

# NATIONAL BRIDGE INVENTORY ANALYSIS

**Big Data: 44517-04**

**Aparna Dondapati, Vedha Sri Gaddam, Venkata Ramana Peddi, Jaya Chandra Eladandi, Anjali Bhogireddy, Meghanaa Malleboina, Naga Mounika Devi Ghanta**

## **Project Idea:**

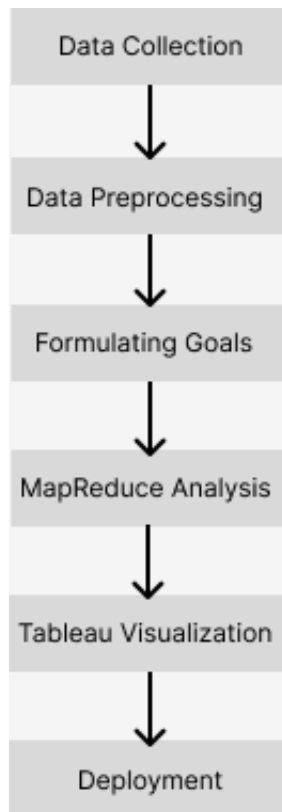
Bridge Inventory data analysis entails the comprehensive assessment of historical bridge data to uncover patterns, forecast deck conditions, and facilitate well-informed decision-making across diverse sectors. This encompasses the identification of regional trends in bridge performance, periodic evaluations of deck conditions, and the analysis of inspection frequency records. In addition to ensuring the longevity and safety of bridges, this data-driven strategy helps communities thrive sustainably and efficiently manage transportation networks. This thorough method of analyzing bridge data adds to the general sustainability and resilience of transportation networks.

## **Technology Summary:**

In this project, we will employ the following tools:

- **Hadoop:** It's role in this project is to provide a robust and scalable infrastructure for storing, processing, and analyzing historical bridge data. It enables efficient data handling, which is crucial for uncovering patterns, forecasting deck conditions, and making well-informed decisions in diverse sectors related to bridge inventory data analysis.
- **MapReduce:** It is to extract valuable insights from the data, using its distributed processing capabilities to handle large datasets efficiently.
- **MS Excel:** It is used for data preprocessing so that the data does not contain any duplicates or missing information.
- **VS Code:** It is used to write the required code for the mapper and reducer phase for the MapReduce which helps in extracting the desired outputs.
- **Tableau:** To present these insights in a clear and visually compelling manner, we will utilize Tableau, a data visualization tool, enabling us to create meaningful and impactful data visualizations that aid in understanding and decision-making.

## **Architecture:**



#### Architecture Summary:

- **Data Collection:** This phase involved collecting the National Bridge Inventory (NBI) dataset from the Federal Highway Administration (FHWA) website. The dataset comprises information related to bridges, encompassing location, maintenance, construction, and condition details.
- **Data Preprocessing:** In this phase, repeated data is deleted to avoid counting the same information multiple times, missing information is filled in to ensure the dataset is complete and unnecessary data is eliminated to make the dataset simpler and more manageable.
- **Goals Formulation:** The required goals are formulated to better understand bridges and make them safer. These goals help us see which bridges need repairs, how traffic affects them, and who is responsible for maintenance and guiding infrastructure decisions.
- **MapReduce Analysis:** It efficiently processes and analyzes large volumes of historical bridge data, extract meaningful insights, identify trends, and make data-driven decisions to optimize bridge maintenance and performance across various sectors.
- **Tableau Visualization:** In this phase, the extracted meaningful insights are presented visually which are used for making the required decisions.
- **Deployment:** This deployment phase involves the implementation and execution of strategies, solutions, or changes based on the analysis and planning stages, with the goal of achieving project objectives.

## **Project Goals:**

### **Goal 1: Minimum Bridge Condition by County**

Calculate the average condition rating for bridges in each county.

### **Goal 2: Traffic Impact Analysis**

Calculate the impact of traffic on bridge condition by grouping bridges based on traffic load.

### **Goal 3: Bridge Age by Owner**

Determine the average age of bridges based on their maintenance ownership (e.g., state-owned, county-owned).

### **Goal 4: Traffic Load Comparison**

Group bridges by the number of traffic lanes and compare their conditions and ages.

## **Project Description:**

### **Project Description:**

#### **1)Project Setup:**

##### **Setting up the Project Environment:**

##### **1. Installing VS Code and Java Extensions:**

Start by installing VS Code along with essential Java modules. VS Code will serve as your primary Integrated Development Environment (IDE) for coding, debugging, and project management. Ensure that Java-related functionalities are integrated smoothly within VS Code to enable seamless Java development.

##### **2. Verifying the Java Environment:**

Verify the correctness of your Java setup on your system. Confirm the presence of Java and its proper configuration. Ensure that system environment variables are accurately configured and that the Java Development Kit (JDK) is correctly added and accessible.

## **Accessing the Bridge Inventory Dataset:**

**Verify Dataset Availability:** Ensure access to the Bridge Inventory dataset, confirming its comprehensive coverage of all bridge records, including essential metrics such as bridge specifications, structural integrity indicators, maintenance history, and relevant performance measures.

