Data Visualization in Processing

Noah Dunn

April 2020

1 Introduction

The past decade of the data revolution has lead to the general public being exposed to large quantities of data on a day to day basis. Gone are the days of Information Systems Analysts and Statisticians being the only groups privy to immense sums of data. From high school classrooms to fast-food restaurants, data science is presented as the new norm for how to proceed in staying relevant in the digital age. However, the idea of looking at hundreds of thousands of rows and columns for a set of data is not something that appeals to a lot of people. This is where the science of *Data Visualization* comes into play, providing a way to present data to a general audience, where everyone can understand what is being presented. Using a modern programming language, IDE, and tool wrapped into one, the power of *Processing* gives even people new to programming a chance to develop their data visualization skills.

2 Data Visualization

2.1 The History of Data Visualization

Some of the earliest forms of data collections existed far before the computation age, and even further back than the creation of the term "Data". The earliest forms of data visualization were "...geometric diagrams, in tables of the positions of stars and other celestial bodies..." [1]. Later on down the timeline, data visualization would be used for mapping purposes, capturing images in the age before cameras, surveying, navigation, and several other purposes all revolving around the era of exploration and land claims. Even further on, during the mid to late 1800s, data visualization would move to structures seen in the modern-day in form of pie charts, bar graphs, line graphs, etc, all used during the era of global industrialization [1].

2.2 Modern Data Visualization

The late 1970s saw the beginnings of what would shape the modern understanding of data visualization. The ability of computing power to generate

images from large amounts of data in a programmatic fashion was both efficient and cleaner than previous hand-drawn techniques. Additionally, during this particular era, the field of data visualization saw a change in diversity, as researchers tried new advanced ways of showing data that would have been too time-consuming to justify drawing previously. In particular, modern data visualization had the potential for a view to interact directly with the visual, providing the ability for scaling, modeling, and visualization [2]. The modern concept of data visualization has changed from a discussion on "Can we visualize this?" to "What is the best way to visualize this?".

2.3 Data Visualization Principles

Data Visualization is intended to exactly as it says, enable our audience to *visualize data.* The overall goal is to present very intricate and elaborate data in a simple but clear format. In general, the book "Designing Data Visualizations: Representing Informational Relationships" offers five purposes for using data visualization [4]:

1. It enables the transmission of a large amount of information to the brain very quickly
2. It takes advantage of the human brain's ability to identify patterns and communicate relationships and meaning
3. It can inspire new questions and further exploration
4. It helps identify additional sub-problems from the data
5. It is really good for identifying trends and outliers, or for identifying interesting specific data points

Particularly in research related fields, data visualization communicates to the reader of a research paper some information about their topic and their findings faster than having to read through an entire block of text. A researcher perusing the topic may formulate questions inspiring future papers, come up with additional problems to tackle in the domain, or may just make note of something interesting they wish to investigate further.

2.4 Data Visualization Best Practices

A universal set of data visualization standards are impossible to lock down due to the necessity and restrictions of different fields. Even within a single field, medical research, for example, there exists differing restrictions between domains [8][3]. However, there are some general "best practices" that are applicable to all fields. In the context of research, Kelleher C. and Wagener T. summarize the succeeding rules [5]:

- Create the simplest visualization that conveys the information you want to convey

- Focus on visualizing patterns or on visualizing details, depending on the purpose of the visual
- Select an appropriate color scheme based on the type of data
- Aggregate larger data sets in meaningful ways

All of these rules are call backs to the overall goal of data visualization. Data needs to be presented in a way that transmits as much information as possible to the intended audience [6].

3 Getting Familiar with Processing

3.1 The History of Processing

There are many tools that can be used to build good data visualizations that follow all the principles and best practices. However, many of these tools fall into one of two categories. Some tools give the user a great deal of control over their visualization but at the cost of programmatic simplicity. The other category is all the tools that are very user friendly but prevent the user from visualizing exactly what they intend to show. The language of Processing, started in the spring of 2001 by Ben Fry and Casey Reas, was created to be a very powerful first programming language for those interested in graphics functionality in programming. [1].

