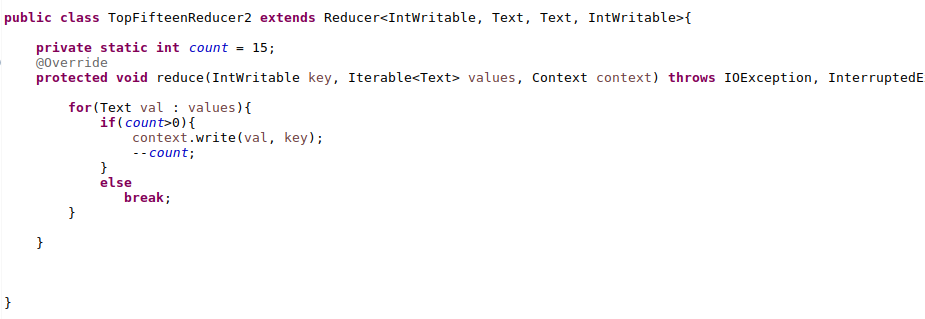


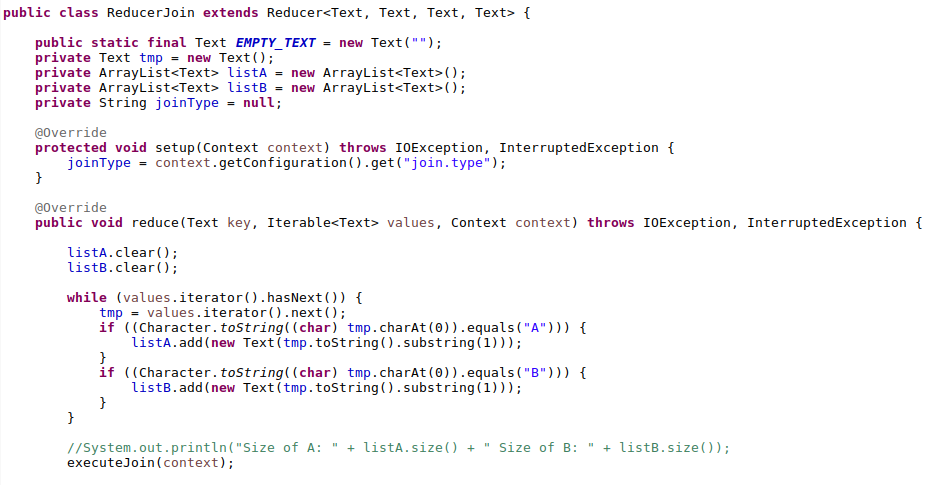


1. Top K busiest airports with respect to delay

Using Chaining of MR jobs, calculating the top K airports with the arrival delay of each airport and summarizing it.  
Using Join Pattern with the data dictionary of airports displaying the top airports with their name and address

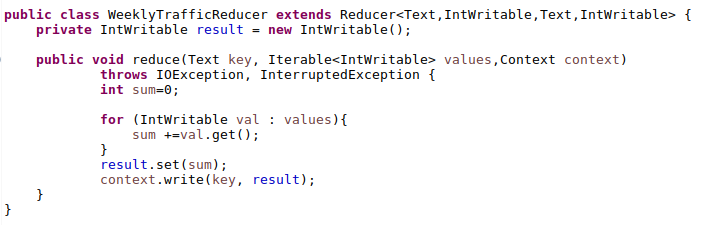






1. Weekly Analysis for the best time to fly

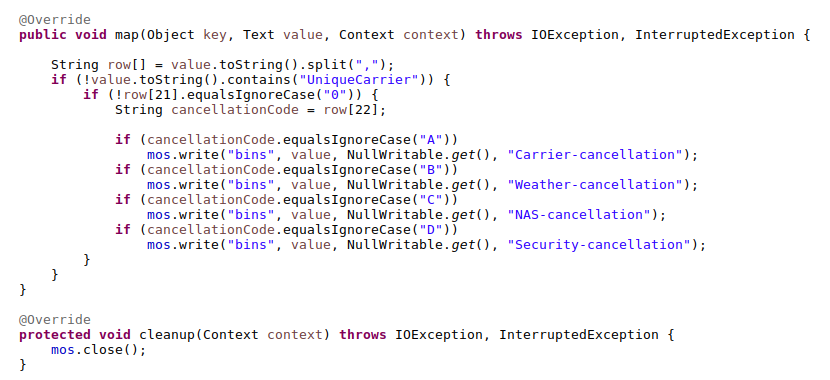
Using Partitioning Data Organization Pattern, partitioned the data based on the day of the week and calculating the weekly traffic for each day



1. Cancellation Reason

Using Binning Data Organization Pattern, binning the data for each of the four-cancellation reason and separating them out.

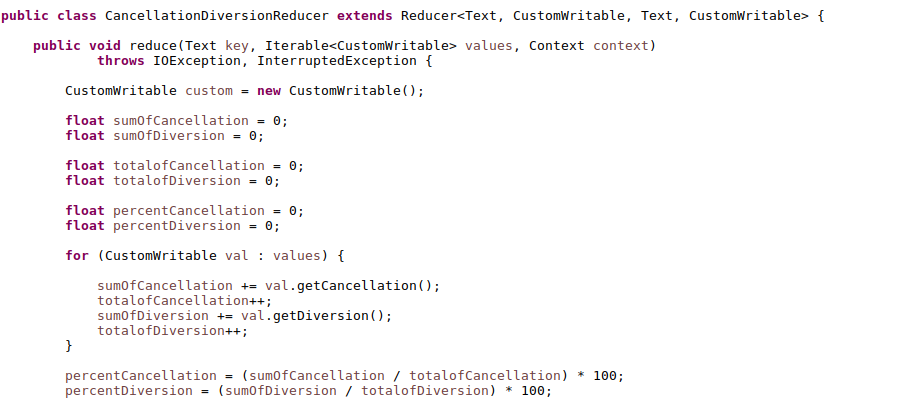
Reason for cancellation (A = carrier, B = weather, C = NAS, D = security)



1. Best Airlines to avoid cancellation and diversion

Using a custom writable class and summarization pattern, calculating the cancellation and diversion rate for each unique carrier, we found out the best airlines to avoid arrival/departure delays

