

Crimes in Chicago

Eploratory analysis using Hadoop

Harshad Pardeshi | Advance Database Management System | April 26, 2017

Contents

[Summary 2](#_Toc480969951)

[Approach for the problem statement 2](#_Toc480969952)

[Dataset 2](#_Toc480969953)

[Analysis performed 4](#_Toc480969954)

[1. Join dataset 4](#_Toc480969955)

[2. Partition data by year 4](#_Toc480969956)

[3. Bin yearly data into months 5](#_Toc480969957)

[4. For each year find the frequency of the crime. 6](#_Toc480969958)

[5. For each month find the frequency of the crime. 6](#_Toc480969959)

[6. For each year find the frequency of the crimes for each district. 7](#_Toc480969960)

[7. For each month find the most common location for crimes. 8](#_Toc480969961)

[8. For each year find the ratio of total domestic crimes vs arrested. 9](#_Toc480969962)

[Visualization 9](#_Toc480969963)

[Code 11](#_Toc480969964)

[Join Datasets 11](#_Toc480969965)

[Partition Data 17](#_Toc480969966)

[Binning By month 20](#_Toc480969967)

[Counting arrests 24](#_Toc480969968)

[Frequency By District (top 25) 27](#_Toc480969969)

[Frequency by Location 32](#_Toc480969970)

# Summary

Chicago is one of the city that has highest number of crimes. This data is available to public for study purpose only. This data sounds interesting as it is possible to derive many patterns and conclusion this data. Further this analysis can be further used for visualization and multiple inferences can be understood better. Based on these inferences we can suggest better strategies and laws to reduce these crime rates and thus we can ensure public safety. We can go a step further to predict where or when a crime will be committed based on some machine learning algorithms.

### Approach for the problem statement

We are using Hadoop map reduce to find some meaningful insights from this data. First of all, data set which we are going to analyze contains many numeric values (Id/Key representation); so we need to consider other data set as well to completely make sense of this huge data. Following steps are performed:

* Join the crime data with district and IUCR data to get names of districts and crime categories.
* Partition data year wise to understand annual behavior of data.
* Bin yearly data month wise to analyze seasonal variations in monthly data.
* For each year find the frequency of the crime.
* For each month find the frequency of the crime.
* For each year find the frequency of the crimes for each district.
* For each month find the most common location for crimes.
* For each year find the ratio of total crimes vs arrested.

# Dataset

**Link to dataset**: <https://www.kaggle.com/currie32/crimes-in-chicago>

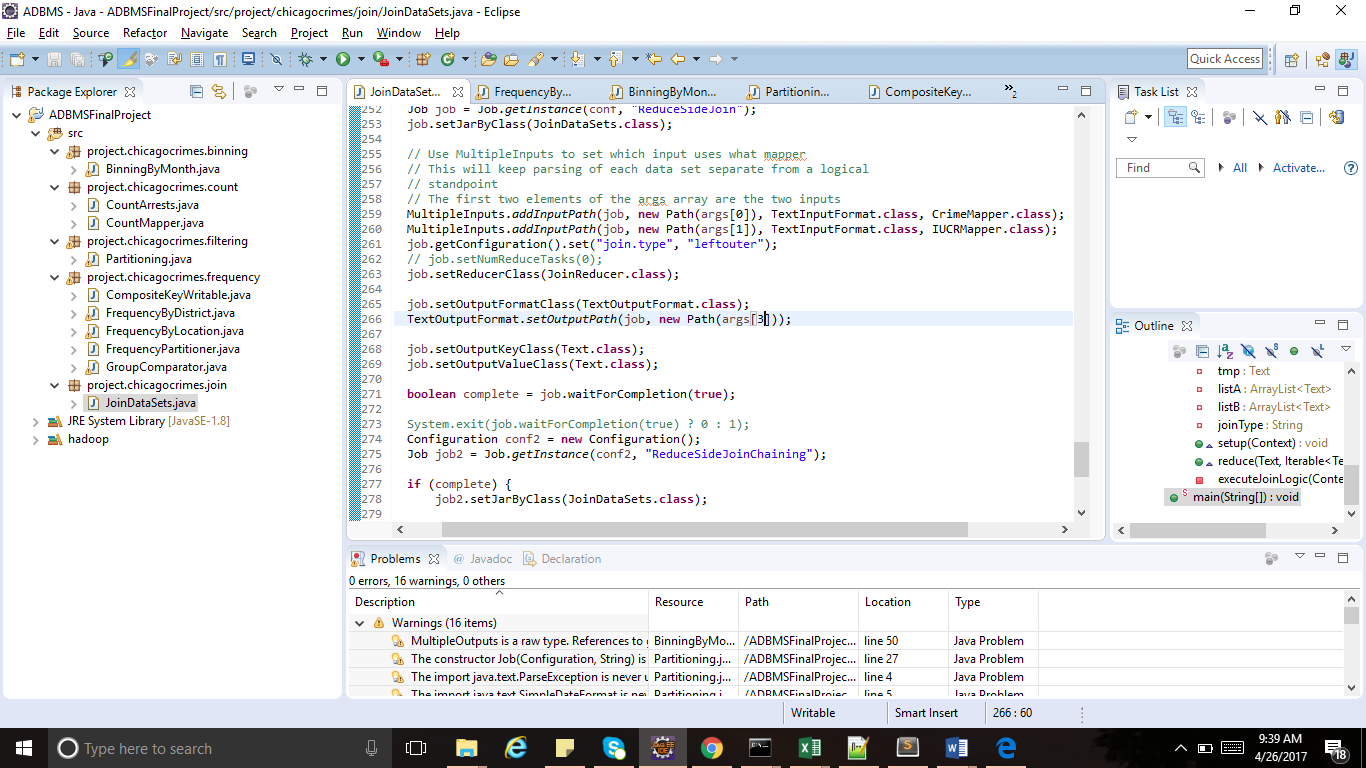
This dataset reflects reported incidents of crime that occurred in the City of Chicago from 2001 to present, minus the most recent seven days. Data is extracted from the Chicago Police Department's CLEAR (Citizen Law Enforcement Analysis and Reporting) system. In order to protect the privacy of crime victims, addresses are shown at the block level only and specific locations are not identified.

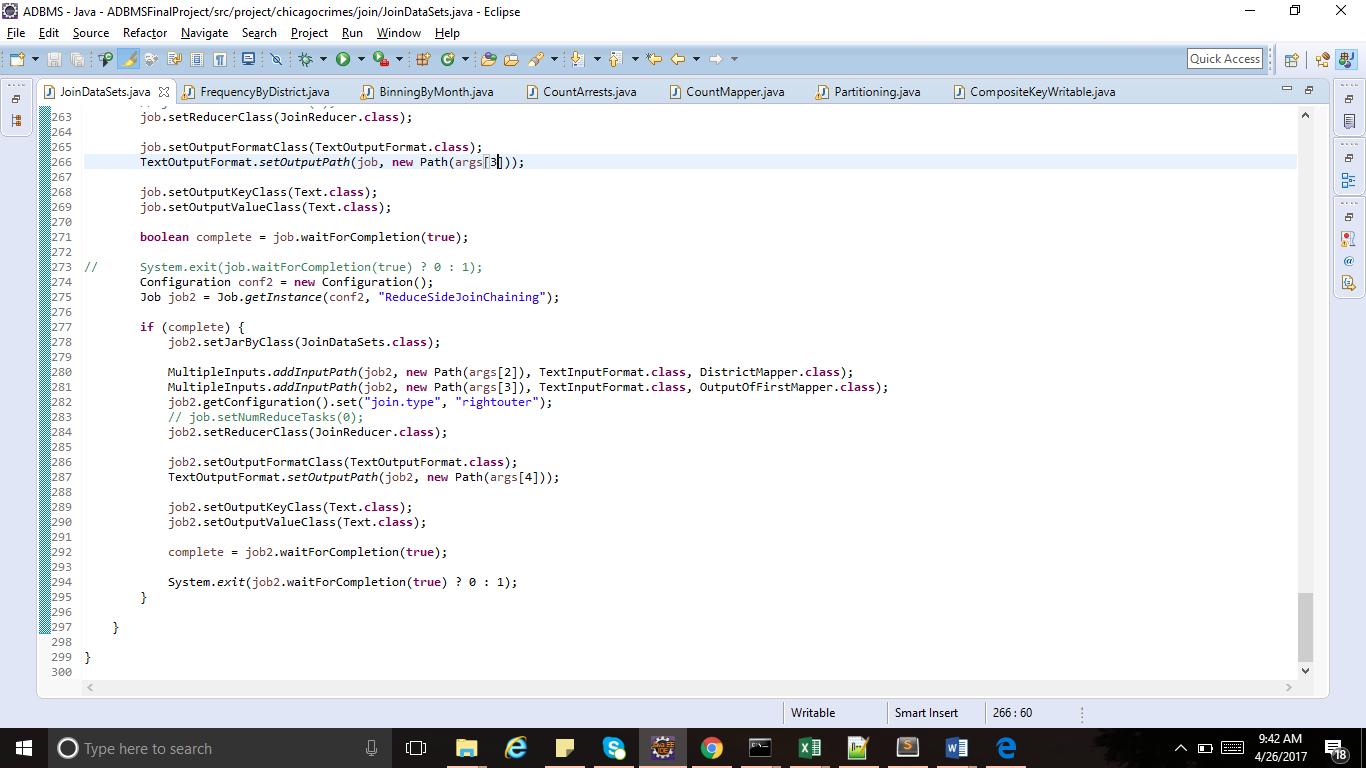
This dataset contains the following columns:

* **ID** - Unique identifier for the record.
* **Case Number** - The Chicago Police Department RD Number (Records Division Number), which is unique to the incident.
* **Date** - Date when the incident occurred. this is sometimes a best estimate.
* **Block** - The partially redacted address where the incident occurred, placing it on the same block as the actual address.
* **IUCR** - The Illinois Uniform Crime Reporting code. This is directly linked to the Primary Type and Description.
* **Primary Type** - The primary description of the IUCR code.
* **Description** - The secondary description of the IUCR code, a subcategory of the primary description.
* **Location Description** - Description of the location where the incident occurred.
* **Arrest** - Indicates whether an arrest was made.
* **Domestic** - Indicates whether the incident was domestic-related as defined by the Illinois Domestic Violence Act.
* **Beat** - Indicates the beat where the incident occurred. A beat is the smallest police geographic area – each beat has a dedicated police beat car. Three to five beats make up a police sector, and three sectors make up a police district.
* **District** - Indicates the police district where the incident occurred.
* **Ward** - The ward (City Council district) where the incident occurred.
* **Community Area** - Indicates the community area where the incident occurred. Chicago has 77 community areas.
* **FBI Code** - Indicates the crime classification as outlined in the FBI's National Incident-Based Reporting System (NIBRS).
* **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.
* **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.
* **Year** - Year the incident occurred.
* **Updated On** - Date and time the record was last updated.
* **Latitude** - The latitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.
* **Longitude** - The longitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.
* **Location** - The location where the incident occurred in a format that allows for creation of maps and other geographic operations on this data portal. This location is shifted from the actual location for partial redaction but falls on the same block.

