CSCE 566
Term Project Final Report

Authors
Xinzheng Zhao xxz4865 - responsible for Method (1)
Yee H. Wong yxw0087 - responsible for Method (2)

Title
Ranking Multi-Dimensional Instances using Partial Ordering Methods

Extended Abstract
This project is specifically devoted to solving ranking problems of elements with multiple attributes with partial order ranking method. We will be using two different existing algorithms to rank the vulnerability of parishes against the environmental hazards by evaluating multiple attributes of these cities such as the median age, birthrate, incomes, ethnicities, unemployed rate of the citizens and many others.

The challenge of the problem we are solving is that there are as many as 42 attributes for each of the 64 parishes. In real situation, each of these attributes should have a different contributions or weights to the problem we are solving. However, there was no such information available to us and we had to make the assumption that they carry the same weight. Another challenge is that the algorithms or methods that we adapted from two research papers about partial ordering may not work perfectly well for our problem. A few changes had to be made to these algorithms so that we can get the desired results.

The dataset used in this project was given to us by Mr. Amirhossein Tavanaei, which is a tabular excel spreadsheet that contains all the 64 parishes of Louisiana in rows and their 42 attributes in columns. We planned to pick two well known existing partial ordering algorithms and apply them to the problem with the code implemented in Java. Our results show two different possible ways to rank these parishes with big dissimilarity in the final rankings.

Introduction

The goal of this project is to rank each and every of the 64 parishes in Louisiana, United States, based on their vulnerabilities to the environmental hazards. A parish can be considered to be more vulnerable to the environmental hazards if, for example, the citizens of the area suffer a lot more when such hazards happen, or if it takes the state government a significantly larger effort to recover the city, and so on. Each of these parishes come with 42 attributes as mentioned above. To properly rank them based on these attributes, it is essential to use some partial ordering methods.

Ordering elements with multiple attributes are not as easy as ordering single-attributed elements. Consider two elements with multiple numerical-valued attributes. They are considered comparable if all attributes of one element are either larger than or smaller than the corresponding attributes of another element. However, if some attributes of element 1 are smaller than the corresponding attributes of another element 2, while some of element 1's attributes are larger than that of element 2, the two elements become uncomparable. Several partial ordering algorithms may handle this problem differently and provide different results. In this project, we reconstruct two partial order ranking algorithms, Hierarchical Partial Order Ranking(HPOR) and pair-wise partial ranking and compared the two algorithms in terms of code complexity and similarities of the results.

Methods

To approach the problem, we choose two known partial ordering methods that were derived by other scientists and test their functionalities in this problem.

The first method employed in this project is what call hierarchical partial order ranking (HPOR) proposed by Lars Carlson[1]. The algorithm tries to order the elements with multiple attributes by means of meta-descriptors, which can be obtained by applying formula (1) to the data.

$$\mathrm{Rk}_{\mathrm{av}}(c_i) = (N+1) - (S(c_i)+1) \times (N+1)/(N+1-U(c_i)) \quad (1)$$

Where N is the number of elements in the diagram, S the number of successor, which is defined here as the element each of whose attributes is less than the corresponding attributes of Ci, the current element in question and U is the number of elements that are incomparable to Ci. The value obtained from this formula servers as the meta-descriptor as the input of next round of calculation. After the first round of calculation, the result is also called as “average rank”, which means the average ordering score in terms of each single attribute. Then in the second step, all the values are multiplied by -1 to reverse the order because, smaller value indicates higher level in average rank.

The algorithm will be best explained with an example. In the paper, the author chose 16 elements with three attributes each of which further has three measurements as shown in the following figure.

Initial dataset

Site No	Da1	Da2	Da3	Db1	Db2	Db3	Dc1	Dc2	Dc3
1	0.44	0.75	0.17	0.93	0.72	0.42	0.87	0.41	0.69
2	0.85	0.32	0.01	0.44	0.67	0.86	0.50	0.84	0.29
3	0.41	0.25	0.39	0.63	0.98	0.77	0.13	0.75	0.15
4	0.01	0.98	0.64	0.40	0.58	0.58	0.15	0.43	0.77
5	0.26	0.07	0.07	0.74	0.42	0.20	0.37	0.51	1.00
6	0.97	0.45	0.95	0.55	0.52	0.68	0.22	0.33	0.27
7	0.97	0.52	0.42	0.06	0.64	0.27	0.72	0.96	0.98
8	0.92	0.10	0.63	0.11	0.74	0.67	0.42	0.68	0.99
9	0.52	0.50	0.87	0.65	0.02	0.99	0.84	0.78	0.81
10	0.43	0.39	0.35	0.30	0.72	0.22	0.69	0.92	0.92
11	0.73	0.08	0.31	0.55	0.81	0.71	0.07	0.40	0.25
12	0.92	0.87	0.89	0.44	0.35	0.45	0.62	0.18	0.65
13	0.02	0.55	0.42	0.58	0.81	0.89	0.63	1.00	0.26
14	0.04	0.19	0.68	0.36	0.96	0.95	0.26	0.05	0.97
15	0.02	0.03	0.79	0.35	0.11	0.37	0.10	0.66	0.24
16	0.74	0.60	0.20	0.01	0.54	0.16	0.24	0.05	0.22

Figure 1. Initial Raw Data

Da’s are the measurements of the first attribute “atmosphere”, Db’s the second attribute “drinking water”, and Dc’s the third “social factors”.

First average rank are calculated by counting successors and incomparable elements and feeding those counts to formula (1). The results are shown in the figure below.

Site No.	Atmospheric impact		Ground water impact		Socio-economic factors	
	Rk_{av}	RLE	Rk_{av}	RLE	Rk_{av}	RLE
1	8.5	8.5	2.4	4.4	2.8	5.3
2	13.6	10.7	4.9	6.3	5.7	7.0
3	12.1	10.7	1.5	2.2	14.6	12.3
4	8.5	8.6	10.6	10.3	12.8	10.8
5	15.6	15.5	11.3	10.0	2.4	4.7
6	1.7	2.9	9.7	8.2	15.1	12.8
7	2.4	4.2	13.6	12.6	1.4	2.1
8	8.5	7.6	10.6	9.5	2.1	4.4
9	4.3	4.5	8.5	8.5	1.9	4.2
10	12.1	10.7	12.8	11.1	3.1	4.2
11	12.1	11.5	3.1	4.4	15.5	14.1
12	1.2	1.4	12.1	11.3	12.1	10.6
13	11.3	9.1	1.5	2.6	2.8	5.5
14	13.6	10.8	2.4	4.6	13.6	11.2
15	13.6	10.7	15.5	14.6	14.9	12.6
16	8.5	8.5	15.6	15.3	15.5	14.1

Figure 2. Average Rank After the First Round

After the first round, each element still has three attributes, but each attributes has only one measurements instead of three. HPOR reduces the number of attributes in this way. With the meta-descriptors or average ranks in hand, we can proceed to the second round of calculation in the exactly the same way. The final ranking is given in Figure 3.