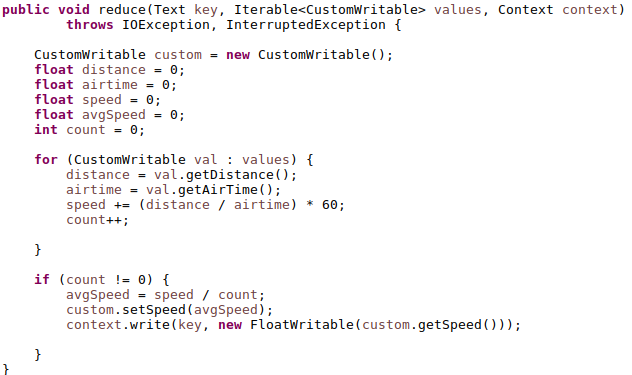




|  |  |
| --- | --- |
|  |  |

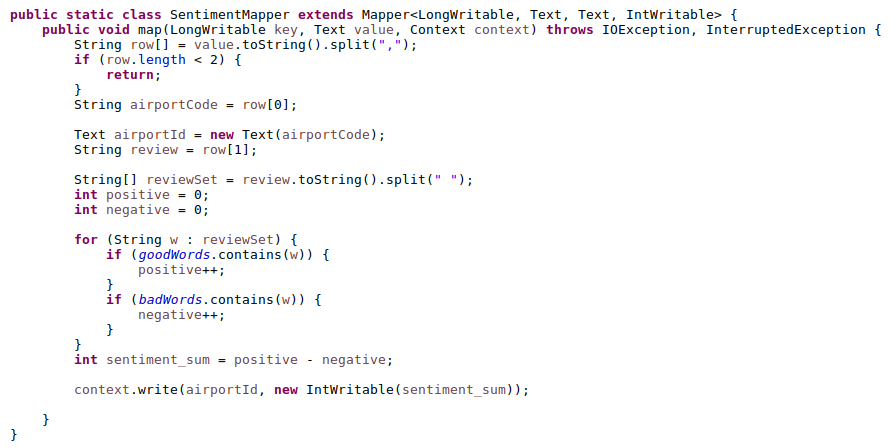
1. Average speed for the airlines

Using Summarization pattern along with custom writable class, calculating the mean speed of each airlines and comparing