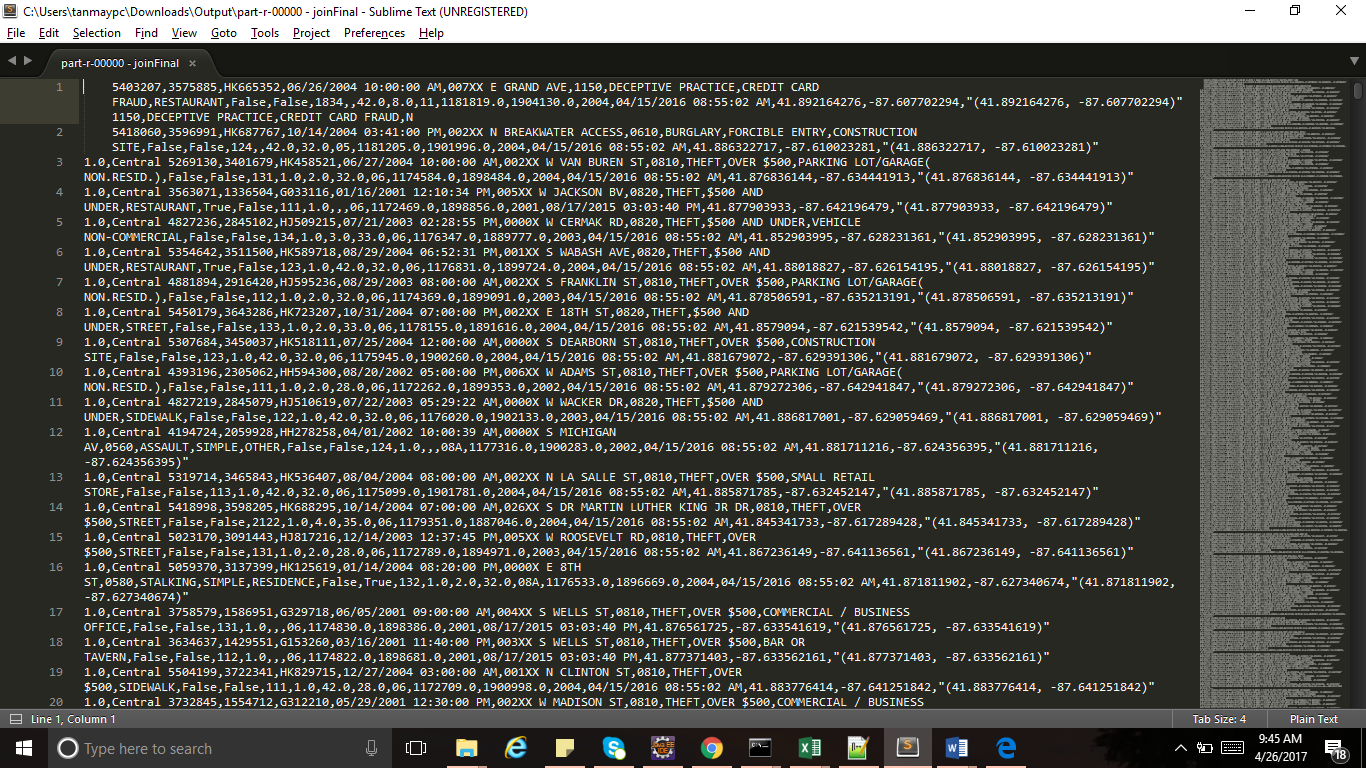
# Analysis performed

### Join dataset

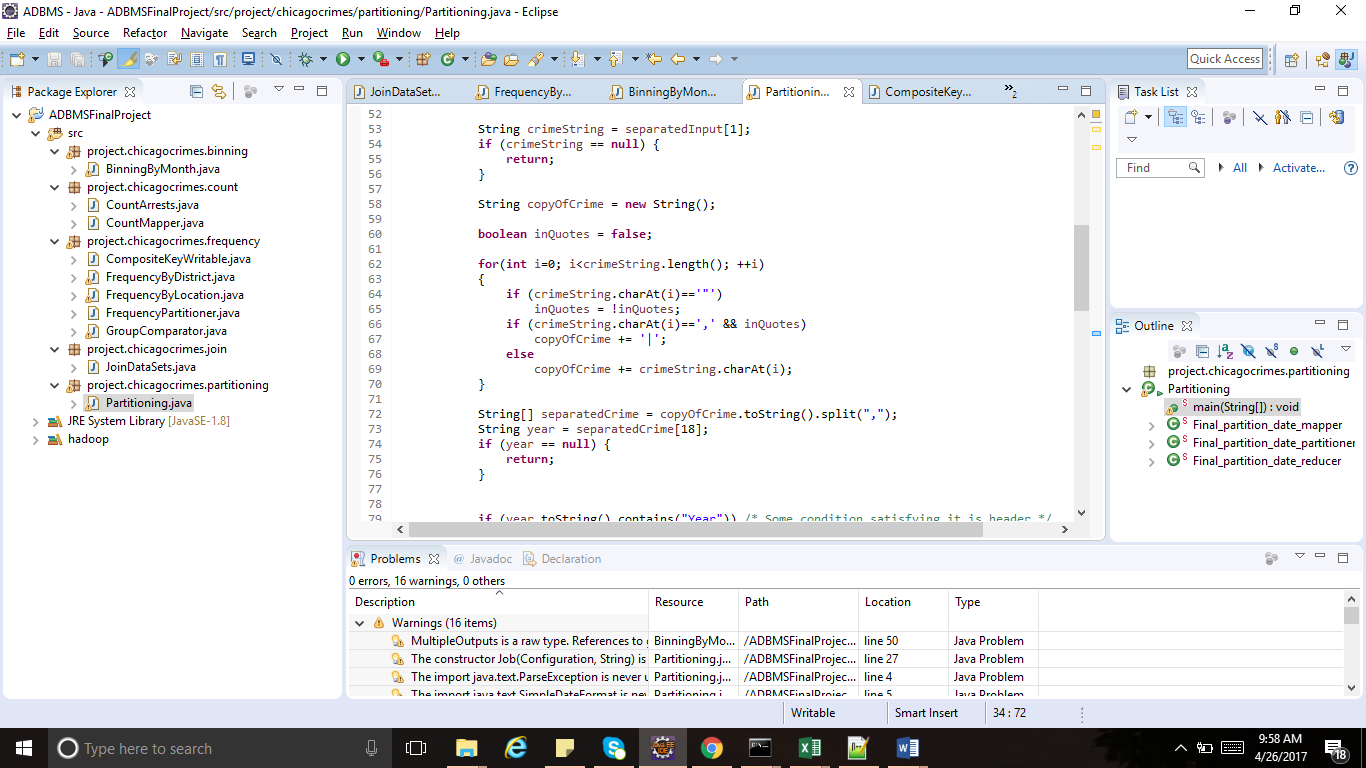
The original dataset contains key/ID columns which can be joined to other datasets to get more useful information. Firstly, original data is combined with [IUCR](https://data.cityofchicago.org/Public-Safety/Chicago-Police-Department-Illinois-Uniform-Crime-R/c7ck-438e/data) (Illinois Uniform Crime Reporting (IUCR) Codes) dataset. This is performed by using map reduce join pattern. A left join is performed to select all the records in crime dataset. 

Output of this operation is given to another map reduce job which will join the chicago district dataset with this output. This achieved through map reduce job chaining. 

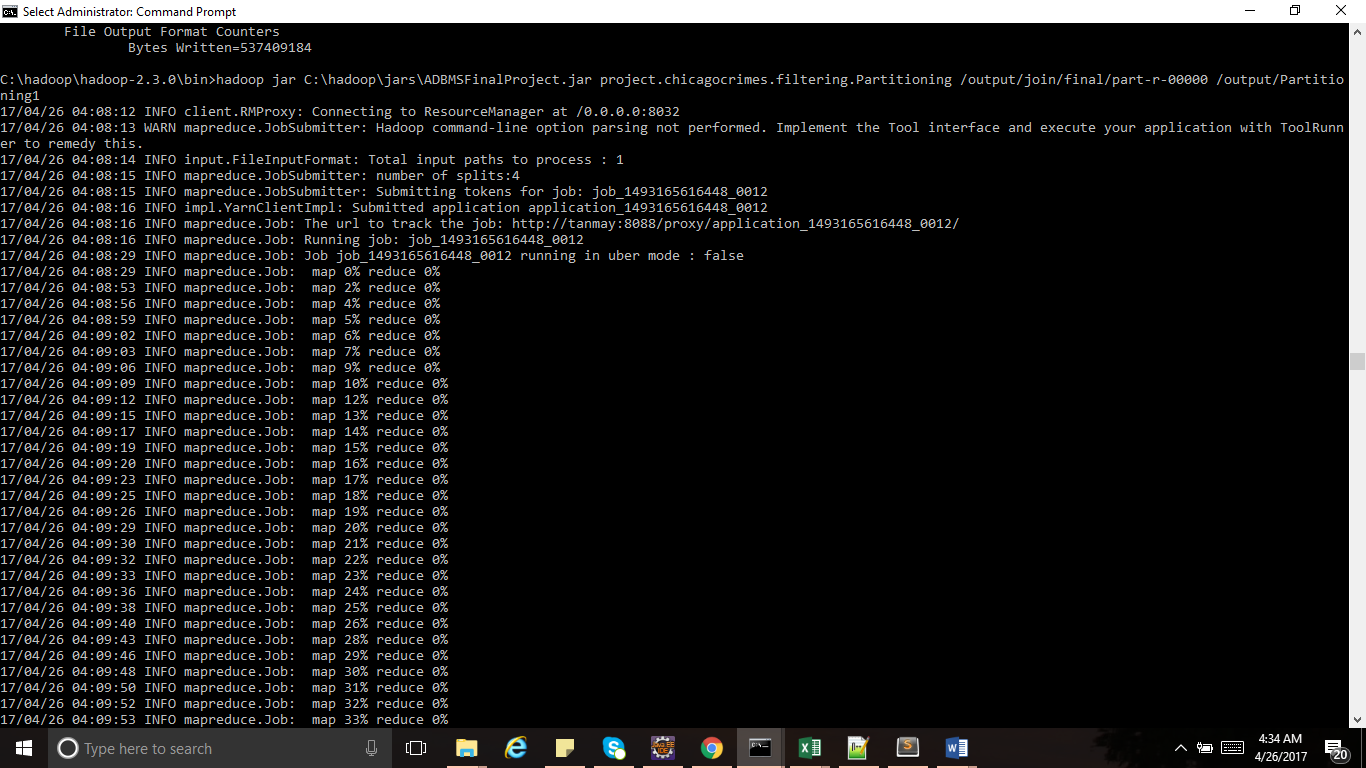
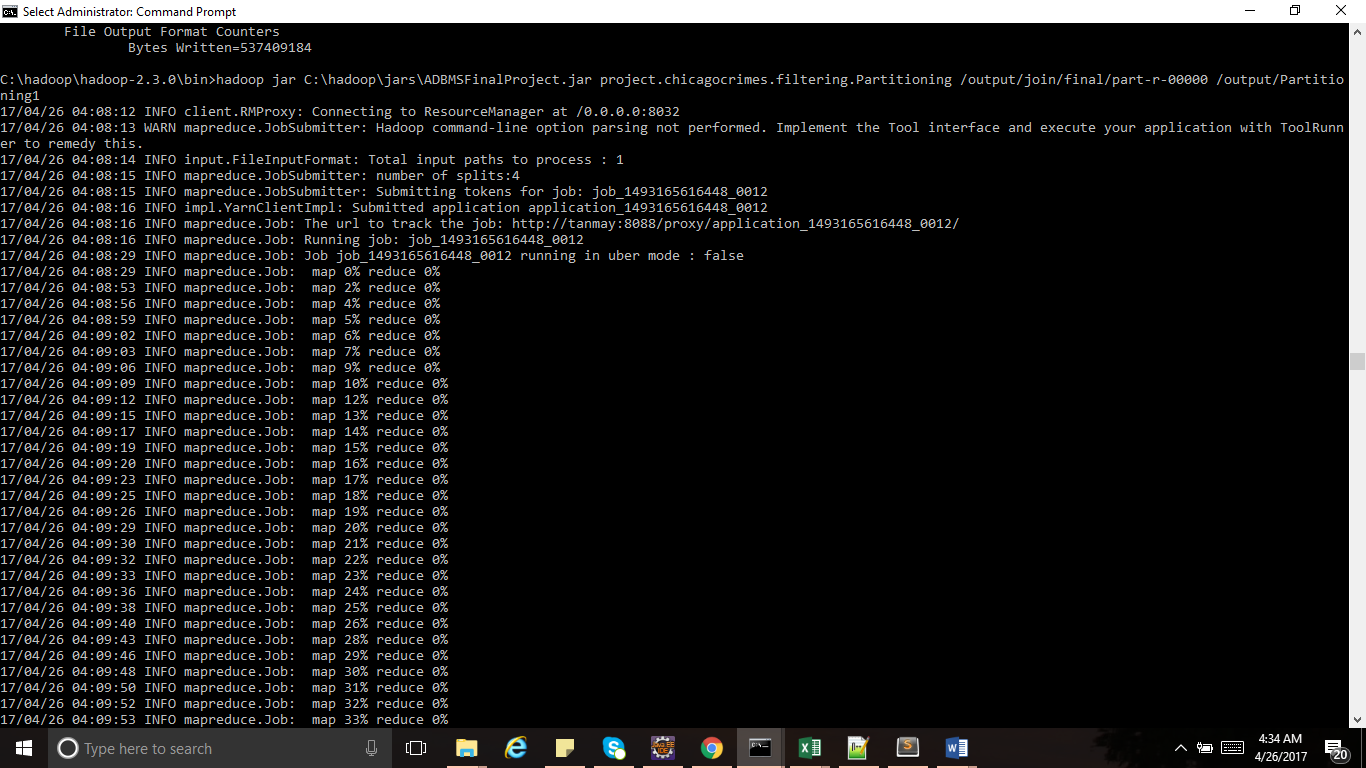
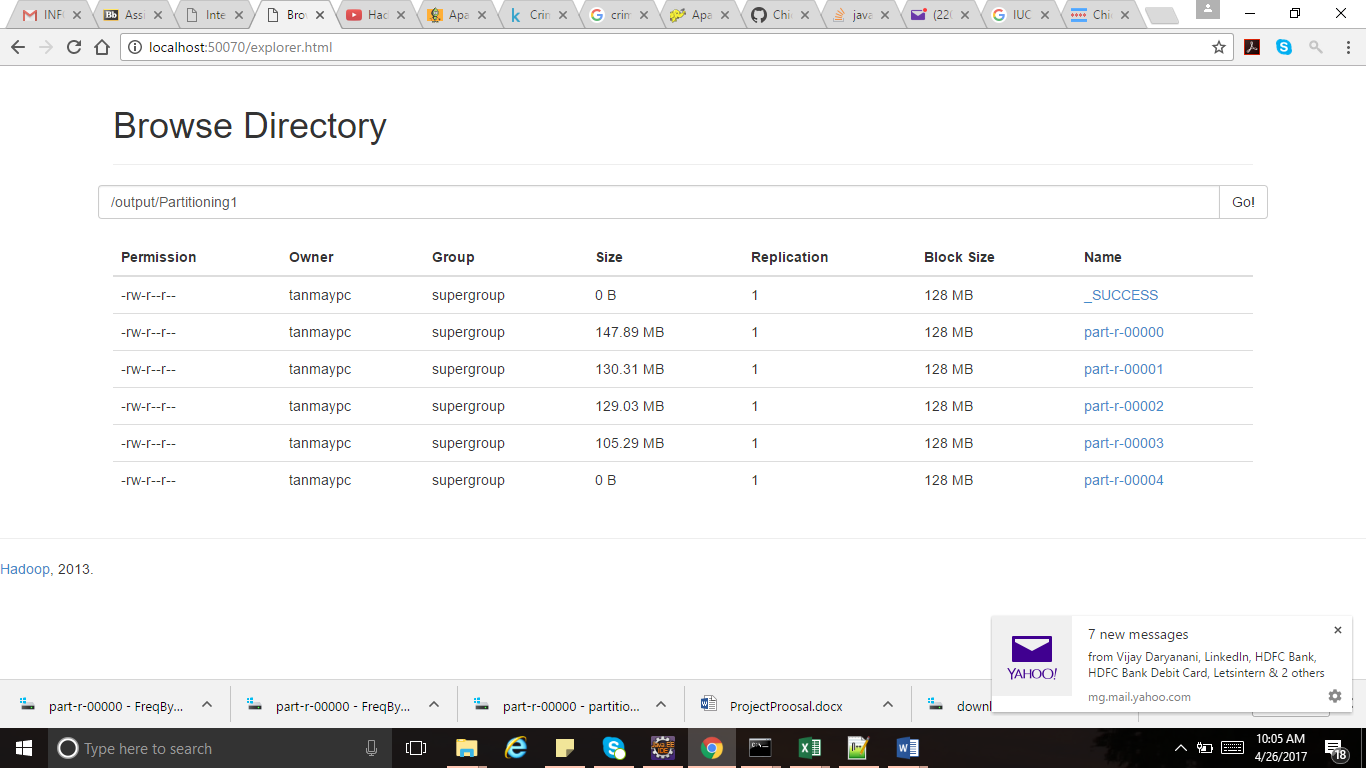
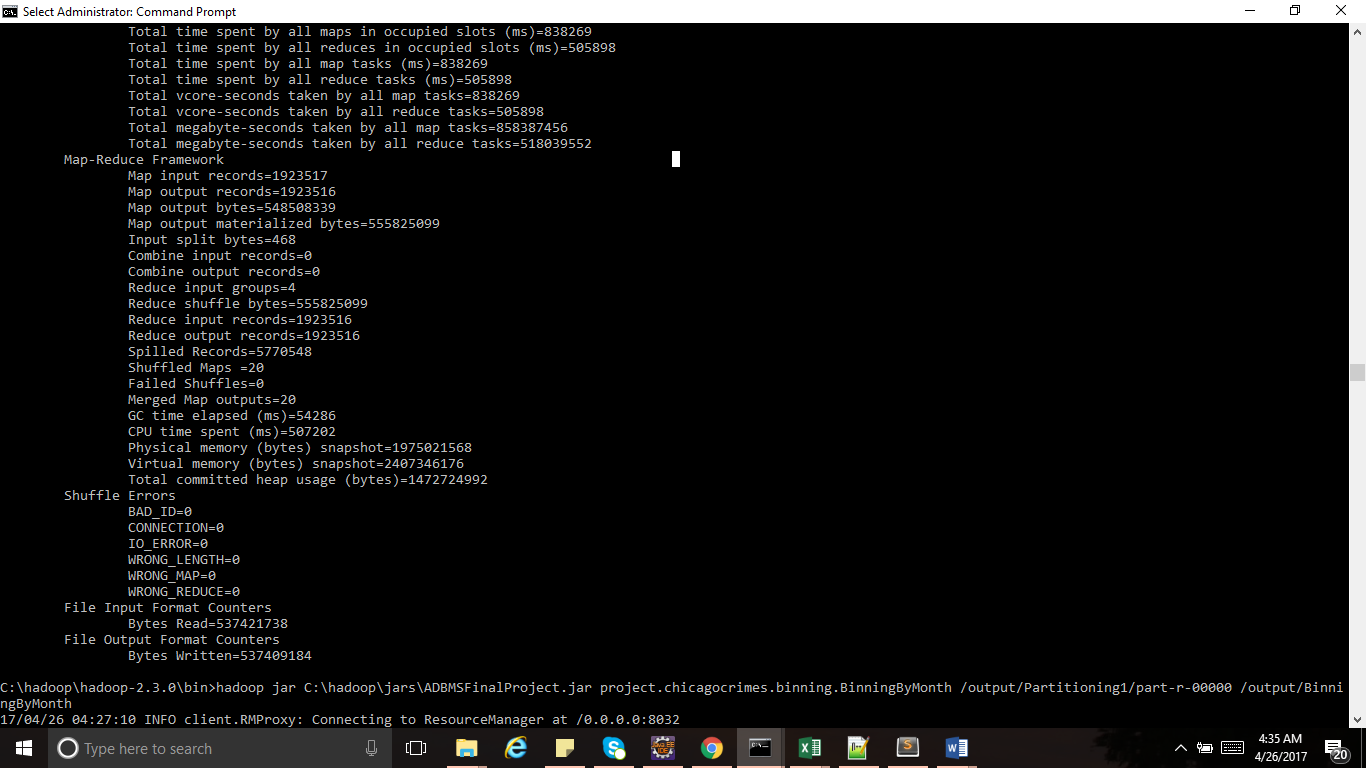
Output of this operation looks something like this (tab seperated columns from 3 files).



