Advance Database Management Systems Final Project Report

Submitted By:
Yamini Sehrawat
NUID: 001617219
Email: sehrawat.y@husky.neu.edu

Table of Contents

Abstract

Dataset: Health Insurance Marketplace
The Health Insurance Marketplace Public Use Files contain data on health and dental plans offered to individuals and small businesses through the US Health Insurance Marketplace. The data is available for various states across USA. The Centers for Medicare & Medicaid Services (CMS) Center for Consumer Information & Insurance Oversight (CCIIO) releases the Marketplace files to improve transparency and increase access to Marketplace data. The files considered for this project belong to business year 2016.
Files considered for analysis:

- **Rate.csv**: Plan-level data on individual rates based on an eligible subscriber's age, tobacco use, and geographic location.
- **ServiceArea**.**csv**: Issuer-level data on the geographic coverage or service area (i.e., where the plan is offered) including state, county, and zip code.
- **Network**.**csv**: Issuer-level data identifying provider network URLs.
- **PlanAttributes**.**csv**: Plan-level data
- **BenefitsCrossSharing**.**csv**: Plan-level data on essential health benefits, coverage limits, and cost sharing.

Link for dataset: https://www.kaggle.com/hhs/health-insurance-marketplace

Dataset size: 701 MB

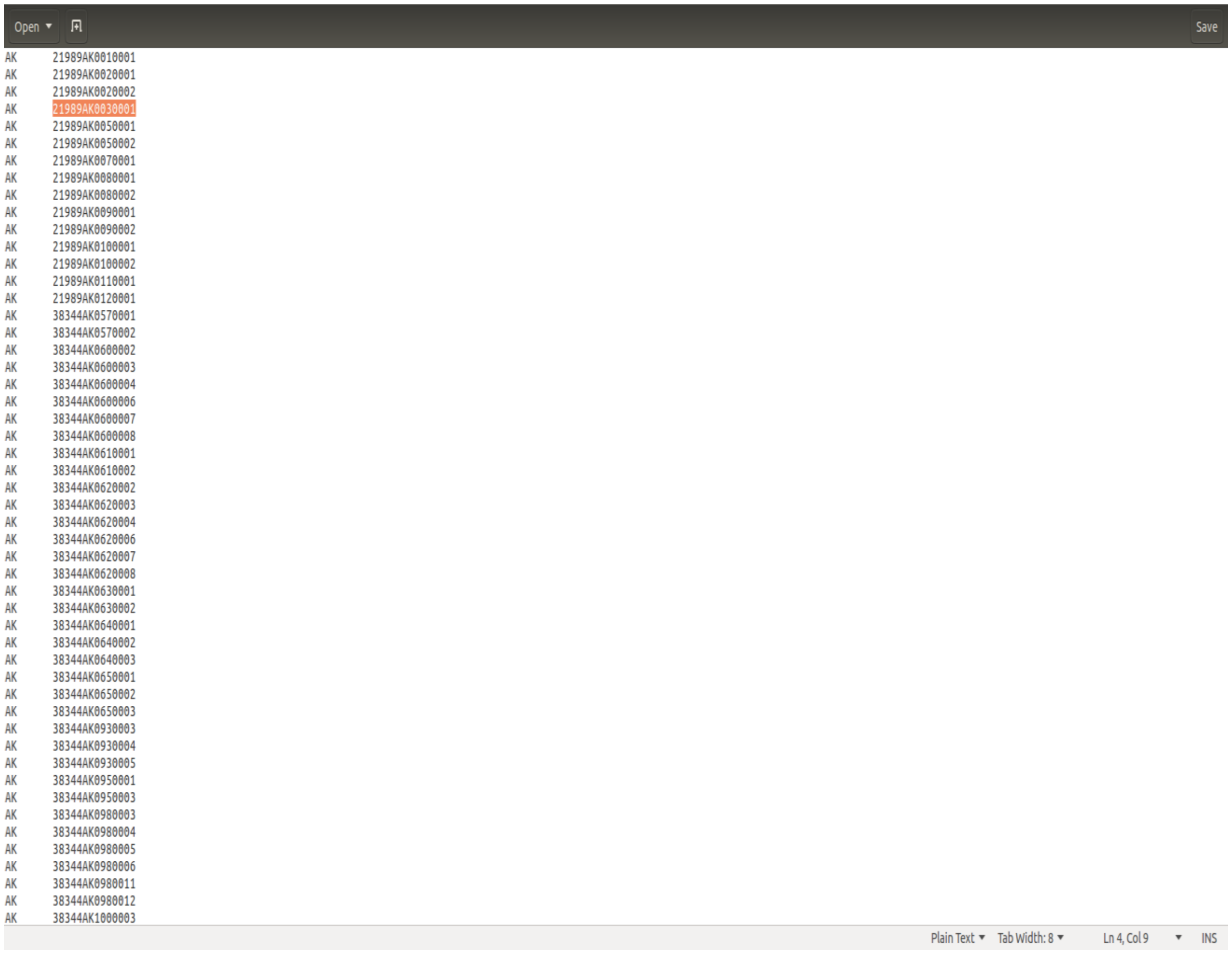
Analysis

1. Distinct Pattern Analysis

Distinct health plans across the states

The dataset contains health plans for various states in US. Analysis is done to get the **distinct** health plans for each state. The input file used here is: Rate.csv

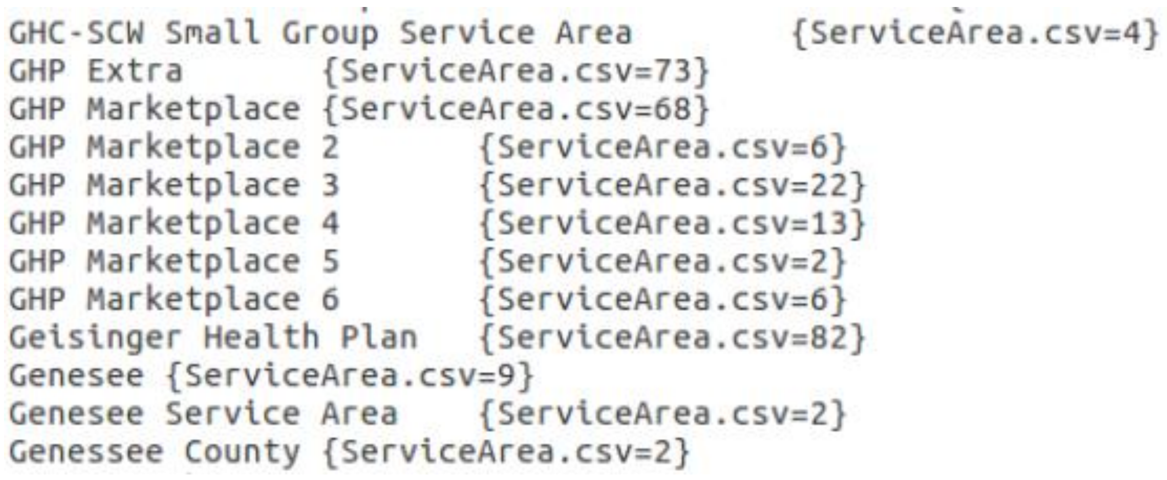
For example: For state - AK, Distinct health plans are---



```
AK	21989AK0010001
AK	21989AK0020001
AK	21989AK0020002
AK	21989AK0030001
AK	21989AK0050001
AK	21989AK0050002
AK	21989AK0070001
AK	21989AK0080001
AK	21989AK0080002
AK	21989AK0090001
AK	21989AK0090002
AK	21989AK0100001
AK	21989AK0100002
AK	21989AK0110001
AK	21989AK0120001
AK	38344AK0570001
AK	38344AK0570002
AK	38344AK0600002
AK	38344AK0600003
AK	38344AK0600004
AK	38344AK0600006
AK	38344AK0600007
AK	38344AK0600008
AK	38344AK0610001
AK	38344AK0610002
AK	38344AK0620002
AK	38344AK0620003
AK	38344AK0620004
AK	38344AK0620006
AK	38344AK0620007
AK	38344AK0620008
AK	38344AK0630001
AK	38344AK0630002
AK	38344AK0640001
AK	38344AK0640002
AK	38344AK0640003
AK	38344AK0650001
AK	38344AK0650002
AK	38344AK0650003
AK	38344AK0930003
AK	38344AK0930004
AK	38344AK0930005
AK	38344AK0950001
AK	38344AK0950003
AK	38344AK0980003
AK	38344AK0980004
AK	38344AK0980005
AK	38344AK0980006
AK	38344AK0980011
AK	38344AK0980012
AK	38344AK1000003
```

2. Text Searching: Inverted Index

   Inverted index is built on ServiceArea.csv file to find out the most commonly words occurring in the file. It will speed up the search process for these words.