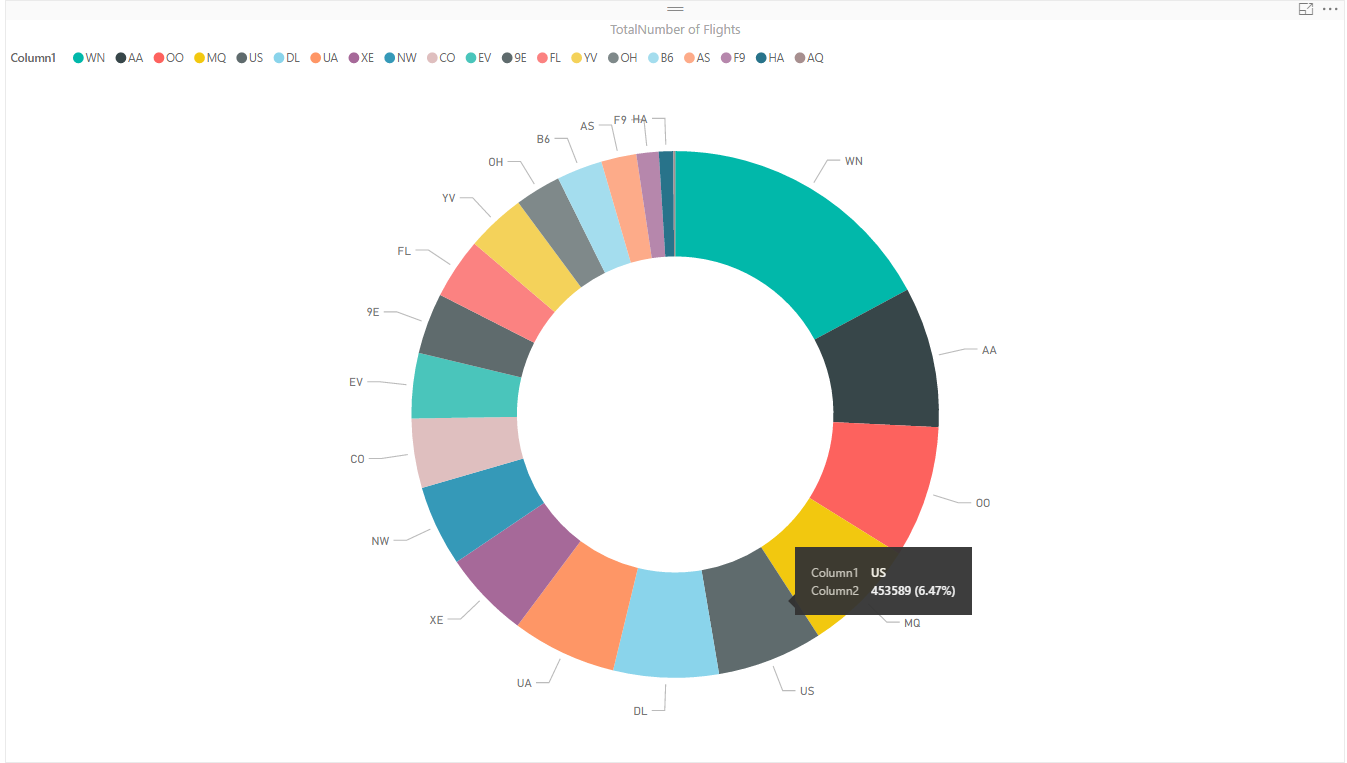


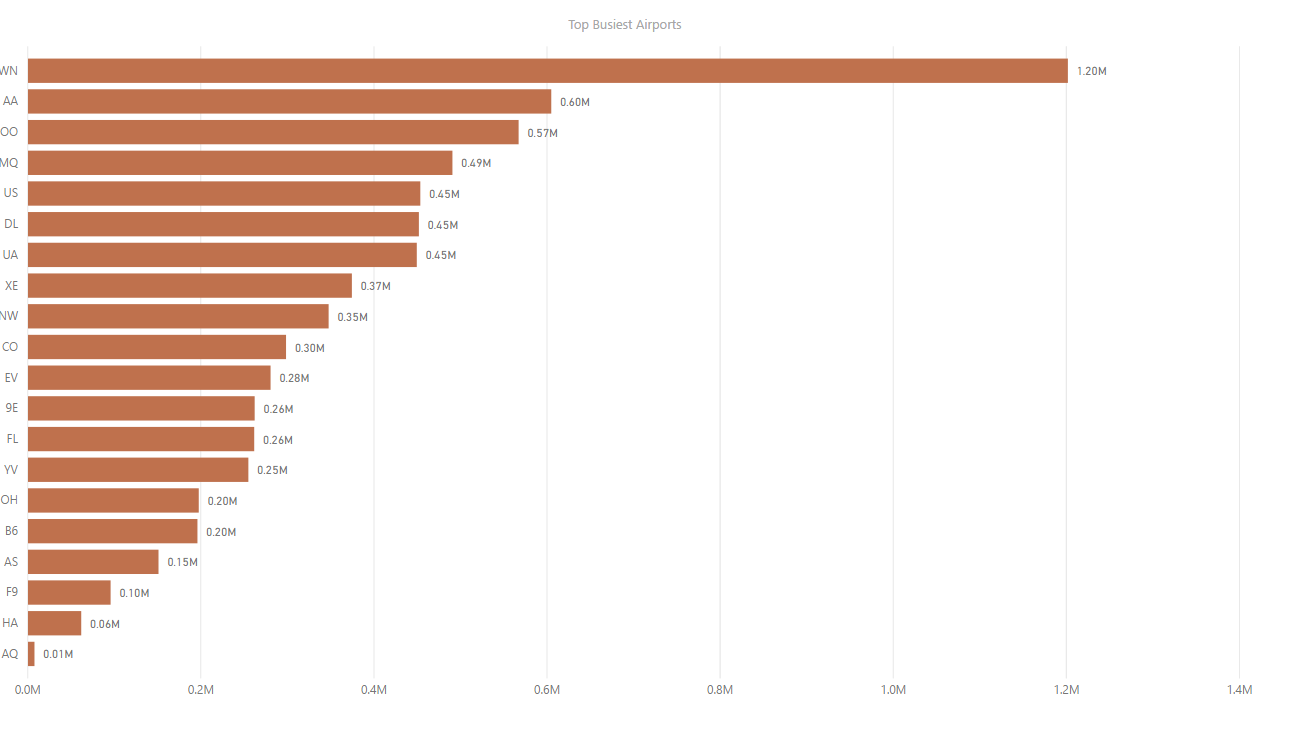
1. Airport Reviews

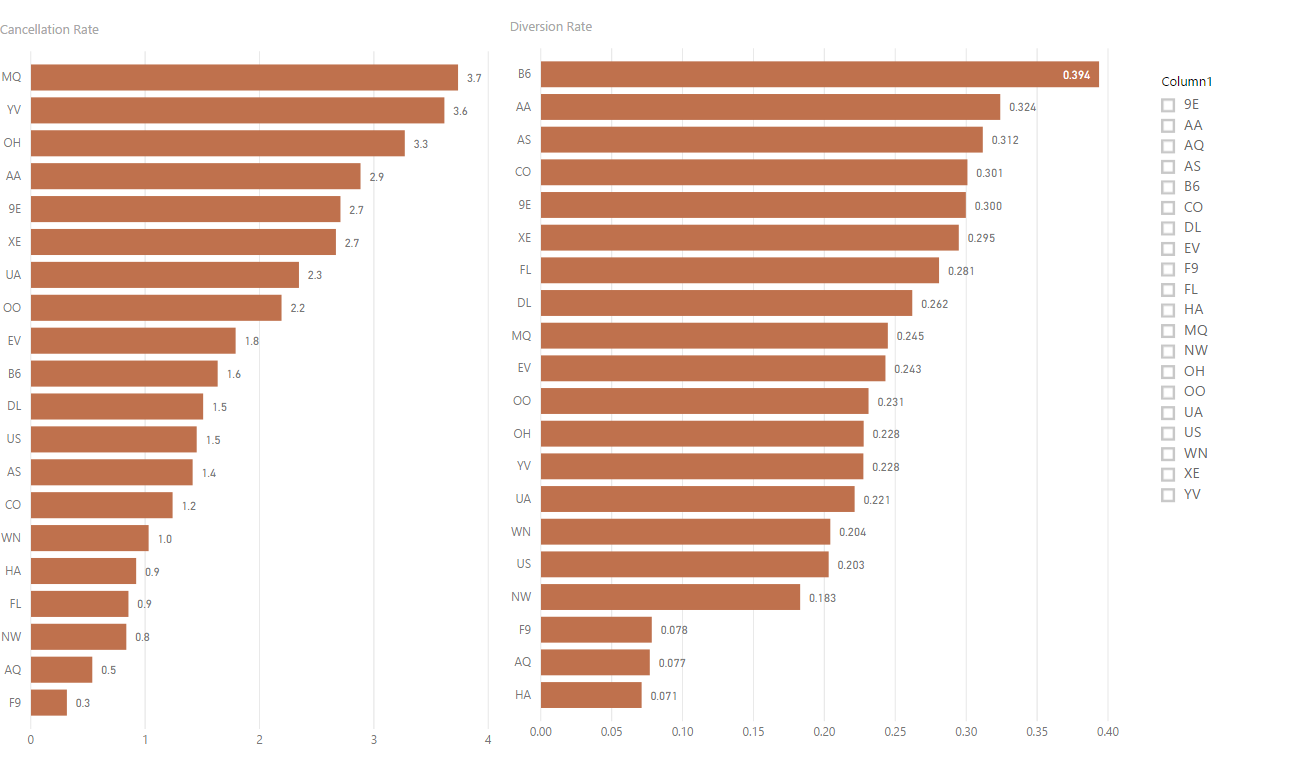
Using Sentiment Analysis, finding the type of review for each airport