### Partition data by year

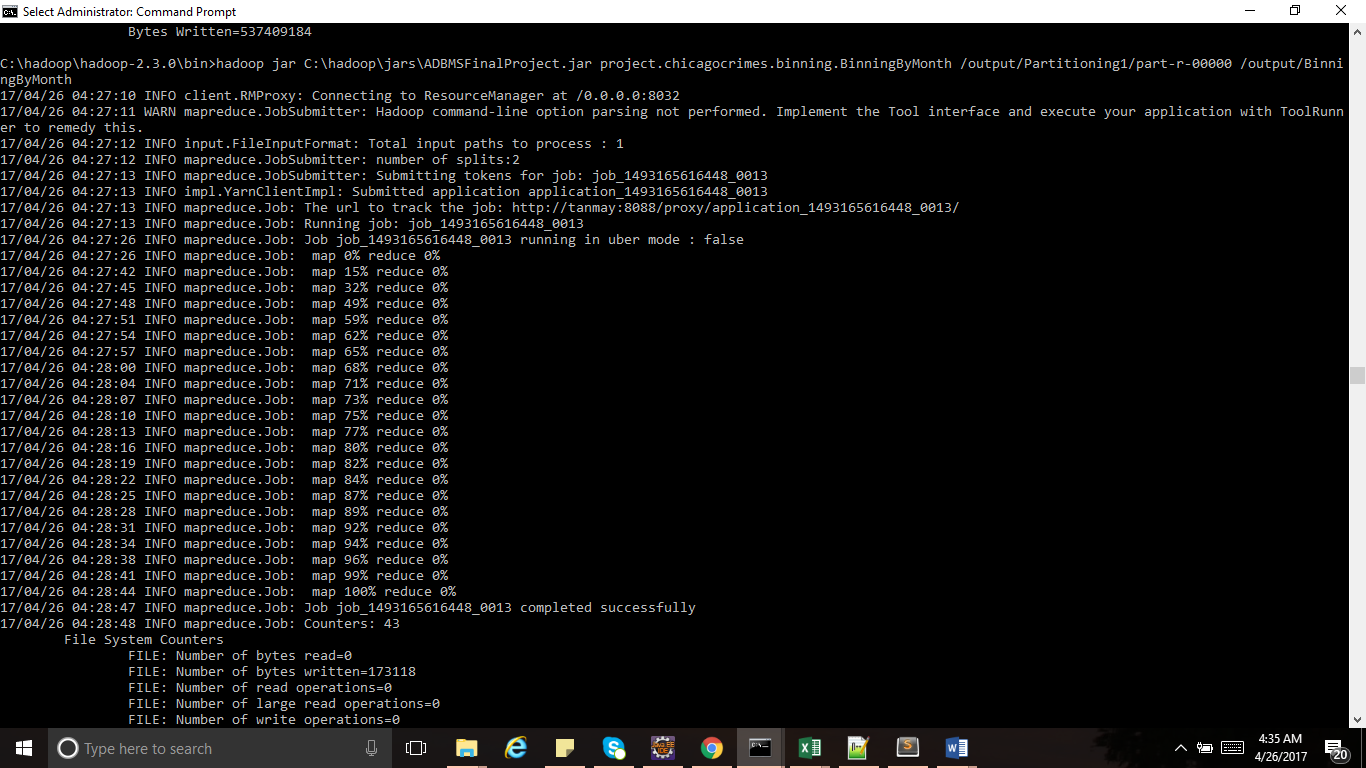
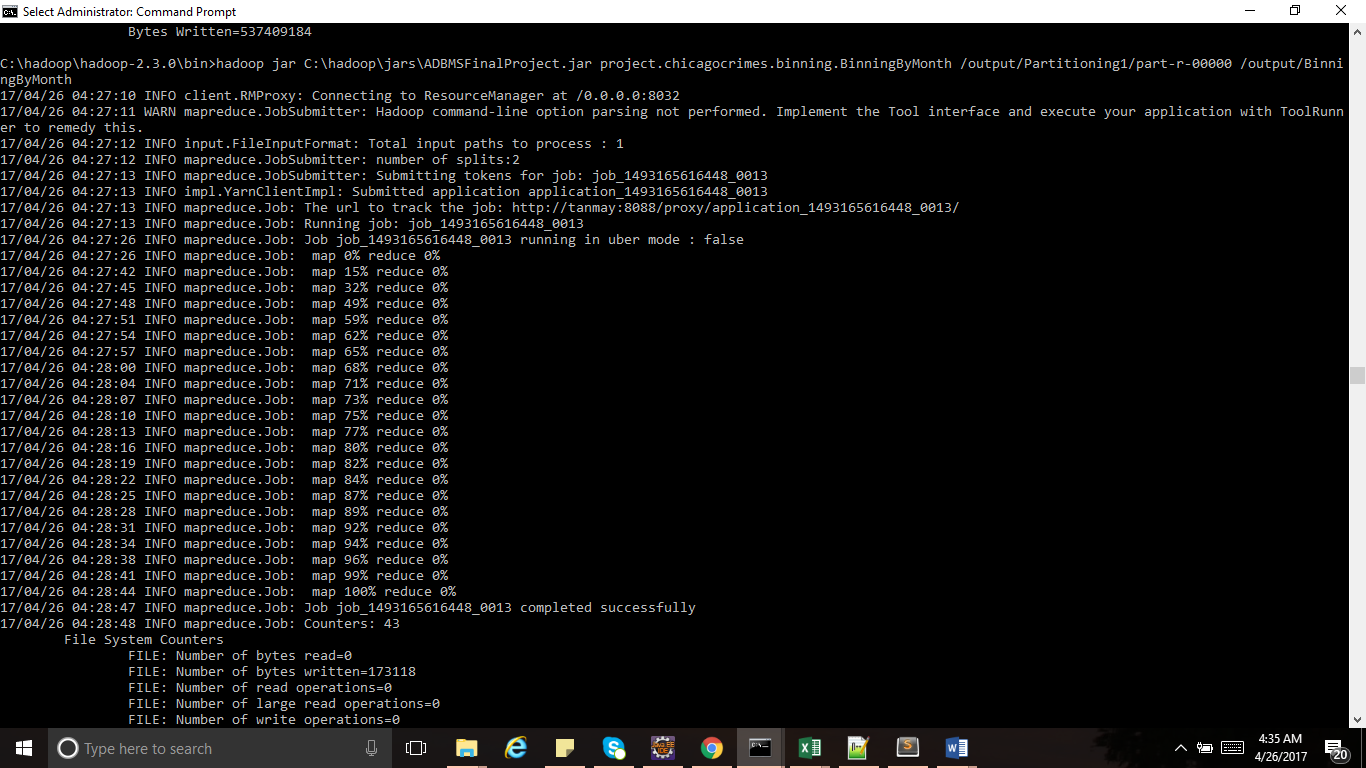
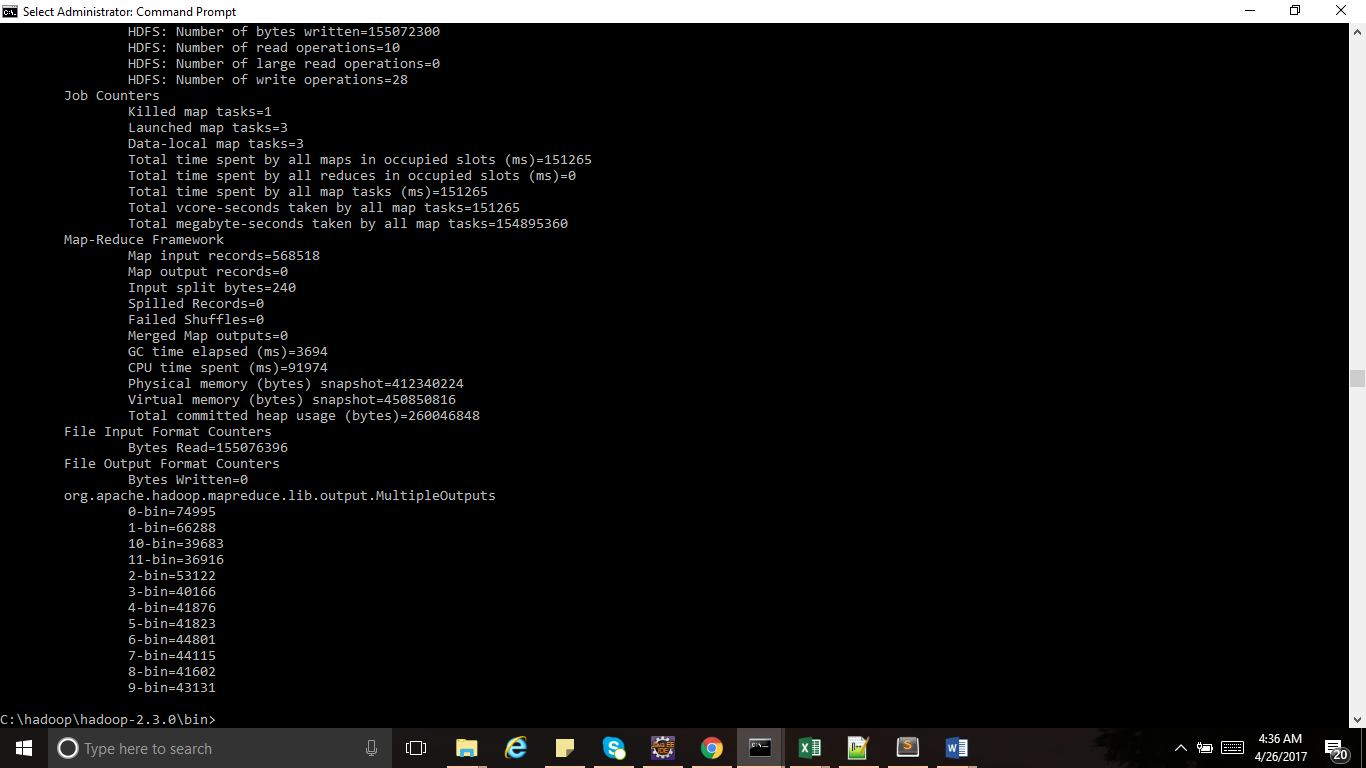
Further, merged data is partitioned by year to analyze crimes in a particular year and compare it with other years. To get the year in which crime was committed, we must parse the string (comma separated); but some columns contains string with comma which makes it difficult to read the string fixed format. Thus, we replace the commas inside a column with a pipe symbol.

