**Crimes in Chicago 2008 - 2017**

**Description :**

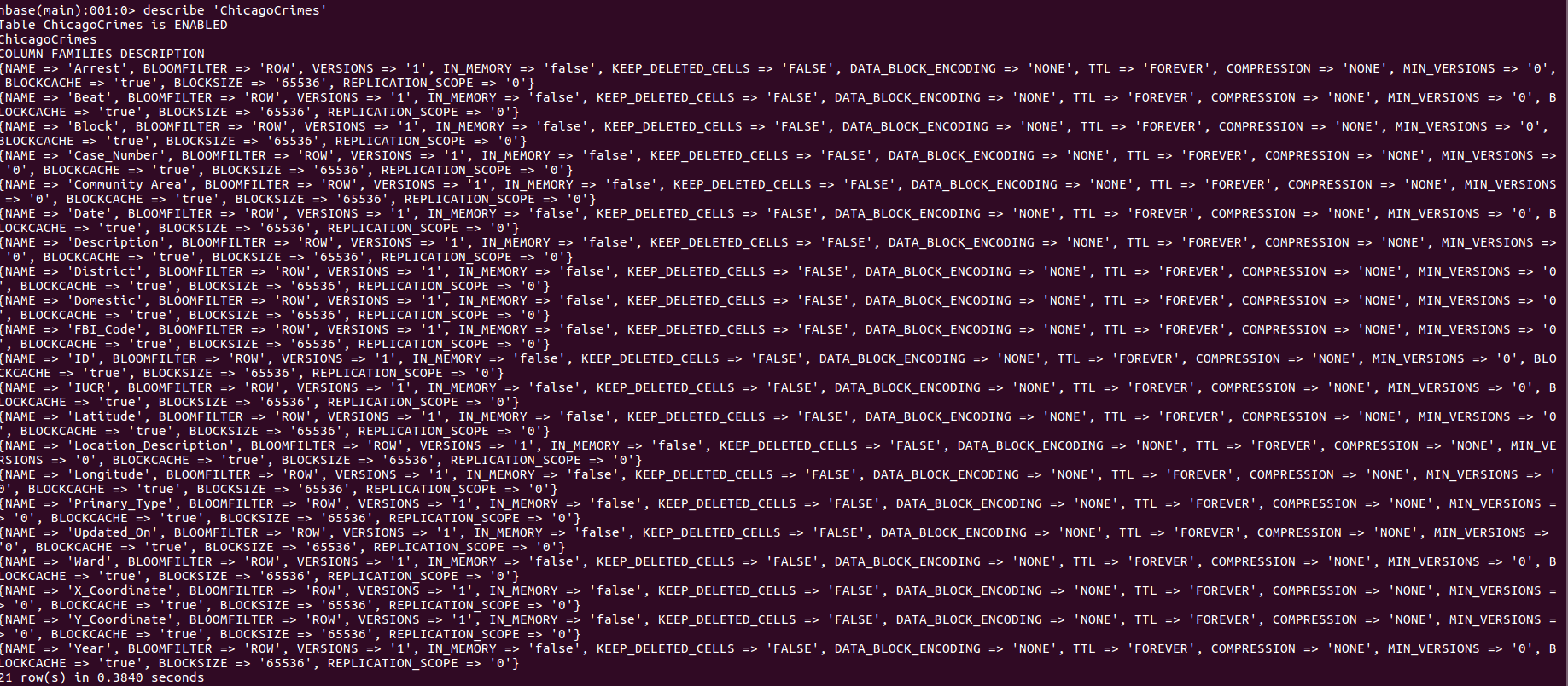
This dataset reflects reported incidents of crime (with the exception of murders where data exists for each victim) that occurred in the City of Chicago from 2008 to present. In order to protect the privacy of crime victims, addresses are shown at the block level only and specific locations are not identified.The dataset consists of a total of 1.6 million records and a total of 22 features out of which I have selected 21 features excluding the location feature. The reason for excluding this column is that the location details are specified in the Latitude and Longitude feature. Thus we can exclude the location feature without having any effect on the overall dataset. Below is a list of all the features and their description.

**Features :**

1. **ID** -  Unique identifier for the record.
2. **Case Number** - The Chicago Police Department RD Number (Records Division Number), which is unique to the incident.
3. **Date** - Date when the incident occurred. this is sometimes a best estimate.
4. **Block** - The partially redacted address where the incident occurred, placing it on the same block as the actual address.
5. **IUCR** - The Illinois Unifrom Crime Reporting code. This is directly linked to the Primary Type and Description.
6. **Primary Type** - The primary description of the IUCR code.
7. **Description** - The secondary description of the IUCR code, a subcategory of the primary description.
8. **Location Description** - Description of the location where the incident occurred.
9. **Arrest** - Indicates whether an arrest was made.
10. **Domestic** - Indicates whether the incident was domestic-related as defined by the Illinois Domestic Violence Act.
11. **Beat** - Indicates the beat where the incident occurred. A beat is the smallest police geographic area – each beat has a dedicated police beat car.
12. **District** - Indicates the police district where the incident occurred.
13. **Ward** - The ward (City Council district) where the incident occurred.
14. **Community Area** - Indicates the community area where the incident occurred. Chicago has 77 community areas.
15. **FBI Code** - Indicates the crime classification as outlined in the FBI's National Incident-Based Reporting System (NIBRS).
16. **X Coordinate** - The x coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block.
17. **Y Coordinate** - The y coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block.
18. **Year** - Year the incident occurred.
19. **Updated On** - Date and time the record was last updated.
20. **Latitude** - The latitude of the location where the incident occurred.
21. **Longitude** - The longitude of the location where the incident occurred.

**Website Link to Dataset :** The website has data starting from 2001. But for my analysis purposes I have focussed on Crimes from year 2008. Following is the link to the dataset on Kaggle.<https://www.kaggle.com/currie32/crimes-in-chicago>

**Analysis Performed:**

1. **HBase:** 
   1. Created an Hbase table ‘ChicagoCrimes’.
   2. 
   3. Created Column Families based on the dataset. Since there were no similar columns that could be grouped in one category. I created a separated Column Family for each column.
   4. From the ChicagoCrimes table retrieved the FBI\_Code values in the Mapper.
   5. On the Reducer side performed a count of all the distinct FBI\_Code values to understand the highly occuring crime types as per the FBI's National Incident-Based Reporting System (NIBRS).
   6. Below is the code to upload the dataset to the Hbase table ‘ChicagoCrimes’.

**HBaseBulkLoadDriver Class(Main Class):**

public class HBaseBulkLoadDriver extends Configured implements Tool {

private static final String DATA\_SEPERATOR = ",";

private static final String TABLE\_NAME = "ChicagoCrimes";

private static final String COLUMN\_FAMILY\_1="Crimes";

private static final String COLUMN\_FAMILY\_2="CrimeDetails";

/\*\*

\* HBase bulk import example

\* Data preparation MapReduce job driver

\*

\* args[0]: HDFS input path

\* args[1]: HDFS output path

\*

\*/

public static void main(String[] args) {

try {

int response = ToolRunner.run(HBaseConfiguration.create(), new HBaseBulkLoadDriver(), args);

if(response == 0) {

System.out.println("Job is successfully completed...");

} else {

System.out.println("Job failed...");

}

} catch(Exception exception) {

exception.printStackTrace();

}

}

@Override

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

int result=0;

String outputPath = args[1];

Configuration configuration = getConf();

configuration.set("data.seperator", DATA\_SEPERATOR);

configuration.set("hbase.table.name",TABLE\_NAME);

configuration.set("COLUMN\_FAMILY\_1",COLUMN\_FAMILY\_1);

configuration.set("COLUMN\_FAMILY\_2",COLUMN\_FAMILY\_2);

Job job = new Job(configuration);

job.setJarByClass(HBaseBulkLoadDriver.class);

job.setJobName("Bulk Loading HBase Table::"+TABLE\_NAME);

job.setInputFormatClass(TextInputFormat.class);

job.setMapOutputKeyClass(ImmutableBytesWritable.class);

job.setMapperClass(HBaseBulkLoadMapper.class);