**Evaluate Dataset Structure:** Assess the dataset's organization, ensuring it adheres to a standardized structure with accurately labeled columns and consistent formatting. If necessary, perform data cleaning and organization to streamline subsequent analysis procedures.

### **Bridge Inventory Data Analysis:**

**Confirm Dataset Availability:** Ensure the availability of the Bridge Inventory dataset encompassing detailed information on various bridges, including structural attributes, maintenance logs, historical performance metrics, and safety evaluations.

**Validate Dataset Structure:** Verify that the dataset is appropriately structured, presenting coherent columns with clear labels and a consistent format. If required, undertake data cleansing and organization to facilitate a smooth analytical process.

## **2. Data Retrieval:**

**Utilizing VS Code for Java Programming:**

Initiate a new Java file within Visual Studio Code (VS Code) for programming purposes.

For retrieving data from Excel files, harness the capabilities of Java, employing tools like Apache POI for Excel-related tasks.

**Adapt Data for Hadoop MapReduce Compatibility:**

Once data is extracted using Java, transform it into a format compatible with Hadoop MapReduce processing. This typically involves structuring the data into an organized file or key-value pairs.

Ensure that the processed files are saved in a standard format readable by MapReduce tools, such as CSV (Comma-Separated Values).

## **3. Data Preprocessing:**

Management of Missing Data:

Inspect the dataset for missing or null values. If identified, strategize on how to handle them, whether through imputation or elimination.

Elimination of Duplicate Entries:

Detect and eliminate any duplicate records within the dataset to maintain data consistency.

Application of Data Filters:

Apply specific filters to extract pertinent data subsets aligned with project objectives.

#### **4. Utilizing Hadoop MapReduce:**

Creation of Java-based MapReduce Programs:

Craft Java-based MapReduce programs to handle the preprocessed data effectively.

Formulation of Mapper Function:

Develop a Mapper function to extract and emit suitable key-value pairs based on the analysis criteria.

Execution of Reducer Function:

Implement a Reducer function to process and aggregate the extracted data effectively.

Optimal Output Formatting:

Ensure that the output is structured in a format conducive to further analysis and visualization.

#### **5. Post-processing and Analysis:**

In-depth Assessment of MapReduce Outputs:

Conduct a thorough analysis of the outcomes derived from the MapReduce executions, unveiling intricate patterns, trends, and actionable insights within the refined data.

Identification of Data Patterns and Trends:

Discern and articulate significant patterns, trends, and insights residing in the processed data.

Execution of Supplementary Calculations and Filtering:

Execute additional calculations or filtering aligned with project-specific requirements.

## 6. Reporting and Visual Representation:

Harnessing Tableau:

Save the processed data in a format compatible with Tableau, such as CSV, for subsequent visualization.

Integration with Tableau:

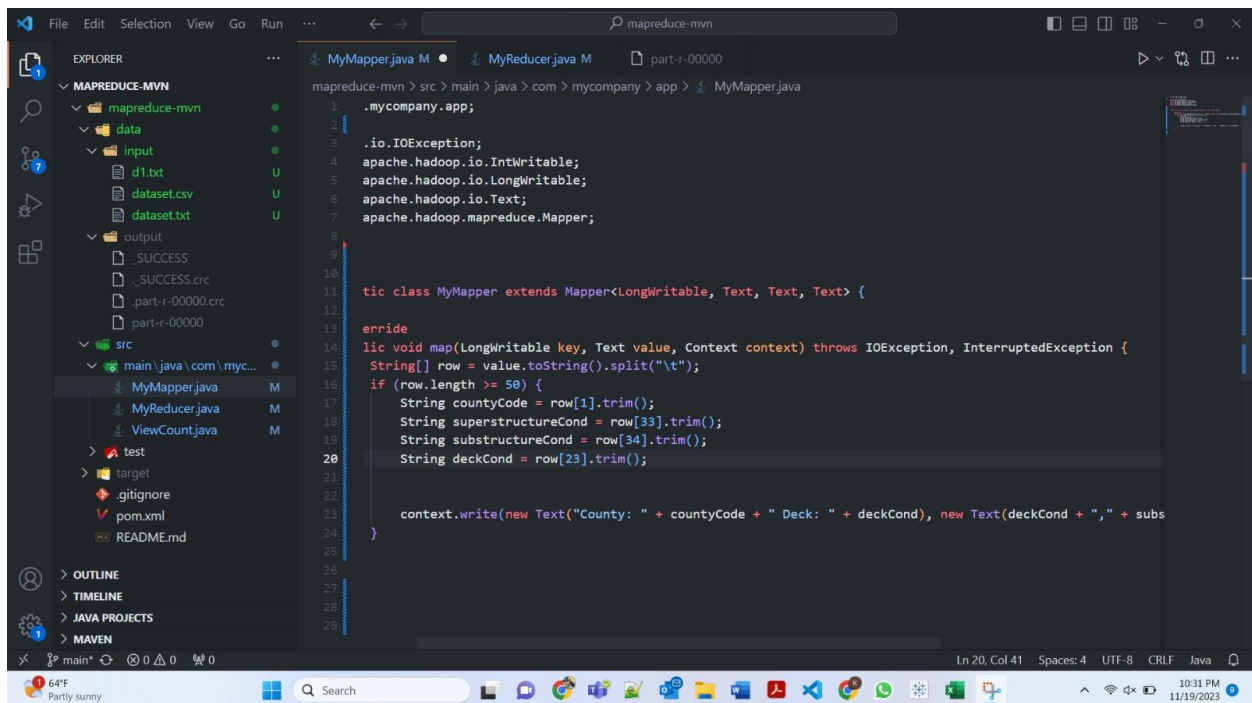
Import the refined data into Tableau to facilitate dynamic visual representation.