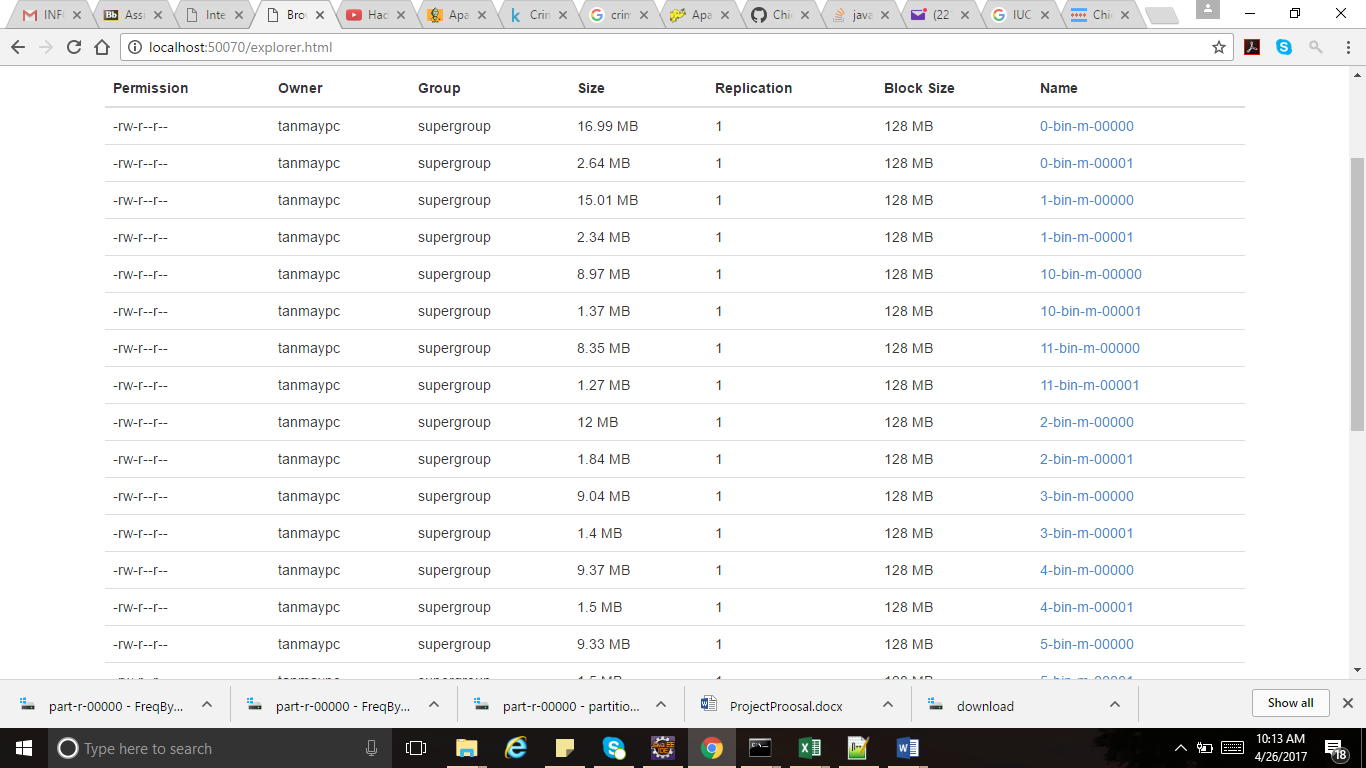
After this cleaning , data is partitioned into different files by year.

### Bin yearly data into months

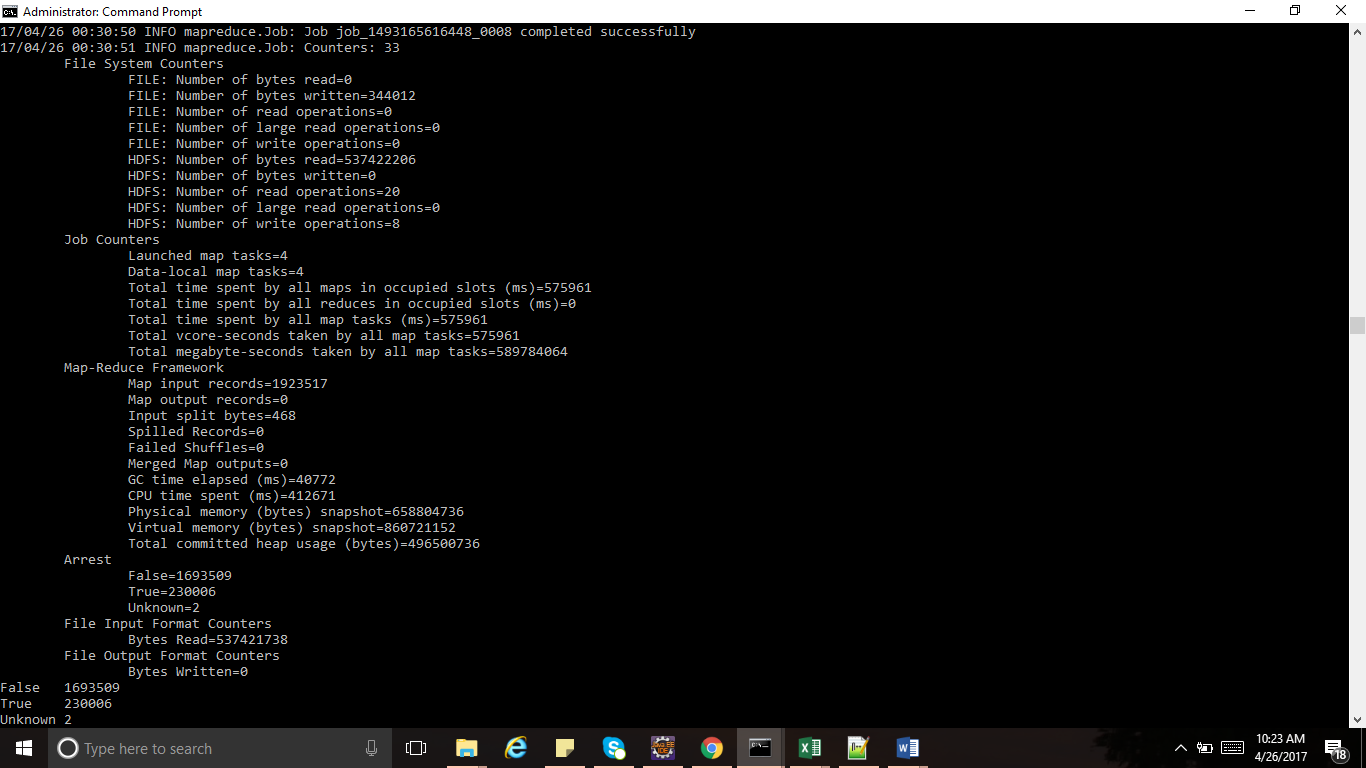
Further, we take data from each year and bin it into months. Twelve bins are created for each year. Binning pattern is implemented to achieve this task.



### For each year find the frequency of the crime.

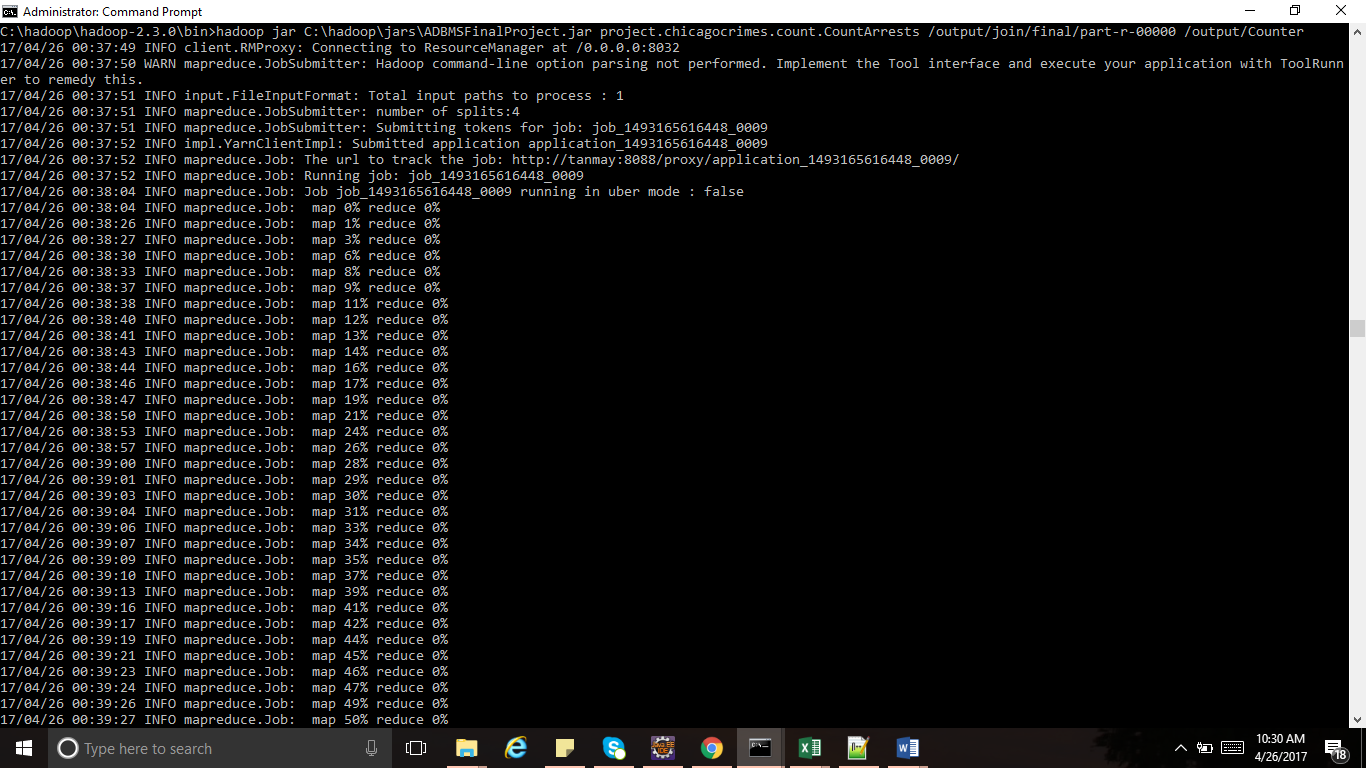
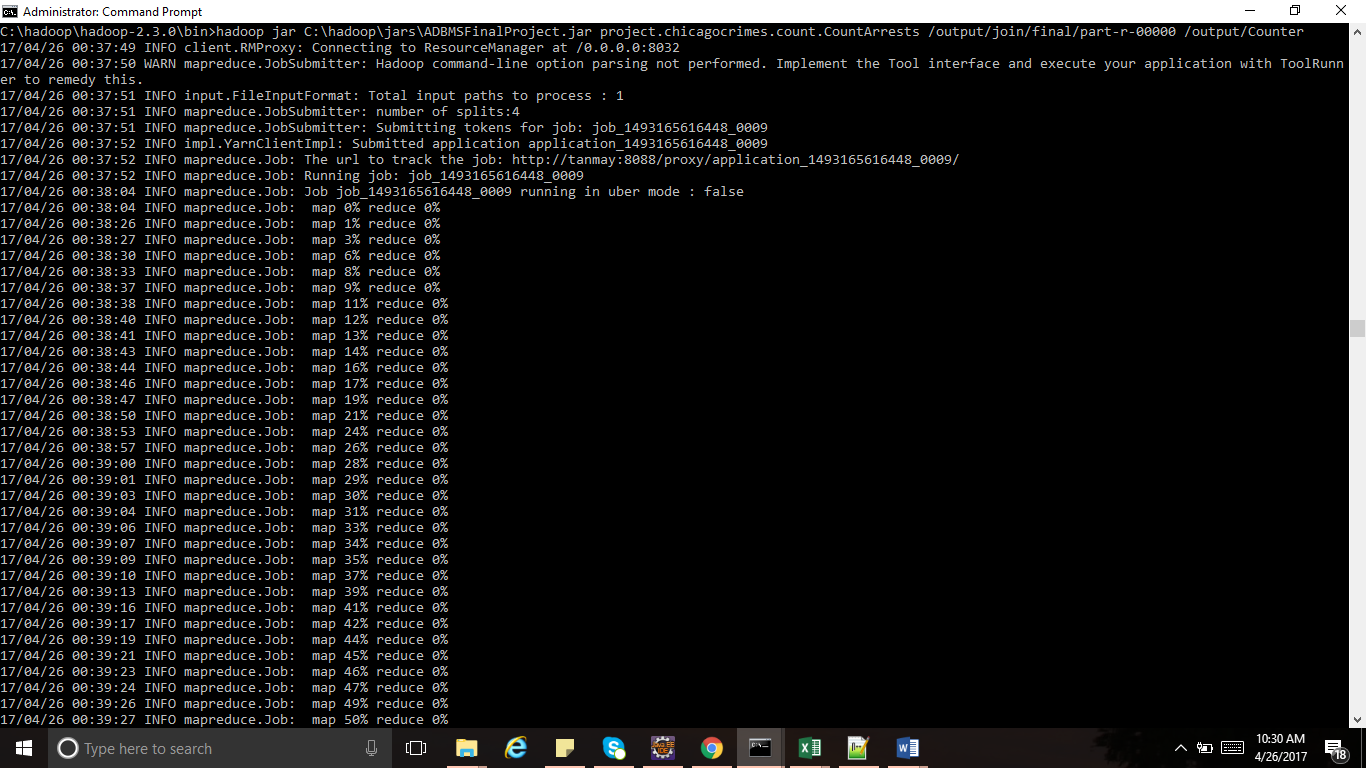
We implement count with counters technique to analyze domestic crimes and number of criminals that were arrested.

This is done for every year and we hav found tha there is no uniform pattern in increase or decrease in crimes over the year. But we can conlude that the 2008 and 2006 had highest number of crimes and there is significant decrease from 2011 in crime rates and this rate keep on decreasing linearly after 2011.