Site No.	Rk_{av}	RLE
1	1.9	3.2
2	10.2	9.7
3	6.8	7.4
4	9.7	9.9
5	12.8	10.7
6	5.7	7.5
7	4.3	7.1
8	4.3	5.4
9	2.1	2.9
10	12.1	10.9
11	13.6	11.8
12	4.3	7.1
13	2.1	2.9
14	10.2	9.7
15	15.7	15.4
16	15.1	14.3

Figure 3. Final Ranking Results

The project was done in Java. Each parish is made into a parish object that has a parish name attribute and an array of 42 elements each of which corresponds to one attribute in the data. The attributes are divided into smaller groups and find the average rank in each group. This procedure repeats until only one average rank is obtained, which is final ranking of each parish.

The second method used in this project is the partial ordering algorithm proposed by Kirkwood and Sarin [2]. This algorithm first computes pair-wise rankings of the relative vulnerabilities between each individual pair of parishes to the environmental hazards using the following formula:

$$\sum_{i=1}^{n} k_i v_i(x_i') > \sum_{i=1}^{n} k_i v_i(x_i''), \qquad (2)$$

where the left hand side may represent parish A and right hand side represents parish B. x_i's are the attributes of each parish starting from i=0, 1, … n, in our case, n equals 42. v_i will be a function to normalize each of these attributes so that they can be more comparable, and k_i is the weight constant for each attribute. In the original algorithm, the weight constant should be different for every attribute according to how important the attribute is to the user. But in our project, we cannot simply determine how each attribute is important or unimportant, since the data was given to us with no such indications from the client, each attribute has to be treated as equally important, which are given the weights of positive one. The only information we know about the data is that attributes that are marked red in the title indicate that they are the attributes that will decrease environmental hazard. These particular attributes will be associated with weights of negative one instead. Therefore, a parish with a lower score With this being said, a parish A that is greater than parish B using equation (2) can be seen as being more undesirable because it is more vulnerable to the environmental hazards.

When the pair-wise rankings are obtained for each possible pairs of parishes, we then proceed to the second step of the algorithm. First we construct a table like this:

	A_1	A_2	A_3	A_4	A_5	A_6
A_1	0	0	1	1	1	1
A_2	0	0	1	1	1	1
A_3	0	0	0	0	1	1
A_4	0	0	0	0	0	1
A_5	0	0	0	0	0	1
A_6	0	0	0	0	0	0
Column totals	0	0	2	2	3	5

Where A1 to A6 are parishes 1 to 6, for example. An entry of 1 is placed for an element of the table if the row parish is preferred to the column parish, otherwise enter a 0. The total number of 1's in each column is determined and listed at the bottom of that column, these numbers stand for the numbers of parishes that are known to be preferred to the parishes listed at the top of the columns.

Next, we construct another table as in:

Required No. of Alternatives	Column Totals	Alternatives with Column Totals	No. of Alternatives with Column Totals	Cumulative No. of Alternatives with Column Totals
1	5	A_6	1	1
2	4	—	0	1
3	3	A_5	1	2
4	2	A_3, A_4	2	4
5	1	—	0	4
6	0	A_1, A_2	2	6

Then we fill in the five columns according to the previous table. Lines are drawn across the table immediately below each row where the entries in the first and fifth columns are equal. In this example, the lines will partition the 6 parishes into three groups. From here, we can conclude that the row on top contains the least preferable parishes (in this case, only A6), and the rows below are increasingly more preferable than the ones above them, with row 6 (parish A1 and A2) being the most preferable (least vulnerability to environmental hazards).

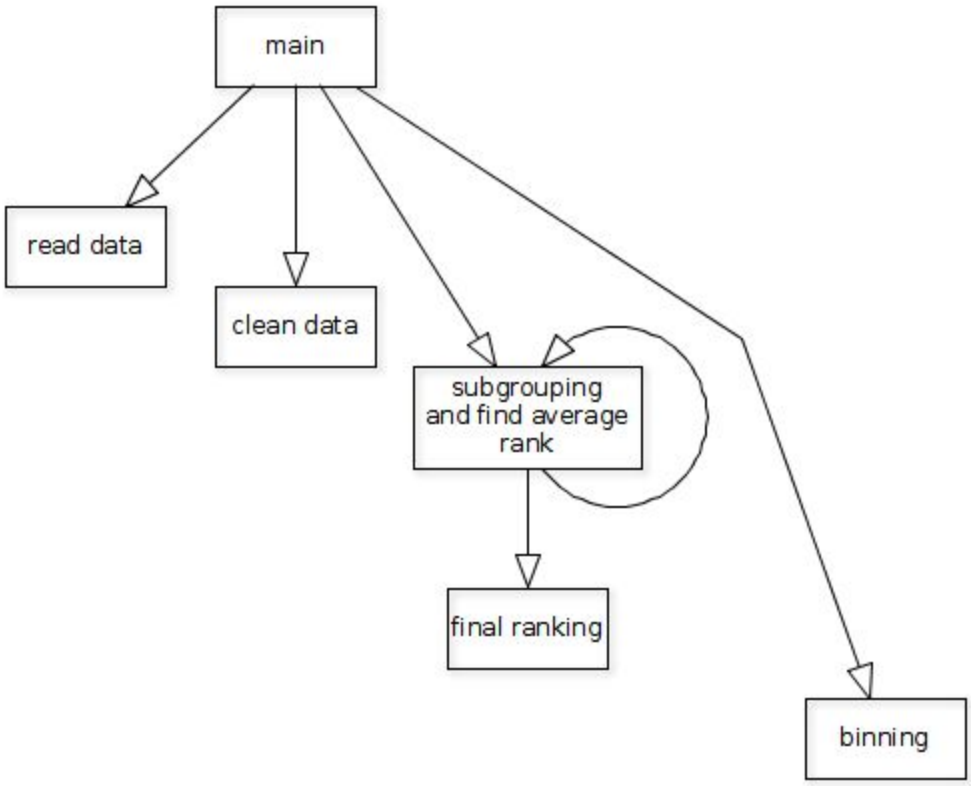
The implementation of the above algorithm was done in Java, with each parish being treated as an object. The original data was transformed into text files and the program will first read the data file to load them into a vector. Equation (2) is then performed on every single pair of parishes, and using a sorting method we can sort the vector into the ranked order that we want. The result of this sorting contain 64 different ranks and may not be too interesting to the user, so a further equiv-width binning method is used such that these 64 ranked objects are partitioned into 7 bins. In each bins, the parishes that are more similar to each other in terms of vulnerability are grouped together. The result will be shown in the result section later in this report.

Pseudo Code

For Method (1),

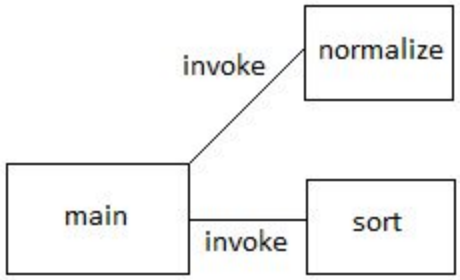
1. Read data from the file and store them each parish object.
2. Divide the attributes into smaller and find the average rank by each group.
3. Repeat step 2 until only one average rank score is generated.
4. Put each parish in different bins depending on their average ranks.