FileInputFormat.addInputPaths(job, args[0]);

FileSystem.getLocal(getConf()).delete(new Path(outputPath), true);

FileOutputFormat.setOutputPath(job, new Path(outputPath));

job.setMapOutputValueClass(Put.class);

HFileOutputFormat.configureIncrementalLoad(job, new HTable(configuration,TABLE\_NAME));

job.waitForCompletion(true);

if (job.isSuccessful()) {

HBaseBulkLoad.doBulkLoad(outputPath, TABLE\_NAME);

} else {

result = -1;

}

return result;

}

}

**HBaseBulkLoadMapper Class:**

public class HBaseBulkLoadMapper extends Mapper<LongWritable, Text, ImmutableBytesWritable, Put> {

private String hbaseTable;

private String dataSeperator;

private String columnFamily1;

private String columnFamily2;

private ImmutableBytesWritable hbaseTableName;

public void setup(Context context) {

Configuration configuration = context.getConfiguration();

hbaseTable = configuration.get("hbase.table.name");

dataSeperator = configuration.get("data.seperator");

columnFamily1 = configuration.get("COLUMN\_FAMILY\_1");

columnFamily2 = configuration.get("COLUMN\_FAMILY\_2");

hbaseTableName = new ImmutableBytesWritable(Bytes.toBytes(hbaseTable));

// System.out.println("In Mapper");

// System.out.println(columnFamily1);

// System.out.println(columnFamily2);

// System.out.println(hbaseTable);

}

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

try {

String[] values = value.toString().split(dataSeperator);

//System.out.println(values[0]);

String rowKey = values[0];

Put put = new Put(Bytes.toBytes(rowKey));

put.add(Bytes.toBytes(columnFamily1), Bytes.toBytes("Case Number"), Bytes.toBytes(values[1]));

System.out.println(columnFamily1);

System.out.println(values[1]);

put.add(Bytes.toBytes(columnFamily1), Bytes.toBytes("Date"), Bytes.toBytes(values[2]));

put.add(Bytes.toBytes(columnFamily1), Bytes.toBytes("Block"), Bytes.toBytes(values[3]));

put.add(Bytes.toBytes(columnFamily1), Bytes.toBytes("IUCR"), Bytes.toBytes(values[4]));

put.add(Bytes.toBytes(columnFamily1), Bytes.toBytes("Primary Type"), Bytes.toBytes(values[5]));

put.add(Bytes.toBytes(columnFamily1), Bytes.toBytes("Description"), Bytes.toBytes(values[6]));

put.add(Bytes.toBytes(columnFamily1), Bytes.toBytes("Location Description"), Bytes.toBytes(values[7]));

put.add(Bytes.toBytes(columnFamily1), Bytes.toBytes("Arrest"), Bytes.toBytes(values[8]));

put.add(Bytes.toBytes(columnFamily1), Bytes.toBytes("Domestic"), Bytes.toBytes(values[9]));

put.add(Bytes.toBytes(columnFamily1), Bytes.toBytes("Beat"), Bytes.toBytes(values[10]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("District"), Bytes.toBytes(values[11]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("Ward"), Bytes.toBytes(values[12]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("Community Area"), Bytes.toBytes(values[13]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("FBI Code"), Bytes.toBytes(values[14]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("X Coordinate"), Bytes.toBytes(values[15]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("Y Coordinate"), Bytes.toBytes(values[16]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("Year"), Bytes.toBytes(values[17]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("Updated On"), Bytes.toBytes(values[18]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("District"), Bytes.toBytes(values[19]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("Latitude"), Bytes.toBytes(values[20]));

put.add(Bytes.toBytes(columnFamily2), Bytes.toBytes("Longitude"), Bytes.toBytes(values[21]));

context.write(hbaseTableName, put);

} catch(Exception exception) {

exception.printStackTrace();

System.out.println("Exception is here" + exception);

}

}

}

**HBaseBulkLoad Class:**

public class HBaseBulkLoad {

/\*\*

\* doBulkLoad.

\*

\* @param pathToHFile path to hfile

\* @param tableName

\*/

public static void doBulkLoad(String pathToHFile, String tableName) {

try {

Configuration configuration = new Configuration();

configuration.set("mapreduce.child.java.opts", "-Xmx300m");

HBaseConfiguration.addHbaseResources(configuration);

LoadIncrementalHFiles loadFfiles = new LoadIncrementalHFiles(configuration);

HTable hTable = new HTable(configuration, tableName);

loadFfiles.doBulkLoad(new Path(pathToHFile), hTable);

System.out.println("Bulk Load Completed..");

} catch(Exception exception) {

exception.printStackTrace();

}

}

}

**Map -Reduce Code Using Hbase table as an Input**

public class Count\_Main {

public static void main( String[] args )

{

try

{

// Setup Hadoop

Configuration conf = HBaseConfiguration.create();

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

job.setJarByClass(Count\_Main.class );

// Create a scan

Scan scan = new Scan();

// Configure the Map process to use HBase

TableMapReduceUtil.initTableMapperJob(

"ChicagoCrimes", // The name of the table

scan, // The scan to execute against the table

testMapper.class, // The Mapper class

LongWritable.class, // The Mapper output key class

LongWritable.class, // The Mapper output value class

job ); // The Hadoop job

// Configure the reducer process

job.setReducerClass( MyReducer.class );

job.setCombinerClass( MyReducer.class );

// Setup the output - we'll write to the file system: HOUR\_OF\_DAY PAGE\_VIEW\_COUNT

job.setOutputKeyClass( LongWritable.class );

job.setOutputValueClass( LongWritable.class );

job.setOutputFormatClass( TextOutputFormat.class );

// We'll run just one reduce task, but we could run multiple

job.setNumReduceTasks( 1 );

// Write the results to a file in the output directory

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

// Execute the job

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

}

catch( Exception e )

{

e.printStackTrace();

}

}

public static class testMapper extends TableMapper<LongWritable,LongWritable> {

private LongWritable ONE = new LongWritable( 1 );

public void map(ImmutableBytesWritable rowKey, Result columns, Context context)

throws IOException, InterruptedException {

try {

// get rowKey and convert it to string

String inKey = new String(rowKey.get());

// set new key having only date

// get sales column in byte format first and then convert it to string (as it is stored as string from hbase shell)

byte[] fbi\_n = columns.getValue(Bytes.toBytes("FBI\_Code"), Bytes.toBytes("FBI\_Code"));

String FBI\_Code = new String(fbi\_n);

Long f= new Long(FBI\_Code);

// emit date and sales values

context.write(new LongWritable(f), ONE);

} catch (RuntimeException e){

e.printStackTrace();

}

}

}

public static class MyReducer extends Reducer<LongWritable, LongWritable, LongWritable, LongWritable>

{

public MyReducer() {

}

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

{

// Add up all of the page views for this hour

long sum = 0;

for( LongWritable val : values)

{

sum+= val.get();

}

// Write out the current hour and the sum

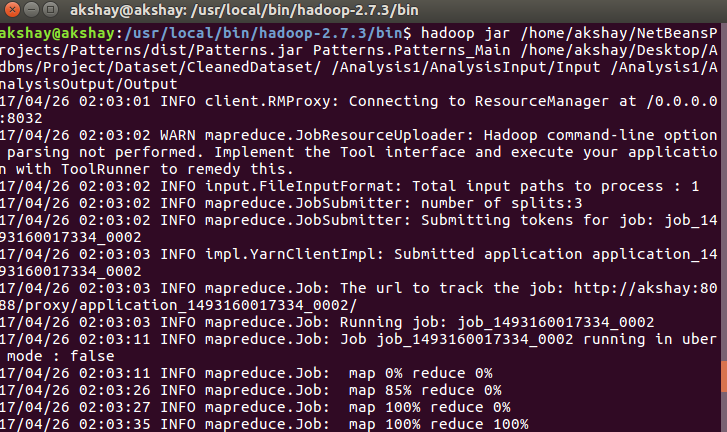
context.write( key, new LongWritable(sum));