Creation of Visual Elements:

Develop customized visualizations tailored to project objectives, crafting charts for average bridge condition rating, average age of bridges, traffic impact analysis, bridge type classification and traffic load comparison.

**Result Summary:**

### Goal 1: Minimum Bridge Condition by County



```
1  .mycompany.app;
2
3  .io.IOException;
4  apache.hadoop.io.IntWritable;
5  apache.hadoop.io.LongWritable;
6  apache.hadoop.io.Text;
7  apache.hadoop.mapreduce.Mapper;
8
9
10
11
12
13
14  tic class MyMapper extends Mapper<LongWritable, Text, Text, Text> {
15
16  erride
17  lic void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
18      String[] row = value.toString().split("\t");
19      if (row.length >= 50) {
20          String countryCode = row[1].trim();
21          String superstructureCond = row[33].trim();
22          String substructureCond = row[34].trim();
23          String deckCond = row[23].trim();
24
25          context.write(new Text("County: " + countryCode + " Deck: " + deckCond), new Text(deckCond + "," + subs
26      }
27
28
29
```

mapreduce-mvn

MyMapper.java M • MyReducer.java M • part-r-00000

mapreduce-mvn > src > main > java > com > mycompany > app > MyReducer.java

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

public static class MyReducer extends Reducer<Text, Text, Text, DoubleWritable> {

    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, Interru
        double minDeck = Double.MAX_VALUE;
        double minSubstructure = Double.MAX_VALUE;
        double minSuperstructure = Double.MAX_VALUE;

        for (Text value : values) {
            String[] conditions = value.toString().split(",");
            if (conditions.length >= 3) {
                double deckCond = Double.parseDouble(conditions[0]);
                double substructureCond = Double.parseDouble(conditions[1]);
                double superstructureCond = Double.parseDouble(conditions[2]);

                minDeck = Math.min(minDeck, deckCond);
                minSubstructure = Math.min(minSubstructure, substructureCond);
                minSuperstructure = Math.min(minSuperstructure, superstructureCond);
            }

            double minOverall = Math.min(minDeck, Math.min(minSubstructure, minSuperstructure));

            context.write(key, new DoubleWritable(minOverall));
        }
    }
```

Ln 33, Col 13 Spaces: 4 UTF-8 CRLF Java

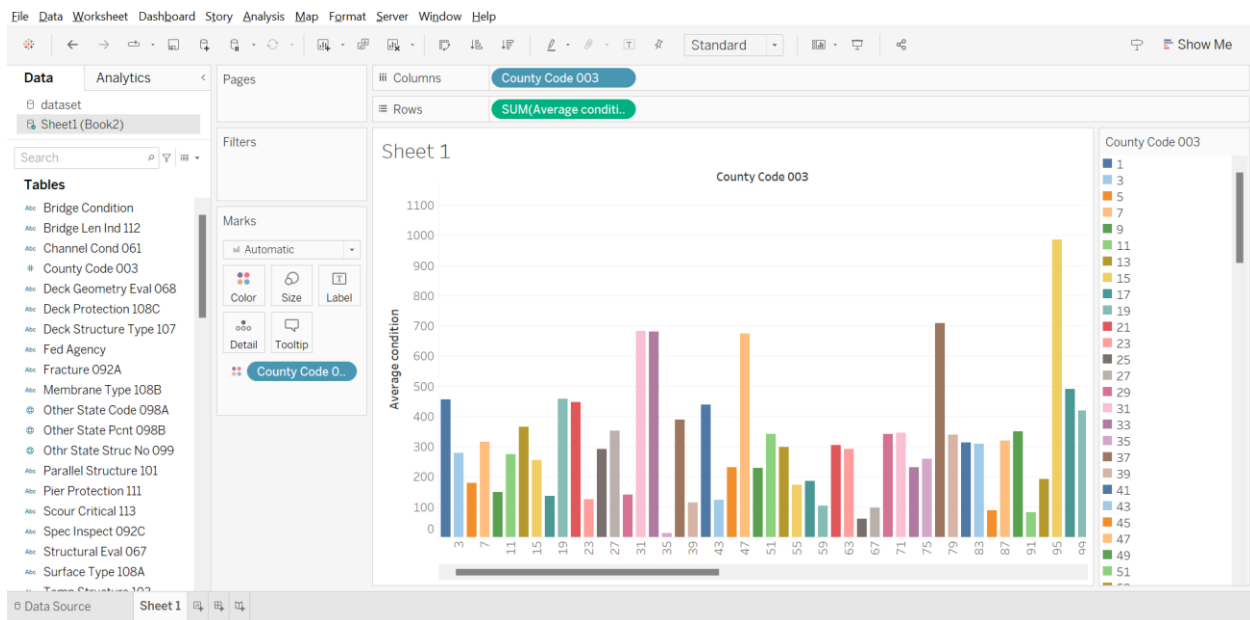
mapreduce-mvn

MyMapper.java M • MyReducer.java M • part-r-00000 x

mapreduce-mvn > data > output > part-r-00000

4	5	5
5	7	4
6	9	4
7	11	4
8	13	4
9	15	4
10	17	3
11	19	4
12	21	3
13	23	4
14	25	3
15	27	5
16	29	5
17	31	2
18	33	3
19	35	3
20	37	4
21	39	4
22	41	3
23	43	4
24	45	5
25	47	3
26	49	4
27	51	4
28	53	5
29	55	3
30	57	2
31	59	4
32	61	4
--	--	--

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Plain Text



**Story:** This chart represents the minimum bridge condition for each county differentiating each county code with a unique color. According to this chart, it can be clearly stated that the average bridge condition is highest for the county with code 95 and the bridge condition is very poor for the county with code 35. This helps us in analyzing the bridge condition to make any repairs required.

## Goal 2: Traffic Impact Analysis