The program creates 64 parish objects each of which encapsulates an array of 42 elements to hold the 42 attribute values.



For Method (2),

1. Read data from file and store into vector
2. Call normalize function
3. Assign weights
4. Carry out equation (2) for each parish
5. Sort the vector
6. Equiv-width binning
7. Display result



The main function invokes two other functions, normalize(data) and sort(parish), where data is the raw data vector and parish is the parish vector that stores the processed data. All three functions are in the same class of HazardRanking.

The main data structure used in this method is vector and object class.

Result Description

For Method (1),

The final average rank was obtained, and all the rank scores were put in different bins. For more exhaustive results, see the result section below.

The time complexity of the algorithm is O(n^2d) where d is the number of attributes. Although it is hard to reduce the time complexity, better results might be obtained by more reasonable grouping of the attributes since the results depends on how to group the attributes.

For Method (2),

The implementation ranks 64 different parishes in Louisiana according to their vulnerabilities to the environment hazards by partial ordering. The partial ordering method was adapted from Kirkwood and Sarin and was not added with any new functionality.

No test cases were required for this implementation, the only test data is the original data which is attached with the submission.

Minor problem with this implementation will be the complexity of the sorting method used, which is the Insertion Sort. It can be changed to use heap sort for lower complexity of O(nlogn) instead of O(n^2).

Future enhancements include asking the client for his choice of weights for each attributes. In this way, the weights will not be just plain 1.0 or -1.0 and could lead the ranking closer to the user's belief. Several researches might be needed to choose some good weights for each attributes.

References

[1] Lars Carlsen, "Hierarchical Partial Order Ranking," in *Proc of ScienceDirect Environmental Pollution 155. Page 247- 253*. 2008

[2] Craig W. Kirkwood and Rakesh K. Sarin, "Ranking with Partial Information: A Method and an Application," in *Proc of Operations Research, Vol. 33, No. 1 (Jan. - Feb., 1985), pp. 38-48*

Source Code

For Method (1),

DataMatrix.java.

```
//this package is used to read and process raw data
package DataMatrix;
import java.io.*;
import java.util.*;
public class DataMatrix // an array list of array lists of
{

   public DataMatrix() // constructor
   {}

   //read the raw data from a csv file to the memory
   public ArrayList<ArrayList<String>> getRawData(String csvFileName) throws
IOException
   {

         String line = null;
         BufferedReader stream = null;
         ArrayList<ArrayList<String>> csvData = new
ArrayList<ArrayList<String>>();
```

```
        try
        {
                stream = new BufferedReader(new FileReader(csvFileName));
                    while ((line = stream.readLine()) != null)
                    {
                String[] splitted = line.split(",");
                ArrayList<String> dataLine = new ArrayList<String>(splitted.length);
                for (String data : splitted)
                        dataLine.add(data);

                csvData.add(dataLine);
                }
        }
        finally
        {
                if (stream != null)
                stream.close();
                }

        return csvData;

    }

// this function is used to clean data. Some data has “18,110” format, this
// function converts it to 18110 format
   public ArrayList<ArrayList<String>> cleanData(ArrayList<ArrayList<String>>
datalists)
    {
        ArrayList<ArrayList<String>> lists = new ArrayList<ArrayList<String>>();

        int rows = datalists.size();
        int cols = datalists.get(2).size();

       for(int i = 2; i < rows; i++)
        {
```

```
                ArrayList<String> list = new ArrayList<String>();
                for(int j = 2; j < cols; j++)
                {
                        if(j >= 3 && j <= 6)
                        {
                                String temp="";
                                String[] parts = datalists.get(i).get(j).split("\"");
                                temp = parts[1];
                                parts = datalists.get(i).get(j+1).split("\"");
                                temp +=parts[0];

                                list.add(temp);
                                j++;
                        }
                        else
                        { list.add(datalists.get(i).get(j));}
        }

                lists.add(list);
        }

        return lists;
   }
}
```