3.2 Downloading and Installing Processing

Every new tool requires a download and an install, and the journey to understanding processing is no different. The download for processing can be found at this link[2]. Processing is both open-source and free; it is available to everyone. Downloading Processing will produce a .zip file that needs to be extracted. Inside the resultant folder is an executable that should be titled "Processing.exe". Running this executable, if everything goes according to plan, will result in the following diagram.

[1] The full history of Processing, as well as where the inspiration behind Processing can be found: `https://processing.org/overview/`

[2] Download Processing 3 here: `https://processing.org/download/`

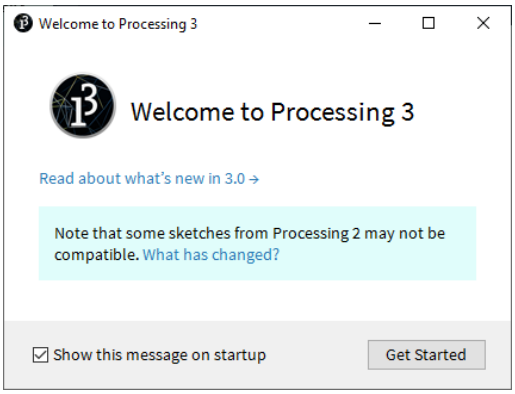

Figure 1: A Sign Everything is Working Properly

Hitting the "Get Started" button or the "X" will reveal the IDE to the user.

3.3 Learning the Processing IDE

As stated previously, the Processing tool is a language, run time environment, and IDE wrapped up all in one. The starting screen for processing should appear as so:

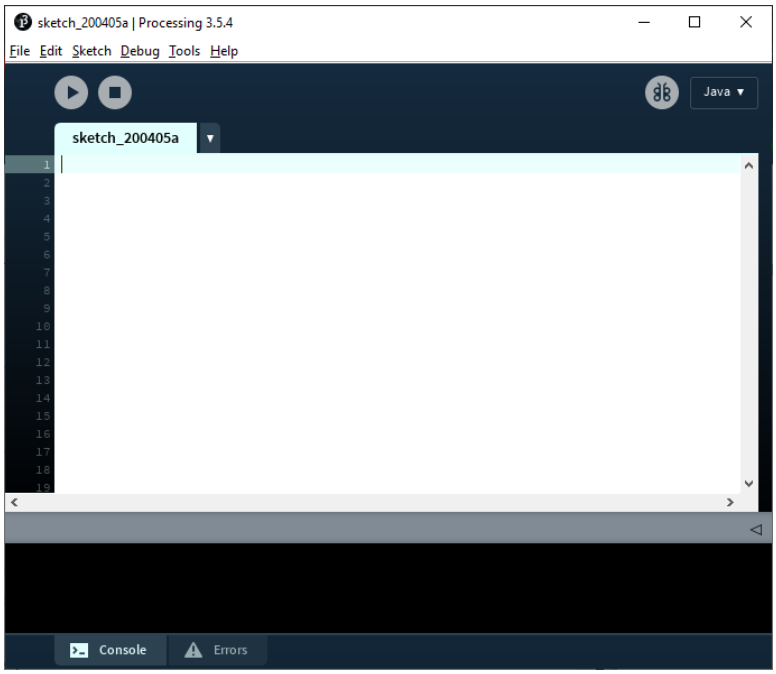

Figure 2: The Blank Processing IDE

A seasoned programmer will believe this to be a very simple IDE and that is exactly the point. The philosophy of Processing is to keep the toolkit as streamlined as possible to enable the user to spend their energy and effort designing complex visuals. For completeness, every piece of the IDE is outlined in the following diagram:

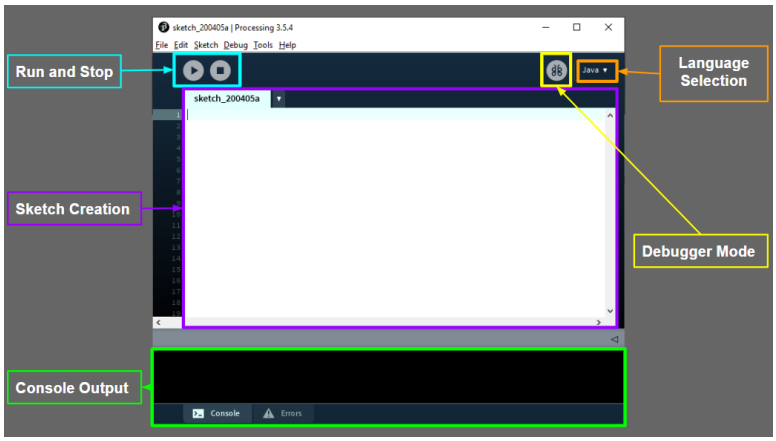

Figure 3: The Blank Processing IDE

The language of choice for this study is *Java*; however, Processing has support for *Python*[3], *Javascript*[4], *Android* [5], and *Raspberry Pi*[6]. The bulk of a programmer's time in Processing will be spent in the "Run and Stop", "Sketch Creation", and "Console Output" sections.

4 Writing Code in Processing

The foundation for all Processing programming is the *sketch*. The sketch is nearly the same as a class in Java, and the term "sketch" refers to the interactive graphical nature of Processing [7]. Those who are familiar with Java will have little trouble writing Processing code. That being said, there are some very important and notable differences.

4.1 Setup and Draw

Unlike Java, Processing sketches do not have a method represented by the standard [7]:

[3]Support for Python: `https://py.processing.org/`
[4]Support for Javascript: `https://p5js.org/`
[5]Support for Android: `https://android.processing.org/`
[6]Support for Raspberry-Pi: `https://pi.processing.org/`
[7]Template available at: `https://tinyurl.com/sshasvg`

```
1 public class HelloWorld {
2    public static void main(String[] args) {
3       System.out.println("Hello World!");
4    }
5 }
```

In Processing, a sketch will execute without the need for any method at all. It is perfectly valid Processing to write the following:

```
1 println("Hello World");
```

Small programs like this, while simple and clean, do not take advantage of the Processing language. The first method in a processing sketch that should be included in the void setup() method[8]. This method is used to declare the value of global variables, set the size of the output screen, and set the frame rate the sketch will run at. It will execute before the void draw() method and any other user-defined method.

```
1    int x;
2    void setup() {
3       x = 2;
4       size(300, 300);
5       frameRate(30);
6    }
7    void draw(){
8       println("I am run number: " + x);
9       x++;
10   }
```

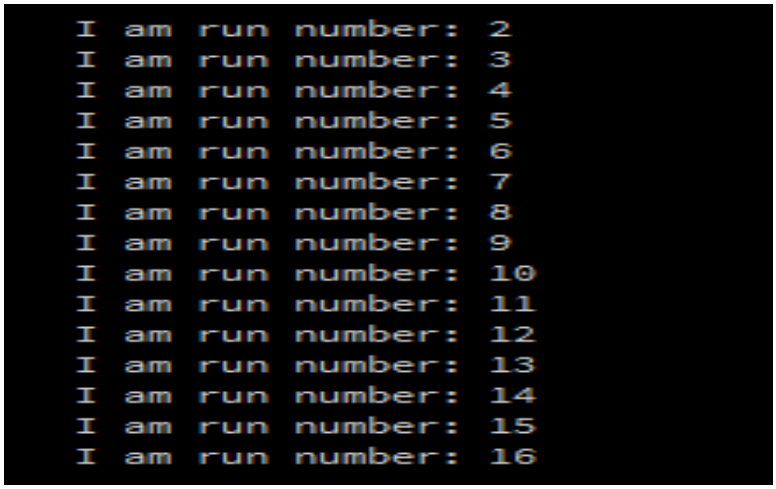

Figure 4: The Looping Nature of the Draw Method

[8] https://processing.org/reference/setup_.html

The draw method[9] is the most important method to start taking advantage of Processing's built-in functionality. The draw method will execute 60 times per second by default, or at whatever frame rate is established in the setup. Any graphical code written inside the draw method will be displayed on the popup window that is created anytime a sketch is run. After each draw loop, everything inside the loop is erased, so anything a user wants to display constantly must be written inside the draw loop, or called from the draw loop.