### For each month find the frequency of the crime.

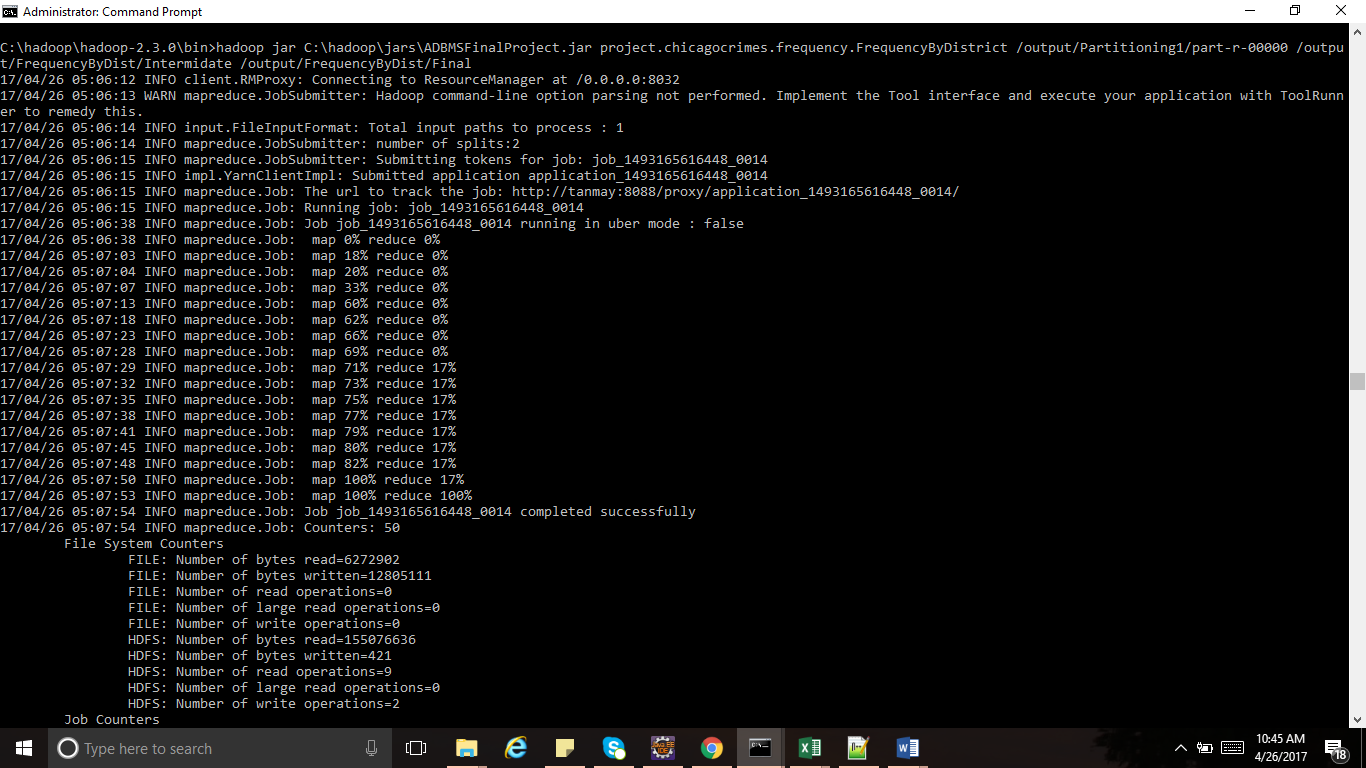
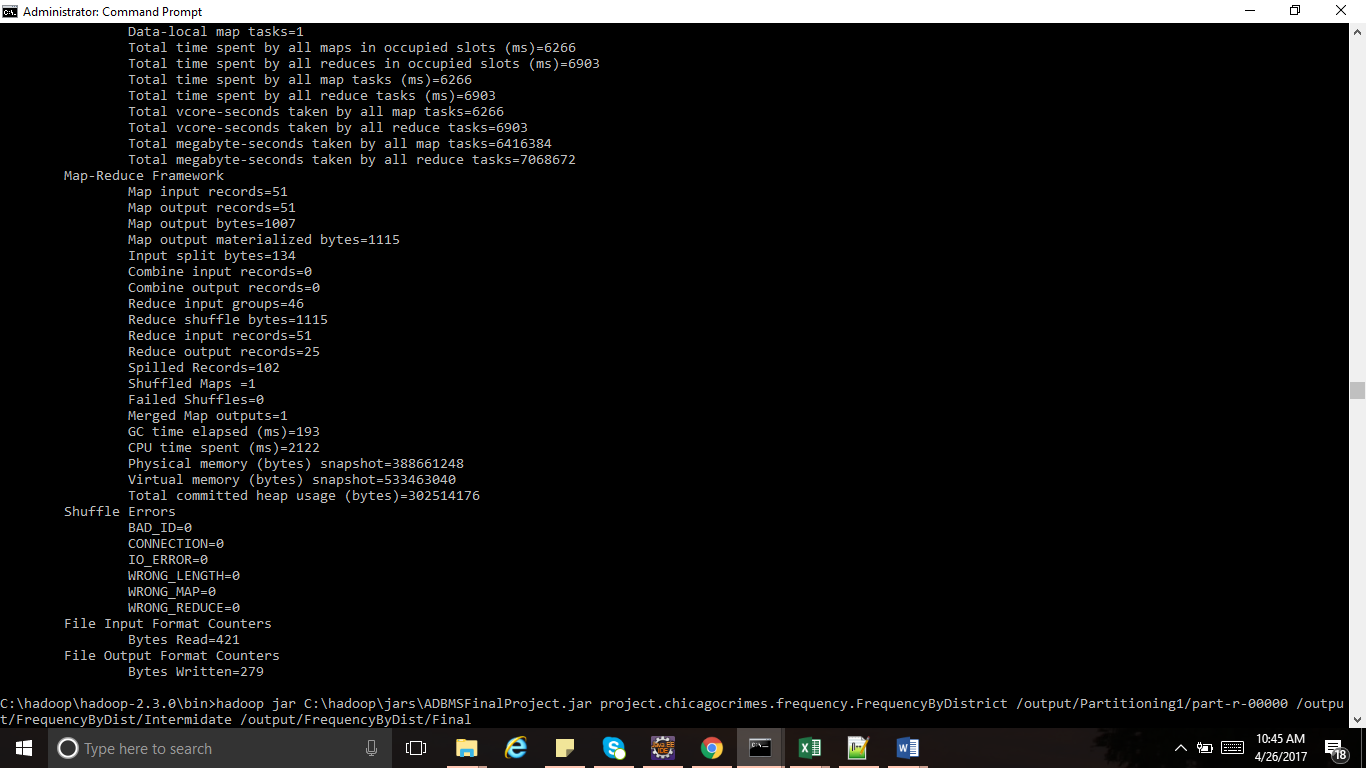
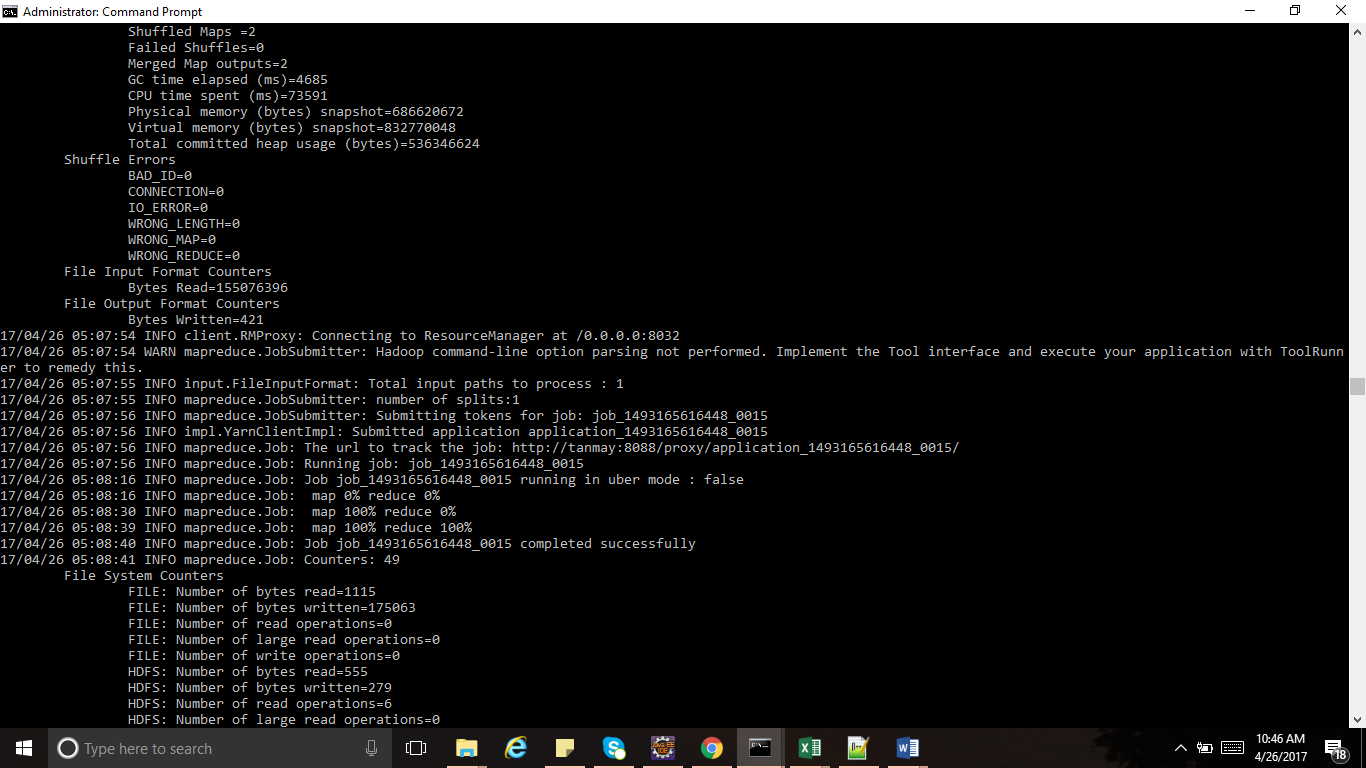
Again, we implement count with counters technique to analyze domestic crimes and number of criminals that were arrested in each month.

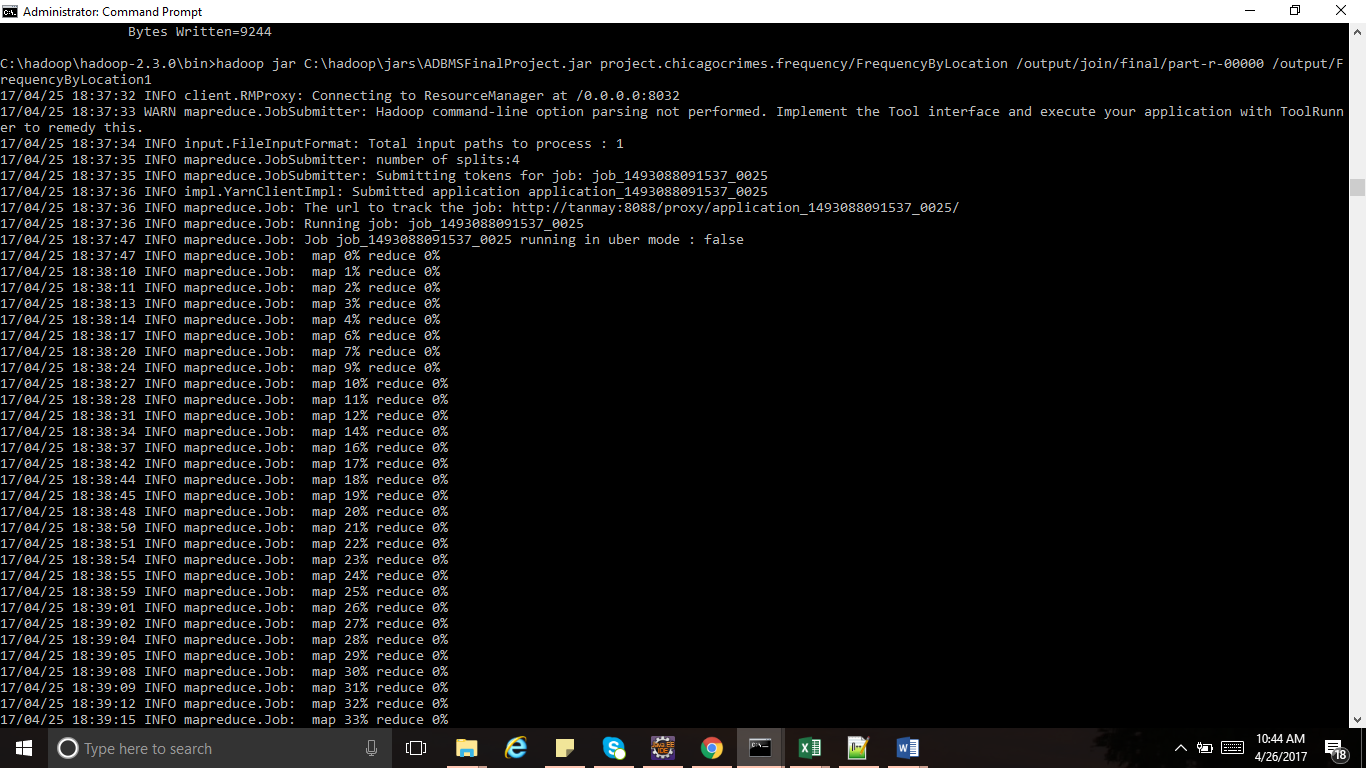
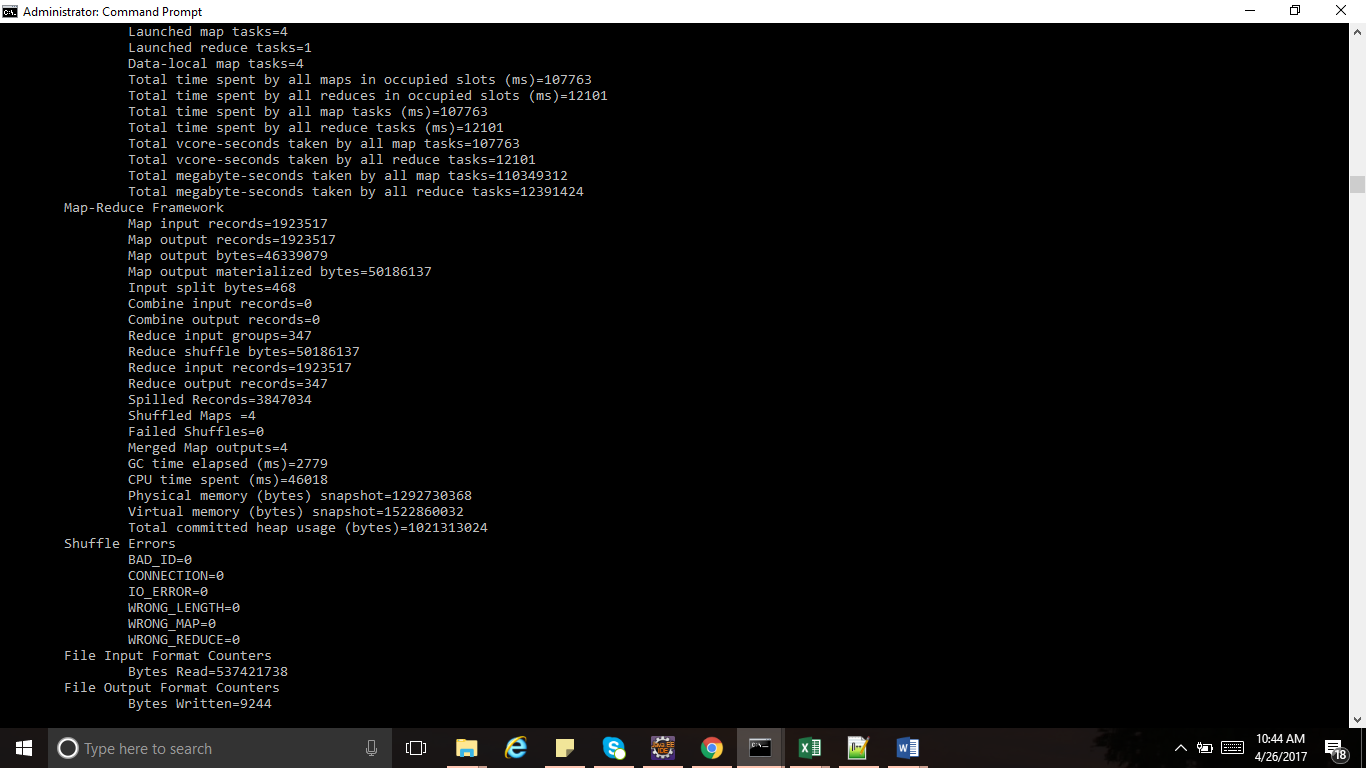
By observing these outputs we can conclude that there are significant amount of increase in crimes in the month of July and febraury and december have least crime rates.