```

1 package com.mycompany.app;
2
3 import java.io.IOException;
4 import org.apache.hadoop.io.IntWritable;
5 import org.apache.hadoop.io.LongWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Mapper;
8
9
10
11 public class BridgeConditionADT implements Tool {
12
13     public static class MyMapper extends Mapper<LongWritable, Text, Text, Text> {
14
15         @Override
16         public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
17             String[] row = value.toString().split(",");
18             if (row.length >= 50) {
19                 String bridgeCondition = row[50].trim();
20                 String adt029 = row[28].trim();
21                 context.write(new Text(bridgeCondition), new Text(adt029));
22             }
23         }
24     }
25 }
26

```

The screenshot shows an IDE with a Java code file named 'MyMapper.java'. The code is a Hadoop MapReduce mapper that processes a CSV file 'dataset.csv'. It defines a 'BridgeConditionADT' class and a 'MyMapper' class that implements the 'Mapper' interface. The 'MyMapper' class has a 'map' method that reads the CSV file, splits the data into rows, and writes the bridge condition and a derived value to the output. The IDE interface includes an 'EXPLORER' pane on the left showing the project structure, a 'Run' button, and a status bar at the bottom.



mapreduce-mvn

EXPLORER

- MAPREDUCE-MVN
  - mapreduce-mvn
    - data
      - d1.txt
      - dataset.csv
      - dataset.txt
    - output
      - .\_SUCCESS
      - .\_SUCCESS.crc
      - part-r-00000.crc
      - part-r-00000
    - src
      - main\java\com\myc...
        - MyMapper.java
        - MyReducer.java
        - ViewCount.java
      - test
      - target
      - .gitignore
      - pom.xml
      - README.md

MyMapper.java M

MyReducer.java M

part-r-00000

dataset.csv U

```
mapreduce-mvn > src > main > java > com > mycompany > app > MyReducer.java
5 import org.apache.hadoop.io.IntWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Reducer;
8
9
10 public static class MyReducer extends Reducer<Text, Text, Text, DoubleWritable> {
11
12     @Override
13     protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
14         double sumADT = 0.0;
15         int count = 0;
16
17         for (Text value : values) {
18             double adt029 = Double.parseDouble(value.toString());
19             sumADT += adt029;
20             count++;
21         }
22
23         if (count > 0) {
24             double avgADT = sumADT / count;
25
26             context.write(key, new DoubleWritable(avgADT));
27         }
28     }
29 }
30
31 }
```

Ln 26, Col 12 Spaces: 4 UTF-8 CRLF Java

64°F Partly sunny 10:46 PM 11/19/2023

mapreduce-mvn

EXPLORER

- MAPREDUCE-MVN
  - mapreduce-mvn
    - data
      - d1.txt
      - dataset.csv
      - dataset.txt
    - output
      - .\_SUCCESS
      - .\_SUCCESS.crc
      - part-r-00000.crc
      - part-r-00000
    - src
      - main\java\com\myc...
        - MyMapper.java
        - MyReducer.java
        - ViewCount.java
      - test
      - target
      - .gitignore
      - pom.xml
      - README.md

MyMapper.java M

MyReducer.java M

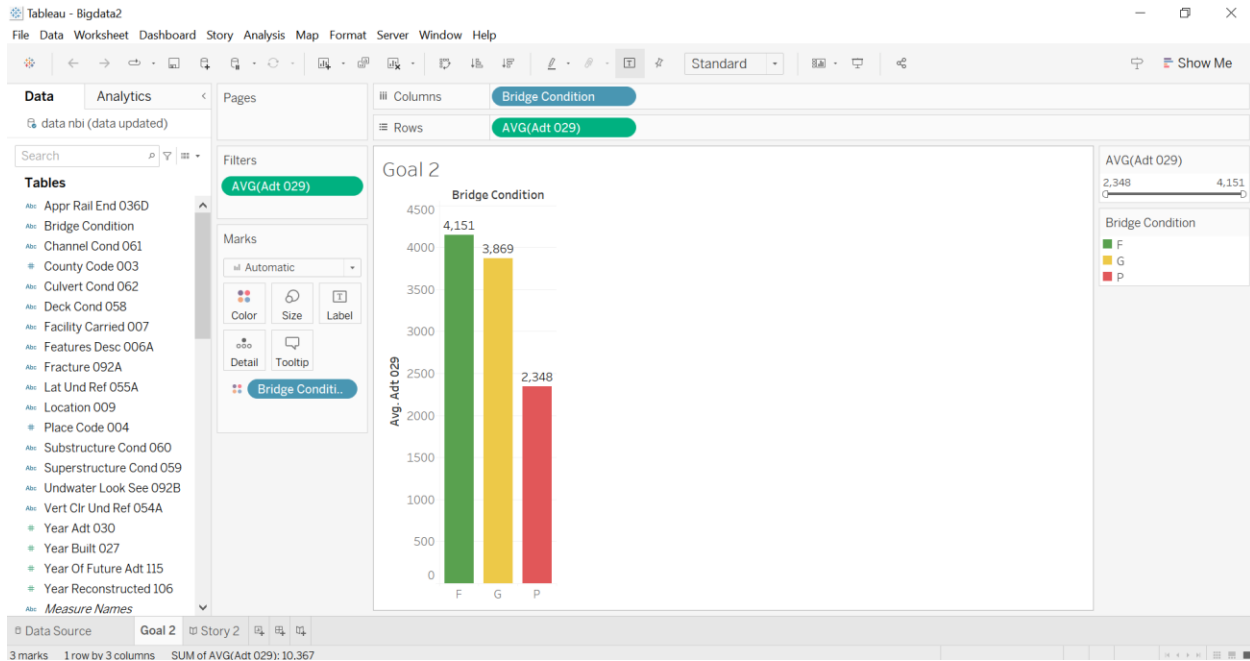
part-r-00000 X

dataset.csv U