4.2 Drawing Points, Lines, and Shapes

In Java, getting to a point to be able to draw a simple shape on a frame takes a lot of overhead, and more lines than should be necessary. Processing streamlines this process in the draw() loop with built-in functionality for point creation, line drawing, and shape rendering. In Processing, the coordinate system begins at (0, 0) in the top left corner of the frame, and ends at (width - 1,height -1) in the bottom right corner. A point[10] in processing is drawn by calling the point method with an x and y value pair. The point can have its color[11] modified by use of the built-in stroke and color functions, and the size of the point can be modified with the strokeweight[12] function.

```
1  void setup() {
2     size(200, 200);
3     frameRate(30);
4  }
5  void draw() {
6     // Set the radius of the point to 15 pixels
7     strokeWeight(15);
8     // Create the color red using standard RGB values
9     color red = color(255, 0, 0);
10    // Make the color of the stroke red
11    stroke(red);
12    // Put it in the middle
13    point(width/2, height/2);
14 }
```

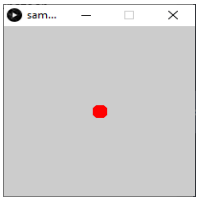

Figure 5: Drawing a Simple Dot

[9] https://processing.org/reference/draw_.html
[10] https://processing.org/reference/point_.html
[11] https://processing.org/reference/color_.html
[12] https://processing.org/reference/strokeWeight_.html

Similar techniques are applied to drawing lines, and shapes[13] can take advantage of the fill function to color not just the border of a given shape, but also the inside of the shape. In addition, the background function can be used to set the background of the output frame to a specified RGB color.

```
void setup() {
  size(200, 200);
  frameRate(30);
}
void draw(){
  // Changes the background color to black
  color black = color(0, 0, 0);
  background(black);
  // Set the desired colors up
  color red = color(255, 0, 0);
  color yellow = color(255, 255, 0);
  color green = color(0, 255, 0);

  //The Top Circle is Red
  strokeWeight(1);
  stroke(red);
  fill(red);
  circle(width/2, height/4, 40);
  //The Middle Circle is Yellow
  stroke(yellow);
  fill(yellow);
  circle(width/2, 2 * height/4, 40);
  //The Last Circle is Green
  stroke(green);
  fill(green);
  circle(width/2, 3 * height/4, 40);
  //Drawing a yellow box around the circles

  // Disable the fill function to draw an outline
  noFill();
  stroke(yellow);
  rect(width/3, height/8, width/3, 6 * height/8);
  // Draw two thick red lines
  stroke(red);
  strokeWeight(10);
  line(30, 30, 170, 170);
  line(170, 30, 30, 170);
}
```

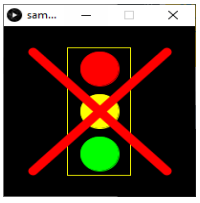

Figure 6: Drawing a Traffic Light

[13] https://processing.org/tutorials/drawing/

4.3 File Input and Output

Processing has built in methods to make reading and writing to files much simpler. To read plaintext files, Processing supplies the loadStrings method[14] which automatically reads an entire inputted file into an array of strings that can be processed like normal. Also, Processing provides the loadTable method[15] which can read in CSV table information in a very straightforward manner [16].

```
// words.txt appears as such
// cheese
// gravy
// a biscuit

//names.csv appears as such
//id,name,number
//0,Noah,17
//1,Nick,20
//2,Josh,13
void setup() {
  size(200, 200);
  frameRate(30);
  String[] words = loadStrings("words.txt");
  // The header flag indicates that the csv file's first
      line is the name of the headers
  Table table = loadTable("names.csv", "header");
  int nameNumber = 0;
  for (TableRow row : table.rows()){
    println(row.getString("name") + " wants " + words[
        nameNumber]);
    nameNumber++;
  }
}
```