Operation.java

```
import DataMatrix.*;
import java.io.*;
import java.util.*;

public class operation
{
   public static void main(String[] args) throws FileNotFoundException,
IOException
   {
```

```
        DataMatrix dm = new DataMatrix(); // make a DataMatrix object to read
data
        ArrayList<ArrayList<String>> rawlists = new
ArrayList<ArrayList<String>>();//store raw data
        ArrayList<ArrayList<String>> cleanedLists = new
ArrayList<ArrayList<String>>();//store cleaned data
        rawlists = dm.getRawData("datafile.csv");//read in raw data
        int numParish = rawlists.size()-2;
        cleanedLists = dm.cleanData(rawlists);



        parish[] parishes = new parish[numParish];
        double[][] tempResults = new double[numParish][1];//store first round
results
        double[][] firstResults = new double[numParish][14];// 14 = 42/3, each 3
cols as a group
        //double[][] secondResults = new double[numParish][
        double[][] secondResults = new double[numParish][7];
        double[][] thirdResults = new double[numParish][2];
        double[][] lastRank = new double[numParish][1];
        initialize(parishes, rawlists, cleanedLists); //objects with values created
        double[][] initialMatrix = getInitialMatrix(parishes);



        double[][] list1 = new double[numParish][1];
        double[][] list2 = new double[numParish][1];



        firstResults = getResults(initialMatrix, parishes,
parishes[0].getAttris().length, 3, numParish);
        secondResults = getResults(firstResults, parishes, firstResults[0].length,
2, numParish);
        subRank(secondResults, parishes, list1, 0, 4);
        subRank(secondResults, parishes, list2, 4, 7);

        for(int i=0; i<numParish; i++)
```

```
        {
                thirdResults[i][0]=list1[i][0];
                thirdResults[i][1]=list2[i][0];
        }
        reverse(thirdResults);
        subRank(thirdResults, parishes, lastRank, 0, 2);

        for(int i = 0;  i < numParish; i++)
        {
                parishes[i].setScore(lastRank[i][0]);
                parishes[i].setBin();
        }

        for(int i=0; i<numParish; i++)
        {
                System.out.printf("%-30s%-7.1f%-12s%-2d\n",
parishes[i].getName(), parishes[i].getScore(), "bin number: ",
parishes[i].getBin());}
        System.out.println("");


   }

   public static double[][] getResults(double[][] matrix, parish[] parishes, int
numAttris, int groupsize, int numParish)
   {
        int cols = numAttris/groupsize;
        double[][] tempResults = new double[numParish][1];
        double[][] firstResults = new double[numParish][cols];
        int numCols = matrix[0].length;
        int nthCol = 0;
        for(int i = 0; i < numCols; i+=groupsize)
        {
                subRank(matrix, parishes, tempResults, i, i+groupsize);
                copyBack(firstResults, tempResults, nthCol);
                nthCol++;
```

```
        }
        reverse(firstResults);
        return firstResults;
    }

    public static void copyBack(double[][] firstResults, double[][] tempResults, int
nthCol)
    {
        int numRow = firstResults.length;
        for(int row = 0; row < numRow; row++)
        {firstResults[row][nthCol] = tempResults[row][0];} //copy the results back to
the first results
    }

    public static void reverse(double[][] firstResults)
    {
        int rows = firstResults.length;
        int cols = firstResults[0].length;
        for(int i = 0; i < rows; i++)
        {
                for(int j = 0; j < cols; j++)
                { firstResults[i][j] *= -1;}
        }
    }

    public static void initialize(parish[] parishes, ArrayList<ArrayList<String>>
rawlists, ArrayList<ArrayList<String>> cleanedLists) //rawlists needed to get
parish name
    {
        for(int i = 0; i < parishes.length; i++)
        {parishes[i] = new parish(cleanedLists.get(i), rawlists.get(i+2).get(0));}
    }

    public static double[][] getInitialMatrix(parish[] parishes)
    {
        int rows = parishes.length;
```

```
        int cols = parishes[0].getAttris().length;
        double[][] matrix = new double[rows][cols];
        for(int i = 0; i < rows; i++)
        {
                double[] temp = parishes[i].getAttris();
                for(int j = 0 ; j < cols; j++)
                { matrix[i][j] = temp[j];}
        }
        return matrix;
    }
    /*
    public static void firstRank(parish[] parishList, parish[] results)
    {
        double avgRank = 0.0;
        int last = parishList.length-1;

        int size = parishList[0].getAttris().length;
        for(int i = 0; i < size; i++)
        {
                for(int j = 0; j < last; j++)
                {
                        for(int k = j+1; k < parishList.length; k++)
                        {
                                if(parishList[j].getAttris()[i] > parishList[k].getAttris()[i])
                                {    parishList[j].successorUp();    }
                                else if(parishList[j].getAttris()[i] <
parishList[k].getAttris()[i])
                                {    parishList[k].successorUp();    }
                        }
                        avgRank = (parishList.length+1) -
(parishList[j].getNumSuccessor()+1);
                        results[j].setAttris(i, avgRank);
                }
                avgRank = (parishList.length+1) -
(parishList[last].getNumSuccessor()+1);
                results[last].setAttris(i, avgRank);
```

```
                resetSuccessors(parishList);
        }

    }
    */
    //public static void subRank(parish[] parishList, double[][] results, int start, int
end)
    /*
    public static void subRank(double[][] matrix, double[][] results, int start, int end)
    {
        double avgRank = 0.0;
        int size = matrix.length;
        int last = size-1;
        //for(int i = 0; i < 1; i++)
        for(int i=0; i<last; i++)
        {
                //for(int j = i+1; j < i+2; j++)
                for(int j=i+1; j<size; j++)
                {
                        if( !parishList[i].isEqualTo(parishList[j], start, end))
                        {
                                if(parishList[i].greaterThan(parishList[j], start, end))
                                { parishList[i].successorUp();}
                                else if(parishList[j].greaterThan(parishList[i], start, end))
                                { parishList[j].successorUp();}
                                else
                                {
                                        parishList[i].unComparatorUp();
                                        parishList[j].unComparatorUp();
                                }
                        }
                }
                //System.out.println(i+"'s
successor"+parishList[i].getNumSuccessor());
                //System.out.println(i+"'s
uncomparator"+parishList[i].getUnComparator());
```

```
                avgRank = (size+1) -
(parishList[i].getNumSuccessor()+1)*(size+1)/(size+1-parishList[i].getUnCompara
tor());
                results[i][0] = avgRank;
                //System.out.println("i is "+i+" and avgRank is "+avgRank);
        }
                avgRank = (size+1) -
(parishList[last].getNumSuccessor()+1)*(size+1)/(size+1-parishList[last].getUnCo
mparator());
                results[last][0] = avgRank;
                reset(parishList);
   }
   */
   public static void subRank(double[][] matrix, parish[] parishList, double[][]
results, int start, int end)
   {
        double avgRank = 0.0;
        int size = matrix.length;
        int last = size-1;
        //for(int i = 0; i < 1; i++)
        for(int i=0; i<last; i++)
        {
                //for(int j = i+1; j < i+2; j++)
                for(int j=i+1; j<size; j++)
                {
                        if( !isEqualTo(matrix[1], matrix[j], start, end))
                        {
                                if(greaterThan(matrix[i], matrix[j], start, end))
                                { parishList[i].successorUp();}
                                else if(greaterThan(matrix[j], matrix[i], start, end))
                                { parishList[j].successorUp();}
                                else
                                {
                                        parishList[i].unComparatorUp();
                                        parishList[j].unComparatorUp();
                                }
```

```
                        }
                }
                //System.out.println(i+"'s
successor"+parishList[i].getNumSuccessor());
                //System.out.println(i+"'s
uncomparator"+parishList[i].getUnComparator());
                avgRank = (size+1) -
(parishList[i].getNumSuccessor()+1)*(size+1)/(size+1-parishList[i].getUnCompara
tor());
                results[i][0] = avgRank;
                //System.out.println("i is "+i+" and avgRank is "+avgRank);
        }
                avgRank = (size+1) -
(parishList[last].getNumSuccessor()+1)*(size+1)/(size+1-parishList[last].getUnCo
mparator());
                results[last][0] = avgRank;
                reset(parishList);
   }


   public static void reset(parish[] plists)
   {
        for(int i = 0; i < plists.length; i++)
        {
                plists[i].resetSuccessor();
                plists[i].resetUnComparator();
        }
   }


   public static boolean isEqualTo(double[] list1, double[] list2, int start, int end)
   {
        boolean correct = true;
        for(int i=start; i<end; i++)
        {
                if(list1[i] != list2[i])
```

```
                { correct = false; break;}
        }

        return correct;
    }

    public static boolean greaterThan(double[] list1, double[] list2, int start, int end)
    {
        boolean correct = true;
        for(int i=start; i<end; i++)
        {
                if(list1[i] < list2[i])
                { correct = false; break;}
        }

        return correct;
    }


}



class parish
{
    private String name;
    private int numSuccessor;
    private int unComparator;
    double[] attributes = new double[42];
    double finalScore;
    int binNumber;
    public parish()
    {
        name = "";
        numSuccessor = 0;
        unComparator = 0;
```

```
        for(int i = 0; i < attributes.length; i++)
        { attributes[i] = 0.0;}
        finalScore=0.00; //added code
        binNumber = 0;
}

public parish(ArrayList<String> initials, String parishName)
{
        name = parishName;
        attributes = toDouble(initials);
        numSuccessor = 0;
        numSuccessor = 0; //added code
        finalScore = 0.00;
        binNumber = 0;
}

private double[] toDouble(ArrayList<String> list)
{
        double[] values = new double[42];
        for(int i = 0; i < list.size(); i++)
        {
                if(i==1 || i==2 || i==5 || i==17 || i== 30 || i==31)
                { values[i] = Double.parseDouble(list.get(i)) * -1;}
                else
                { values[i] = Double.parseDouble(list.get(i));}
        }

        return values;
}

public void successorUp()
{numSuccessor++;}

public void resetSuccessor()
{ numSuccessor = 0;}
```

```
public int getNumSuccessor()
{ return numSuccessor;}

public void unComparatorUp()
{ unComparator++;}

public void resetUnComparator()
{ unComparator = 0;}

public int getUnComparator()
{ return unComparator;}

public void showAttris()
{
	for(int i = 0; i < attributes.length; i++)
	{ System.out.print(attributes[i]+" ");}
	System.out.println("\n");
}

public double[] getAttris()
{ return attributes;}

public String getName()
{ return name;}

public void setAttris(int index, double value)
{ attributes[index] = value;}


public boolean greaterThan(parish p)
{
	boolean correct = true;
	double[] target = p.getAttris();

	for(int i=1; i<attributes.length; i++)
	{
```

```
            if(attributes[i] < target[i])
            { correct = false; break;}
        }

        return correct;
    }

    public boolean greaterThan(parish p, int start, int end)
    {
        boolean correct = true;
        double[] target = p.getAttris();

        for(int i=start; i<end; i++)
        {
            if(attributes[i] < target[i])
            { correct = false; break;}
        }

        return correct;
    }


    public void negate()
    {
        for(int i=0; i<attributes.length; i++)
        { attributes[i] *= -1;}
    }

    public void setScore(double value)
    { finalScore = value;}

    public double getScore()
    { return finalScore;}

    public void setBin()
    {
```

```
        if(finalScore >=1.0 && finalScore < 10.0)
        { binNumber = 1;}
        else if( finalScore >=10.0 && finalScore < 20.0)
        { binNumber = 2;}
        else if( finalScore >=20.0 && finalScore < 30.0)
        { binNumber = 3;}
        else if( finalScore >=30.0 && finalScore < 40.0)
        { binNumber = 4;}
        else if( finalScore >=40.0 && finalScore < 50.0)
        { binNumber = 5;}
        else if( finalScore >=50.0 && finalScore < 60.0)
        { binNumber = 6;}
        else
        { binNumber = 7;}
    }

    public int getBin()
    { return binNumber;}
}
```