```
mapreduce-mvn > data > output > part-r-00000
1 F 4151.035192
2 G 3868.866104
3 P 2347.543759
4
5 |
```

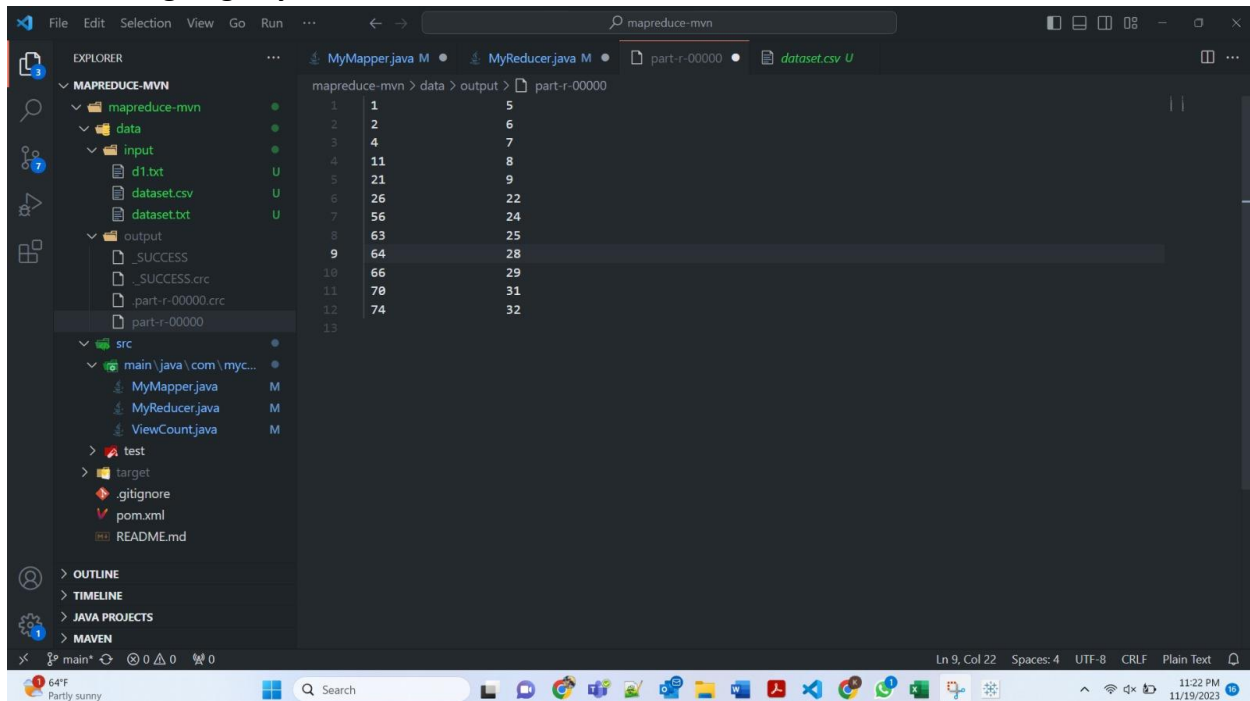
Ln 5, Col 1 Spaces: 4 UTF-8 CRLF Plain Text

64°F Partly sunny 10:43 PM 11/19/2023



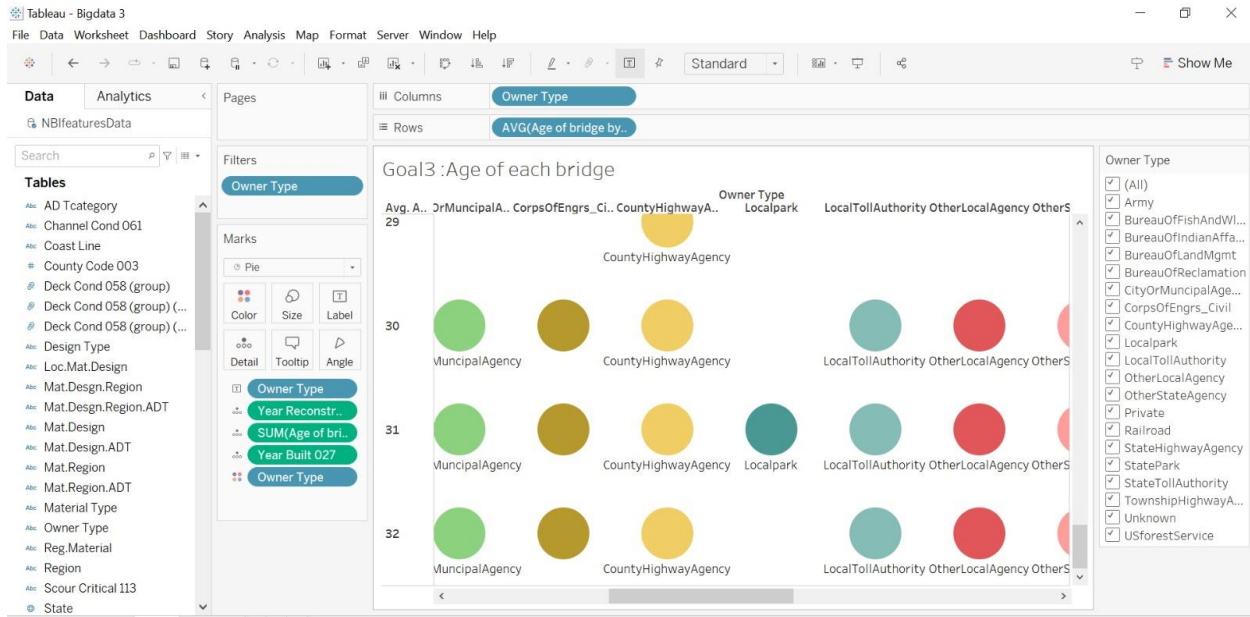
**Story:** This bar chart shows average daily traffic on bridges grouped according to their condition. Each condition of bridge is represented in each color. And filters can be applied to both Average traffic condition and bridge condition.

### Goal 3: Bridge Age by Owner