}

}

}

**2. Identify if the crime was domestic-related or not. This would help the Police to analyse the ratio of domestic crimes to the non-domestic crimes and be more cautious for the later as they are more of threat to the community.**

* 1. I haved used the concept of putmerge taught in lecture and considered both files as input from the local directory and output them as a single merged file on the HDFS.
  2. This file is then taken as an input by the Hadoop Job. In the Mapper the domestic feature is taken and a count of all the True, False which indicates whether it was a domestic violence or not is taken and the results are fed in the HDFS output directory.
  3. Below are the screenshots that show the running of the job.
  4. 
  5. 
  6. Code Snippet:
     1. **Main Class:**

public class Patterns\_Main {

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

Configuration conf = new Configuration();

FileSystem hdfs = FileSystem.get(conf);

FileSystem local = FileSystem.getLocal(conf);

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

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

try{

FileStatus[] inputfiles = local.listStatus(inputDir);

FSDataOutputStream out = hdfs.create(hdfsfile);

for(int i=0; i<inputfiles.length; i++){

FSDataInputStream in = local.open(inputfiles[i].getPath());

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();

}

Job job = Job.getInstance(conf,"domestic\_true\_false");

job.setJarByClass(Patterns\_Main.class);

job.setMapperClass(Patterns\_Mapper.class);

job.setReducerClass(Patterns\_Reducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, hdfsfile);

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

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

}

}

* + 1. **Mapper Class:**

public class Patterns\_Mapper extends Mapper<LongWritable, Text, Text, IntWritable>{

private Text domestic\_incident = new Text();

private IntWritable one = new IntWritable(1);

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

String header = value.toString();

if(header.contains("Domestic")){

return;

}else{

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

//System.out.println(values[0] + "-" + values[1] + "-" + values[2] + "-" + values[3] + "- " + values[4] + "-" + values[5] + "-" + values[6] + "-" + values[7] +"####" + values[8] + "#################" + values[9]);

String domestic = values[9];

domestic\_incident.set(domestic);

context.write(domestic\_incident, one);

}

}

}

* + 1. **Reducer Class:**

public class Patterns\_Reducer extends Reducer<Text, IntWritable, Text, IntWritable>{

private Text true\_false = new Text();

private IntWritable count = 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();

}

count.set(sum);

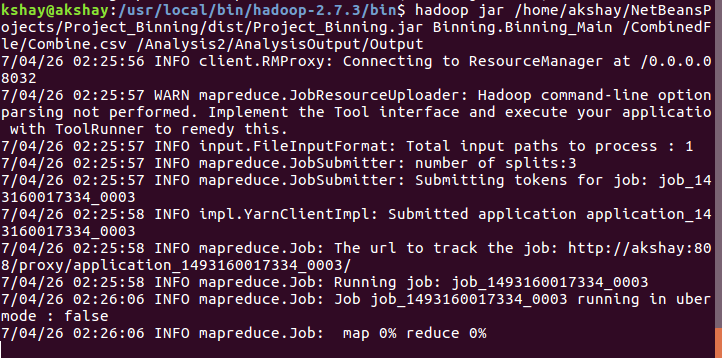
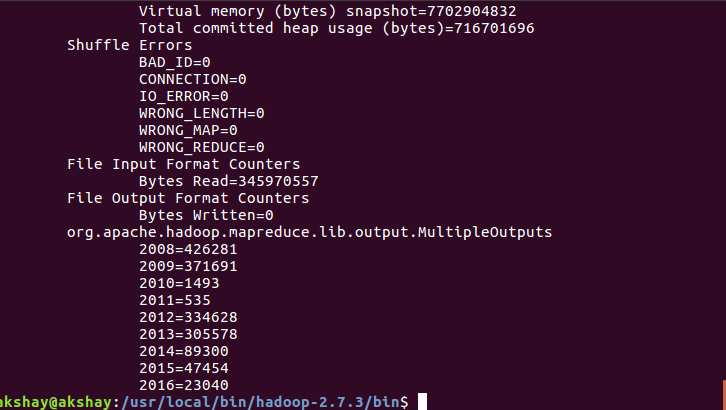
true\_false.set(key);

context.write(true\_false, count);

}

}

**3. Categorise the entire dataset based on the year in which crime happened.(Binning)**

* 1. This would help in further analysis since we have a group of crime records on a yearly basis.
  2. For this analysis I have used Binning Pattern taught in class where a bin is created for every year that contains the yearly criminal activity.
  3. Below is the screenshot of the runnning job:
  4. 
  5. 
  6. Code Snippet:
     1. **Main Class:**

public class Binning\_Main {

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

Configuration conf = new Configuration();

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

job.setJarByClass(Binning\_Main.class);

job.setJobName("Group By Year");

//job.setMapOutputKeyClass(Text.class);

//job.setMapOutputValueClass(NullWritable.class);

job.setMapperClass(Binning\_Mapper.class);

//job.setOutputKeyClass(Text.class);

//job.setOutputValueClass(NullWritable.class);

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

MultipleOutputs.setCountersEnabled(job, true);

//job.setNumReduceTasks(0);

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

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

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

}

}

* + 1. **Mapper Class:**

public class Binning\_Mapper extends Mapper<Object, Text, Text, NullWritable>{

private MultipleOutputs<Text, NullWritable> mos = null;

protected void setup(Mapper.Context context){

mos = new MultipleOutputs(context);

}

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

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

String year = records[17].trim();

if(year.equalsIgnoreCase("2008"))

mos.write("bins",value, NullWritable.get(), "2008");

if(year.equalsIgnoreCase("2009"))

mos.write("bins",value, NullWritable.get(), "2009");

if(year.equalsIgnoreCase("2010"))

mos.write("bins",value, NullWritable.get(), "2010");

if(year.equalsIgnoreCase("2011"))

mos.write("bins",value, NullWritable.get(), "2011");

if(year.equalsIgnoreCase("2012"))

mos.write("bins",value, NullWritable.get(), "2012");

if(year.equalsIgnoreCase("2013"))

mos.write("bins",value, NullWritable.get(), "2013");

if(year.equalsIgnoreCase("2014"))

mos.write("bins",value, NullWritable.get(), "2014");

if(year.equalsIgnoreCase("2015"))

mos.write("bins",value, NullWritable.get(), "2015");

if(year.equalsIgnoreCase("2016"))

mos.write("bins",value, NullWritable.get(), "2016");