For Method (2),

HazardRanking.java:

```
package hazardranking;

import java.io.FileReader;
import java.io.IOException;
import java.util.Collections;
import java.util.Scanner;
import java.util.Vector;

public class HazardRanking {

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

```
Vector<Vector<Double>> data = new Vector<Vector<Double>>();
Vector<Parish> parish = new Vector<Parish>();
Vector<Double> weight = new Vector<Double>();
String fileName;

try {
   // Read and store the data into a data vector
   fileName = "Data.txt";
   FileReader fileReader = new FileReader(fileName);
   Scanner scanner = new Scanner(fileReader);

   for (int i = 0; i < 64; i++) {
      Vector<Double> line = new Vector<Double>();
      for (int j = 0; j < 42; j++) {
         double tempData = scanner.nextDouble();
         line.add(tempData);
      }
      data.add(line);
      scanner.nextLine();
   }

   // Read and store the names of parishes into another vector
   fileName = "Parish Name.txt";
   fileReader = new FileReader(fileName);
   scanner = new Scanner(fileReader);

   for (int i = 0; i < 64; i++) {
      Parish p = new Parish(scanner.nextLine());
      parish.add(p);
   }

   normalize(data);

   // Assign weight of 1.0 to all attributes
   // Except attributes at place 1, 2, 5, 17, 30 and 31 with respect to
   // the original data that were marked red
```

```
        for (int i = 0; i < 42; i++) {
            if (i == 1 || i == 2 || i == 5 || i == 17 || i == 30 || i == 31) {
                weight.add(-1.0);
            } else {
                weight.add(1.0);
            }
        }

        // Compute sum of weighted attributes for each parish
        for (int i = 0; i < data.size(); i++) {
            for (int j = 0; j < 42; j++) {
                parish.elementAt(i).sum += weight.elementAt(j) *
data.elementAt(i).elementAt(j);
            }
        }

        // Pair-wise ranking, increase score of a parish if it has greater
        // sum of weights than any other
        for (int i = 0; i < data.size(); i++) {
            for (int j = 0; j < data.size(); j++) {
                if (i == j) {
                    continue;
                } else if (parish.elementAt(i).sum > parish.elementAt(j).sum) {
                    parish.elementAt(i).score++;
                }
            }
        }

        sort(parish);

        // Output the relative score of each parish to other parishes
        // 0 being the best and 63 being the worst
        System.out.println("Sorting result: ");
        for (int i = 0; i < parish.size(); i++) {
            System.out.print(parish.elementAt(i).name + " with the score of: ");
            System.out.println(parish.elementAt(i).score);
```

```
}

System.out.println("\nUsing equiv-width binning: ");

// Partition the data into 7 bins, each with width of 2.
// Parishes with similar vulnerabilities will fall into same bin
for (int i = 0; i < parish.size(); i++) {

   // Because we know that no data is less than 2.0, we skip that bin
   if(parish.elementAt(i).sum < 4.0) {
      parish.elementAt(i).rank = 1;
   }
   else if(parish.elementAt(i).sum < 6.0) {
      parish.elementAt(i).rank = 2;
   }
   else if(parish.elementAt(i).sum < 8.0) {
      parish.elementAt(i).rank = 3;
   }
   else if(parish.elementAt(i).sum < 10.0) {
      parish.elementAt(i).rank = 4;
   }
   else if(parish.elementAt(i).sum < 12.0) {
      parish.elementAt(i).rank = 5;
   }
   else if(parish.elementAt(i).sum < 14.0) {
      parish.elementAt(i).rank = 6;
   }
   else if(parish.elementAt(i).sum < 16.0) {
      parish.elementAt(i).rank = 7;
   }
}

// Display result
for (int i = 0; i < parish.size(); i++) {
   System.out.print(parish.elementAt(i).name + " with the rank of: ");
   System.out.println(parish.elementAt(i).rank);
```

```
        }

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

}

// Normalize the data vector
public static void normalize(Vector<Vector<Double>> data) {

    Vector<Double> min = new Vector<Double>();
    Vector<Double> max = new Vector<Double>();

    // First assign min and max to the first entry in the vector
    for (int i = 0; i < data.elementAt(0).size(); i++) {
        min.add(data.elementAt(0).elementAt(i));
        max.add(data.elementAt(0).elementAt(i));
    }

    // Determine the min and max value for each attribute
    for (int i = 1; i < data.size(); i++) {
        Vector<Double> temp = data.elementAt(i);
        for (int j = 0; j < min.size(); j++) {
            if (temp.elementAt(j) < min.elementAt(j)) {
                min.setElementAt(temp.elementAt(j), j);
                continue;
            }
            if (temp.elementAt(j) > max.elementAt(j)) {
                max.setElementAt(temp.elementAt(j), j);
                continue;
            }
        }
    }