```
public static class MyReducer extends Reducer<Text, Text, Text, DoubleWritable> {  
    @Override  
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, Interru  
        double sumAge = 0.0;  
        int count = 0;  
  
        for (Text value : values) {  
            String[] years = value.toString().split(",");  
            if (years.length >= 2) {  
                int yearBuilt = Integer.parseInt(years[0]);  
                int yearReconstructed = Integer.parseInt(years[1]);  
  
                int currentYear = java.time.Year.now().getValue();  
                int age = Math.max(currentYear - yearBuilt, currentYear - yearReconstructed);  
  
                sumAge += age;  
                count++;  
            }  
        }  
  
        if (count > 0) {  
            double avgAge = sumAge / count;  
  
            context.write(key, new DoubleWritable(avgAge));  
        }  
    }  
}
```

```
package com.mycompany.app;  
  
import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Mapper;  
  
public static class MyMapper extends Mapper<LongWritable, Text, Text, Text> {  
    @Override  
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedExcep  
        String[] row = value.toString().split(",");  
        if (row.length >= 50) {  
            String structureNumber = row[37].trim();  
            String yearBuilt = row[27].trim();  
            String yearReconstructed = row[106].trim();  
  
            context.write(new Text(structureNumber), new Text(yearBuilt + "," + yearReconstructed));  
        }  
    }  
}
```



**Story:** This pie chart with bubbles shows age of each bridge labelled in the pie chart with bubbles which is calculated based on the year built and year of reconstruction(if happened). It is represented based on each individual structure number. Each bubble on the pie chart represents a different structure and can show different details including year built, year of reconstruction and age calculated using max function.

#### Goal 4: Traffic Load Comparison