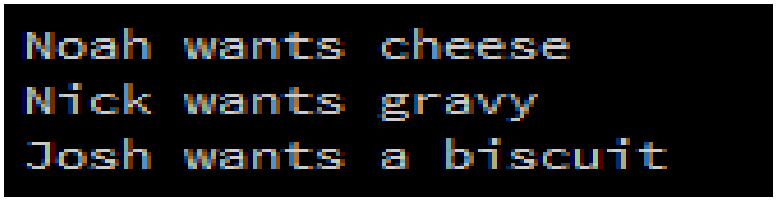

Figure 7: CSV and Plaintext File Input in Processing

For data visualization in Processing, reading files will be significantly more common than the the infrequent write to a file. Nevertheless, Processing does also streamline these processes with the inclusions of the saveTable method[17] and the createWriter method[18].

[14] https://processing.org/reference/loadStrings_.html
[15] https://processing.org/reference/loadTable_.html
[16] Processing also supports XML: https://processing.org/reference/loadXML_.html
[17] https://processing.org/reference/saveTable_.html
[18] https://processing.org/reference/createWriter_.html

4.4 PImage and the Map Function

Particularly in cases where a researcher or data scientist is intending to work with images of maps or diagrams, the PImage class[19] in Processing is an easy way for displaying files. A PImage object reads in the pixels of a desired image and then allows for direct manipulation of the individual pixels on that image. This is particularly useful for demonstrations requiring highlighting or filtering of certain areas. It comes complete with a wide toolkit include a resize method, the .pixels field, and a loadPixels method, which are what allow reading and writing from and to individual pixels respectively.

```
PImage chicagoMapDark;
PImage chicagoMapLight;
// The Chicago map is a 792 x 844 pixel image
void setup() {
  chicagoMapLight = loadImage("ChicagoMap.PNG");
  // This scales the image to half size while keeping the
      width/height aspect ratio
  chicagoMapLight.resize(0, 844/2);
  chicagoMapDark = negative(chicagoMapLight);
  // Size cannot use variables as arguments, they must be
      literals
  size(792, 422);
  float xPosition = 0.0;
  float yPosition = 0.0;
  // The position where the first image ends
  float xPosition2 = chicagoMapLight.width;
  float yPosition2 = 0.0;


  image(chicagoMapLight, xPosition, yPosition);
  image(chicagoMapDark, xPosition2, yPosition2);
}

/* A helper method to turn the map black/white */
PImage negative(PImage source){
  PImage result = createImage(source.width, source.height,
      RGB);
  result.loadPixels();
  // in result, store darker version of what is in source
  for(int i = 0; i < source.pixels.length; i++){
   // First, extract the rgb components of the source image
       :
   float r = red(source.pixels[i]);
   float g = green(source.pixels[i]);
   float b = blue(source.pixels[i]);

   float avg = (r + g + b) / 3;

   result.pixels[i] = color(255 - avg, 255 - avg, 255 - avg
       );
  }
  return result;
}
```

[19] https://processing.org/reference/PImage.html

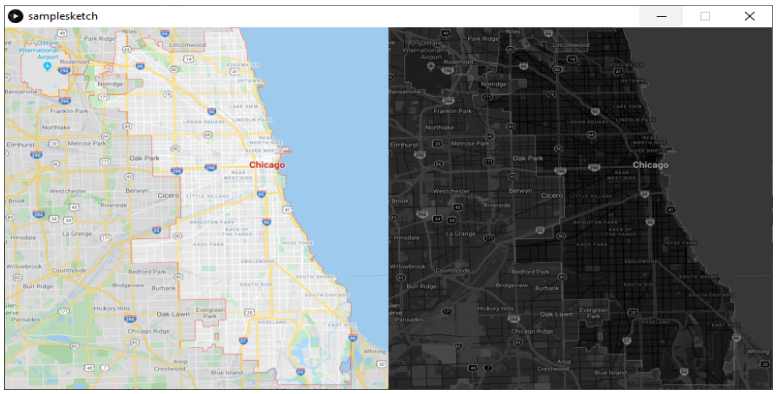

Figure 8: Using PImage to Display and Modify a Map of Chicago

It is often the case that one or several data points need to be transformed from one scale in another, specifically for mapping and graphing visualization. Processing also has support for this in the form of their map function [20].