}

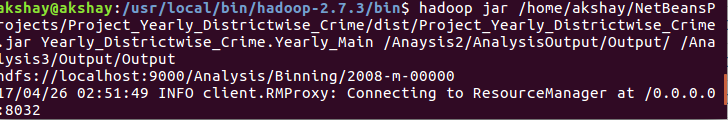
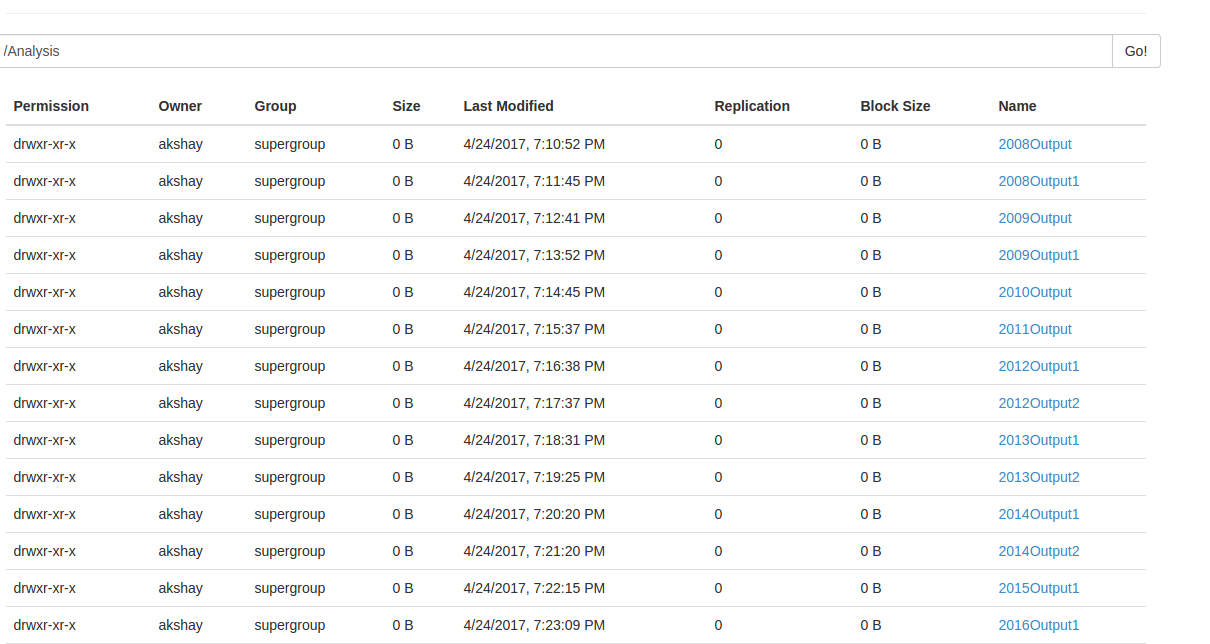
protected void cleanup(Mapper.Context context) throws IOException, InterruptedException{

mos.close();

}

}

**4. Yearly Districtwise Crime Count(Multiple Jobs) : Based on the bins done in the previous analysis, I have considered using those bins to analyse the**

* 1. Each record in the dataset contains information about the district code where the crime happened. This could be used to analyse the crime count in a particular district that reflects how unsafe the district and the Chicago Police should enforce some extra Policeman’s in those district to control the crime rate.
  2. The yearly trend in districtwise crime could also help in analysing how effective the Chicago Police has been in controlling the crime rate in a highly criminal district.
  3. Here, I haved used a different job for each of the input year files. The Mapper takes that file and counts the number of criminal records districtwise and then outputs the results on the HDFS output directory. I
  4. Below is the screenshot of the running job.
  5. 
  6. 
  7. Code Snippet:
     1. **Main Class:**

public class Yearly\_Main{

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

Configuration conf = new Configuration();

String[] names ={"2008-m-00000","2008-m-00001","2009-m-00000","2009-m-00001","2010-m-00000","2011-m-00000","2012-m-00001","2012-m-00002","2013-m-00001","2013-m-00002","2014-m-00001","2014-m-00002","2015-m-00001","2016-m-00001"};

String[] outNames = {"2008Output", "2008Output1", "2009Output","2009Output1","2010Output","2011Output","2012Output1","2012Output2","2013Output1","2013Output2","2014Output1","2014Output2","2015Output1","2016Output1"};

// Path inPath = new Path(args[0]);

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

int c = 0;

// String fileName = inPath.getName() + "/2008-m-00000" ;

for(int i=0; i<14; i++){

//Path in = new Path(fileName);

String hdfsPath="hdfs://localhost:9000/Analysis/Binning/";

String outPath = "hdfs://localhost:9000/Analysis/";

String fileName = hdfsPath + names[i];

String outFileName = outPath + outNames[i];

System.out.println(fileName);

Job job = Job.getInstance(conf, "Yearly\_Districtwise\_Count");

job.setJarByClass(Yearly\_Main.class);

job.setJobName("Group By Year");

job.setMapperClass(Yearly\_Mapper.class);

job.setReducerClass(Yearly\_Reducer.class);

job.setNumReduceTasks(14);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

//MultipleOutputs.setCountersEnabled(job, true);

FileInputFormat.setInputDirRecursive(job, true);

FileInputFormat.addInputPath(job, new Path(fileName));

FileOutputFormat.setOutputPath(job, new Path(outFileName));

//FileInputFormat.setInputPaths(job, in);

//FileOutputFormat.setOutputPath(job, outPath);

c++;

//job.waitForCompletion(true);

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

if(i==14){

System.exit(status);

}

}

}

}

* + 1. **Mapper Class:**

public class Yearly\_Mapper extends Mapper<Object, Text, Text, IntWritable>{

private Text district = new Text();

private IntWritable one = new IntWritable(1);

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

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

String distric\_code = values[11].trim();

district.set(distric\_code);

context.write(district, one);

}

}

* + 1. **Reducer Class:**

public class Yearly\_Reducer extends Reducer<Text, IntWritable, Text, IntWritable>{

public IntWritable dist\_count = new IntWritable();

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

int count = 0;

for(IntWritable val : values){

count++;

}

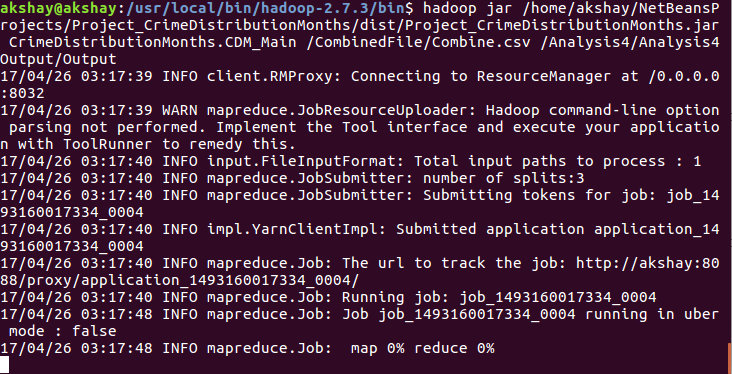
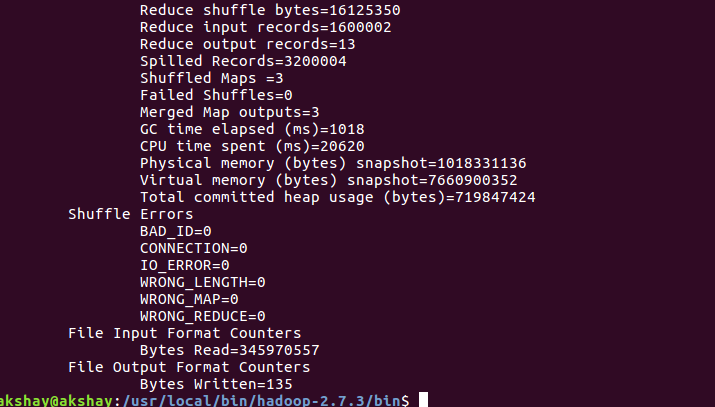
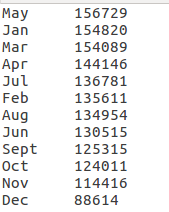
dist\_count.set(count);

context.write(key, dist\_count);

}

}

**5. Crime Distribution Monthly(Secondary Sorting): This analysis includes identifying months of the year when crime is more often bound to happen based on historical trends.**

* 1. Here, I have made a count of the total crime records monthly starting from 2008 until 2011.
  2. Since we need to identify the months with maximum crime records I first took a count of all the records monthly and then later used another job that takes the output of this job as an input and sorts the count of monthly crime records descendingly.
  3. For the sorting I have used Secondary sorting as taught in lecture.
  4. Below are the screenshots of the running job:
  5. 
  6. 
  7. 
  8. Code Snippet:

**Main Class:**

public class CDM\_Main extends Configured implements Tool{

public static void main(String[] args){

try{

int res = ToolRunner.run(new Configuration(), new CDM\_Main(), args);

}catch (Exception e){

}

}

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf,"domestic\_true\_false");

job.setJarByClass(CDM\_Main.class);

job.setMapperClass(CDM\_Mapper.class);

job.setReducerClass(CDM\_Reducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

boolean success = job.waitForCompletion(true);

Configuration conf2 = new Configuration();