### For each year find the frequency of the crimes for each district.

We have implemented summarization pattern to find the top 25 district with highest crime rates. First, we find the frequency on crimes in each district and then sort the output by frequency and select the top 25 districts with highest crimes. Similarly, we also find top 25 districts with least number of crimes. These districts can be considered as safe areas to live.

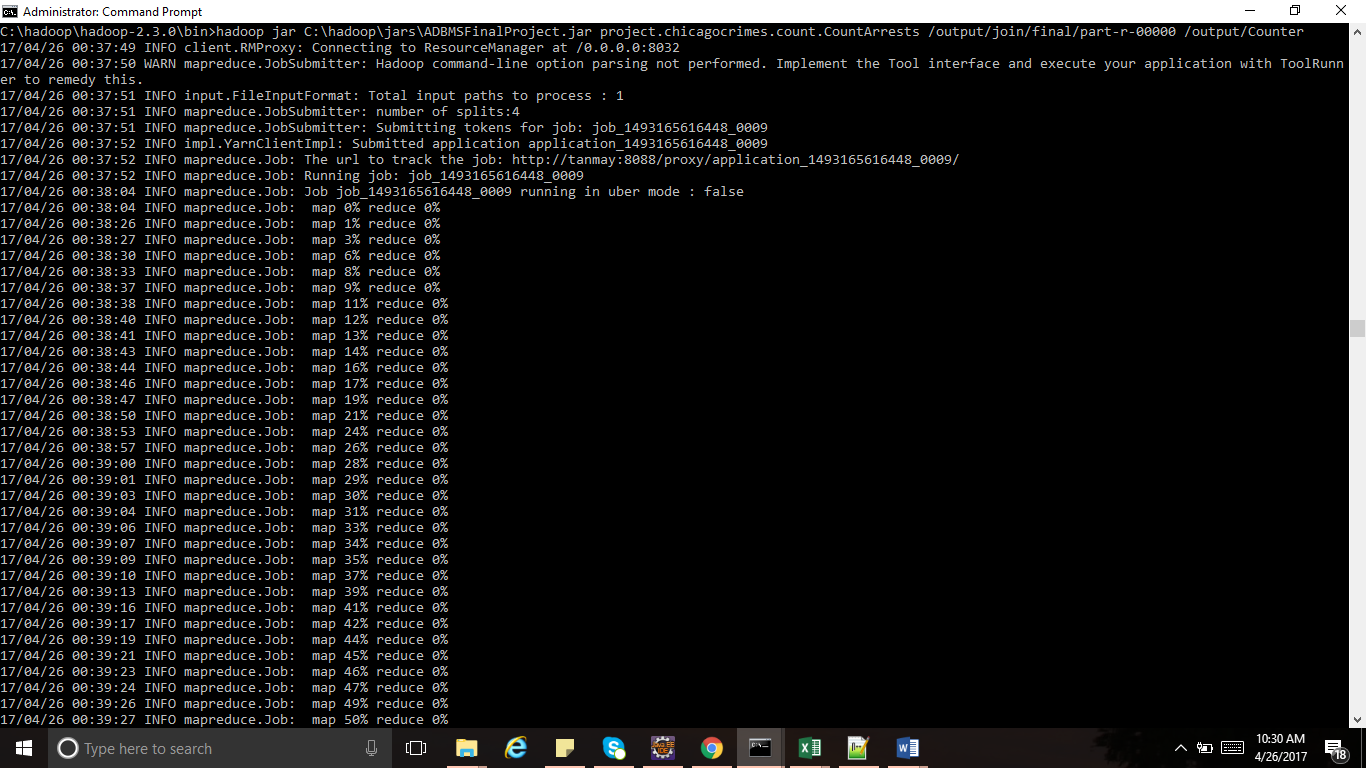
### For each month find the most common location for crimes.

We have implemented summarization pattern to observe places with highest crime rates. First, we find the frequency on crimes in each location and then sort the output by frequency.  

We observe that Resident is the most common location for crime.

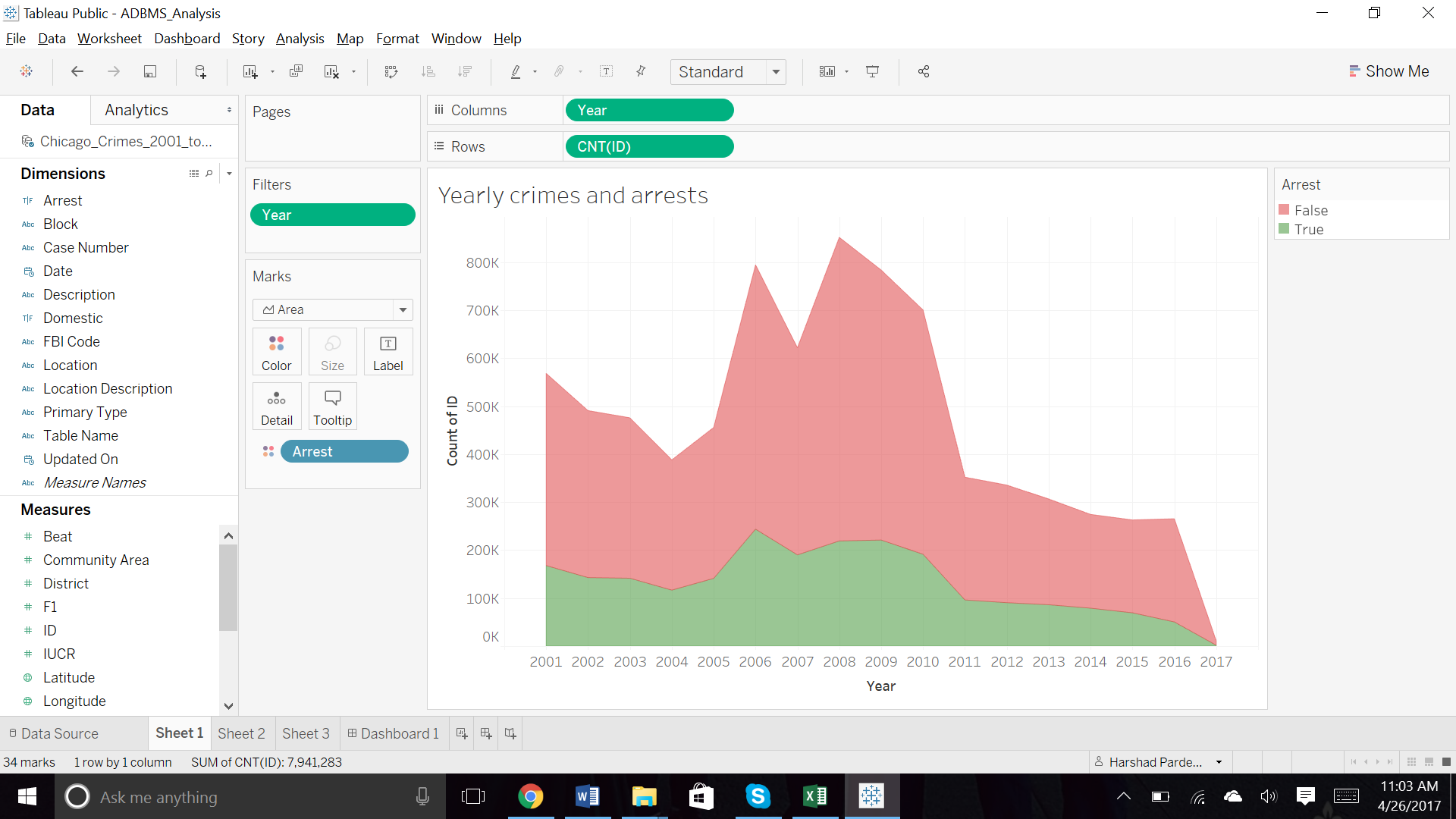
### For each year find the ratio of total domestic crimes vs arrested.

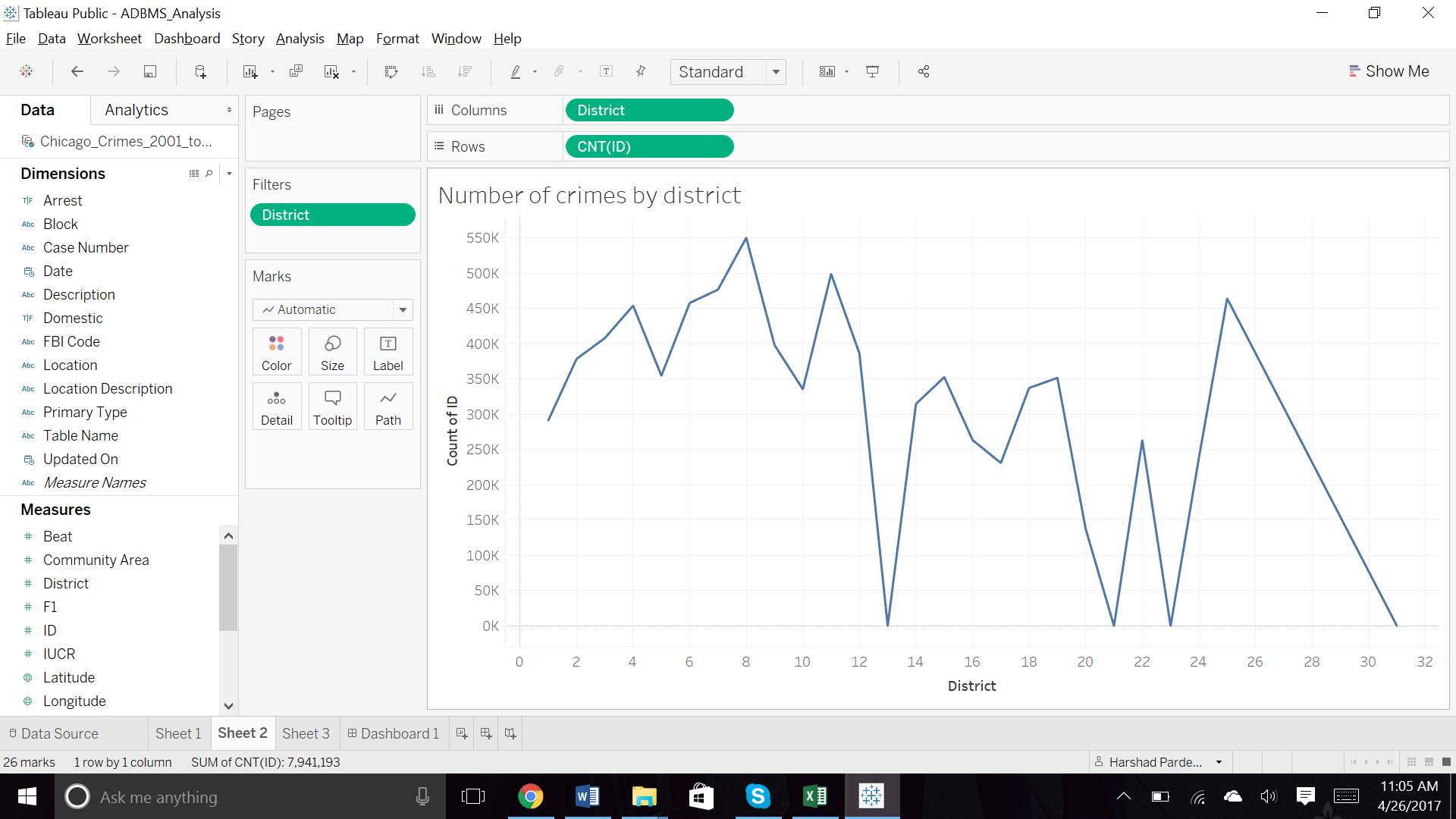
We calculate the number of domestic crimes which were arrested. We implement counting with counters wo calculate number of crimes where domestic crimes are true and calculate number of arrests for these crimes.