4.5 Interactive Functionality

Action listeners and mouse listeners can be cumbersome to implement and bind correctly in many languages. Processing has support for both, and much in the spirit of all the other elements present, the philosophy is to make the process easier. At any point during the run time of a sketch, the program has access to two reserved global keywords: mouseX and mouseY. These give the X and Y value of a mouse pointer at any given point that the mouse is inside the frame. Processing comes equipped with all the following mouse listeners to compliment the global mouseX and mouseY: mouseClicked[21], mousePressed[22], mouseDragged[23], mouseReleased[24], and mouseMoved[25].

Complimenting the mouse listener suite, the two key listeners that Processing offers are the keyPressed[26] and keyReleased[27] listeners. All of these can be used to build interactive visualizations and put power into the user's hands to manipulate a model.

4.6 Shortcomings of Processing

Processing does well at efficient graphics manipulation, but it is not a bulletproof software. One flagrant feature Processing lacks is console related input from the

[20] https://processing.org/reference/map_.html
[21] https://processing.org/reference/mouseClicked_.html
[22] https://processing.org/reference/mousePressed_.html
[23] https://processing.org/reference/mouseDragged_.html
[24] https://processing.org/reference/mouseReleased_.html
[25] https://processing.org/reference/mouseMoved_.html
[26] https://processing.org/reference/keyPressed_.html
[27] https://processing.org/reference/keyReleased_.html

keyboard. All user input is required to be handled using keyboard listeners. This can prove quite tedious if a developer is attempting to test out different values for variables and has to modify their code every time they wish to test. As another problem, Processing is notoriously slow, due to the nature of it being an interpreted language off an already rather slow language. For this reasoning, data processing and cleansing should be done separately or before executing a visualization, to avoid abhorrent slowdown.

5 Data Visualization in Processing

The preceding list of methods and tools in Processing is not exhaustive, but it does provide a foundation for individuals of all programming backgrounds to piece together a powerful visual with very few steps. To give some understanding of the types of powerful visuals that can be created with the preceding techniques, it would be instructive to provide some examples of these visuals. Open Processing[28] is a great online resource for Processing developers, as it demonstrates both a sketch running as well as access to the source code to build it.

5.1 Energy Consumption Data Visualization

In research related to energy usage across the world, it would be beneficial to have a visual understanding of just how much energy some countries use compared to others. User Jem Brown on Open Processing provides such an example shown below[29]:

[28] https://www.openprocessing.org/
[29] https://www.openprocessing.org/sketch/198208