Job job2 = Job.getInstance(conf2,"SecondReducerJoin");

if(success){

job2.setJarByClass(CDM\_Main.class);

job2.setMapperClass(CDM\_Mapper2.class);

job2.setReducerClass(CDM\_Reducer2.class);

job2.setOutputKeyClass(CompositeKeyWritable.class);

job2.setOutputValueClass(NullWritable.class);

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

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

}

boolean complete = job2.waitForCompletion(true);

return complete ? 0 : 2;

}

**Mapper1:**

public static class CDM\_Mapper extends Mapper<Object, Text, Text, IntWritable>{

private Text month = new Text();

private IntWritable one = new IntWritable(1);

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

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

String date = values[2].trim();

String month\_number = date.substring(0,2);

if(month\_number.equalsIgnoreCase("01"))

month.set("Jan");

else if(month\_number.equalsIgnoreCase("02"))

month.set("Feb");

else if(month\_number.equalsIgnoreCase("03"))

month.set("Mar");

else if(month\_number.equalsIgnoreCase("04"))

month.set("Apr");

else if(month\_number.equalsIgnoreCase("05"))

month.set("May");

else if(month\_number.equalsIgnoreCase("06"))

month.set("Jun");

else if(month\_number.equalsIgnoreCase("07"))

month.set("Jul");

else if(month\_number.equalsIgnoreCase("08"))

month.set("Aug");

else if(month\_number.equalsIgnoreCase("09"))

month.set("Sept");

else if(month\_number.equalsIgnoreCase("10"))

month.set("Oct");

else if(month\_number.equalsIgnoreCase("11"))

month.set("Nov");

else if(month\_number.equalsIgnoreCase("12"))

month.set("Dec");

context.write(month, one);

}

}

Reducer1:

public static class CDM\_Reducer extends Reducer<Text, IntWritable, Text, IntWritable>{

private IntWritable months\_count = new IntWritable();

private Text peak\_months = new Text();

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

int sum = 0;

for(IntWritable val: values){

sum+= val.get();

}

months\_count.set(sum);

peak\_months.set(key);

context.write(peak\_months, months\_count);

}

}

**Mapper 2:**

public static class CDM\_Mapper2 extends Mapper<Object, Text, CompositeKeyWritable, NullWritable>{

private Text month1 = new Text();

private IntWritable count = new IntWritable();

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

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

// int crime\_count = (Integer.parseInt(values[1].trim()));

// month1.set(values[0]);

// context.write(new IntWritable(crime\_count), month1);

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

CompositeKeyWritable cw = new CompositeKeyWritable(values[0], Integer.parseInt(values[1]));

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

}

}

**Reducer 2:**

public static class CDM\_Reducer2 extends Reducer<CompositeKeyWritable, NullWritable, CompositeKeyWritable, NullWritable>{

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

for(NullWritable val : values){

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

}

}

}

**Writable Comparable Class:**

public static class CompositeKeyWritable implements Writable, WritableComparable<CompositeKeyWritable> {

private String month1;

private Integer crime\_count;

public CompositeKeyWritable(){

}

public CompositeKeyWritable(String m, int c){

this.month1 = m;

this.crime\_count = c;

}

public String getMonth1() {

return month1;

}

public void setMonth1(String month1) {

this.month1 = month1;

}

public int getCrime\_count() {

return crime\_count;

}

public void setCrime\_count(int crime\_count) {

this.crime\_count = crime\_count;

}

@Override

public void write(DataOutput d) throws IOException {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

WritableUtils.writeString(d, month1);

WritableUtils.writeVInt(d, crime\_count);

}

@Override

public void readFields(DataInput di) throws IOException {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

month1 = WritableUtils.readString(di);

crime\_count = WritableUtils.readVInt(di);

}

@Override

public int compareTo(CompositeKeyWritable o) {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

int result = crime\_count.compareTo(o.crime\_count);

if(result ==0){

result = month1.compareTo(o.month1);

}

return -1 \* result;

}

public String toString()

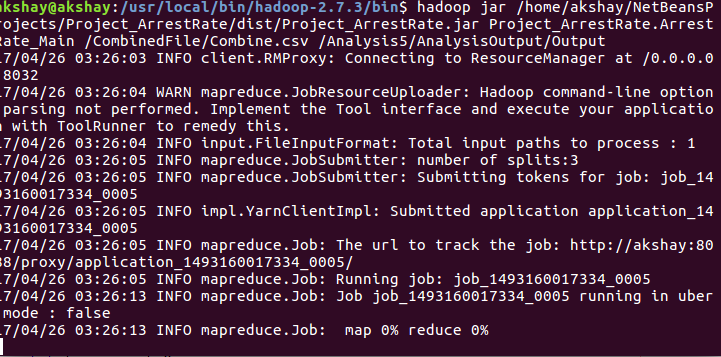
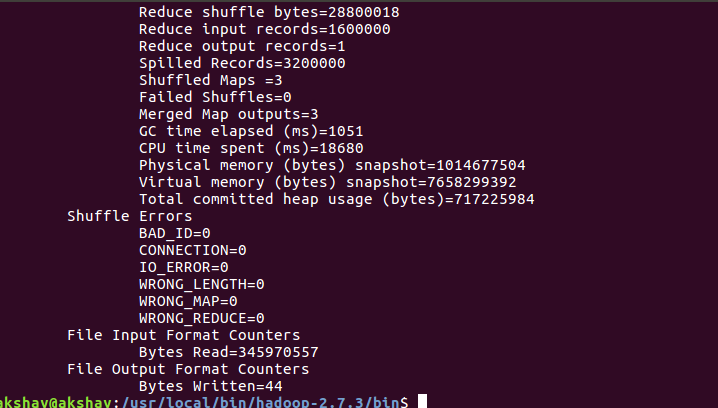
{

return (new StringBuilder().append(month1).append("\t").append(crime\_count).toString());

}

}

**6. Arrest Rate Analysis(Writable Class to represent Percentage): This analysis indicates the Chicago Police performance in arresting criminals.**

* 1. The dataset contains a feature called Arrest that indicates True or False. I have taken a count of all the True and False and then later used a Writable class to represent the precentage of Arrest and Count of the same.
  2. The counting and taking the precentage is done in the Reducer part.
  3. Below is the screenshot of the runnnig job:
  4. 
  5. 
  6. 
  7. Code Snippet:

**Main Class:**

public class ArrestRate\_Main {

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

Configuration conf= new Configuration();

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

job.setJarByClass(ArrestRate\_Main.class);

job.setMapperClass(ArrestRate\_Mapper.class);

job.setReducerClass(ArrestRate\_Reducer.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(ArrestRate\_True\_False.class);

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

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

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

}

**Mapper Class:**

public static class ArrestRate\_Mapper extends Mapper<LongWritable, Text, NullWritable, ArrestRate\_True\_False>{

private Text arrest = new Text();

private ArrestRate\_True\_False atf = new ArrestRate\_True\_False();

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

String a = value.toString();

if(a.contains("Arrest")){

return;

}

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

String arrest\_type = values[8].trim();

if(arrest\_type.equals("True")){

//arrest.set(arrest\_type);

atf.setTrue\_count(1);

atf.setFalse\_count(0);

//System.out.println("Im True");

atf.setPercentTrue(0);

}else{

//arrest.set(arrest\_type);

atf.setFalse\_count(1);

atf.setTrue\_count(0);

//System.out.println("Im False");

atf.setPercentFalse(0);

}

// System.out.println(arrest\_type);

// System.out.println(atf.getTrue\_count());

// System.out.println(atf.getFalse\_count());

//

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

}

}

**Writable Class:**

public static class ArrestRate\_True\_False implements Writable {

private int true\_count;

private int false\_count;

private float percentTrue;

private float percentFalse;

public float getPercentTrue() {

return percentTrue;

}

public void setPercentTrue(float percentTrue) {

this.percentTrue = percentTrue;

}

public float getPercentFalse() {

return percentFalse;