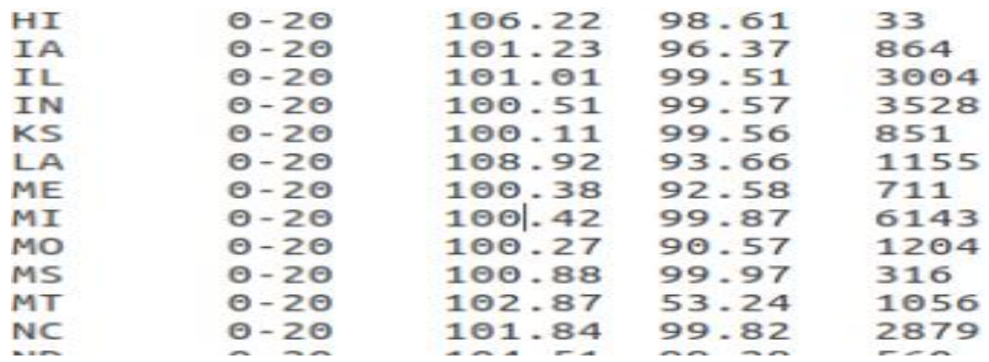


```
GHC-SCW Small Group Service Area        {ServiceArea.csv=4}
GHP Extra       {ServiceArea.csv=73}
GHP Marketplace {ServiceArea.csv=68}
GHP Marketplace 2       {ServiceArea.csv=6}
GHP Marketplace 3       {ServiceArea.csv=22}
GHP Marketplace 4       {ServiceArea.csv=13}
GHP Marketplace 5       {ServiceArea.csv=2}
GHP Marketplace 6       {ServiceArea.csv=6}
Geisinger Health Plan   {ServiceArea.csv=82}
Genesee {ServiceArea.csv=9}
Genesee Service Area    {ServiceArea.csv=2}
Genessee County {ServiceArea.csv=2}
```

3. How plan rates vary across states? – Min, Max & Count

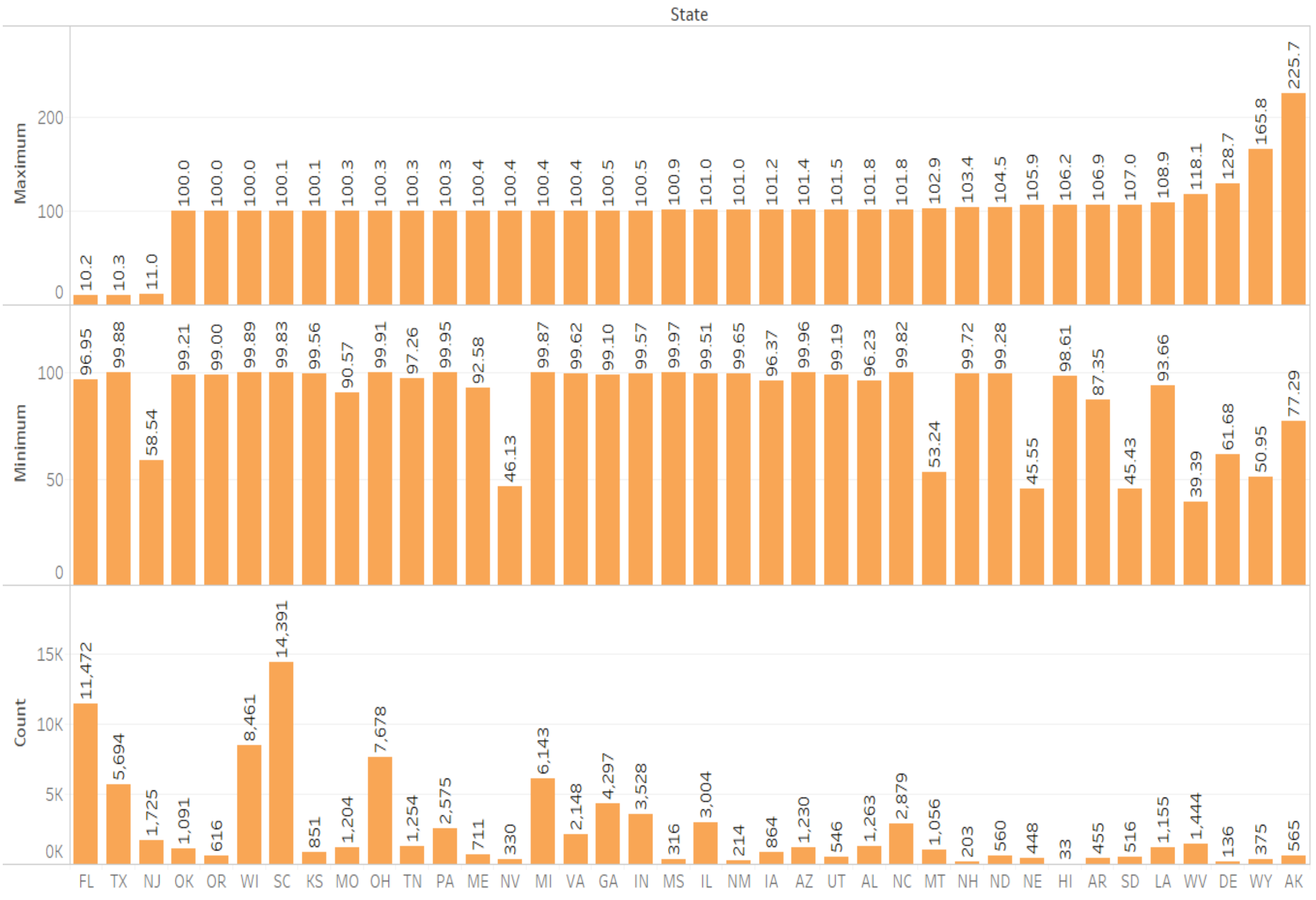
Min and Max analysis is done to find out how plan rates vary across different states for different age groups. What are the maximum and minimum rates considering the age groups?

For ex: ScreenShot of Output for age-group 0-20



```
HI      0-20    106.22  98.61   33
IA      0-20    101.23  96.37   864
IL      0-20    101.01  99.51   3004
IN      0-20    100.51  99.57   3528
KS      0-20    100.11  99.56   851
LA      0-20    108.92  93.66   1155
ME      0-20    100.38  92.58   711
MI      0-20    100.42  99.87   6143
MO      0-20    100.27  90.57   1204
MS      0-20    100.88  99.97   316
MT      0-20    102.87  53.24   1056
NC      0-20    101.84  99.82   2879
```

Max, Min and Count for Age-group(0-20) for all states



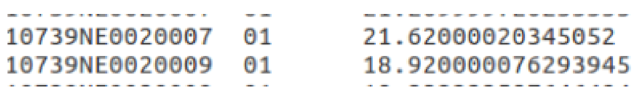
Sum of Maximum, sum of Minimum and sum of Count for each State. The data is filtered on Age Group, which keeps 0-20.

4. How Average Price for the plans vary with Time? – Moving Average

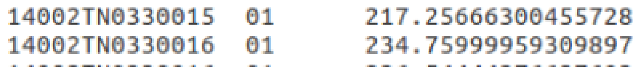
The mean (or average) of time series data (observations equally spaced in time, such as per hour or per day) from several consecutive periods is called the moving average. It is called moving because the average is continually recomputed as new time series data becomes available, and it progresses by dropping the earliest value and adding the most recent.
In this, I have calculated average price for health plans over the months of Rate effective date
Input file is Rate.csv.
This shows Planid average rates over months.
I have set Number of reduce tasks to 10.

For example: For 1st month, Moving Avg for different plan IDs:

PlandID Month Moving Average



```
10739NE0020007  01      21.62000020345052
10739NE0020009  01      18.920000076293945
```

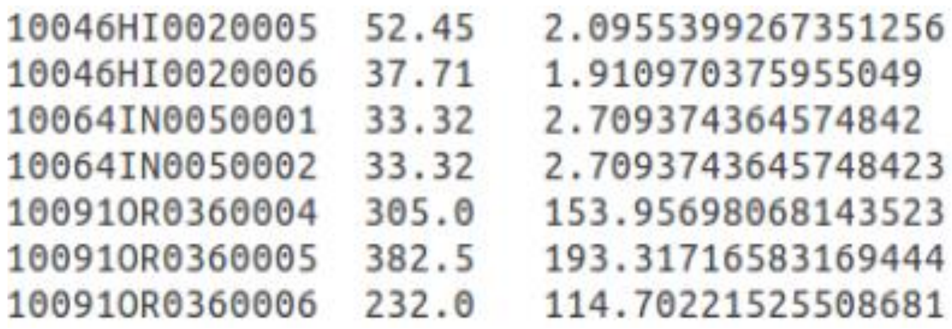


```
14002TN0330015  01      217.25666300455728
14002TN0330016  01      234.75999959309897
```

5. Mean and standard deviation of individual rates for all the health plans

Next, **Mean and Standard Deviations** of individual rates of health plans is calculated to get the beneficial plans. The code consists of Mapper class, Reducer class where actual calculations are performed and finally the driver class.