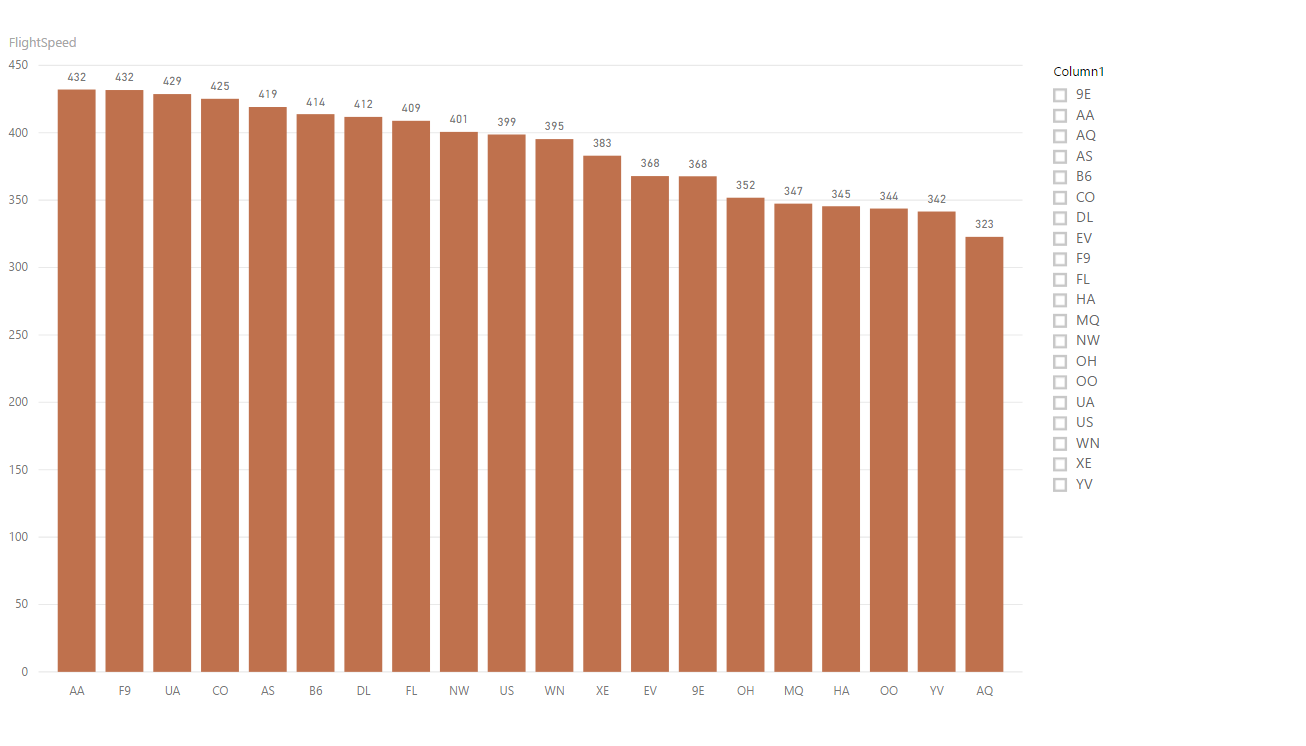


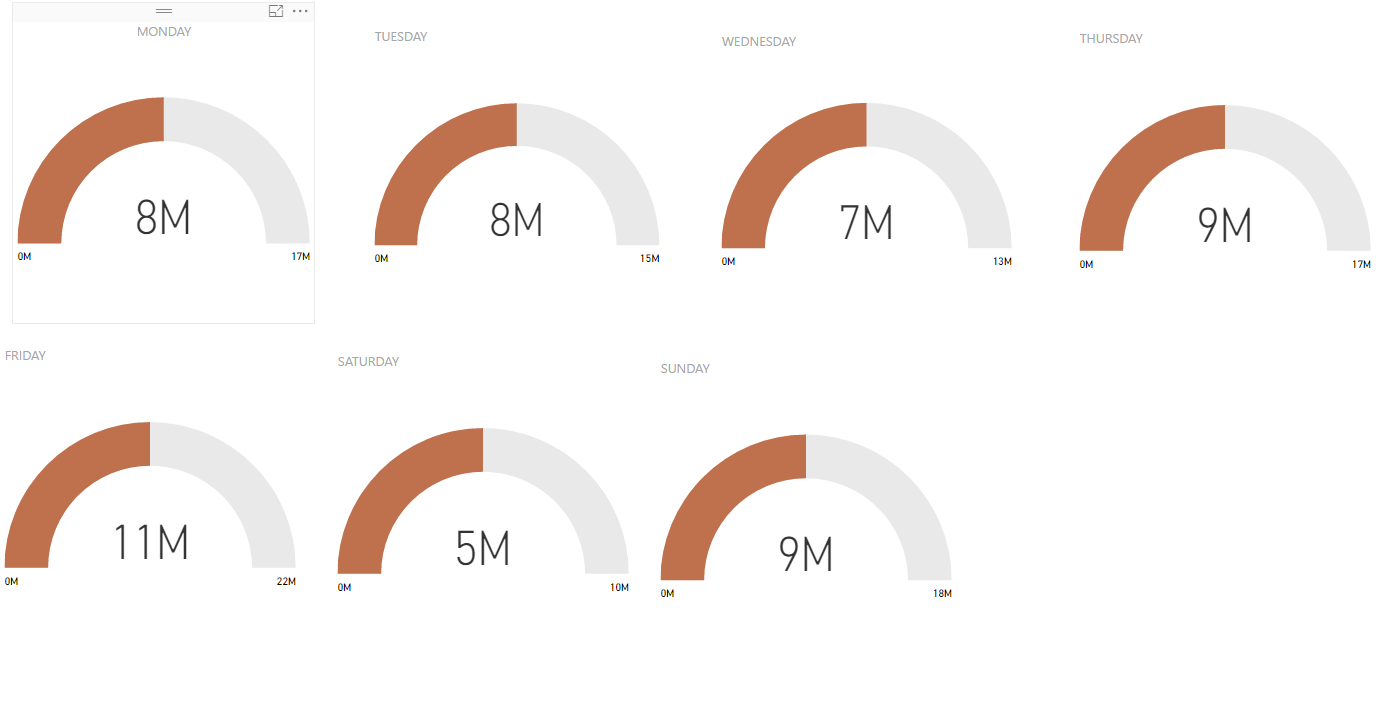
Dashboards for Visualization







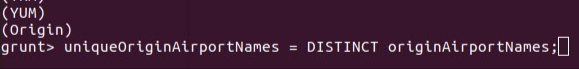




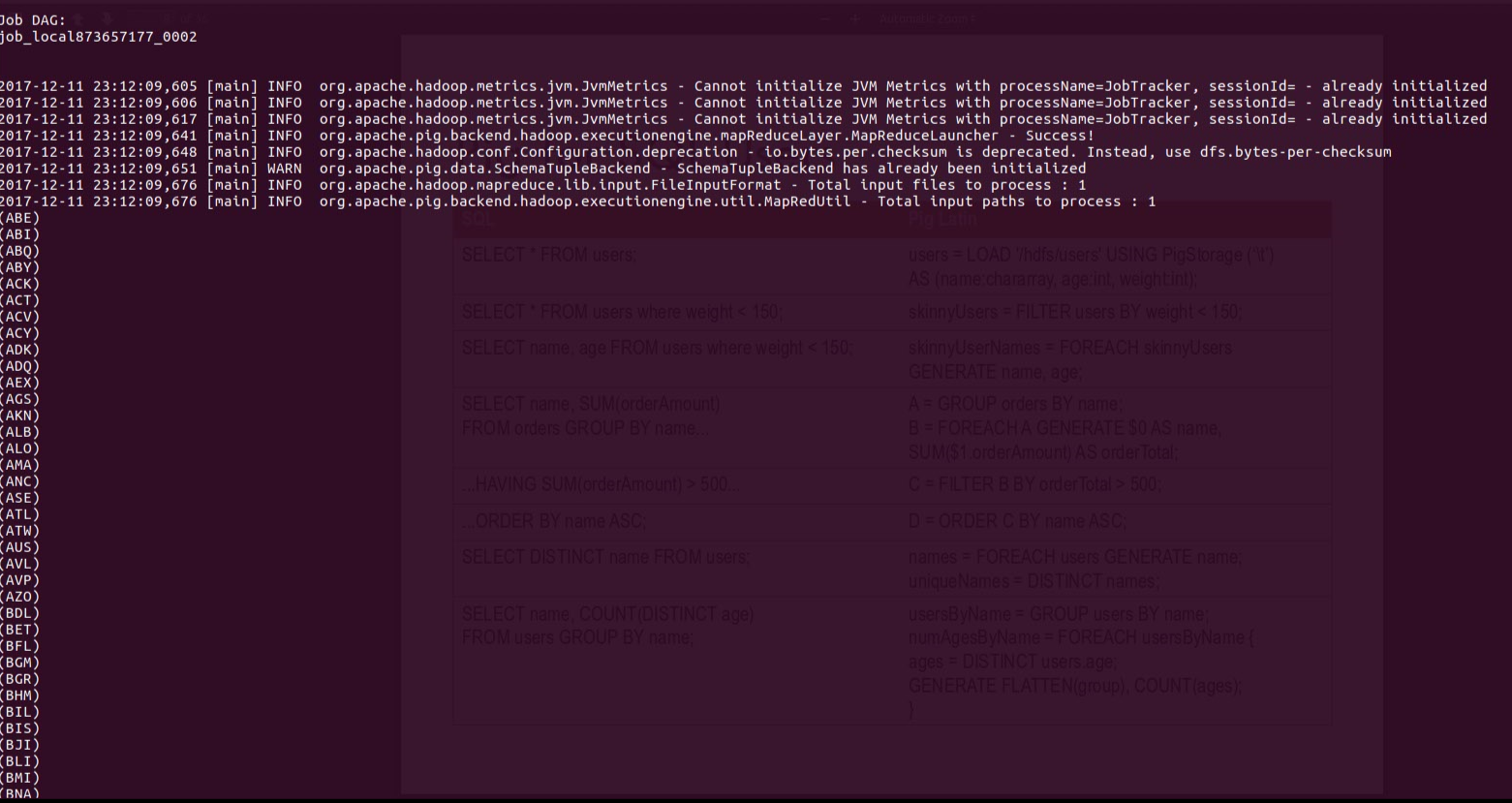
NoSQL Databases

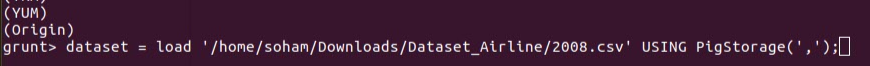
Pig

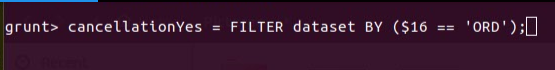


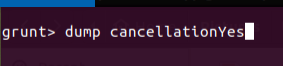


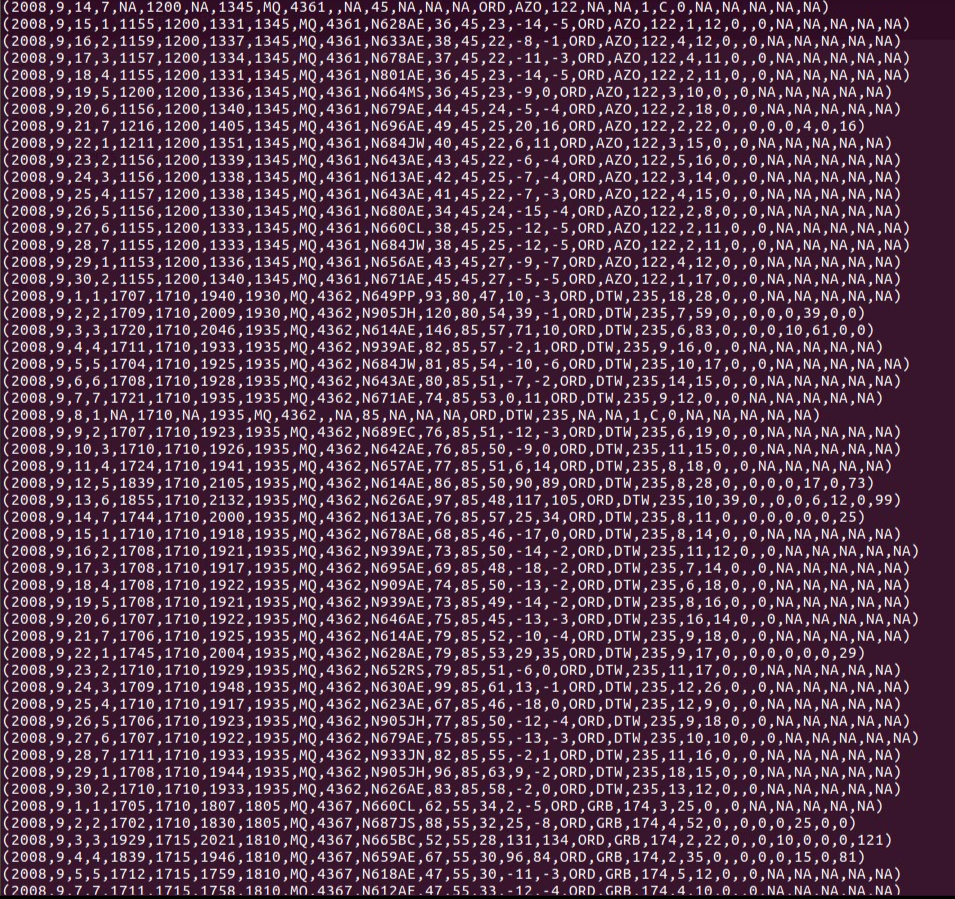












HIVE



APPENDIX

1.

package partitioning.weeklyAnalysis;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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

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

public class Driver {

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Weekly Partitioning count");

job.setJarByClass(Driver.class);

job.setMapperClass(WeeklyTrafficMapper.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setPartitionerClass(WeeklyPartitioner.class);

job.setNumReduceTasks(7);

job.setCombinerClass(WeeklyTrafficReducer.class);

job.setReducerClass(WeeklyTrafficReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}

}

package partitioning.weeklyAnalysis;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Partitioner;

public class WeeklyPartitioner extends Partitioner<Text, IntWritable> {

@Override

public int getPartition(Text key, IntWritable value, int numOfPartitions) {

int week = Integer.parseInt(key.toString());

return (week % numOfPartitions);

}

}

package partitioning.weeklyAnalysis;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class WeeklyTrafficMapper extends Mapper<Object, Text, Text, IntWritable> {

private Text dayOfTheWeek = new Text();

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

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

if (!value.toString().contains("DayOfWeek")) {

if (!row[14].equalsIgnoreCase("NA")) {

dayOfTheWeek.set(row[3]);

context.write(dayOfTheWeek, new IntWritable(Integer.parseInt(row[14])));

}

}

}

}

package partitioning.weeklyAnalysis;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,Context context)

throws IOException, InterruptedException {

int sum=0;

for (IntWritable val : values){

sum +=val.get();

}

result.set(sum);

context.write(key, result);

}

}

2.

package TestMRJob.testMR;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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

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