}

public void setPercentFalse(float percentFalse) {

this.percentFalse = percentFalse;

}

public int getTrue\_count() {

return true\_count;

}

public void setTrue\_count(int true\_count) {

this.true\_count = true\_count;

}

public int getFalse\_count() {

return false\_count;

}

public void setFalse\_count(int false\_count) {

this.false\_count = false\_count;

}

@Override

public void write(DataOutput d) throws IOException {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

d.writeInt(true\_count);

d.writeInt(false\_count);

d.writeFloat(percentTrue);

d.writeFloat(percentFalse);

}

@Override

public void readFields(DataInput di) throws IOException {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

true\_count = di.readInt();

false\_count = di.readInt();

percentTrue = di.readFloat();

percentFalse = di.readFloat();

}

public String toString(){

return "True" + "\t" + true\_count + "\t" + percentTrue + "\n" + "False" + "\t" + false\_count + "\t" + percentFalse;

}

}

**Reducer Class:**

public static class ArrestRate\_Reducer extends Reducer<NullWritable, ArrestRate\_True\_False, NullWritable, ArrestRate\_True\_False>{

private ArrestRate\_True\_False result = new ArrestRate\_True\_False();

public void reduce(NullWritable key, Iterable<ArrestRate\_True\_False> values, Context context)throws IOException, InterruptedException{

float sum=0;

int true\_count=0;

int false\_count=0;

for(ArrestRate\_True\_False val : values){

true\_count += val.getTrue\_count();

false\_count += val.getFalse\_count();

sum+= val.getTrue\_count() + val.getFalse\_count();

}

result.setPercentTrue((true\_count/sum)\*100);

result.setPercentFalse((false\_count/sum)\*100);

result.setTrue\_count(true\_count);

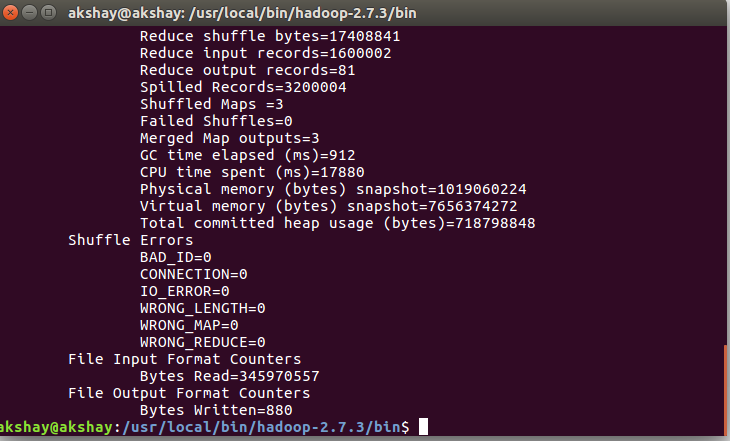
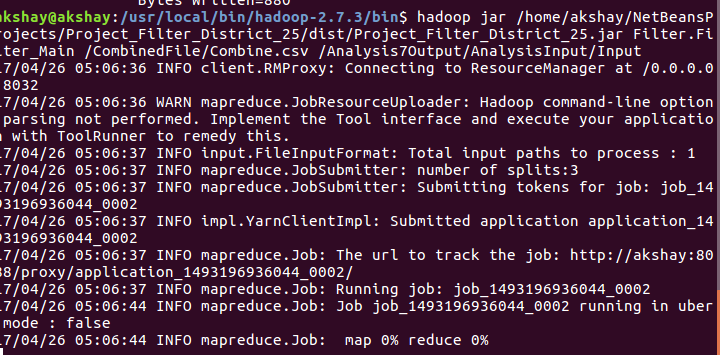
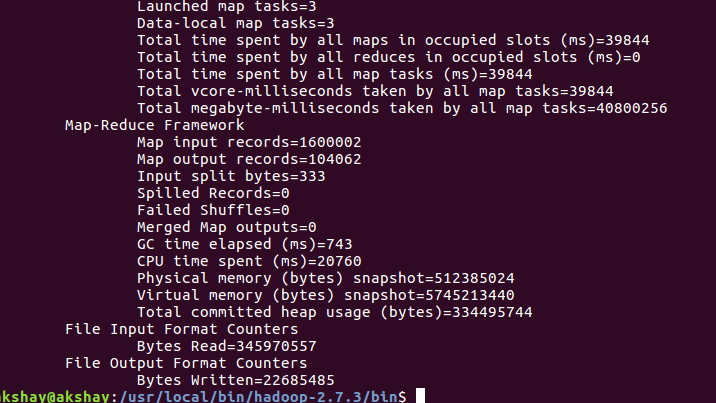
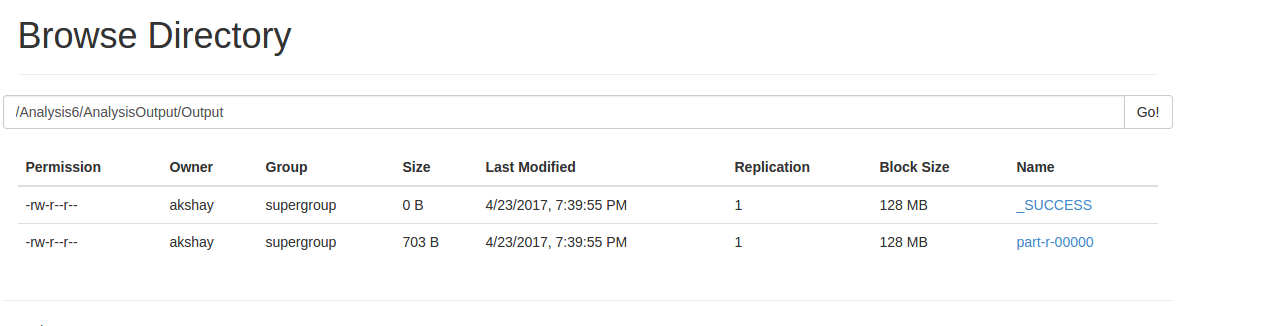
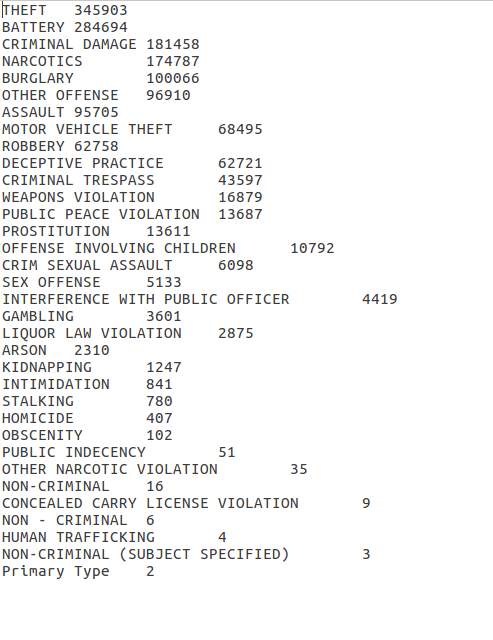
result.setFalse\_count(false\_count);

context.write(key, result);

}

}

**7. Primary Crime Types(Chaining): The dataset contains a feature called as crime type that indicates the nature of the crime whether it was a theft or something else.**

* 1. This analysis can help to identify the most occuring crime types that could help the Chicago Police to implement better systems for a particular type of crime occurences.
  2. Here I have used the concept of chaining multiple jobs taught in class.
  3. The first job here gets a count of all the unique crime types for the records from 2008 until 2017.
  4. The second job sorts the crime type count by descending order to gain an insight into the most frequently occuring crime type.
  5. The dataset indicates that THEFT is the most prominent crime type that occurs in the city of Chicago in the past 9 years.
  6. Below are screenshots of the running code:
  7. 
  8. 
  9. Filter Records
  10. 
  11. 
  12. 
  13. Code Snippet:

**Main Class:**

public class PrimaryCrimeTypes\_Main extends Configured implements Tool {

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

try{

int res = ToolRunner.run(new Configuration(), new PrimaryCrimeTypes\_Main(), args);

}catch (Exception e){

}

}

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf,"domestic\_true\_false");

job.setJarByClass(PrimaryCrimeTypes\_Main.class);

job.setMapperClass(PCT\_Mapper.class);

job.setReducerClass(PCT\_Reducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

boolean complete = job.waitForCompletion(true);

Configuration conf2 = new Configuration();

Job job2 = Job.getInstance(conf2,"SecondReducerJoin");

if(complete){

job2.setJarByClass(PrimaryCrimeTypes\_Main.class);

job2.setMapperClass(PCT\_Mapper2.class);

job2.setReducerClass(PCT\_Reducer2.class);

job2.setOutputKeyClass(PrimaryCrimeTypesWritable.class);

job2.setOutputValueClass(NullWritable.class);

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

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

}

boolean success = job2.waitForCompletion(true);

return success ? 0 : 2;

}

public static class PCT\_Mapper extends Mapper<LongWritable, Text, Text, IntWritable>{

private IntWritable one = new IntWritable(1);

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

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

String crime\_type = values[5].trim();

context.write(new Text(crime\_type), one);

}

}

public static class PCT\_Reducer extends Reducer<Text, IntWritable, Text, IntWritable>{

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

int count=0;

for(IntWritable val: values){

count++;

}

context.write(key, new IntWritable(count));

}

}

public static class PCT\_Mapper2 extends Mapper<Object, Text, PrimaryCrimeTypesWritable, NullWritable>{

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

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

PrimaryCrimeTypesWritable pct = new PrimaryCrimeTypesWritable(values[0],Integer.parseInt(values[1]));

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

}

}

public static class PrimaryCrimeTypesWritable implements Writable, WritableComparable<PrimaryCrimeTypesWritable>{

private String theft;

private Integer count;

public PrimaryCrimeTypesWritable(){

}

public PrimaryCrimeTypesWritable(String t,int c){

this.theft = t;

this.count = c;

}

@Override

public void write(DataOutput d) throws IOException {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

WritableUtils.writeString(d, theft);

WritableUtils.writeVInt(d, count);

}

public String getTheft() {

return theft;

}

public void setTheft(String theft) {

this.theft = theft;

}

public int getCount() {

return count;

}

public void setCount(int count) {

this.count = count;

}

@Override

public void readFields(DataInput di) throws IOException {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

theft = WritableUtils.readString(di);

count = WritableUtils.readVInt(di);

}

@Override

public int compareTo(PrimaryCrimeTypesWritable o) {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

int result = count.compareTo(o.count);

if(result ==0){

result = theft.compareTo(o.theft);

}

return -1 \* result;

}

public String toString()

{

return (new StringBuilder().append(theft).append("\t").append(count).toString());

}

}

public static class PCT\_Reducer2 extends Reducer<Text, IntWritable, PrimaryCrimeTypesWritable, NullWritable>{

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

for(NullWritable val : values){

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

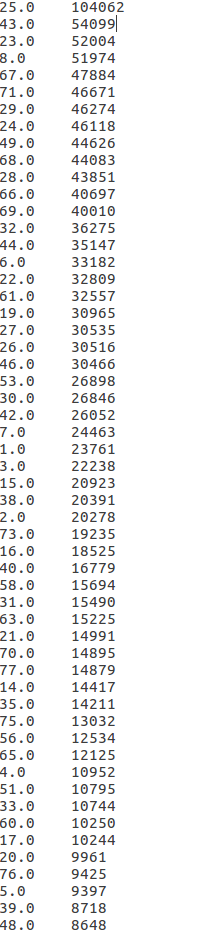
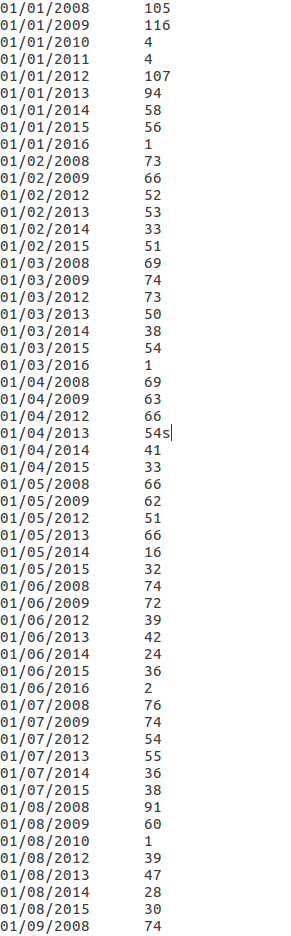
}

}

}

}

**8. Community Area with Max Crime and Later Filter that community area to find the Average Crime Per Day in that community(Filtering and Chaining):**

* 1. In this analysis first a separate analysis has been done that finds out the Community Area with the maximum crime count. The output received from this shows that community area 25.0 has the maximum crime count.
  2. In the next step I have done Filtering on community area 25.0 and generated only those area specific records.
  3. In the third job I have calculated the distinct days when crimes occured. Took a count of all thes days along with the count of crimes on that particular day.
  4. Reducer then takes an average of the sum of the counts divided by the total number of days that gives us the Average Crime Per day in community area 25.0 which is 52 Crimes.
  5. The results show that it is a highly criminal Area.
  6. Below is the Output for the above analysis:
  7. Max Crime Count in area 25
  8. Daily Crime Count:
  9. 
  10. Code Snippet:

**Area With Max Crime:**

public class AreaWithMaxCrime\_Main extends Configured implements Tool {

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

try{

int res = ToolRunner.run(new Configuration(), new AreaWithMaxCrime\_Main(), args);

}catch (Exception e){

}

}

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf,"domestic\_true\_false");

job.setJarByClass(AreaWithMaxCrime\_Main.class);

job.setMapperClass(AreaWithMaxCrime\_Mapper.class);

job.setReducerClass(AreaWithMaxCrime\_Reducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

boolean success = job.waitForCompletion(true);

Configuration conf2 = new Configuration();

Job job2 = Job.getInstance(conf2,"SecondReducerJoin");

if(success){

job2.setJarByClass(AreaWithMaxCrime\_Main.class);

job2.setMapperClass(AreaWithMaxCrime\_Mapper2.class);

job2.setReducerClass(AreaWithMaxCrime\_Reducer2.class);

job2.setOutputKeyClass(CompositeKeyWritable.class);

job2.setOutputValueClass(NullWritable.class);

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

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

}

boolean complete = job2.waitForCompletion(true);

return complete ? 0 : 2;

}

public static class AreaWithMaxCrime\_Mapper extends Mapper<Object, Text, Text, IntWritable>{

private IntWritable one = new IntWritable();

private Text area = new Text();

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

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

String area\_code = values[13].trim();

area.set(area\_code);

context.write(area, one);

}

}

public static class AreaWithMaxCrime\_Reducer extends Reducer<Text, IntWritable, Text, IntWritable>{

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

int count = 0;

for(IntWritable val : values){

count++;

}

context.write(key, new IntWritable(count));

}

}

public static class AreaWithMaxCrime\_Mapper2 extends Mapper<Object, Text, CompositeKeyWritable, NullWritable>{

private Text area1 = new Text();

private IntWritable count = new IntWritable();

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

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

// int crime\_count = (Integer.parseInt(values[1].trim()));

// month1.set(values[0]);

// context.write(new IntWritable(crime\_count), month1);

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

CompositeKeyWritable cw = new CompositeKeyWritable(values[0], Integer.parseInt(values[1]));

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

}

}

public static class AreaWithMaxCrime\_Reducer2 extends Reducer<CompositeKeyWritable, NullWritable, CompositeKeyWritable, NullWritable>{

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

for(NullWritable val : values){

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

}

}

}

public static class CompositeKeyWritable implements Writable, WritableComparable<CompositeKeyWritable> {

private String area1;

private Integer crime\_count;

public CompositeKeyWritable(){

}

public CompositeKeyWritable(String m, int c){

this.area1 = m;

this.crime\_count = c;