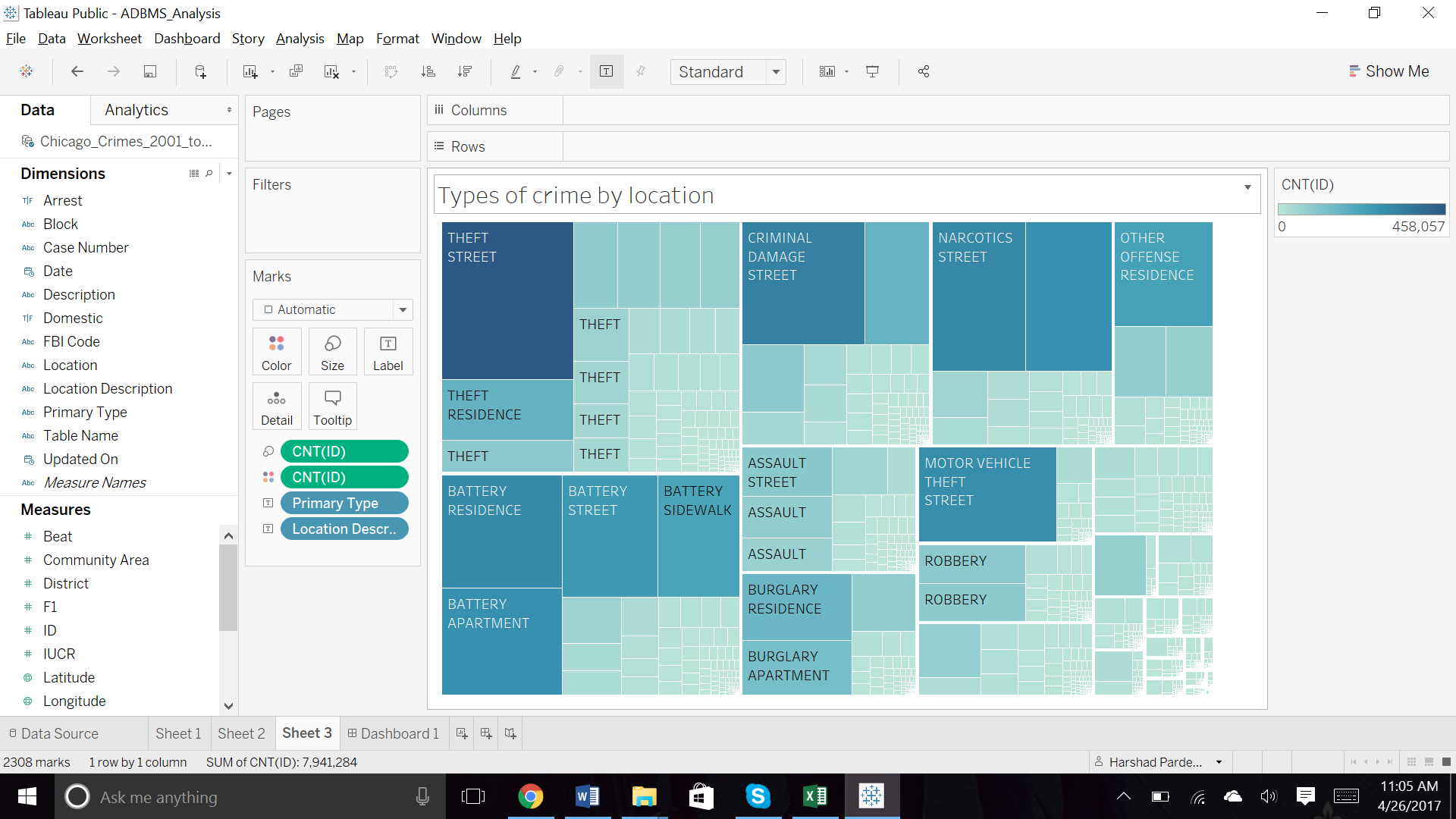


# Visualization

All the above analysis are visualized using Tableau.











These visualizations can be found on the following link:

<https://public.tableau.com/profile/publish/Chicago_2000_0/Dashboard1#!/publish-confirm>

<https://public.tableau.com/profile/publish/ADBMS_Analysis/Dashboard1#!/publish-confirm>

# Code

### Join Datasets

**package** project.chicagocrimes.join;

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

**import** java.io.IOException;

**import** java.util.ArrayList;

**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.Mapper;

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

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

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

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

/\*\*

\*

\* **@author** harshad

\*/

**public** **class** JoinDataSets {

**public** **static** **class** CrimeMapper **extends** Mapper<Object, Text, Text, Text> {

**private** Text outkey = **new** Text();

**private** Text outvalue = **new** Text();

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

// Parse the input string into a nice map

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

String iucrnNumber = separatedInput[5];

// String bookName = separatedInput[1];

**if** (iucrnNumber == **null**) {

**return**;

}

// The foreign join key is the user ID

outkey.set(iucrnNumber);

// Flag this record for the reducer and then output

outvalue.set("A|" + value);

context.write(outkey, outvalue);

}

}

**public** **static** **class** IUCRMapper **extends** Mapper<Object, Text, Text, Text> {

**private** Text outkey = **new** Text();

**private** Text outvalue = **new** Text();

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

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

String iucrNumber = separatedInput[0];

// String rating = separatedInput[2];

**if** (iucrNumber == **null**) {

**return**;

}

// The foreign join key is the user ID

outkey.set(iucrNumber);

// Flag this record for the reducer and then output

outvalue.set("B|" + value);

context.write(outkey, outvalue);

}

}

**public** **static** **class** DistrictMapper **extends** Mapper<Object, Text, Text, Text> {

**private** Text outkey = **new** Text();

**private** Text outvalue = **new** Text();

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

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

String districtId = separatedInput[0];

// String rating = separatedInput[2];

**if** (districtId == **null**) {

**return**;

}

// The foreign join key is the user ID

outkey.set(districtId);

// Flag this record for the reducer and then output

outvalue.set("A|" + value);

context.write(outkey, outvalue);

}

}

**public** **static** **class** OutputOfFirstMapper **extends** Mapper<Object, Text, Text, Text> {

**private** Text outkey = **new** Text();

**private** Text outvalue = **new** Text();

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

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

String crimeString = separatedInput[0];

**if** (crimeString == **null**) {

**return**;

}

String copyOfCrime = **new** String();

**boolean** inQuotes = **false**;

**for**(**int** i=0; i<crimeString.length(); ++i)

{

**if** (crimeString.charAt(i)=='"')

inQuotes = !inQuotes;

**if** (crimeString.charAt(i)==',' && inQuotes)

copyOfCrime += '|';

**else**

copyOfCrime += crimeString.charAt(i);

}

String[] separatedCrime = copyOfCrime.toString().split(",");

String districtId = separatedCrime[12];

**if** (districtId == **null**) {

**return**;

}

// The foreign join key is the user ID

outkey.set(districtId);

// Flag this record for the reducer and then output

outvalue.set("B|" + value);

context.write(outkey, outvalue);

}

}

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

**private** **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**;

**public** **void** setup(Context context) {

// Get the type of join from our configuration

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

}

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

// Clear our lists

listA.clear();

listB.clear();

// iterate through all our values, binning each record based on what

// it was tagged with. Make sure to remove the tag!

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

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

System.***out***.println(Character.*toString*((**char**) tmp.charAt(0)));

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

&& (Character.*toString*((**char**) tmp.charAt(1)).equals("|"))) {

listA.add(**new** Text(tmp.toString().substring(2)));

}

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

&& (Character.*toString*((**char**) tmp.charAt(1)).equals("|"))) {

listB.add(**new** Text(tmp.toString().substring(2)));

}

}

// Execute our join logic now that the lists are filled

System.***out***.println(listB.size());

executeJoinLogic(context);

}

**private** **void** executeJoinLogic(Context context) **throws** IOException, InterruptedException {

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

// If both lists are not empty, join A with B

// System.out.println("here3");

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

System.***out***.println("here");

**for** (Text A : listA) {

// System.out.println("here1");

**for** (Text B : listB) {

// System.out.println("here2");

context.write(A, B);

}

}

}

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

// For each entry in A,

**for** (Text A : listA) {

// If list B is not empty, join A and B

**if** (!listB.isEmpty()) {

**for** (Text B : listB) {

context.write(A, B);

}

} **else** {

// Else, output A by itself

context.write(A, ***EMPTY\_TEXT***);

}

}

} **else** **if** (joinType.equalsIgnoreCase("rightouter")) {

// For each entry in B,

**for** (Text B : listB) {

// If list A is not empty, join A and B

**if** (!listA.isEmpty()) {

**for** (Text A : listA) {

context.write(A, B);

}

} **else** {

// Else, output B by itself

context.write(***EMPTY\_TEXT***, B);

}

}

} **else** **if** (joinType.equalsIgnoreCase("fullouter")) {

// If list A is not empty

**if** (!listA.isEmpty()) {

// For each entry in A

**for** (Text A : listA) {

// If list B is not empty, join A with B

**if** (!listB.isEmpty()) {

**for** (Text B : listB) {

context.write(A, B);

}

} **else** {

// Else, output A by itself

context.write(A, ***EMPTY\_TEXT***);

}

}

} **else** {

// If list A is empty, just output B

**for** (Text B : listB) {

context.write(***EMPTY\_TEXT***, B);

}

}

} **else** **if** (joinType.equalsIgnoreCase("anti")) {

// If list A is empty and B is empty or vice versa

**if** (listA.isEmpty() ^ listB.isEmpty()) {

// Iterate both A and B with null values

// The previous XOR check will make sure exactly one of

// these lists is empty and therefore the list will be

// skipped

**for** (Text A : listA) {

context.write(A, ***EMPTY\_TEXT***);

}

**for** (Text B : listB) {

context.write(***EMPTY\_TEXT***, B);

}

}

}

}

}

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

Configuration conf = **new** Configuration();

Job job = Job.*getInstance*(conf, "ReduceSideJoin");

job.setJarByClass(JoinDataSets.**class**);

// Use MultipleInputs to set which input uses what mapper

// This will keep parsing of each data set separate from a logical

// standpoint

// The first two elements of the args array are the two inputs

MultipleInputs.*addInputPath*(job, **new** Path(args[0]), TextInputFormat.**class**, CrimeMapper.**class**);

MultipleInputs.*addInputPath*(job, **new** Path(args[1]), TextInputFormat.**class**, IUCRMapper.**class**);

job.getConfiguration().set("join.type", "leftouter");

// job.setNumReduceTasks(0);

job.setReducerClass(JoinReducer.**class**);

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

TextOutputFormat.*setOutputPath*(job, **new** Path(args[3]));

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

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

**boolean** complete = job.waitForCompletion(**true**);

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

Configuration conf2 = **new** Configuration();

Job job2 = Job.*getInstance*(conf2, "ReduceSideJoinChaining");

**if** (complete) {

job2.setJarByClass(JoinDataSets.**class**);

MultipleInputs.*addInputPath*(job2, **new** Path(args[2]), TextInputFormat.**class**, DistrictMapper.**class**);

MultipleInputs.*addInputPath*(job2, **new** Path(args[3]), TextInputFormat.**class**, OutputOfFirstMapper.**class**);

job2.getConfiguration().set("join.type", "rightouter");

// job.setNumReduceTasks(0);

job2.setReducerClass(JoinReducer.**class**);

job2.setOutputFormatClass(TextOutputFormat.**class**);

TextOutputFormat.*setOutputPath*(job2, **new** Path(args[4]));

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

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

complete = job2.waitForCompletion(**true**);

System.*exit*(job2.waitForCompletion(**true**) ? 0 : 1);

}

}

}

### Partition Data

**package** project.chicagocrimes.partitioning;

**import** java.io.IOException;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Calendar;

**import** java.util.logging.Level;

**import** java.util.logging.Logger;

**import** org.apache.hadoop.conf.Configurable;

**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.Partitioner;

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

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

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

**public** **class** Partitioning {

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

Configuration conf = **new** Configuration();

Job job = **new** ~~Job~~(conf, "PostCommentHierarchy");

job.setJarByClass(Partitioning.**class**);

job.setMapperClass(Final\_partition\_date\_mapper.**class**);

job.setMapOutputKeyClass(IntWritable.**class**);

job.setMapOutputValueClass(Text.**class**);

job.setPartitionerClass(Final\_partition\_date\_partitioner.**class**);

Final\_partition\_date\_partitioner.*setMinLastAccessDate*(job, 2001);

job.setNumReduceTasks(5);

job.setReducerClass(Final\_partition\_date\_reducer.**class**);

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);

}

**public** **static** **class** Final\_partition\_date\_mapper **extends** Mapper<Object, Text, IntWritable, Text> {

**private** IntWritable outKey = **new** IntWritable();

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

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

String crimeString = separatedInput[1];

**if** (crimeString == **null**) {

**return**;

}

String copyOfCrime = **new** String();

**boolean** inQuotes = **false**;

**for**(**int** i=0; i<crimeString.length(); ++i)

{

**if** (crimeString.charAt(i)=='"')

inQuotes = !inQuotes;

**if** (crimeString.charAt(i)==',' && inQuotes)

copyOfCrime += '|';

**else**

copyOfCrime += crimeString.charAt(i);

}

String[] separatedCrime = copyOfCrime.toString().split(",");

String year = separatedCrime[18];

**if** (year == **null**) {

**return**;

}

**if** (year.toString().contains("Year")) /\* Some condition satisfying it is header \*/

**return**;

**else** {

**try** {

outKey.set(Integer.*parseInt*(year));

} **catch** (Exception e) {

// **TODO**: handle exception

}

context.write(outKey, value);

}

}

}

**public** **static** **class** Final\_partition\_date\_partitioner **extends** Partitioner<IntWritable, Text>

**implements** Configurable {

**private** **static** **final** String ***MIN\_LAST\_ACCESS\_DATE\_MONTH*** = "min.last.access.date.year";

**private** Configuration conf = **null**;

**private** **static** **int** *minLastAccessDateMonth* = 0;

**public** **int** getPartition(IntWritable key, Text value, **int** numPartitions) {

**if** (key.get() == 2001) {

**return** 0;

}

**if** (key.get() == 2002) {

**return** 1;

}

**if** (key.get() == 2003) {

**return** 2;

}

**if** (key.get() == 2004)

**return** 3;

**else**

**return** 4;

}

**public** Configuration getConf() {

**return** conf;

}

**public** **void** setConf(Configuration conf) {

**this**.conf = conf;

*minLastAccessDateMonth* = conf.getInt(***MIN\_LAST\_ACCESS\_DATE\_MONTH***, 0);