PlanId Median Standard Deviation



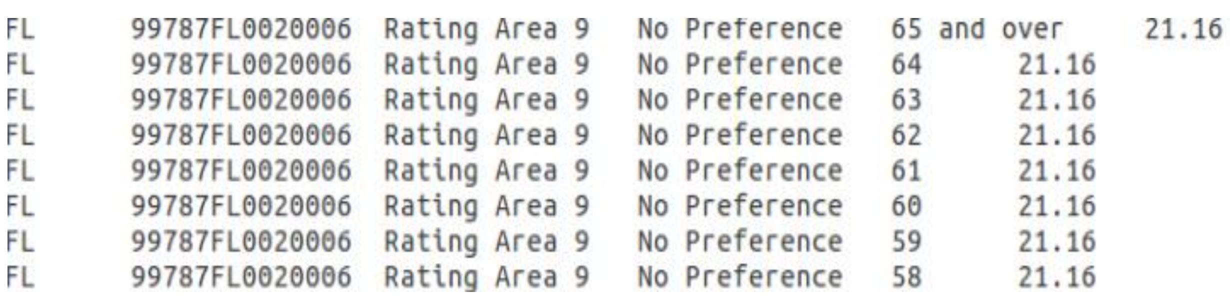
```
10046HI0020005  52.45   2.0955399267351256
10046HI0020006  37.71   1.910970375955049
10064IN0050001  33.32   2.709374364574842
10064IN0050002  33.32   2.7093743645748423
10091OR0360004  305.0   153.95698068143523
10091OR0360005  382.5   193.31716583169444
10091OR0360006  232.0   114.70221525508681
```

6. Secondary Sorting

This is performed to sort the values coming to the Reducer of Hadoop Map/Reduce job by using combination of composite key, Practitioner and Writable Comparator created using plan ID, rating area, Tobacco preference, age -groups and rate.
Here, I have set Number of Reduce tasks as 6.

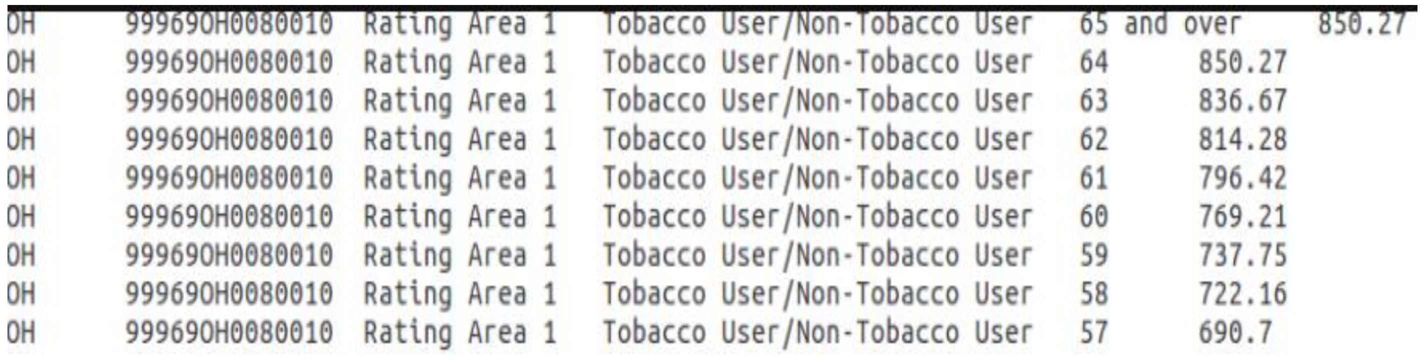
The Rates for different plans are sorted in descending order based on selected preferences.
For example: In case of Tobacco as "No Preference", how rates can can sorted for different age-groups in a State.

State Plan Rating Area Tobacco Age Rate



```
FL      99787FL0020006  Rating Area 9   No Preference   65 and over     21.16
FL      99787FL0020006  Rating Area 9   No Preference   64      21.16
FL      99787FL0020006  Rating Area 9   No Preference   63      21.16
FL      99787FL0020006  Rating Area 9   No Preference   62      21.16
FL      99787FL0020006  Rating Area 9   No Preference   61      21.16
FL      99787FL0020006  Rating Area 9   No Preference   60      21.16
FL      99787FL0020006  Rating Area 9   No Preference   59      21.16
FL      99787FL0020006  Rating Area 9   No Preference   58      21.16
```

Tobacco : "TobaccoUser/Non-Tobacco User".

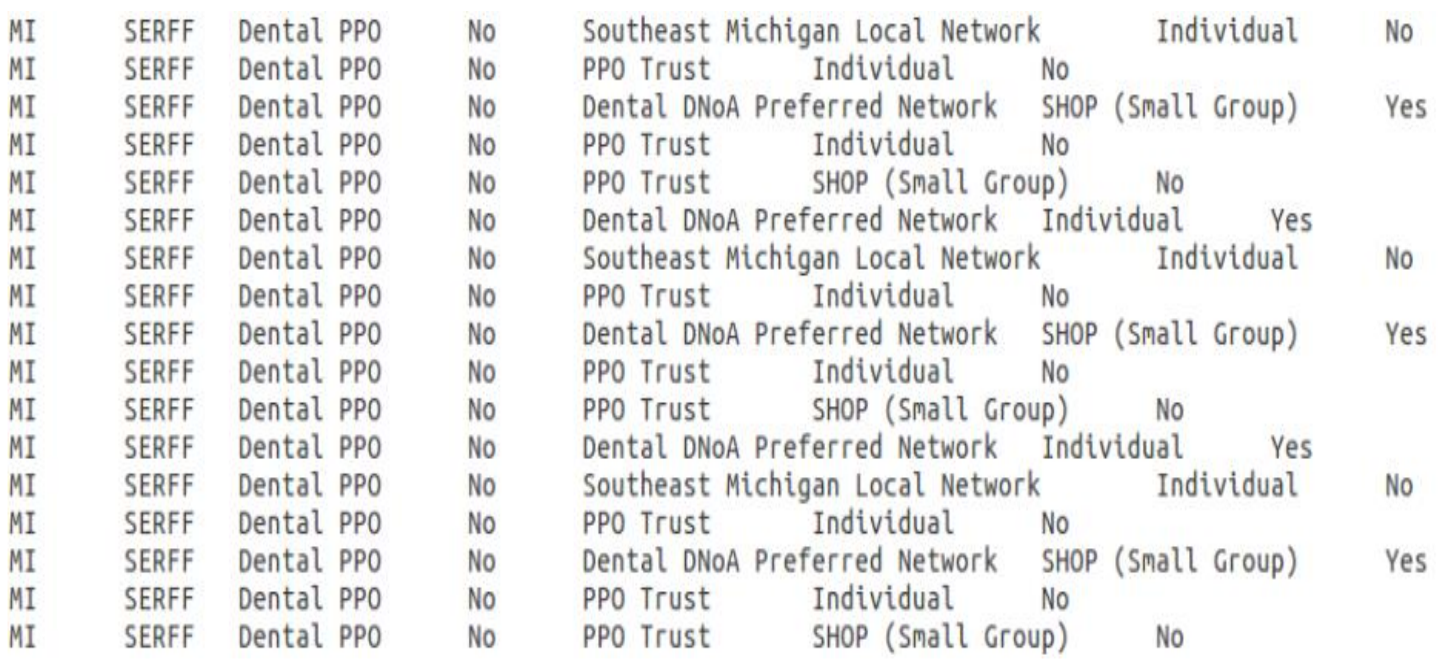


```
OH      99969OH0080010  Rating Area 1   Tobacco User/Non-Tobacco User   65 and over     850.27
OH      99969OH0080010  Rating Area 1   Tobacco User/Non-Tobacco User   64      850.27
OH      99969OH0080010  Rating Area 1   Tobacco User/Non-Tobacco User   63      836.67
OH      99969OH0080010  Rating Area 1   Tobacco User/Non-Tobacco User   62      814.28
OH      99969OH0080010  Rating Area 1   Tobacco User/Non-Tobacco User   61      796.42
OH      99969OH0080010  Rating Area 1   Tobacco User/Non-Tobacco User   60      769.21
OH      99969OH0080010  Rating Area 1   Tobacco User/Non-Tobacco User   59      737.75
OH      99969OH0080010  Rating Area 1   Tobacco User/Non-Tobacco User   58      722.16
OH      99969OH0080010  Rating Area 1   Tobacco User/Non-Tobacco User   57      690.7
```

7. Which service areas cover what plans? – Reduce Side Join

Input files: ServiceArea.csv and Network.csv.
Analysis done using Reduce-Side Join to find out which service areas (combination of state and zip code) provide which health plans, do they cover entire state, Is MarketCoverage Individual or Small Group and is the Source in that network provide DentalOnlyPlans?
The code consists of two Mapper classes for two input files and one Reducer class which performs the join function.