```

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

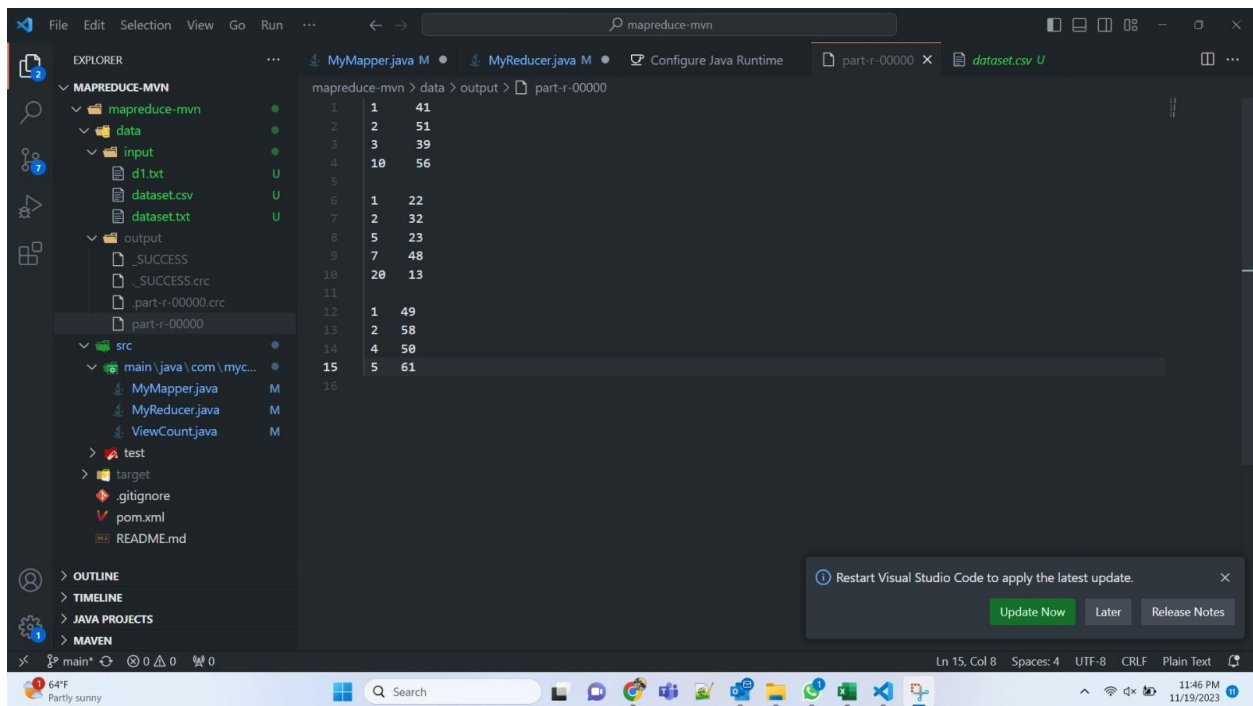
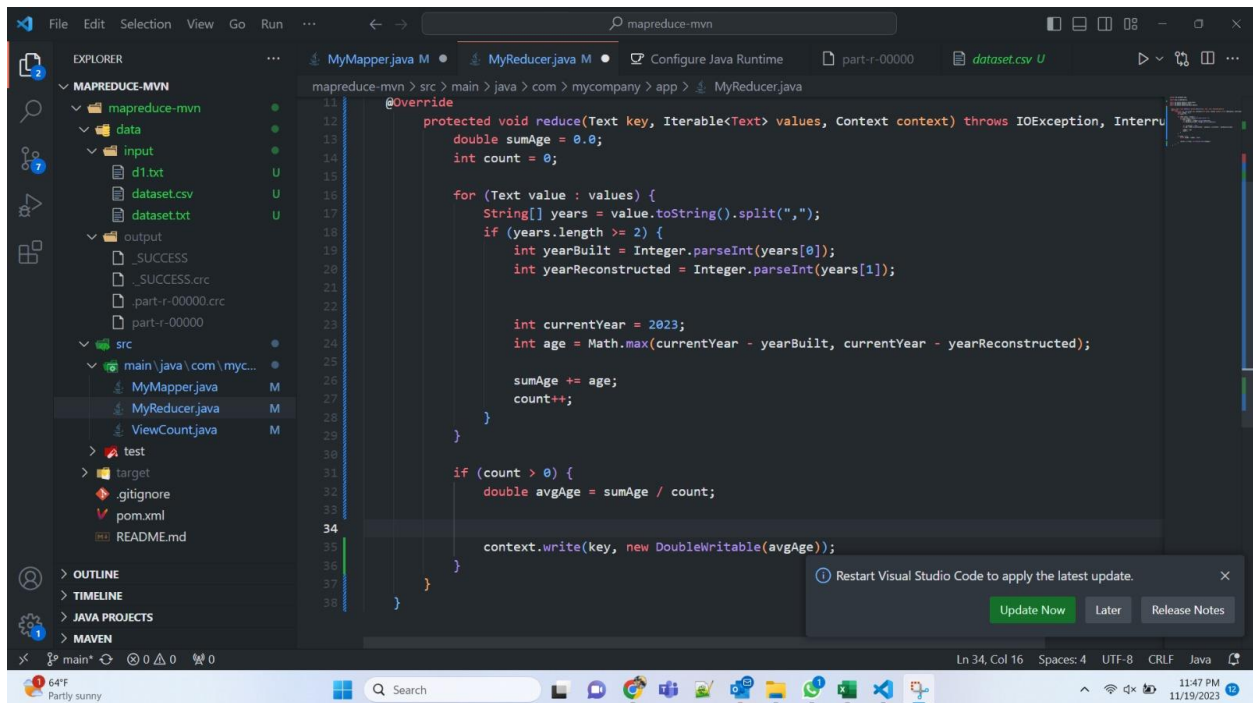
public static class MyMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] row = value.toString().split(",");
        if (row.length >= 50) {
            String bridgeCondition = row[58].trim();
            String trafficLanes = row[28].trim();
            String yearBuilt = row[27].trim();
            String yearReconstructed = row[106].trim();