}

public String getArea1() {

return area1;

}

public void setArea1(String area1) {

this.area1 = area1;

}

public int getCrime\_count() {

return crime\_count;

}

public void setCrime\_count(int crime\_count) {

this.crime\_count = crime\_count;

}

@Override

public void write(DataOutput d) throws IOException {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

WritableUtils.writeString(d, area1);

WritableUtils.writeVInt(d, crime\_count);

}

@Override

public void readFields(DataInput di) throws IOException {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

area1 = WritableUtils.readString(di);

crime\_count = WritableUtils.readVInt(di);

}

@Override

public int compareTo(CompositeKeyWritable o) {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

int result = crime\_count.compareTo(o.crime\_count);

if(result ==0){

result = area1.compareTo(o.area1);

}

return -1 \* result;

}

public String toString()

{

return (new StringBuilder().append(area1).append("\t").append(crime\_count).toString());

}

}

}

**Filtering into Area 25.0:**

public class Filter\_Main {

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf,"domestic\_true\_false");

job.setJarByClass(Filter\_Main.class);

job.setMapperClass(Filter\_Mapper.class);

job.setNumReduceTasks(0);

job.getConfiguration().set("community\_area\_code", "25.0");

//job.setReducerClass(Filter\_Reducer.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

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

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

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

}

public static class Filter\_Mapper extends Mapper<Object, Text, NullWritable, Text>{

private String community\_area = null;

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

community\_area = context.getConfiguration().get("community\_area\_code");

}

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

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

String area\_code = values[13].trim();

System.out.println(area\_code);

if(area\_code.contains(community\_area)){

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

}

else

return;

}

}

}

**Average Crime Per Day:**

public class AverageCrime\_Main extends Configured implements Tool{

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

try{

int res = ToolRunner.run(new Configuration(), new AverageCrime\_Main(), args);

}catch (Exception e){

}

}

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

Configuration conf= new Configuration();

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

job.setJarByClass(AverageCrime\_Main.class);

job.setMapperClass(AverageCrime\_Mapper.class);

//job.setCombinerClass(AverageCrime\_Reducer.class);

job.setReducerClass(AverageCrime\_Reducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.setInputDirRecursive(job, true);

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

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

boolean success = job.waitForCompletion(true);

Configuration conf2 = new Configuration();

Job job2 = Job.getInstance(conf2,"SecondReducerJoin");

if(success){

job2.setJarByClass(AverageCrime\_Main.class);

job2.setMapperClass(AverageCrime\_Mapper2.class);

job2.setMapOutputKeyClass(NullWritable.class);

job2.setMapOutputValueClass(CompositeKeyWritable.class);

job2.setReducerClass(AverageCrime\_Reducer2.class);

job2.setOutputKeyClass(NullWritable.class);

job2.setOutputValueClass(IntWritable.class);

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

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

}

boolean complete = job2.waitForCompletion(true);

return complete ? 0 : 2;

}

public static class AverageCrime\_Mapper extends Mapper<Object, Text, Text, IntWritable>{

private Text date\_year = new Text();

private IntWritable one = new IntWritable();

//private CompositeKeyWritable cw = new CompositeKeyWritable();

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

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

String date = values[2].trim();

String month\_date\_year = date.substring(0,10);

date\_year.set(month\_date\_year);

//cw.setCount(1);

context.write(date\_year, one);

}

}

public static class AverageCrime\_Reducer extends Reducer<Text, IntWritable, Text, IntWritable>{

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

int count = 0;

for(IntWritable val : values){

count++;

}

context.write(key, new IntWritable(count));

}

}

public static class AverageCrime\_Mapper2 extends Mapper<Object, Text, NullWritable, CompositeKeyWritable>{

private CompositeKeyWritable cw = new CompositeKeyWritable();

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

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

String date\_year = values[0].trim();

int crime\_count = Integer.parseInt(values[1].trim());

cw.setCount(1);

cw.setAvg\_crime\_count(crime\_count);

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

}

}

public static class AverageCrime\_Reducer2 extends Reducer<NullWritable, CompositeKeyWritable, NullWritable, IntWritable>{

private IntWritable avg\_crime\_per\_day = new IntWritable();

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

int sum = 0;

int count=0;

int avg = 0;

for(CompositeKeyWritable val : values){

sum+= val.getCount() \* val.getAvg\_crime\_count();

count+= val.getCount();

}

System.out.println(sum);

System.out.println(count);

avg = sum/count;

System.out.println(avg);

avg\_crime\_per\_day.set(avg);

context.write(NullWritable.get(), avg\_crime\_per\_day);

}

}

public static class CompositeKeyWritable implements Writable{

private int count;

private int avg\_crime\_count;

public CompositeKeyWritable(){

}

public int getCount() {

return count;

}

public void setCount(int count) {

this.count = count;

}

public int getAvg\_crime\_count() {

return avg\_crime\_count;

}

public void setAvg\_crime\_count(int avg\_crime\_count) {

this.avg\_crime\_count = avg\_crime\_count;

}

@Override

public void write(DataOutput d) throws IOException {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

d.writeInt(count);

d.writeInt(avg\_crime\_count);

}

@Override

public void readFields(DataInput di) throws IOException {

//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

count = di.readInt();

avg\_crime\_count = di.readInt();

}

public String toString(){

return count + "\t" + avg\_crime\_count;

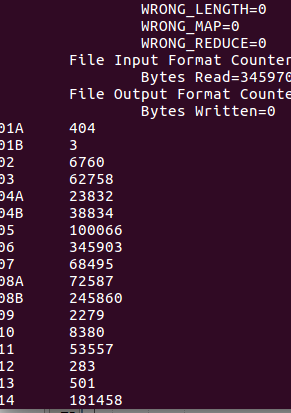
}

}

}

**9. Analysis using Map Reduce inbuilt Counters(Counter):**

**The dataset has a feature called FBI\_Code that indicates a number that corresponds to the crime category according the FBI's National Incident-Based Reporting System (NIBRS).**

* 1. Here I have used Map Reduce inbuilt counters to count the number of crime activities based on the FBI code that provides an efficient means of counting as compared to to the simple count variable used for counting.
  2. 
  3. Below is the code snippet:

public class Counters\_Main {

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

Configuration conf= new Configuration();

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

job.setJarByClass(Counters\_Main.class);

job.setMapperClass(Counters\_Mapper.class);

//job.setCombinerClass(AverageCrime\_Reducer.class);

//job.setReducerClass(AverageCrime\_Reducer.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(NullWritable.class);

//FileInputFormat.setInputDirRecursive(job, true);

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

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

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

if(code== 0){

for(Counter counter : job.getCounters().getGroup(Counters\_Mapper.FBI\_COUNTER\_GROUP)){

System.out.println(counter.getDisplayName() + "\t" + counter.getValue());

}

}

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

System.exit(code);

}

public static class Counters\_Mapper extends Mapper<Object, Text, NullWritable, NullWritable>{

public static final String FBI\_COUNTER\_GROUP = "FBI\_CODE";

public static final String UNKNOWN\_COUNTER = "Unknown";

public static final String NULL\_OR\_EMPTY\_COUNTER = "Null or Empty";

private String[] codeArray = new String[] {"08B","24","06","04B","15","03","07","08A","26","11","14","05","02","18","17","04A","20","01A","10","16","19","13","09","22","12","01B"};

private HashSet<String> codes = new HashSet<String>(Arrays.asList(codeArray));

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

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

String fbi\_code = values[14].trim();

//System.out.println(fbi\_code);

boolean unknown = true;

if(codes.contains(fbi\_code)){

context.getCounter(FBI\_COUNTER\_GROUP, fbi\_code).increment(1);

unknown = false;

}

if(unknown){

context.getCounter(FBI\_COUNTER\_GROUP, UNKNOWN\_COUNTER).increment(1);

}

}

}

}