State Source ServiceAreaName CoverEntireState NetworkName MarketCoverage DentalOnlyPlan

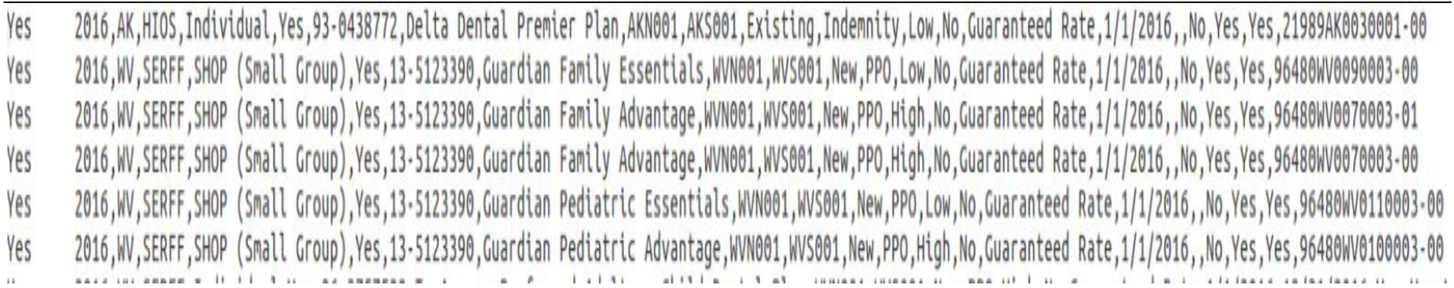


```
MI      SERFF   Dental PPO      No      Southeast Michigan Local Network        Individual      No
MI      SERFF   Dental PPO      No      PPO Trust       Individual      No
MI      SERFF   Dental PPO      No      Dental DNoA Preferred Network   SHOP (Small Group)      Yes
MI      SERFF   Dental PPO      No      PPO Trust       Individual      No
MI      SERFF   Dental PPO      No      PPO Trust       SHOP (Small Group)      No
MI      SERFF   Dental PPO      No      Dental DNoA Preferred Network   Individual      Yes
MI      SERFF   Dental PPO      No      Southeast Michigan Local Network        Individual      No
MI      SERFF   Dental PPO      No      PPO Trust       Individual      No
MI      SERFF   Dental PPO      No      Dental DNoA Preferred Network   SHOP (Small Group)      Yes
MI      SERFF   Dental PPO      No      PPO Trust       Individual      No
MI      SERFF   Dental PPO      No      PPO Trust       SHOP (Small Group)      No
MI      SERFF   Dental PPO      No      Dental DNoA Preferred Network   Individual      Yes
MI      SERFF   Dental PPO      No      Southeast Michigan Local Network        Individual      No
MI      SERFF   Dental PPO      No      PPO Trust       Individual      No
MI      SERFF   Dental PPO      No      Dental DNoA Preferred Network   SHOP (Small Group)      Yes
MI      SERFF   Dental PPO      No      PPO Trust       Individual      No
MI      SERFF   Dental PPO      No      PPO Trust       SHOP (Small Group)      No
```

8. Partitioning and Binning

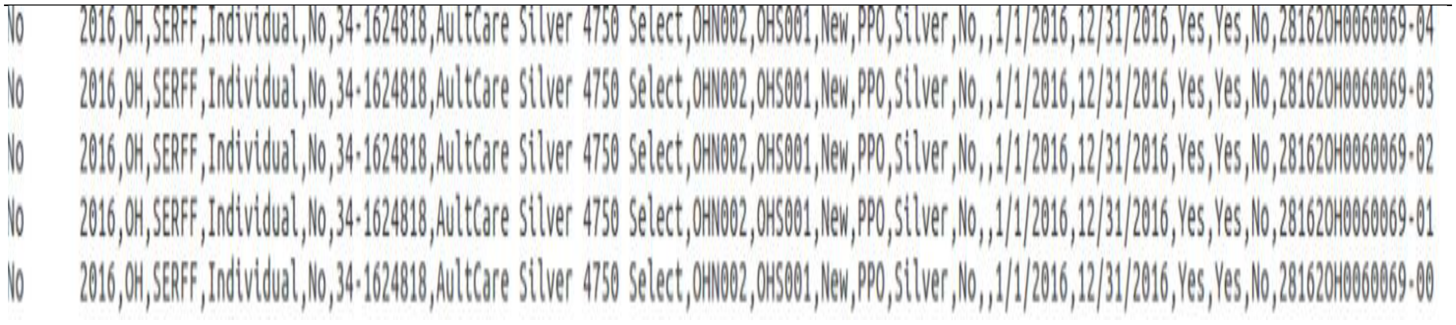
Here, I have done Partitioning on Network.csv file to partition the plans which are Dental Only Plans and which are not. There are two Partition to separate out the plans coverage. So, It is a Partitioning by categorical variable technique.
In Binning, I have created state bins to get insight of what plans each state offer and how plans vary across states.

Partitioning:
Dental Only Plan: Yes



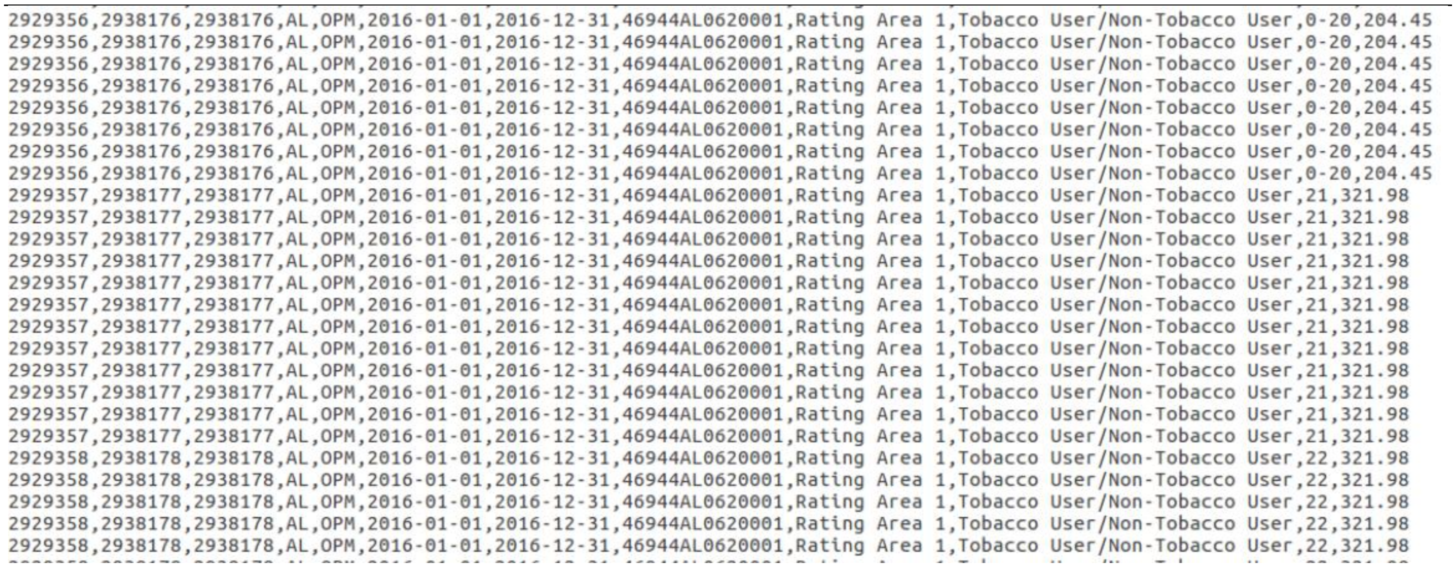
```
Yes     2016,AK,HIOS,Individual,Yes,93-0438772,Delta Dental Premier Plan,AKN001,AKS001,Existing,Indemnity,Low,No,Guaranteed Rate,1/1/2016,,No,Yes,Yes,21989AK0030001-00
Yes     2016,WV,SERFF,SHOP (Small Group),Yes,13-5123390,Guardian Family Essentials,WVN001,WVS001,New,PPO,Low,No,Guaranteed Rate,1/1/2016,,No,Yes,Yes,96480WV0090003-00
Yes     2016,WV,SERFF,SHOP (Small Group),Yes,13-5123390,Guardian Family Advantage,WVN001,WVS001,New,PPO,High,No,Guaranteed Rate,1/1/2016,,No,Yes,Yes,96480WV0070003-01
Yes     2016,WV,SERFF,SHOP (Small Group),Yes,13-5123390,Guardian Family Advantage,WVN001,WVS001,New,PPO,High,No,Guaranteed Rate,1/1/2016,,No,Yes,Yes,96480WV0070003-00
Yes     2016,WV,SERFF,SHOP (Small Group),Yes,13-5123390,Guardian Pediatric Essentials,WVN001,WVS001,New,PPO,Low,No,Guaranteed Rate,1/1/2016,,No,Yes,Yes,96480WV0110003-00
Yes     2016,WV,SERFF,SHOP (Small Group),Yes,13-5123390,Guardian Pediatric Advantage,WVN001,WVS001,New,PPO,High,No,Guaranteed Rate,1/1/2016,,No,Yes,Yes,96480WV0100003-00
```