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

public class Driver {

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

if (args.length != 2) {

System.err.println("Usage: MaxSubmittedCharge <input path> <output path>");

System.exit(-1);

}

Path inputPath = new Path(args[0]);

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

// configuration

Configuration conf = new Configuration(true);

// job

Job job = new Job(conf, "SecondarySort");

job.setJarByClass(TestMapper.class);

// mapreduce

job.setMapperClass(TestMapper.class);

job.setReducerClass(TestReducer.class);

job.setNumReduceTasks(1);

// key value

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

// input

FileInputFormat.addInputPath(job, inputPath);

job.setInputFormatClass(TextInputFormat.class);

// output

FileOutputFormat.setOutputPath(job, outputDir);

// delete output if exist

FileSystem hdfs = FileSystem.get(conf);

if (hdfs.exists(outputDir))

hdfs.delete(outputDir, true);

// Execute job

int code = job.waitForCompletion(true) ? 0 : 1;

System.exit(code);

}

}

package TestMRJob.testMR;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

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

@Override

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

String line = value.toString();

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

String carrier = carrierList[8];

try {

if (key.get() == 0 && value.toString().contains("UniqueCarrier") /\*Some condition satisfying it is header\*/)

return;

else {

context.write(new Text(carrier), one);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

package TestMRJob.testMR;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

private IntWritable total = new IntWritable();

@Override

public void reduce(Text key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

total.set(sum);

context.write(key, total);

}

}

3.

package sentimental.reviewAnalysis;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.HashSet;

import java.util.Set;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

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.TextInputFormat;

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

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

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

public class SentimentAnalysis extends Configured implements Tool {