    // Normalize the vector with (value - min)/(max - min)
```

```
        for (int i = 0; i < data.size(); i++) {
            Vector<Double> temp = data.elementAt(i);
            for (int j = 0; j < min.size(); j++) {
                double normalized = (temp.elementAt(j) - min.elementAt(j)) /
(max.elementAt(j) - min.elementAt(j));
                temp.setElementAt(normalized, j);
            }
        }
    }

    // Sorts parish vector using insertion sort method
    public static void sort(Vector<Parish> parish) {

        Parish temp;

        for (int i = 1; i < parish.size(); i++) {
            int j = i;
            while (j > 0 && parish.elementAt(j - 1).score > parish.elementAt(j).score) {
                temp = parish.elementAt(j - 1);
                parish.setElementAt(parish.elementAt(j), j - 1);
                parish.setElementAt(temp, j);
                j--;
            }
        }
    }
}
```

Parish.java:

```
package hazardranking;

public class Parish {
    int score;
    double sum;
    String name;
    int rank;
```

```
    public Parish(String name) {
        score = 0;
        sum = 0.0;
        this.name = name;
        rank = 0;
    }
}
```

Actual Results

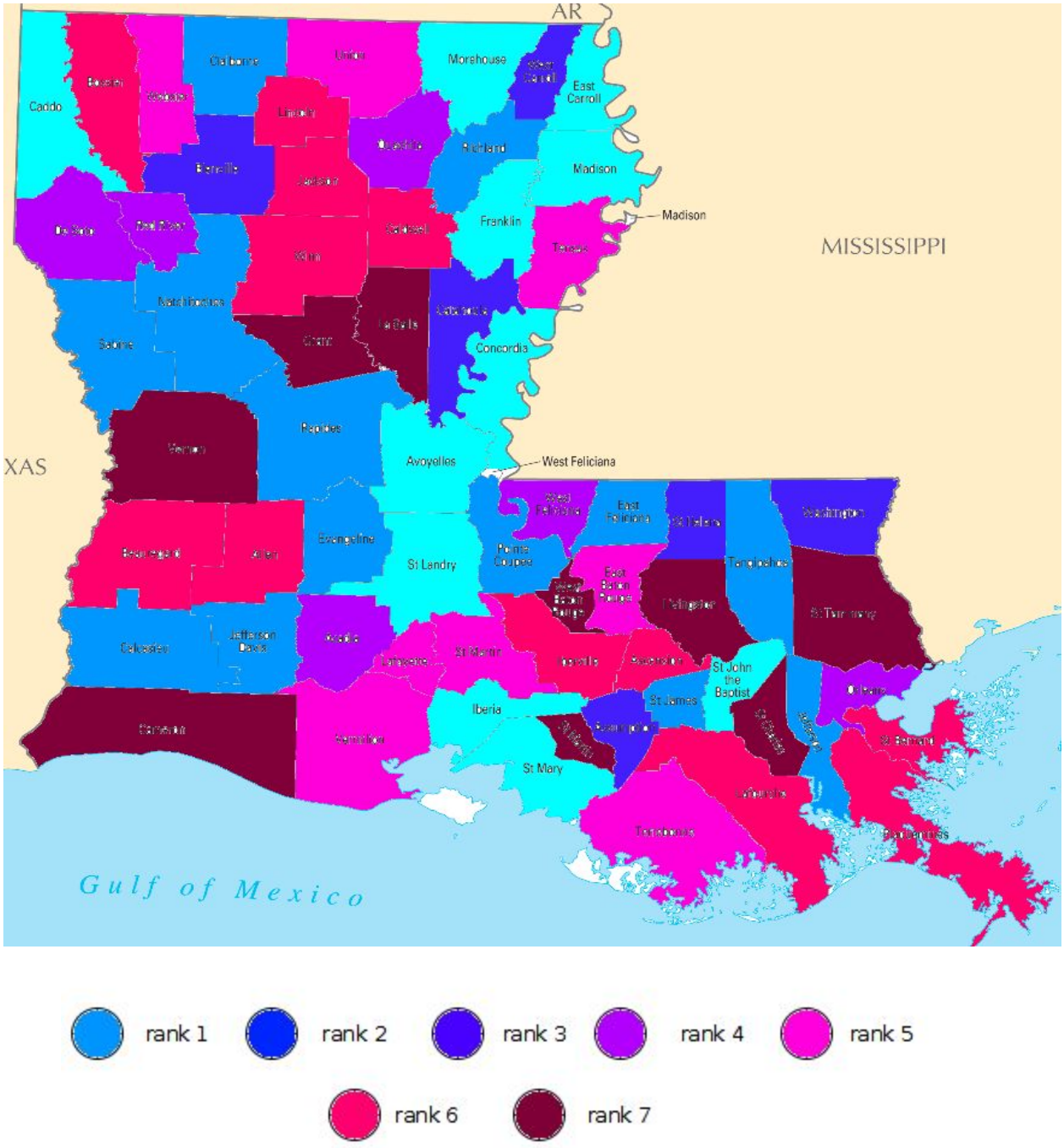
From Method (1),

Acadia Parish	33.0	bin number: 4
Allen Parish	54.0	bin number: 6
Ascension Parish	58.0	bin number: 6
Assumption Parish	26.0	bin number: 3
Avoyelles Parish	5.0	bin number: 1
Beauregard Parish	55.0	bin number: 6
Bienville Parish	28.0	bin number: 3
Bossier Parish	58.0	bin number: 6
Caddo Parish	5.0	bin number: 1
Calcasieu Parish	13.0	bin number: 2
Caldwell Parish	54.0	bin number: 6
Cameron Parish	64.0	bin number: 7
Catahoula Parish	26.0	bin number: 3
Claiborne Parish	11.0	bin number: 2
Concordia Parish	7.0	bin number: 1
De Soto Parish	33.0	bin number: 4
East Baton Rouge Parish	47.0	bin number: 5
East Carroll Parish	6.0	bin number: 1
East Feliciana Parish	15.0	bin number: 2
Evangeline Parish	13.0	bin number: 2
Franklin Parish	6.0	bin number: 1
Grant Parish	61.0	bin number: 7
Iberia Parish	4.0	bin number: 1
Iberville Parish	58.0	bin number: 6
Jackson Parish	51.0	bin number: 6
Jefferson Parish	14.0	bin number: 2

Jefferson Davis Parish	19.0	bin number: 2
Lafayette Parish	41.0	bin number: 5
Lafourche Parish	50.0	bin number: 6
La Salle Parish	63.0	bin number: 7
Lincoln Parish	51.0	bin number: 6
Livingston Parish	62.0	bin number: 7
Madison Parish	9.0	bin number: 1
Morehouse Parish	6.0	bin number: 1
Natchitoches Parish	17.0	bin number: 2
Orleans Parish	37.0	bin number: 4
Ouachita Parish	31.0	bin number: 4
Plaquemines Parish	52.0	bin number: 6
Pointe Coupee Parish	11.0	bin number: 2
Rapides Parish	11.0	bin number: 2
Red River Parish	38.0	bin number: 4
Richland Parish	15.0	bin number: 2
Sabine Parish	19.0	bin number: 2
St. Bernard Parish	55.0	bin number: 6
St. Charles Parish	63.0	bin number: 7
St. Helena Parish	26.0	bin number: 3
St. James Parish	19.0	bin number: 2
St. John the Baptist Parish	4.0	bin number: 1
St. Landry Parish	1.0	bin number: 1
St. Martin Parish	43.0	bin number: 5
St. Mary Parish	3.0	bin number: 1
St. Tammany Parish	62.0	bin number: 7
Tangipahoa Parish	16.0	bin number: 2
Tensas Parish	48.0	bin number: 5
Terrebonne Parish	42.0	bin number: 5
Union Parish	49.0	bin number: 5
Vermilion Parish	46.0	bin number: 5
Vernon Parish	61.0	bin number: 7
Washington Parish	21.0	bin number: 3
Webster Parish	42.0	bin number: 5
West Baton Rouge Parish	51.0	bin number: 6
West Carroll Parish	23.0	bin number: 3

West Feliciana Parish	34.0	bin number: 4
Winn Parish	56.0	bin number: 6

After this ranking is obtained, the final output is made by coloring Louisiana's map of all parishes according to their ranks, with light blue = 1 (least vulnerable), all the way to purple = 7 (most vulnerable), as shown:



From Method (2),

Sorting result:
Livingston Parish, Louisiana with the score of: 0
Cameron Parish, Louisiana with the score of: 1
St. Charles Parish, Louisiana with the score of: 2
St. Tammany Parish, Louisiana with the score of: 3
Ascension Parish, Louisiana with the score of: 4
West Feliciana Parish, Louisiana with the score of: 5
Grant Parish, Louisiana with the score of: 6
Bossier Parish, Louisiana with the score of: 7
West Baton Rouge Parish, Louisiana with the score of: 8
Vernon Parish, Louisiana with the score of: 9
Beauregard Parish, Louisiana with the score of: 10
La Salle Parish, Louisiana with the score of: 11
Lafourche Parish, Louisiana with the score of: 12
Plaquemines Parish, Louisiana with the score of: 13
St. Martin Parish, Louisiana with the score of: 14
St. James Parish, Louisiana with the score of: 15
St. John the Baptist Parish, Louisiana with the score of: 16
Iberville Parish, Louisiana with the score of: 17
St. Bernard Parish, Louisiana with the score of: 18
Tangipahoa Parish, Louisiana with the score of: 19
Calcasieu Parish, Louisiana with the score of: 20
Caldwell Parish, Louisiana with the score of: 21
Allen Parish, Louisiana with the score of: 22
De Soto Parish, Louisiana with the score of: 23
Vermilion Parish, Louisiana with the score of: 24
Ouachita Parish, Louisiana with the score of: 25

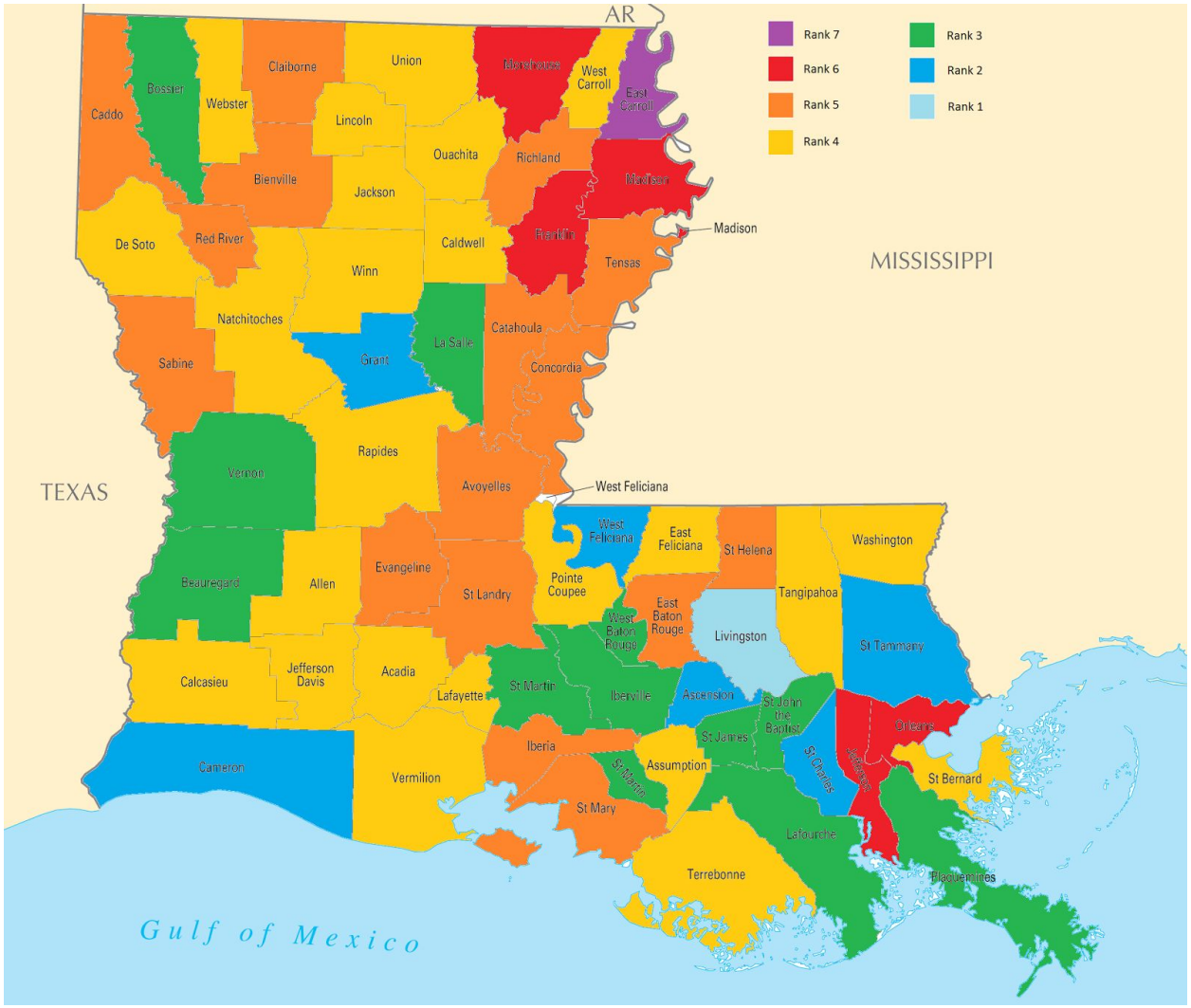
Lincoln Parish, Louisiana with the score of: 26
Terrebonne Parish, Louisiana with the score of: 27
Union Parish, Louisiana with the score of: 28
Jackson Parish, Louisiana with the score of: 29
Assumption Parish, Louisiana with the score of: 30
Rapides Parish, Louisiana with the score of: 31
East Feliciana Parish, Louisiana with the score of: 32
Lafayette Parish, Louisiana with the score of: 33
Webster Parish, Louisiana with the score of: 34
Winn Parish, Louisiana with the score of: 35
Acadia Parish, Louisiana with the score of: 36
Jefferson Davis Parish, Louisiana with the score of: 37
Pointe Coupee Parish, Louisiana with the score of: 38
Washington Parish, Louisiana with the score of: 39
Natchitoches Parish, Louisiana with the score of: 40
West Carroll Parish, Louisiana with the score of: 41
Red River Parish, Louisiana with the score of: 42
East Baton Rouge Parish, Louisiana with the score of: 43
Evangeline Parish, Louisiana with the score of: 44
Claiborne Parish, Louisiana with the score of: 45
St. Mary Parish, Louisiana with the score of: 46
Iberia Parish, Louisiana with the score of: 47
Sabine Parish, Louisiana with the score of: 48
Bienville Parish, Louisiana with the score of: 49
Caddo Parish, Louisiana with the score of: 50
Avoyelles Parish, Louisiana with the score of: 51
Catahoula Parish, Louisiana with the score of: 52
St. Helena Parish, Louisiana with the score of: 53
Richland Parish, Louisiana with the score of: 54
Tensas Parish, Louisiana with the score of: 55
St. Landry Parish, Louisiana with the score of: 56
Concordia Parish, Louisiana with the score of: 57
Franklin Parish, Louisiana with the score of: 58
Morehouse Parish, Louisiana with the score of: 59
Jefferson Parish, Louisiana with the score of: 60
Madison Parish, Louisiana with the score of: 61

Orleans Parish, Louisiana with the score of: 62
East Carroll Parish, Louisiana with the score of: 63

Using equiv-width binning:
Livingston Parish, Louisiana with the rank of: 1
Cameron Parish, Louisiana with the rank of: 2
St. Charles Parish, Louisiana with the rank of: 2
St. Tammany Parish, Louisiana with the rank of: 2
Ascension Parish, Louisiana with the rank of: 2
West Feliciana Parish, Louisiana with the rank of: 2
Grant Parish, Louisiana with the rank of: 2