Dental Only Plan: No



```
No     2016,OH,SERFF,Individual,No,34-1624818,AultCare Silver 4750 Select,OHN002,OHS001,New,PPO,Silver,No,,1/1/2016,12/31/2016,Yes,Yes,No,28162OH0060069-04
No     2016,OH,SERFF,Individual,No,34-1624818,AultCare Silver 4750 Select,OHN002,OHS001,New,PPO,Silver,No,,1/1/2016,12/31/2016,Yes,Yes,No,28162OH0060069-03
No     2016,OH,SERFF,Individual,No,34-1624818,AultCare Silver 4750 Select,OHN002,OHS001,New,PPO,Silver,No,,1/1/2016,12/31/2016,Yes,Yes,No,28162OH0060069-02
No     2016,OH,SERFF,Individual,No,34-1624818,AultCare Silver 4750 Select,OHN002,OHS001,New,PPO,Silver,No,,1/1/2016,12/31/2016,Yes,Yes,No,28162OH0060069-01
No     2016,OH,SERFF,Individual,No,34-1624818,AultCare Silver 4750 Select,OHN002,OHS001,New,PPO,Silver,No,,1/1/2016,12/31/2016,Yes,Yes,No,28162OH0060069-00
```

Binning: For example, for state AL



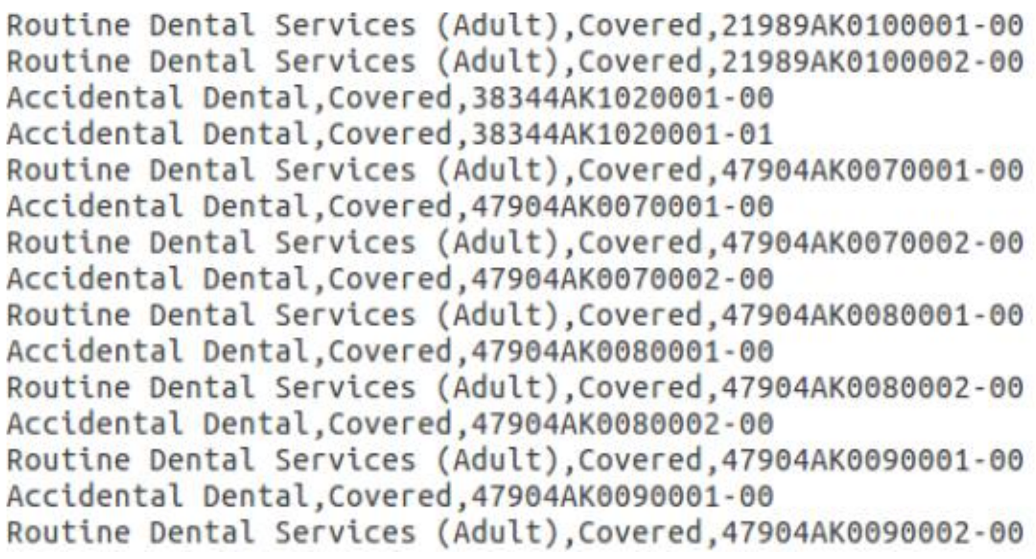
```
2929356,2938176,2938176,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,0-20,204.45
2929356,2938176,2938176,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,0-20,204.45
2929356,2938176,2938176,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,0-20,204.45
2929356,2938176,2938176,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,0-20,204.45
2929356,2938176,2938176,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,0-20,204.45
2929356,2938176,2938176,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,0-20,204.45
2929356,2938176,2938176,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,0-20,204.45
2929356,2938176,2938176,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,0-20,204.45
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929357,2938177,2938177,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,21,321.98
2929358,2938178,2938178,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,22,321.98
2929358,2938178,2938178,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,22,321.98
2929358,2938178,2938178,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,22,321.98
2929358,2938178,2938178,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,22,321.98
2929358,2938178,2938178,AL,OPM,2016-01-01,2016-12-31,46944AL0620001,Rating Area 1,Tobacco User/Non-Tobacco User,22,321.98
```

9. Bloom Filter:

To filter out for some specific benefits of plan, I have used Bloom filter technique as Bloom filter will tell me whether an element is present in a set.

Here, I have find out what plans cover Accidental Dental Benefits, Routine Dental Services (Adult) and Basic Dental Care - Child.

The code consists of PlanBenefits class defining Funnel. A Funnel describes how to decompose an object type into primitive field values. As this is Map Only job, the Mapper class filters the key, value pairs as per the criteria set.



```
Routine Dental Services (Adult),Covered,21989AK0100001-00
Routine Dental Services (Adult),Covered,21989AK0100002-00
Accidental Dental,Covered,38344AK1020001-00
Accidental Dental,Covered,38344AK1020001-01
Routine Dental Services (Adult),Covered,47904AK0070001-00
Accidental Dental,Covered,47904AK0070001-00
Routine Dental Services (Adult),Covered,47904AK0070002-00
Accidental Dental,Covered,47904AK0070002-00
Routine Dental Services (Adult),Covered,47904AK0080001-00
Accidental Dental,Covered,47904AK0080001-00
Routine Dental Services (Adult),Covered,47904AK0080002-00
Accidental Dental,Covered,47904AK0080002-00
Routine Dental Services (Adult),Covered,47904AK0090001-00
Accidental Dental,Covered,47904AK0090001-00
Routine Dental Services (Adult),Covered,47904AK0090002-00
```

Appendix

Below are the source codes for analysis done:

Distinct Pattern

Mapper class:

```
public class Distinct_Mapper extends Mapper<Object, Text, Text, NullWritable>{
  @Override
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException
    {
      String[] tokens = value.toString().split(",");
      try{
        Text planId = new Text();
        Text state = new Text();
        state.set(tokens[10].substring(5, 7));
        planId.set((tokens[10]));
        Text planState = new Text();
        planState.set(state + "\t" + planId);
        context.write(planState, NullWritable.get());
        }catch(Exception e){
          System.out.println(e);
          }
    }
}
```

Reducer Class:

```
public class Distinct_Reducer extends Reducer<Text, NullWritable, Text, NullWritable>{
  public void reduce(Text key, Iterable<NullWritable> values, Context context)
      throws IOException, InterruptedException
  {
    context.write(key, NullWritable.get());
  }
}
```

Driver Class

```
public class Project_DistinctPattern {
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
    try{
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "Project_DistinctPattern");
      job.setJarByClass(Project_DistinctPattern.class);
      job.setMapperClass(Distinct_Mapper.class);
      job.setCombinerClass(Distinct_Reducer.class);
      job.setReducerClass(Distinct_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);
            }catch(Exception e){
                System.out.print("Exception:"+e);
            }
    }
    }
```

Min and Max Analysis

AgeCompositeKeyWritable

```
public class AgeCompostieKeyWritable implements
Writable,WritableComparable<AgeCompostieKeyWritable>{

    private String state;
    private String age;
     public AgeCompostieKeyWritable(){
    }
    public AgeCompostieKeyWritable(String y,String a){
        this.state = y;
        this.age = a;
    }

    public String getState() {
        return state;
    }
public void setState(String state) {
        this.state = state;
    }
public String getAge() {
        return age;
    }
public void setAge(String age) {
        this.age = age;
    }

    @Override
    public void write(DataOutput d) throws IOException {
      WritableUtils.writeString(d, state);
      WritableUtils.writeString(d, age);
    }
@Override
    public void readFields(DataInput di) throws IOException {
      state = WritableUtils.readString(di);
      age = WritableUtils.readString(di);
    }
@Override
    public int compareTo(AgeCompostieKeyWritable o) {
        int result = age.compareTo(o.age);
        if(result == 0){
            result = state.compareTo(o.state);
        }
        return result;
    }

public String toString(){
        return state + "\t" + age;
    }
}
```

MinMaxCountTuple

```
public class MinMaxCountTuple implements Writable{
    private String min;
    private String max;
    private long count;
    public String getMin() {
        return min;
    }
public void setMin(String min) {
        this.min = min;
    }
public String getMax() {
        return max;
    }
public void setMax(String max) {
        this.max = max;
    }
public long getCount() {
        return count;
    }
public void setCount(long count) {
        this.count = count;
    }
    @Override
    public void write(DataOutput d) throws IOException {
        d.writeLong(count);
        WritableUtils.writeString(d, min);
        WritableUtils.writeString(d, max);
    }

    @Override
    public void readFields(DataInput di) throws IOException {
        count = di.readLong();
        min = WritableUtils.readString(di);
        max = WritableUtils.readString(di);
    }
    public String toString(){
        return min + "\t" + max + "\t" + count ;
    }
```

Mapper:

```
public class Rates_Mapper extends Mapper<Object, Text, AgeCompostieKeyWritable, MinMaxCountTuple>{
    MinMaxCountTuple tuple = new MinMaxCountTuple();
    public void map(Object key, Text value, Context context)
    {
        String[] tokens = value.toString().split(",");
        if(tokens[1].contains("state") && tokens[13].contains("age")){
        return;
        }
        else{
            AgeCompostieKeyWritable stateAge = new       AgeCompostieKeyWritable(tokens[1],tokens[13]);
        try{
            tuple.setMin(tokens[14]);
            tuple.setMax(tokens[14]);
            tuple.setCount(1);

            context.write(stateAge, tuple);
        }catch(IOException | InterruptedException | NumberFormatException  e){
            System.out.println("Error in Mapper" + e.getMessage());
        }
        }
     }
    }
```

Partitioner

```
public class Rates_Partitioner extends Partitioner<AgeCompostieKeyWritable, MinMaxCountTuple>{
    @Override
    public int getPartition(AgeCompostieKeyWritable key, MinMaxCountTuple value, int i) {
        if(!key.getAge().contains("Family")){
            return (key.getAge().hashCode()%i);
        }
        else{
            return 2;
        }
    }
}
```