public static Set<String> goodWords = new HashSet<String>();

public static Set<String> badWords = new HashSet<String>();

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

if (args.length != 4) {

System.err.println("Usage: Sentiment Analysis <positive> <negative> <input path> <output path>");

System.exit(-1);

}

ToolRunner.run(new SentimentAnalysis(), args);

}

public int run(String[] args) throws Exception {

listOfPositiveWords(args[0]); // Path to positive words file

listOfNegativeWords(args[1]); // Path to negative words file

Path inputpath = new Path(args[2]);

Path final\_output = new Path(args[3]);

Configuration conf = new Configuration(true);

Job job = new Job(conf, "SentimentAnalysis");

job.setJarByClass(SentimentAnalysis.class);

job.setMapperClass(SentimentMapper.class);

job.setReducerClass(SentimentReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, inputpath);

FileOutputFormat.setOutputPath(job, final\_output);

FileSystem hdfs = FileSystem.get(conf);

if (hdfs.exists(final\_output))

hdfs.delete(final\_output, true);

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

return 0;

}

private void listOfPositiveWords(String p) {

try {

BufferedReader fis = new BufferedReader(new FileReader(new File(p)));

String word;

while ((word = fis.readLine()) != null) {

goodWords.add(word);

}

fis.close();

} catch (IOException ioe) {

System.err.println("Caught exception..File not found");

}

}

private void listOfNegativeWords(String p) {

try {

BufferedReader fis = new BufferedReader(new FileReader(new File(p)));

String word;

while ((word = fis.readLine()) != null) {

badWords.add(word);

}

fis.close();

} catch (IOException ioe) {

System.err.println("Caught exception..File not found");

}

}

public static class SentimentMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

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

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

if (row.length < 2) {

return;

}

String airportCode = row[0];

Text airportId = new Text(airportCode);

String review = row[1];

String[] reviewSet = review.toString().split(" ");

int positive = 0;

int negative = 0;

for (String w : reviewSet) {

if (goodWords.contains(w)) {

positive++;

}

if (badWords.contains(w)) {

negative++;

}

}

int sentiment\_sum = positive - negative;

context.write(airportId, new IntWritable(sentiment\_sum));