}

**public** **static** **void** setMinLastAccessDate(Job job, **int** minLastAccessDateYear) {

job.getConfiguration().setInt(***MIN\_LAST\_ACCESS\_DATE\_MONTH***, *minLastAccessDateMonth*);

}

}

**public** **static** **class** Final\_partition\_date\_reducer **extends** Reducer<IntWritable, Text, Text, NullWritable> {

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

**throws** IOException, InterruptedException {

**for** (Text t : values) {

context.write(t, NullWritable.*get*());

}

}

}

}

### Binning By month

**package** project.chicagocrimes.binning;

**import** java.io.IOException;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Calendar;

**import** java.util.logging.Level;

**import** java.util.logging.Logger;

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

**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.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** project.chicagocrimes.partitioning.Partitioning.Final\_partition\_date\_mapper;

**public** **class** BinningByMonth {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**try** {

Configuration conf = **new** Configuration();

Job job = Job.*getInstance*(conf, "BinningByYear");

job.setJarByClass(BinningByMonth.**class**);

job.setMapperClass(MapperClass.**class**);

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

MultipleOutputs.*setCountersEnabled*(job, **true**);

job.setNumReduceTasks(0);

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

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

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

} **catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

}

}

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

**private** MultipleOutputs<Text, NullWritable> mos = **null**;

**protected** **void** setup(Context context) {

mos = **new** MultipleOutputs(context);

}

**private** **static** SimpleDateFormat *frmt* = **new** SimpleDateFormat("MM/dd/yyyy");

Calendar cal = Calendar.*getInstance*();

@Override

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

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

String crimeString = separatedInput[1];

**if** (crimeString == **null**) {

**return**;

}

String copyOfCrime = **new** String();

**boolean** inQuotes = **false**;

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

**if** (crimeString.charAt(i) == '"')

inQuotes = !inQuotes;

**if** (crimeString.charAt(i) == ',' && inQuotes)

copyOfCrime += '|';

**else**

copyOfCrime += crimeString.charAt(i);

}

String[] separatedCrime = copyOfCrime.toString().split(",");

String timeStamp = separatedCrime[3];

**if** (timeStamp == **null** || timeStamp == "") {

**return**;

}

**if** (timeStamp.toString().contains("Date")) /\* Some condition satisfying it is header \*/

**return**;

**else** {

String[] words = timeStamp.toString().split(" ");

**try** {

cal.setTime(*frmt*.parse(words[0].trim()));

} **catch** (ParseException ex) {

Logger.*getLogger*(Final\_partition\_date\_mapper.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

}

String month = String.*valueOf*(cal.get(Calendar.***MONTH***));

// The foreign join key is the user ID

// int yearBin = Integer.parseInt(year);

mos.write("bins", value, NullWritable.*get*(), month + "-bin");

/\*

\* for(String s:str) { if("00".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"12am-bin"); }

\* if("01".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"01am-bin"); }

\* if("02".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"02am-bin"); }

\* if("03".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"03am-bin"); }

\* if("04".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"04am-bin"); }

\* if("05".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"05am-bin"); }

\* if("06".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"06am-bin"); }

\* if("07".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"07am-bin"); }

\* if("08".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"08am-bin"); }

\* if("09".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"09am-bin"); }

\* if("10".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"10am-bin"); }

\* if("11".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"11am-bin"); }

\* if("12".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"12pm-bin"); }

\* if("13".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"01pm-bin"); }

\* if("14".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"02pm-bin"); }

\* if("15".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"03pm-bin"); }

\* if("16".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"04pm-bin"); }

\* if("17".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"05pm-bin"); }

\* if("18".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"06pm-bin"); }

\* if("19".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"07pm-bin"); }

\* if("20".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"08pm-bin"); }

\* if("21".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"09pm-bin"); }

\* if("22".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"10pm-bin");

\* }if("23".equals(s1)) {

\* mos.write("bins",value,NullWritable.get(),"11pm-bin"); } }

\*/

}

@Override

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

// super.cleanup(context); //To change body of generated methods,

// choose Tools | Templates.

mos.close();

}

}

}

### Counting arrests

**package** project.chicagocrimes.count;

**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.mapreduce.Counter;

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

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

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

**public** **class** CountArrests {

/\*\*

\* **@param** args the command line arguments

\*/

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

// **TODO** code application logic here

Path input = **new** Path(args[0]);

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

Configuration conf = **new** Configuration();

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

job.setJarByClass(CountArrests.**class**);

job.setMapperClass(CountMapper.**class**);

job.setNumReduceTasks(0);

job.setOutputKeyClass(NullWritable.**class**);

job.setOutputValueClass(NullWritable.**class**);

//job.setInputFormatClass(FileInputFormat.class);

FileInputFormat.*addInputPath*(job, input);

FileOutputFormat.*setOutputPath*(job, outputDir);

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

**if** (code == 0) {

**for** (Counter counter : job.getCounters().getGroup(CountMapper.***ARREST\_TYPE***)) {

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

}

}

FileSystem.*get*(conf).delete(outputDir, **true**);

System.*exit*(code);

}

}

**package** project.chicagocrimes.count;

**import** java.util.Arrays;

**import** java.util.HashSet;

**import** org.apache.hadoop.io.NullWritable;

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

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

**public** **class** CountMapper **extends** Mapper<Object, Text, NullWritable, NullWritable> {

**public** **static** **final** String ***ARREST\_TYPE*** = "Arrest ";

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

**public** **static** **final** String ***OTHER\_COUNTER*** = "Other";

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

**private** String[] arrestArray = **new** String[]{"TRUE", "FALSE","True","False"};

**private** HashSet<String> arrests = **new** HashSet<String>(Arrays.*asList*(arrestArray));

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

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

String crimeString = separatedInput[1];

**if** (crimeString == **null**) {

**return**;

}

String copyOfCrime = **new** String();

**boolean** inQuotes = **false**;

**for**(**int** i=0; i<crimeString.length(); ++i)

{

**if** (crimeString.charAt(i)=='"')

inQuotes = !inQuotes;

**if** (crimeString.charAt(i)==',' && inQuotes)

copyOfCrime += '|';

**else**

copyOfCrime += crimeString.charAt(i);

}

String[] separatedCrime = copyOfCrime.toString().split(",");

String arrested = separatedCrime[9];

String domestic = separatedCrime[10];

**if** (arrested == **null**) {

**return**;

}

**boolean** unknown = **true**;

**if** (arrests.contains(arrested) && domestic.equalsIgnoreCase("True")) {

context.getCounter(***ARREST\_TYPE***, arrested).increment(1);

unknown = **false**;

}

**if** (unknown) {

context.getCounter(***ARREST\_TYPE***, ***UNKNOWN\_COUNTER***).increment(1);

} /\*else {

context.getCounter(ARREST\_TYPE, NULL\_OR\_EMPTY);

}\*/

}

}

### Frequency By District (top 25)

**package** project.chicagocrimes.frequency;

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

**import** java.io.IOException;

**import** java.util.logging.Level;

**import** java.util.logging.Logger;

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

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

**import** org.apache.hadoop.io.FloatWritable;

**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.FileInputFormat;

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

/\*\*

\*

\* **@author** harshad

\*/

**public** **class** FrequencyByDistrict {

**public** **static** **class** FrequencyByDistrictMapper **extends** Mapper<Object, Text, Text, LongWritable> {

LongWritable one = **new** LongWritable(1);

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

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

String crimeString = separatedInput[1];

**if** (crimeString == **null**) {

**return**;

}

String copyOfCrime = **new** String();

**boolean** inQuotes = **false**;

**for**(**int** i=0; i<crimeString.length(); ++i)

{

**if** (crimeString.charAt(i)=='"')

inQuotes = !inQuotes;

**if** (crimeString.charAt(i)==',' && inQuotes)

copyOfCrime += '|';

**else**

copyOfCrime += crimeString.charAt(i);

}

String[] separatedCrime = copyOfCrime.toString().split(",");

String districtId = separatedCrime[13];

**if** (districtId == **null**) {

**return**;

}

// The foreign join key is the user ID

Text outKey = **new** Text(districtId);

context.write(outKey, one);

}

}

**public** **static** **class** FrequencyByDistrictReducer **extends** Reducer<Text, LongWritable, Text, LongWritable>{

@Override

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

LongWritable result = **new** LongWritable();

**long** count = 0;

**for**(LongWritable val :values){

count++;

}

result.set(count);

context.write(key,result);

}

}

**public** **static** **class** FrequencyMapper2 **extends** Mapper<LongWritable, Text, CompositeKeyWritable, IntWritable>{

@Override

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

Text districtId = **new** Text();

LongWritable count = **new** LongWritable();

**if**(values.toString().length()>0)

{

**try**{

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

districtId.set(str[0]);

count.set(Long.*parseLong*(str[1]));

CompositeKeyWritable cw = **new** CompositeKeyWritable(districtId.toString(), count.get());

context.write(cw, **new** IntWritable(1));

}**catch**(IOException | InterruptedException ex){

Logger.*getLogger*(FrequencyMapper2.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

}

}

}

**public** **static** **class** FrequencyReducer2 **extends** Reducer<CompositeKeyWritable, IntWritable, CompositeKeyWritable, IntWritable> {

**public** **static** **int** *count* = 0;

@Override

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

**for** (IntWritable val : values) {

**if** (*count* < 25) {

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

*count*++;

} **else** {

**break**;

}

}

}

}

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

Configuration conf = **new** Configuration();

Job job = Job.*getInstance*(conf, "FrequencyByDistrict");

job.setJarByClass(FrequencyByDistrict.**class**);

job.setMapperClass(FrequencyByDistrictMapper.**class**);

// job.setCombinerClass(MovieReducer1.class);

job.setReducerClass(FrequencyByDistrictReducer.**class**);

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

job.setOutputValueClass(LongWritable.**class**);

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

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

**boolean** complete = job.waitForCompletion(**true**);

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

Configuration conf2 = **new** Configuration();

Job job2 = Job.*getInstance*(conf2, "Chaining");

**if** (complete) {

job2.setJarByClass(FrequencyByDistrict.**class**);

job2.setMapperClass(FrequencyMapper2.**class**);

// job2.setCombinerClass(MovieReducer2.class);

job2.setReducerClass(FrequencyReducer2.**class**);

job2.setOutputKeyClass(CompositeKeyWritable.**class**);

job2.setOutputValueClass(IntWritable.**class**);

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

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

System.*exit*(job2.waitForCompletion(**true**) ? 0 : 1);

}

}

}

**package** project.chicagocrimes.frequency;

**import** java.io.DataInput;

**import** java.io.DataOutput;

**import** java.io.IOException;

**import** org.apache.hadoop.io.Writable;

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

**import** org.apache.hadoop.io.WritableUtils;

/\*\*

\*

\* **@author** harshad

\*/

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

**private** String location;

**private** Long totalCount;

**public** CompositeKeyWritable()

{

}

**public** CompositeKeyWritable(String l, Long c)

{

**this**.location = l;

**this**.totalCount = c;

}

**public** String getLocation() {

**return** location;

}

**public** **void** setLocation(String location) {

**this**.location = location;

}

**public** Long getTotalCount() {

**return** totalCount;

}

**public** **void** setTotalCount(Long totalCount) {

**this**.totalCount = totalCount;

}

@Override

**public** **void** write(DataOutput d) **throws** IOException {

WritableUtils.*writeString*(d, location);

d.writeLong(totalCount);

}

@Override

**public** **void** readFields(DataInput di) **throws** IOException {

location = WritableUtils.*readString*(di);

totalCount = di.readLong();

}

@Override

**public** **int** compareTo(CompositeKeyWritable o) {

/\* int result = totalCount.compareTo(o.totalCount);

if(result ==0)

{

result = location.compareTo(o.location);

}

return result;

\*/

**return** -1\*(totalCount.compareTo(o.totalCount));

}

@Override

**public** String toString() {

**return** location + "\t" + totalCount;

}

}

**package** project.chicagocrimes.frequency;

**import** org.apache.hadoop.io.NullWritable;

**import** org.apache.hadoop.mapreduce.Partitioner;

**public** **class** FrequencyPartitioner **extends** Partitioner<CompositeKeyWritable, NullWritable>{

@Override

**public** **int** getPartition(CompositeKeyWritable key, NullWritable value, **int** numOfPartitions) {

**return** (key.getLocation().hashCode() % numOfPartitions);

}

}

### Frequency by Location

**package** project.chicagocrimes.frequency;

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

**import** java.io.IOException;

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

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

**import** org.apache.hadoop.io.LongWritable;

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

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

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

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

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

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

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

/\*\*

\*

\* **@author** harshad

\*/

**public** **class** FrequencyByLocation {

**public** **static** **class** FrequencyByLocationMapper **extends** Mapper<Object, Text, Text, LongWritable> {

LongWritable one = **new** LongWritable(1);

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

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

String crimeString = separatedInput[1];

**if** (crimeString == **null**) {

**return**;

}

String copyOfCrime = **new** String();

**boolean** inQuotes = **false**;

**for**(**int** i=0; i<crimeString.length(); ++i)

{

**if** (crimeString.charAt(i)=='"')

inQuotes = !inQuotes;

**if** (crimeString.charAt(i)==',' && inQuotes)

copyOfCrime += '|';

**else**

copyOfCrime += crimeString.charAt(i);

}

String[] separatedCrime = copyOfCrime.toString().split(",");

String districtId = separatedCrime[8];

**if** (districtId == **null**) {

**return**;

}

// The foreign join key is the user ID

Text outKey = **new** Text(districtId);

context.write(outKey, one);

}

}

**public** **static** **class** FrequencyByLocationReducer **extends** Reducer<Text, LongWritable, Text, LongWritable>{

@Override

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

LongWritable result = **new** LongWritable();

**long** count = 0;

**for**(LongWritable val :values){

count++;

}

result.set(count);

context.write(key,result);

}

}

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

Configuration conf = **new** Configuration();

Job job = Job.*getInstance*(conf, "FrequencyByLocation");

job.setJarByClass(FrequencyByLocation.**class**);

job.setMapperClass(FrequencyByLocationMapper.**class**);

// job.setCombinerClass(MovieReducer1.class);

job.setReducerClass(FrequencyByLocationReducer.**class**);

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

job.setOutputValueClass(LongWritable.**class**);

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

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

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

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

}

}