Reducer:

```
public class Rates_Reducer extends
Reducer<AgeCompostieKeyWritable,MinMaxCountTuple,AgeCompostieKeyWritable,MinMaxCountTuple>{
    private MinMaxCountTuple result = new MinMaxCountTuple();
     public void reduce(AgeCompostieKeyWritable key, Iterable<MinMaxCountTuple> values, Context context)
throws IOException, InterruptedException{
         result.setMax(null);
         result.setMin(null);
        result.setCount(0);
        long sum = 0;
        for(MinMaxCountTuple val : values)
        {
        if(result.getMin()== null || val.getMin().compareTo(result.getMin()) <0)
        {
            result.setMin(val.getMin());
        }
       if(result.getMax()==null || val.getMax().compareTo(result.getMax())>0)
        {result.setMax(val.getMax());
        }
       sum +=  val.getCount();
      }
       result.setCount(sum);
        context.write(key,result);
   }
}
```

Driver class:

```
public class Project_RatesMinMax {
public static void main(String[] args) {
    try {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "RatesMinMax");
        job.setJarByClass(Project_RatesMinMax.class);
        job.setMapperClass(Rates_Mapper.class);
        job.setMapOutputKeyClass(AgeCompostieKeyWritable.class);
        job.setMapOutputValueClass(MinMaxCountTuple.class);
        job.setCombinerClass(Rates_Reducer.class);
        job.setReducerClass(Rates_Reducer.class);
        job.setOutputKeyClass(AgeCompostieKeyWritable.class);
        job.setOutputValueClass(MinMaxCountTuple.class);
        job.setPartitionerClass(Rates_Partitioner.class);
        //job.setNumReduceTasks(10);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        } catch (IOException | InterruptedException | ClassNotFoundException ex) {
         System.out.println("Main Error" + ex.getMessage());
         }
    }
}
```

Inverted Index

Mapper

```
public class Inverted_Mapper extends Mapper<Object, Text, Text, Text>{

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

        String fileName = ((FileSplit) context.getInputSplit()).getPath().getName();
        String[] line = value.toString().split(",");

        for(String s:line){
            context.write(new Text(s), new Text(fileName));
        }
    }

}
```

Reducer:

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

    private Text result = new Text();
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws InterruptedException, IOException
    {
      HashMap m=new HashMap();
      int count=0;

      for(Text t:values){
          String str=t.toString();
          if(m!=null &&m.get(str)!=null)
          {
              count=(int)m.get(str);
              m.put(str, ++count);
          }else{
              m.put(str, 1);
          }
       }

       context.write(key, new Text(m.toString()));

    }

}
```

Driver Class

```
public class Project_InvertedIndex {
public static void main(String[] args) {
       try{
            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "Inverted Index");
            job.setJarByClass(Project_InvertedIndex.class);
            job.setMapperClass(Inverted_Mapper.class);
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(Text.class);
            job.setReducerClass(Inverted_Reducer.class);
            job.setOutputKeyClass(Text.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);
            }catch(Exception e){
               System.out.print("Exception:"+e);
            }
    }
```

```
        }
```

Moving Average

```
public class PlanDateTuple implements Writable, WritableComparable<PlanDateTuple>{
    public String plan;
    public String month;

    public PlanDateTuple(){
    }
    public PlanDateTuple(String p, String m){
        this.plan = p;
        this.month = m;
    }
public String getPlan() {
        return plan;
    }
 public void setPlan(String plan) {
        this.plan = plan;
    }
 public String getMonth() {
        return month;
    }

    public void setMonth(String month) {
        this.month = month;
    }
@Override
    public void write(DataOutput d) throws IOException {
        WritableUtils.writeString(d, plan);
        WritableUtils.writeString(d, month);
    }

    @Override
    public void readFields(DataInput di) throws IOException {
        plan = WritableUtils.readString(di);
        month = WritableUtils.readString(di);
    }

    @Override
    public int compareTo(PlanDateTuple o) {
        int result = month.compareTo(o.month);
        if(result == 0){
            result = plan.compareTo(o.plan);
        }    return result;
    }

    public String toString(){
        return  "Plan:"+ plan + "\t" + "Month:" + month + "Moving Average:";
    }
}
```

Mapper:

```
public class Avg_Mapper extends Mapper<Object, Text, PlanDateTuple, DoubleWritable>{
    DoubleWritable rate = new DoubleWritable();
    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException
    {
        try {
            String[] values = value.toString().split(",");
            String date = values[8];
            DateFormat frmt = new SimpleDateFormat("mm/dd/yyyy");
            Date newDate = frmt.parse(date);
            DateFormat df = new SimpleDateFormat("mm");
            String month = df.format(newDate);
            PlanDateTuple planDate = new PlanDateTuple(values[10], month);
            rate.set(Double.parseDouble(values[14]));
            context.write(planDate, rate);
        } catch (ParseException ex) {
            Logger.getLogger(Avg_Mapper.class.getName()).log(Level.SEVERE, null, ex);
        }
}
}
```

Reducer:

```
public class Avg_Reducer extends Reducer<PlanDateTuple, DoubleWritable, PlanDateTuple, DoubleWritable>
{
    private  int windowSize = 3; // default
    private final ArrayList<Float> rates = new ArrayList<>();
  @Override
    public void reduce(PlanDateTuple key, Iterable<DoubleWritable> values, Context context)
            throws IOException, InterruptedException
    {
        double sum = 0;
        rates.clear(); //clear the list
        double mvAvg;
        for(DoubleWritable val: values)
        {rates.add((float)val.get());
        }
        for (int i=0; i < windowSize-1; i++)
        {
            sum += rates.get(i);
        }
        for (int i = windowSize-1; i < rates.size(); i++)
        {
            sum += rates.get(i);
            System.out.println(sum);
            mvAvg = (sum /windowSize);
            DoubleWritable movingAverage = new DoubleWritable();
            movingAverage.set(mvAvg);
            context.write(key, movingAverage);
            sum -= rates.get(i-windowSize+1);
        }
    }
}
```

Driver Class:

```
public class Project_MovingAvg {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try{
            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "Moving Average");
            job.setJarByClass(Project_MovingAvg.class);
            job.setMapperClass(Avg_Mapper.class);
            job.setMapOutputKeyClass(PlanDateTuple.class);
            job.setMapOutputValueClass(DoubleWritable.class);
            job.setCombinerClass(Avg_Reducer.class);
            job.setReducerClass(Avg_Reducer.class);
            job.setOutputKeyClass(PlanDateTuple.class);
            job.setOutputValueClass(DoubleWritable.class);
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            System.exit(job.waitForCompletion(true) ? 0 : 1);
            }catch(Exception e){
                System.out.print("Exception:"+e);
        }
    }

}
```

Median and Standard Deviation

```
public class MedStdCompositeKey implements Writable{

    double median;
    double standardDeviation;

    public MedStdCompositeKey(){

    }

    public MedStdCompositeKey(double median, double standardDeviation){
        this.median = median;
        this.standardDeviation = standardDeviation;
    }

    public void readFields(DataInput dataInput) throws IOException {
        median = Double.parseDouble(WritableUtils.readString(dataInput));
        standardDeviation = Double.parseDouble(WritableUtils.readString(dataInput));
    }

    public void write(DataOutput dataOutput) throws IOException {
         WritableUtils.writeString(dataOutput, String.valueOf(median));
         WritableUtils.writeString(dataOutput, String.valueOf(standardDeviation));
    }

    public double getMedian() {
        return median;
    }

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

    public double getStandardDeviation() {
        return standardDeviation;
```

```
   }

   public void setStandardDeviation(double standardDeviation) {
          this.standardDeviation = standardDeviation;
   }


   @Override
   public String toString() {
     // TODO Auto-generated method stub
          return (median + "\t"+standardDeviation);
   }

}
```

Mapper

```
public class MedStd_Mapper extends Mapper<Object, Text, Text, DoubleWritable> {

   private Text plan = new Text();
   private DoubleWritable rate = new DoubleWritable();

   public void map(Object key,Text value,Context context)
         //throws IOException, InterruptedException
   {
      try{
         String input = value.toString();
         String[] inputs = input.split(",");
             if(inputs[14].contains("[0-9]+") || inputs[14].contains("."))
         {
            rate.set(Double.parseDouble(inputs[14]));
            plan.set((inputs[10]));
            context.write(plan, rate);
         }

}catch (Exception ex) {
         Logger.getLogger(MedStd_Mapper.class.getName()).log(Level.SEVERE, null, ex);
      }
}
}
```

Reducer

```
public class MedStd_Reducer extends Reducer<Text, DoubleWritable, Text, MedStdCompositeKey>{


          ArrayList<Double> inputs = new ArrayList<Double>();
```

```
    MedStdCompositeKey mstd = new MedStdCompositeKey();

      public void reduce(Text key, Iterable<DoubleWritable> values,Context context)
        throws IOException, InterruptedException
    {
            double sum = 0.0;
            int count = 0;
            inputs.clear();

            for(DoubleWritable value:values)
       {
          inputs.add(value.get());
              sum += value.get();
          count++;
            }