}

}

public static class SentimentReducer extends Reducer<Text, IntWritable, Text, Text> {

public void reduce(Text key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

int totalsentiment = 0;

Text result = new Text("NEUTRAL");

for (IntWritable val : values) {

totalsentiment += val.get();

}

if (totalsentiment > 0) {

result.set("POSITIVE");

} else if (totalsentiment < 0) {

result.set("NEGATIVE");

}

context.write(key, result);

}

}

}

4.

package speed.flightSpeed;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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

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

public class Driver {

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Flight Speed");

job.setJarByClass(Driver.class);

job.setMapperClass(FlightSpeedMapper.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(CustomWritable.class);

job.setReducerClass(FlightSpeedReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(FloatWritable.class);

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

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

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

}

}

package speed.flightSpeed;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class FlightSpeedMapper extends Mapper<Object, Text, Text, CustomWritable> {

@Override

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

CustomWritable flightSpeed = new CustomWritable();

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

String carrier;

if (!value.toString().contains("DayOfWeek")) {

carrier = row[8];

if (!row[18].equalsIgnoreCase("NA") && !row[13].equalsIgnoreCase("NA") && !row[18].equalsIgnoreCase("0") && !row[13].equalsIgnoreCase("0") ) {

flightSpeed.setDistance(Float.parseFloat(row[18]));

flightSpeed.setAirTime(Float.parseFloat(row[13]));

context.write(new Text(carrier), flightSpeed);

}

}

}

}

5.

package cancellationReason.cancellationReason;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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

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

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

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

public class Driver {

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

Configuration conf = new Configuration();

if (args.length != 2) {

System.err.println("Usage: Binning <input> <output>");

System.exit(2);

}

Path inputPath = new Path(args[0]);

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

Job job = new Job(conf, "Binning");

job.setJarByClass(BinningMapper.class);

job.setMapperClass(BinningMapper.class);

job.setNumReduceTasks(0);

TextInputFormat.setInputPaths(job, inputPath);

FileOutputFormat.setOutputPath(job, outputDir);

MultipleOutputs.addNamedOutput(job, "bins", TextOutputFormat.class, Text.class, NullWritable.class);

MultipleOutputs.setCountersEnabled(job, true);

FileSystem hdfs = FileSystem.get(conf);

if (hdfs.exists(outputDir))

hdfs.delete(outputDir, true);

// Execute job

int code = job.waitForCompletion(true) ? 0 : 1;

System.exit(code);

}

}

package cancellationReason.cancellationReason;

import java.io.IOException;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

public class BinningMapper 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 (!value.toString().contains("UniqueCarrier")) {

if (!row[21].equalsIgnoreCase("0")) {

String cancellationCode = row[22];

if (cancellationCode.equalsIgnoreCase("A"))

mos.write("bins", value, NullWritable.get(), "Carrier-cancellation");

if (cancellationCode.equalsIgnoreCase("B"))

mos.write("bins", value, NullWritable.get(), "Weather-cancellation");

if (cancellationCode.equalsIgnoreCase("C"))

mos.write("bins", value, NullWritable.get(), "NAS-cancellation");

if (cancellationCode.equalsIgnoreCase("D"))

mos.write("bins", value, NullWritable.get(), "Security-cancellation");

}

}

}

@Override

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

mos.close();

}

}

6.

package customWritable.cancellationAndDiversionRate;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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

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

import java.io.IOException;

public class Driver {

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Cancellation and Diversion Rate");

job.setJarByClass(Driver.class);

job.setMapperClass(CancellationDiversionMapper.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(CustomWritable.class);

job.setReducerClass(CancellationDiversionReducer.class);

//job.setOutputKeyClass(CustomWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(CustomWritable.class);

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

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

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

}

}

package customWritable.cancellationAndDiversionRate;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class CustomWritable implements Writable {

private float cancellation;

private float diversion;

public CustomWritable() {

}

public CustomWritable(float cancellation, float diversion) {

this.cancellation = cancellation;

this.diversion = diversion;

}

public float getCancellation() {

return cancellation;

}

public float getDiversion() {

return diversion;

}

public void setCancellation(float cancellation) {

this.cancellation = cancellation;

}

public void setDiversion(float diversion) {

this.diversion = diversion;

}

public void readFields(DataInput in) throws IOException {

cancellation = in.readFloat();

diversion = in.readFloat();

}

public void write(DataOutput out) throws IOException {

out.writeFloat(cancellation);

out.writeFloat(diversion);

}

@Override

public String toString() {

return (cancellation + " " + diversion);

}

}

package customWritable.cancellationAndDiversionRate;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

public void reduce(Text key, Iterable<CustomWritable> values, Context context)

throws IOException, InterruptedException {

CustomWritable custom = new CustomWritable();

float sumOfCancellation = 0;

float sumOfDiversion = 0;

float totalofCancellation = 0;

float totalofDiversion = 0;

float percentCancellation = 0;

float percentDiversion = 0;

for (CustomWritable val : values) {

sumOfCancellation += val.getCancellation();

totalofCancellation++;

sumOfDiversion += val.getDiversion();

totalofDiversion++;

}

percentCancellation = (sumOfCancellation / totalofCancellation) \* 100;

percentDiversion = (sumOfDiversion / totalofDiversion) \* 100;

// System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// System.out.println("TOTAL CANCELLATION " + sumOfCancellation);

// System.out.println("TOTAL DIVERSION " + totalofCancellation);

// System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

custom.setCancellation(percentCancellation);

custom.setDiversion(percentDiversion);

context.write(key, custom);

}

}

package customWritable.cancellationAndDiversionRate;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class CancellationDiversionMapper extends Mapper<Object, Text, Text, CustomWritable> {

@Override

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

CustomWritable cancelAndDivert = new CustomWritable();

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

String carrier;

if (!value.toString().contains("DayOfWeek")) {

carrier = row[8];

if (!row[21].equalsIgnoreCase("NA") && !row[23].equalsIgnoreCase("NA")) {

cancelAndDivert.setCancellation(Float.parseFloat(row[21]));

cancelAndDivert.setDiversion(Float.parseFloat(row[23]));

context.write(new Text(carrier), cancelAndDivert);

}

}

}

}

7.

package topK.airports;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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

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

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 {

// TODO code application logic here

Configuration conf = new Configuration();