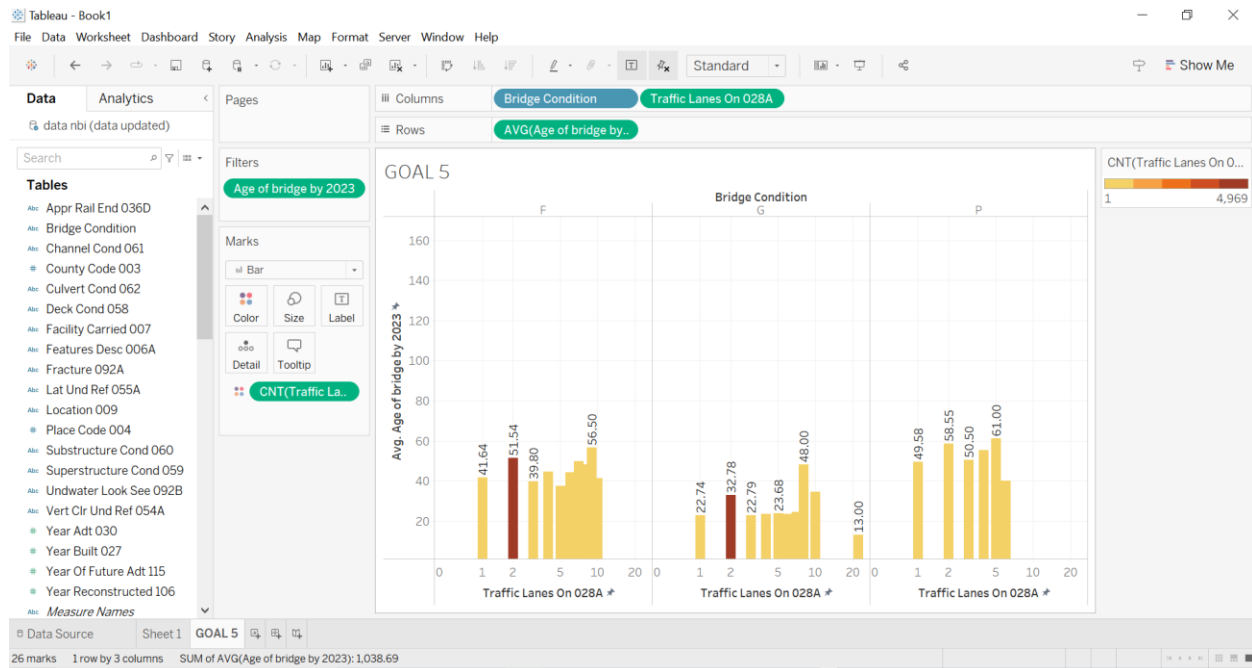
            context.write(new Text(bridgeCondition + "-" + trafficLanes), new Text(yearBuilt + "-" + yearReconstructed));
        }
    }
}

```

Restart Visual Studio Code to apply the latest update. [Update Now] [Later] [Release Notes]







**Story:** This graph compares the bridge condition with the average age of the bridge by the year 2023. Here the bridges are grouped according to the number of traffic lanes present and the average age and the condition of these grouped bridges are compared here.

## Conclusion:

### Goal 1: Average Bridge Condition by County

The output of this analysis will provide the average condition rating for bridges in each county. This information is crucial for identifying counties with bridges that might require more maintenance or reconstruction. It can help allocate resources efficiently to address the condition of bridges in different areas.

## Goal 2: Traffic Impact Analysis

Analyzing the impact of traffic on bridge condition by grouping bridges based on traffic load is valuable for transportation planning. It allows us to understand how heavily trafficked bridges correlate with their condition. This information can guide decisions on infrastructure improvements and expansions in areas with high traffic loads.

## Goal 3: Bridge Age by Owner

Determining the average age of bridges based on their maintenance ownership provides insights into the aging infrastructure for different owners (e.g., state, county). This knowledge is essential for planning maintenance schedules, budgeting, and prioritizing infrastructure investments.

## Goal 4: Traffic Load Comparison

Grouping bridges by the number of traffic lanes and comparing their conditions and ages provides valuable information on how the design and capacity of bridges influence their longevity. This knowledge is crucial for optimizing traffic flow, identifying potential bottlenecks, and planning for future infrastructure needs.

In conclusion, the analysis of these goals provides actionable insights for infrastructure planning, resource allocation, and decision-making to ensure the safety and sustainability of the bridge network. It enables authorities to prioritize maintenance efforts, plan for infrastructure upgrades, and make informed decisions on future projects.

## Citations:

<https://www.fhwa.dot.gov/bridge/nbi/ascii2023.cfm>

GitHub repo link: [https://github.com/ghantanagamounikadevi/BigData\\_Project-Team7-.git](https://github.com/ghantanagamounikadevi/BigData_Project-Team7-.git)