            Collections.sort(inputs);
            double mean = 0.0;
            if(count != 0)
       {
             mean = sum / count;
            }else{
          mean = sum / 1;
            }


            //Find Median
            if(inputs.size() % 2 == 0){
                 int index = (int)(inputs.size()/2);
                 int index1 = index - 1;
                 mstd.setMedian( ((inputs.get(index1) + inputs.get(index))/2.0) );


            }else{
                 mstd.setMedian(inputs.get((int)(inputs.size()/2)));
            }

            //Find Standard Deviation
            double sumOfSquares = 0.0;
            for(Double eachValue: inputs){
                 sumOfSquares += (eachValue - mean)*(eachValue - mean);
            }

            if((inputs.size() - 1) != 0){
                 mstd.setStandardDeviation( Math.sqrt(sumOfSquares/(inputs.size()-1)) );
```

```
                }else{
                        mstd.setStandardDeviation( Math.sqrt(sumOfSquares/1) );
                }

                context.write(key, mstd);
        }

        public ArrayList<Double> getArrayList(){
                if(this.inputs == null){
                        inputs = new ArrayList<Double>();
                }
                return inputs;
        }
}
```

Driver Class

```
public class Project_MeanStdDev {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Configuration conf = new Configuration();
            try{
Job job = Job.getInstance(conf, "Median & Standard Deviation");
              job.setJarByClass(Project_MeanStdDev.class);
              job.setMapperClass(MedStd_Mapper.class);
              job.setReducerClass(MedStd_Reducer.class);
              job.setMapOutputKeyClass(Text.class);
              job.setMapOutputValueClass(DoubleWritable.class);
              job.setOutputKeyClass(Text.class);
              job.setOutputValueClass(MedStdCompositeKey.class);
              FileInputFormat.addInputPath(job, new Path(args[0]));
          FileOutputFormat.setOutputPath(job, new Path(args[1]));
              System.exit(job.waitForCompletion(true) ? 0 : 1);
          }catch (IOException e) {
                  System.out.println("IOException Inside Stock Price Driver");
              }catch (ClassNotFoundException e) {
                  System.out.println("Class Not Found Exception in Stock Price Driver");
              }catch(InterruptedException e){
                  System.out.println("Interrupted Exception in Stock Price Driver");
              }
        }
```

Secondary Sort

```
public class StateAreaRateTuple implements Writable, WritableComparable<StateAreaRateTuple>{
    public String state;
    public String ratingArea;
    @Override
    public void write(DataOutput d) throws IOException {
        WritableUtils.writeString(d,state);
        WritableUtils.writeString(d, ratingArea);
    }

    @Override
    public void readFields(DataInput di) throws IOException {
        state = WritableUtils.readString(di);
        ratingArea = WritableUtils.readString(di);
    }

    @Override
    public int compareTo(StateAreaRateTuple o) {
        int compareValue = this.ratingArea.compareTo(o.getRatingArea());
        if(compareValue==0){
            compareValue = state.compareTo(o.getState());
        }
        return -1*compareValue;
    }

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

    public String getRatingArea() {
        return ratingArea;
    }

    public void setRatingArea(String ratingArea) {
        this.ratingArea = ratingArea;
    }

    public String toString(){
        return state + "\t" + ratingArea;
    }

}

public class Project_GroupingComparator extends WritableComparator{

    public Project_GroupingComparator(){
        super(StateAreaRateTuple.class,true);
    }


    @Override
    public int compare(WritableComparable wc1, WritableComparable wc2){
        StateAreaRateTuple tuple1 = (StateAreaRateTuple)wc1;
        StateAreaRateTuple tuple2 = (StateAreaRateTuple)wc2;
        return tuple1.getRatingArea().compareTo(tuple2.getRatingArea());
    }
}

public class SecondarySort_Mapper extends Mapper<Object, Text, StateAreaRateTuple, NullWritable>{

    private final static StateAreaRateTuple tuple = new StateAreaRateTuple();
    private Text rate = new Text();

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

        String[] tokens = value.toString().split(",");
        rate.set(tokens[11]);
        tuple.setState(tokens[3]);
        tuple.setRatingArea(tokens[7]+"\t"+tokens[8] +"\t"+ tokens[9]+"\t"+tokens[10] +"\t"+tokens[11] );
        context.write(tuple, NullWritable.get());
    }

}

*/
public class Project_Partitioner extends Partitioner<StateAreaRateTuple,NullWritable>{

    @Override
    public int getPartition(StateAreaRateTuple key, NullWritable value, int i) {
        return Math.abs(key.state.hashCode()%i);
    }

}

*/
public class SecondarySort_Reducer extends Reducer<StateAreaRateTuple, NullWritable, StateAreaRateTuple,
NullWritable>{

    @Override
    public void reduce(StateAreaRateTuple key, Iterable<NullWritable> values, Context context) throws
IOException{
        try{
            for(NullWritable val:values){
                context.write(key, val);
            }
        }catch(Exception e){
            System.out.println(e);
        }
    }

}

public class Project_SecondarySort {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
            try {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "SecondarySort");
        job.setJarByClass(Project_SecondarySort.class);
        job.setMapperClass(SecondarySort_Mapper.class);
        job.setMapOutputKeyClass(StateAreaRateTuple.class);
        job.setMapOutputValueClass(NullWritable.class);
        //job.setCombinerClass(SecondarySort_Reducer.class);
        job.setPartitionerClass(Project_Partitioner.class);
        job.setGroupingComparatorClass(Project_GroupingComparator.class);
        job.setNumReduceTasks(6);
        job.setReducerClass(SecondarySort_Reducer.class);
        job.setOutputKeyClass(StateAreaRateTuple.class);
        job.setOutputValueClass(NullWritable.class);
        job.setPartitionerClass(Project_Partitioner.class);
        //job.setNumReduceTasks(10);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        } catch (IOException | InterruptedException | ClassNotFoundException ex) {
         System.out.println("Main Error" + ex.getMessage());
         }
    }

}
```

Reduce-Side Join

Area_mapper

```
public class Area_Mapper extends Mapper<Object, Text, Text, Text> {

    private Text issuerID = new Text();
    private Text areaValue = new Text();

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException
    {
        try{
        String[] tokens = value.toString().split(",");
             String id = tokens[3];

        if (id == null)
        {
            return;
    }

        issuerID.set(id);

        areaValue.set("A" + tokens[2]+"\t"+tokens[4] +"\t"+tokens[10]+ "\t"+tokens[11]);
        context.write(issuerID, areaValue);
  }catch(Exception e){
    System.out.println("Exception is:"+ e.getMessage());
  }
  }
}
```

Network_Mapper

```
public class Network_Mapper extends Mapper<Object, Text, Text, Text>{

  private Text issuerID = new Text();
  private Text rateValue = new Text();

  public void map(Object key, Text value, Context context)
      throws IOException, InterruptedException
  {
    try{
    String[] tokens = value.toString().split(",");
        String id = tokens[3];

        if (id == null)
    {
      return;
    }

    issuerID.set(id);
        rateValue.set("R" + tokens[9] + "\t" + tokens[13] + "\t" + tokens[14]);
        context.write(issuerID, rateValue);
  }catch(Exception e){
    System.out.println("Exception is:"+e.getMessage());
  }
  }
}
```

Reducer

```
public class Joins_Reducer 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> listR = 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();
     listR.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"))
        {
           System.out.println("here4");
           listA.add(new Text(tmp.toString().substring(1)));
        }


        if (Character.toString((char) tmp.charAt(0)).equals("R"))
        {
           System.out.println("here5");
           listR.add(new Text(tmp.toString().substring(1)));
        }

           System.out.println(tmp);
        }


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

        System.out.println(listR.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
            if (!listA.isEmpty() && !listR.isEmpty()) {
                System.out.println("here");
                for (Text A : listA) {
                    //System.out.println("here1");
                    for (Text R : listR) {
                        //System.out.println("here2");
                        context.write(A, R);
                    }
                }
            }
        } 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 (!listR.isEmpty()) {
                    for (Text R : listR) {
                        context.write(A, R);
                    }
                } else {
                    // Else, output A by itself
                    context.write(A, EMPTY_TEXT);
                }
            }
        }
    }
}
```

Driver Class

```
public static void main(String[] args) throws InterruptedException, ClassNotFoundException {
    try {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Joins");
        job.setJarByClass(Project_Joins.class);
        MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, Area_Mapper.class);
        MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, Network_Mapper.class);
        job.getConfiguration().set("join.type", "inner");
        //job.setNumReduceTasks(0);
        job.setReducerClass(Joins_Reducer.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        TextOutputFormat.setOutputPath(job, new Path(args[2]));
```

```
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);
            job.waitForCompletion(true);
        } catch (IOException ex) {
            Logger.getLogger(Project_Joins.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