Job job1 = Job.getInstance(conf, "Chaining");

job1.setJarByClass(Driver.class);

job1.setMapperClass(TopFifteenMapper1.class);

job1.setMapOutputKeyClass(Text.class);

job1.setMapOutputValueClass(IntWritable.class);

job1.setReducerClass(TopFifteenReducer1.class);

job1.setOutputKeyClass(Text.class);

job1.setOutputValueClass(IntWritable.class);

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

job1.setInputFormatClass(TextInputFormat.class);

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

//boolean complete = job1.waitForCompletion(true);

boolean firstComplete = job1.waitForCompletion(true);

boolean secondComplete=false;

if (firstComplete) {

Configuration conf2 = new Configuration();

Job job2 = Job.getInstance(conf, "Chaining");

job2.setJarByClass(Driver.class);

job2.setMapperClass(TopFifteenMapper2.class);

job2.setMapOutputKeyClass(IntWritable.class);

job2.setMapOutputValueClass(Text.class);

job2.setSortComparatorClass(Comparator.class);

job2.setReducerClass(TopFifteenReducer2.class);

job2.setOutputKeyClass(Text.class);

job2.setOutputValueClass(IntWritable.class);

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

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

secondComplete=job2.waitForCompletion(true);

}

Configuration conf3 = new Configuration();

Job job3 = Job.getInstance(conf3, "Join");

if(secondComplete){

job3.setJarByClass(Driver.class);

job3.setMapperClass(DelayRatingMapper.class);

job3.setMapperClass(AirportMapper.class);

job3.setMapOutputKeyClass(Text.class);

job3.setMapOutputValueClass(Text.class);

job3.setReducerClass(ReducerJoin.class);

MultipleInputs.addInputPath(job3, new Path(args[2]), TextInputFormat.class, DelayRatingMapper.class);

MultipleInputs.addInputPath(job3, new Path(args[3]), TextInputFormat.class, AirportMapper.class);

job3.getConfiguration().set("join.type", "inner");

job3.setOutputKeyClass(Text.class);

job3.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job3, new Path(args[2]));

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

System.exit(job3.waitForCompletion(true)? 0 :1);

}

}

}

package topK.airports;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class TopFifteenMapper1 extends Mapper<LongWritable, Text, Text, IntWritable> {

@Override

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

if (!value.toString().contains("UniqueCarrier")) {

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

if (!row[14].equalsIgnoreCase("NA")) {

String destAirport = row[17];

// String arrDelay = row[14].trim();

try {

IntWritable busyRating = new IntWritable(Integer.parseInt(row[14]));

context.write(new Text(destAirport), busyRating);

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

}

package topK.airports;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class TopFifteenReducer1 extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable result = new IntWritable();

protected void reduce(Text key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

//super.reduce(key, values, context); //To change body of generated methods, choose Tools | Templates.

//int count=0;

int sum=0;

//double avg=0;

for(IntWritable val : values){

sum += val.get();

//++count;

}

//avg=sum/count;

result.set(sum);

context.write(key, result);

}

}

package topK.airports;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class TopFifteenMapper2 extends Mapper<LongWritable, Text, IntWritable, Text>{

//private final static IntWritable sales = new IntWritable();

//private Text employee = new Text();

@Override

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

//super.map(key, value, context); //To change body of generated methods, choose Tools | Templates.

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

//IntWritable movieId = new IntWritable(Integer.parseInt(row[0]));

//Text airport = new Text(row[0]);

String rating = row[1].trim();

try {

IntWritable rating\_int = new IntWritable(Integer.parseInt(rating));

context.write(rating\_int, new Text(row[0]));

} catch (Exception e) {

e.printStackTrace();

}

}

}

package topK.airports;

import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class TopFifteenReducer2 extends Reducer<IntWritable, Text, Text, IntWritable>{

private static int count = 15;

@Override

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

for(Text val : values){

if(count>0){

context.write(val, key);

--count;

}

else

break;

}

}

}

package topK.airports;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class DelayRatingMapper extends Mapper<Object, Text, Text, Text> {

private Text outkey = new Text();

private Text outValue = new Text();

@Override

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

String airportList[] = value.toString().split("\t");

String airport = airportList[0].replace("\"", "");

if (airport != null) {

// System.out.println("Airport name ->" + airport);

outkey.set(airport);

outValue.set("A" + value.toString().replace("\"", ""));

context.write(outkey, outValue);

}

}

}

package topK.airports;

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class AirportMapper extends Mapper<Object, Text, Text, Text>{

private Text outkey= new Text();

private Text outValue= new Text();

@Override

public void map(Object key, Text value, Context context)

throws IOException, InterruptedException {

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

if (!value.toString().contains("iata")) {

String iata = aiportList[0].replace("\"", "");

//System.out.println("Airport name ->" + iata);

if(iata!=null){

outkey.set(iata);

outValue.set("B"+ value.toString().replace("\"", "").replace(",", " "));

context.write(outkey, outValue);

}

}

}

}

package topK.airports;

import java.io.IOException;

import java.util.ArrayList;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

public static final Text EMPTY\_TEXT = new Text("");

private Text tmp = new Text();

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

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

private String joinType = null;

@Override

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

joinType = context.getConfiguration().get("join.type");

}

@Override

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

listA.clear();

listB.clear();

while (values.iterator().hasNext()) {

tmp = values.iterator().next();

if ((Character.toString((char) tmp.charAt(0)).equals("A"))) {

listA.add(new Text(tmp.toString().substring(1)));

}

if ((Character.toString((char) tmp.charAt(0)).equals("B"))) {

listB.add(new Text(tmp.toString().substring(1)));

}

}

//System.out.println("Size of A: " + listA.size() + " Size of B: " + listB.size());

executeJoin(context);

}

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

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

// for (Text B : listB) {

// System.out.println(B);

// }

if (!listA.isEmpty() && !listB.isEmpty()) {

for (Text A : listA) {

for (Text B : listB) {

System.out.println("ListA B contains : " + A + " " + B);

context.write(A, B);

}

}

}

} else if (joinType.equalsIgnoreCase("leftouter")) {

for (Text A : listA) {

if (!listB.isEmpty()) {

for (Text B : listB) {

context.write(A, B);

}

} else {

context.write(A, EMPTY\_TEXT);

}

}

}

}

}