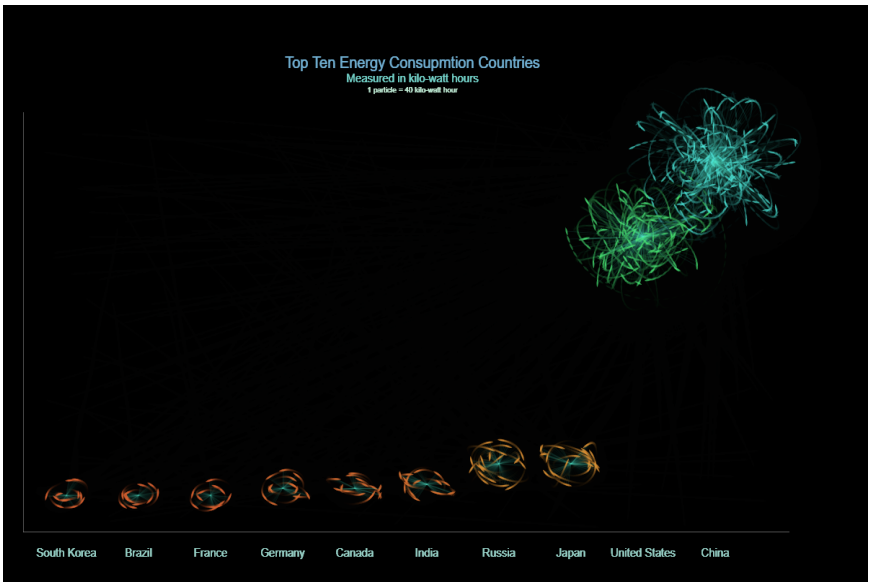

Figure 9: Energy Consumption Worldwide Visualization Built in Processing

In terms of the data visualization best practices discussed earlier, this visual does transmit large amounts of information quickly, it identifies a trend, and it does inspire a pattern in the brain. For the confines of best practices, the visual is simple with all information shown, and it does successfully aggregate larger data sets in a meaningful way.

The developer makes use of several of Processing's features already discussed here: fill, line, rect, stroke, to name a few. The developer also uses some additional ones like the text method [30], and the PVector class[31] to add titles and physics driven processes. The total diagram makes up only 152 lines of code.

5.2 Organ Transplant Data Visualization

Developer Shannon Balke on Open Processing tackles a medical problem with their visualization[32]. This interactive visual allows users to determine the various causes of why patients were removed from different organ transplant waiting lists in different years.

[30] https://processing.org/reference/text_.html
[31] https://processing.org/reference/PVector.html
[32] https://openprocessing.org/sketch/183729

Figure 10: Organ Transplant Rejection Visualization Built in Processing

This interactive model is good for pointing out outliers, inspiring new questions, and prompting additional sub-problems. The sheer quantity of descriptive data generates a lot of questions that can be solved through additional research. Also, this diagram excels in particular at the best practices of data aggregation and meaningful color (a lot of these colors seem consistent with medical color palettes).

The creator of this sketch uses some familiar methods: loadStrings, rect, noFill, mouseMoved, mousePressed. Also, the author makes use of several self defined classes that add a layer of complexity onto existing Processing features, totalling a complete 635 lines of code.

6 Conclusion

Processing is a versatile kit that makes for a powerful data visualization tool. The goal of Processing is to make building impressive visuals as quick and easy as possible. Newcomers to the programming world will find that they can create complex visuals from understanding and mastering the built-in methods. Experts in Programming can find quicker development cycles to spin up

exploratory data analysis faster than they would be able to otherwise. Data visualization as a field is not disappearing anytime soon, and as such, people of all backgrounds should have access to exploring and showing off real-world information.

As a final note, I would be amiss if I did not include a note of thanks to Prof. Norm Krumpe from Miami University for giving me my first introduction to the technology of Processing. Without him, this paper and study would not have happened or been possible.

References

[1] M. Friendly. A brief history of data visualization. In C. Chen, W. Härdle, and A Unwin, editors, *Handbook of Computational Statistics: Data Visualization*, volume III. Springer-Verlag, Heidelberg, 2006. (In press).

[2] Michael Friendly and Daniel J Denis. Milestones in the History of Thematic Cartography, Statistical Graphics, and Data Visualization. *URL http://www.datavis.ca/milestones*, 32:13, 2001.

[3] C. Hildebrand, J. Stausberg, K. H. Englmeier, and G. Kopanitsa. Visualization of medical data based on EHR standards. *Methods of Information in Medicine*, 52(01):43–50, 2013.

[4] Noah Iliinsky and Julie Steele. *Designing Data Visualizations: Representing Informational Relationships*. "O'Reilly Media, Inc.", 2011.

[5] Christa Kelleher and Thorsten Wagener. Ten guidelines for Effective Data Visualization in Scientific Publications. *Environmental Modelling Software*, 26(6):822 – 827, 2011.

[6] Prasanthi Korada. 5 Best Practices for Data Visualization. *Oracle AI and Data Science Blog*, Jul 2018.

[7] Casey Reas and Ben Fry. *Getting Started with Processing: A Hands-on introduction to making interactive Graphics*. Maker Media, Inc., 2015.

[8] Matthew O Ward, Georges Grinstein, and Daniel Keim. *Interactive Data Visualization: Foundations, Techniques, and Applications*. CRC Press, 2010.