Partitioning and Binning

Binning

```
public static class Binning_Mapper extends Mapper<Object, Text, Text, Text>{
        private MultipleOutputs<Text,Text> mos = null;
        private Text outKey = new Text();
        protected void setup(Context context)
        {
            mos = new MultipleOutputs(context);
        }

        protected void map(Object key, Text value, Context context) throws IOException, InterruptedException
        {
            String[] rows = value.toString().split(",");
            String s = rows[3];

for(String k: rows){
               String StateCode = s;
if(StateCode.equalsIgnoreCase("AK")){
                    mos.write("bins",value,NullWritable.get(),"AK");
                }
                if(StateCode.equalsIgnoreCase("AL")){
                    mos.write("bins",value,NullWritable.get(),"AL");
                }
                if(StateCode.equalsIgnoreCase("AZ")){
                    mos.write("bins",value,NullWritable.get(),"AZ");
                }
                if(StateCode.equalsIgnoreCase("FL")){
                    mos.write("bins", value,NullWritable.get(),"FL");
                }
                if(StateCode.equalsIgnoreCase("GA")){
                    mos.write("bins", value,NullWritable.get(),"GA");
                }
                if(StateCode.equalsIgnoreCase("IN")){
                    mos.write("bins", value,NullWritable.get(),"IN");
                }
                if(StateCode.equalsIgnoreCase("LA")){
                    mos.write("bins", value,NullWritable.get(),"LA");
                }
                if(StateCode.equalsIgnoreCase("MO")){
                    mos.write("bins", value,NullWritable.get(),"MO");
                }
                if(StateCode.equalsIgnoreCase("MS")){
                    mos.write("bins",value,NullWritable.get(),"MS");
                }
                if(StateCode.equalsIgnoreCase("NC")){
                    mos.write("bins", value,NullWritable.get(),"NC");
                }
                if(StateCode.equalsIgnoreCase("NJ")){
                    mos.write("bins", value,NullWritable.get(),"NJ");
                }
                if(StateCode.equalsIgnoreCase("OK")){
                    mos.write("bins", value,NullWritable.get(),"OK");
                }
                if(StateCode.equalsIgnoreCase("PA")){
                    mos.write("bins", value,NullWritable.get(),"PA");
                }
                if(StateCode.equalsIgnoreCase("SC")){
                    mos.write("bins", value,NullWritable.get(),"SC");
                }

                if(StateCode.equalsIgnoreCase("TN")){
                    mos.write("bins", value,NullWritable.get(),"TN");
                }
                if(StateCode.equalsIgnoreCase("TX")){
                    mos.write("bins", value,NullWritable.get(),"TX");
                }
                if(StateCode.equalsIgnoreCase("WI")){
                    mos.write("bins", value,NullWritable.get(),"WI");
                }
                if(StateCode.equalsIgnoreCase("WY")){
                    mos.write("bins", value,NullWritable.get(),"WY");
                }

            }

        }

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

```
        }
    }

public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Binning");
        job.setJarByClass(Project_BenefitsBinning.class);
        job.setMapperClass(Binning_Mapper.class);
        job.setNumReduceTasks(0);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        //MultipleOutputs.addNamedOutput(job, namedOutput, outputFormatClass, keyClass, valueClass);
        MultipleOutputs.addNamedOutput(job, "bins", TextOutputFormat.class, Text.class, NullWritable.class);
        MultipleOutputs.setCountersEnabled(job, true);

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

Parttitioning

```
public class Partitioning_Mapper extends Mapper<Object,Text,Text,Text>{
    private Text outKey = new Text();
    public void map(Object key, Text value, Context context)
            throws IOException,InterruptedException
    {
        try{
        String[] rows = value.toString().split(",");
        outKey.set(rows[4]);
        context.write(outKey, new Text(value));
        }catch(Exception e){
            System.out.println("Exception e"+ e.getMessage());
        }
    }
}
```

Partitioner:

```
public class Partitioning_Partitioner extends Partitioner<Text,Text> {
    @Override
    public int getPartition(Text key, Text value, int numOfPartitions) {
        return(key.hashCode()%numOfPartitions);
    }
}
```

Reducer

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

    public void reduce(Text key,Iterable<Text> values, Context context)
            throws IOException, InterruptedException
    {
        for(Text t: values)
        {
            context.write(key,t);
        }

    }

}
```

Driver Class

```
public class Project_Partitioning {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Partitioning");
        job.setJarByClass(Project_Partitioning.class);
        job.setMapperClass(Partitioning_Mapper.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        job.setPartitionerClass(Partitioning_Partitioner.class);
        job.setReducerClass(Partitioning_Reducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setNumReduceTasks(12);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true)?0:1);
    }
}
```

Bloom Filter: Plan Benefits

```
//PlanBenefits class
public class PlanBenefits {
  final String benefitName;
   final String isCovered;
   PlanBenefits(String bn, String c) {
      this.benefitName = bn;
      this.isCovered = c;
  }
}
```

Mapper

```
public class BloomFilter_Mapper extends Mapper<Object, Text, Text, NullWritable> {


    Funnel<PlanBenefits> pfunnel = new Funnel<PlanBenefits>() {

      @Override
      public void funnel(PlanBenefits p, Sink into) {
          into.putString(p.benefitName, Charsets.UTF_8).putString(p.isCovered, Charsets.UTF_8);

       }
   };


   private BloomFilter<PlanBenefits> planBenefitsFilter = BloomFilter.create(pfunnel, 500, 0.1);

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

      PlanBenefits p1 = new PlanBenefits("Accidental Dental", "Covered");
      ArrayList<PlanBenefits> planBenefitsList = new ArrayList<PlanBenefits>();
          planBenefitsList.add(p1);

      for (PlanBenefits pb : planBenefitsList) {
          planBenefitsFilter.put(pb);
      }

   }


   @Override
```

```
    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException
    {
        String values[] = value.toString().split(",");
             PlanBenefits b = new PlanBenefits(values[0], values[1]);
             if (planBenefitsFilter.mightContain(b)) {
            context.write(value, NullWritable.get());

        }
    }

}
```

Driver Class

```
public class Project_BloomFIlter {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
        try {
            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "Bloom Filter");
            job.setJarByClass(Project_BloomFIlter.class);
            job.setMapperClass(BloomFilter_Mapper.class);
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(NullWritable.class);
            job.setNumReduceTasks(0);
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            boolean success = job.waitForCompletion(true);
            System.out.println(success);
        } catch (InterruptedException ex) {
            Logger.getLogger(Project_BloomFIlter.class.getName()).log(Level.SEVERE, null, ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(Project_BloomFIlter.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

}
```

Top N Analysis

```
public class SortMapper extends Mapper<LongWritable, Text, LongWritable, Text>
```

```
{
   Text planId = new Text();
   LongWritable count = new LongWritable();

   public void map(LongWritable key, Text value, Context context)
   {
      try{
         String[] tokens =value.toString().split("\t");
         planId.set(tokens[0]);
         count.set(Long.parseLong(tokens[1]));
         context.write(count, planId);
      }catch(Exception e){
         System.out.println(e);
      }
   }
}

public class SortReducer extends Reducer<LongWritable, Text, LongWritable, Text>
{
   public void reduce(LongWritable key, Iterable<Text> values, Context context){
      for(Text val: values){
         try {
            context.write(key, val);
         } catch (Exception ex) {
            Logger.getLogger(SortReducer.class.getName()).log(Level.SEVERE, null, ex);
         }
      }
   }
}

public class TopMapper extends Mapper<Object, Text, Text, IntWritable>
{
      Text planId = new Text();
      IntWritable result = new IntWritable(1);


      public void map(Object key, Text value, Context context)
      {
         String[] values = value.toString().split(",");

         if(values[1].matches("Covered")){
            try {
               System.out.println("covered");

               planId.set(values[2]);
```

```
                context.write(planId, result);
            } catch (IOException ex) {
                Logger.getLogger(TopMapper.class.getName()).log(Level.SEVERE, null, ex);
            } catch (InterruptedException ex) {
                Logger.getLogger(TopMapper.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
}
}

public class TopReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
    int sum = 0;
    IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
    {
        try {
            for(IntWritable val:values){
                sum +=val.get();
            }

            result.set(sum);
            context.write(key, result);
        } catch (IOException ex) {
            Logger.getLogger(TopReducer.class.getName()).log(Level.SEVERE, null, ex);
        } catch (InterruptedException ex) {
            Logger.getLogger(TopReducer.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
  }

public class Project_TopN {

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {

      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "Top10");
      job.setJarByClass(Project_TopN.class);
      job.setMapperClass(TopMapper.class);
      job.setMapOutputKeyClass(Text.class);
```

```
        job.setMapOutputValueClass(IntWritable.class);
        job.setReducerClass(TopReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);

        //Configuration conf2 = new Configuration();
        Job job2 = new Job(conf, "Sort");
        job2.setMapperClass(SortMapper.class);
        job2.setMapOutputKeyClass(LongWritable.class);
        job2.setMapOutputValueClass(Text.class);
        job2.setSortComparatorClass(LongWritable.DecreasingComparator.class);
        System.out.println("job2running");

        job2.setReducerClass(SortReducer.class);
        job2.setOutputKeyClass(LongWritable.class);
        job2.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job2, new Path(args[1]));
        FileOutputFormat.setOutputPath(job2, new Path(args[2]));
        System.exit(job2.waitForCompletion(true) ? 0 : 1);

    }

}
```