Bossier Parish, Louisiana with the rank of: 3
West Baton Rouge Parish, Louisiana with the rank of: 3
Vernon Parish, Louisiana with the rank of: 3
Beauregard Parish, Louisiana with the rank of: 3
La Salle Parish, Louisiana with the rank of: 3
Lafourche Parish, Louisiana with the rank of: 3
Plaquemines Parish, Louisiana with the rank of: 3
St. Martin Parish, Louisiana with the rank of: 3
St. James Parish, Louisiana with the rank of: 3
St. John the Baptist Parish, Louisiana with the rank of: 3
Iberville Parish, Louisiana with the rank of: 3
St. Bernard Parish, Louisiana with the rank of: 4
Tangipahoa Parish, Louisiana with the rank of: 4
Calcasieu Parish, Louisiana with the rank of: 4
Caldwell Parish, Louisiana with the rank of: 4
Allen Parish, Louisiana with the rank of: 4
De Soto Parish, Louisiana with the rank of: 4
Vermilion Parish, Louisiana with the rank of: 4
Ouachita Parish, Louisiana with the rank of: 4
Lincoln Parish, Louisiana with the rank of: 4
Terrebonne Parish, Louisiana with the rank of: 4
Union Parish, Louisiana with the rank of: 4
Jackson Parish, Louisiana with the rank of: 4
Assumption Parish, Louisiana with the rank of: 4
Rapides Parish, Louisiana with the rank of: 4

East Feliciana Parish, Louisiana with the rank of: 4
Lafayette Parish, Louisiana with the rank of: 4
Webster Parish, Louisiana with the rank of: 4
Winn Parish, Louisiana with the rank of: 4
Acadia Parish, Louisiana with the rank of: 4
Jefferson Davis Parish, Louisiana with the rank of: 4
Pointe Coupee Parish, Louisiana with the rank of: 4
Washington Parish, Louisiana with the rank of: 4
Natchitoches Parish, Louisiana with the rank of: 4
West Carroll Parish, Louisiana with the rank of: 4
Red River Parish, Louisiana with the rank of: 5
East Baton Rouge Parish, Louisiana with the rank of: 5
Evangeline Parish, Louisiana with the rank of: 5
Claiborne Parish, Louisiana with the rank of: 5
St. Mary Parish, Louisiana with the rank of: 5
Iberia Parish, Louisiana with the rank of: 5
Sabine Parish, Louisiana with the rank of: 5
Bienville Parish, Louisiana with the rank of: 5
Caddo Parish, Louisiana with the rank of: 5
Avoyelles Parish, Louisiana with the rank of: 5
Catahoula Parish, Louisiana with the rank of: 5
St. Helena Parish, Louisiana with the rank of: 5
Richland Parish, Louisiana with the rank of: 5
Tensas Parish, Louisiana with the rank of: 5
St. Landry Parish, Louisiana with the rank of: 5
Concordia Parish, Louisiana with the rank of: 5
Franklin Parish, Louisiana with the rank of: 6
Morehouse Parish, Louisiana with the rank of: 6
Jefferson Parish, Louisiana with the rank of: 6
Madison Parish, Louisiana with the rank of: 6
Orleans Parish, Louisiana with the rank of: 6
East Carroll Parish, Louisiana with the rank of: 7

After this ranking is obtained, the final output is made by coloring Louisiana's map of all parishes according to their ranks, with light blue = 1 (least vulnerable), all the way to purple = 7 (most vulnerable), as shown:


AR
Rank 7
Rank 6
Rank 5
Rank 4
Rank 3
Rank 2
Rank 1
Bossier
Caddo
Webster
Claiborne
Union
Morehouse
West Carroll
East Carroll
Lincoln
Ouachita
Richland
Bienville
Jackson
Madison
Franklin
Tensas
De Soto
Red River
Caldwell
Winn
Natchitoches
Catahoula
La Salle
Sabine
Grant
Concordia
MISSISSIPPI
Rapides
Vernon
Avoyelles
TEXAS
West Feliciana
East Feliciana
St Helena
Washington
Beauregard
Allen
Evangeline
St Landry
Pointe Coupee
Tangipahoa
East Baton Rouge
West Baton Rouge
Livingston
St Tammany
Calcasieu
Jefferson Davis
Acadia
Lafayette
St Martin
Iberville
Ascension
St John the Baptist
Orleans
St James
Cameron
Vermilion
Iberia
St Mary
Assumption
St Charles
Jefferson
St Bernard
Lafourche
Terrebonne
Plaquemines
Gulf of Mexico