Paper 275-2012

Let the Data Paint the Picture

Data-Driven, Interactive and Animated Visualizations Using SAS®, Java and the Processing Graphics

Ryan Snyder, Institute for Advanced Analytics, North Carolina State University, Raleigh, N.C.

Patrick Hall, Institute for Advanced Analytics, North Carolina State University, Raleigh, N.C.

ABSTRACT

This paper introduces a scalable technique that combines the data manipulation capabilities of Base SAS® 9.2 with the Java Processing graphics library to generate customizable visualizations. With an instructive example, the reader is guided through connecting to the sashelp.iris sample data using a SAS/SHARE® server to build a visualization applet. The explanation includes details for adding animation and simple mouse and keyboard interactions into the visualization. A basic understanding of Object Oriented (OO) programming is assumed. The included example was designed for the Windows platform; however, Java and Processing are designed to support many other operating systems. Additional reference material and sample visualizations are also included.

INTRODUCTION

By considering audience and purpose, harnessing the procedural efficiency of SAS, and drawing with Processing, effective visual representations of data can be produced for a broad array of subject matter through the creation of data-driven static images, animations, or interactive Java applications and applets (apps). Utilizing SAS to organize the data behind a complicated visualization can reduce both the time spent writing data manipulation code and the time required to render the image or animation. SAS also provides a myriad of techniques to classify data, while Processing offers a wide-ranging, high-level interface for 2-D and 3-D graphics. Moreover, the Processing methodology *encourages* animation, user-interaction and the creation of web applets.

WHAT IS PROCESSING?

Processing is an open source programming language and environment designed to create interactive images and animations. It was originally developed to be an intuitive tool for novice programmers and non-technical users. Processing has since grown into a complete graphics Application Programming Interface (API), while still retaining much of its accessibility. Throughout its evolution, the on-line Processing community has published in-depth documentation and detailed tutorials available freely on the web at http://www.processing.org.

Processing is available as a Java library, allowing Processing classes and methods to be used in any standard Java construct or Java app. Because they are Java objects, Processing classes can be instantiated directly in a SAS data step using the Base SAS® Javaobj construct, and SAS data can be accessed by Processing classes locally or over the web using Java Database Connectivity (JDBC) and data base servers such as SAS/SHARE or MySQL.

BEFORE BEGINNING

A fundamental consideration for any visualization is the target audience, specifically what they already know about the topic to be visualized, and what they want to know. Having a clear understanding of these two notions will assist in creating a visual analysis that is interesting, comprehensible and free of bias. To ensure the visualization is understandable for the audience, consider whether the data and message of the visualization are better suited for a more immediately recognizable form of static chart or graph, or a more complex, interactive or animated visualization. For many types of quantitative visual communications, a line graph, bar chart, scatter plot or pie chart is preferable. When choosing to forgo the familiarity of these standards, there must be a significant reward for the audience.

Knowing the purpose of the visualization can be immensely helpful in resolving the accessibility versus complexity tradeoff. Generally, visualizations meant for presentation should be more directly recognizable, with only a small number of highlighted visual narratives. Conversely, visualizations and visual apps meant for exploration and discovery may be highly complex, interactive, animated or any combination thereof. When choosing to construct an interactive visual app, the interactive functionality should support the audience's analytic goals. Common interactive capabilities the audience may expect include: selecting data, reorganizing data, zooming and data filtering. Animation in visualization is typically used for multi-dimensional analyses involving time, to increase audience engagement, or to emphasize trends, patterns or outliers. Two dimensional problem domains with time as the independent variable can typically be expressed clearly using a static line chart. The method of *Small Multiples*, or a series of small charts having similar axes within the same larger graphic, is often employed for multi-dimensional analysis where animation may be inappropriate.

Accuracy and engagement are also important aspects of purpose. If the infographic is meant to display the analyzed data as accurately as possible, care should be taken to ensure all graphic attributes are to scale, that aesthetic complexity is uniform throughout the image(s), and that all information contained in the data, or an explicitly defined subset, is displayed. If the purpose of the visualization is to increase or motivate the audience's engagement with the portrayed data or topic, then a less rigorous and more aesthetically focused approach may be warranted, provided any artistic distortions to the character of the data are noted. While many enhancement techniques and effects can be applied to digital images at an analyst's discretion, simply dropping less critical information from a visualization and increasing the aesthetic detail of more critical information can suffice to increase audience engagement.

GUIDELINES FOR COMPLEX VISUALIZATIONS

In creating complex visualizations the authors have found several guidelines useful:

- **Keep dimensionality as low as possible.** Keep dimensionality low, both in terms of audience perspective and data dimensions. Avoid 3-D plots except when natural or intuitive to the data, and do not attempt to display more than seven variables within the same visualization. Even when displaying a smaller number of variables, the combination of hue, luminance, and texture (e.g. size, orientation or dot density) for attributes must be carefully chosen to maximize visual differences between dissimilar data records. The small multiples technique can aid in clearly displaying several variables in the same image.
- **Keep axes honest**. In an already complex visualization, tampering with audience expectations of scale is typically undesirable. Inconsistent ranges for the axes of small multiples are a common source of data misinterpretation.
- **Color and texture are easily recognizable for most audience members**. When choosing how to communicate different types of information about a given attribute in an image, the easiest variations for most audience members to recognize visually are, in order:
    1. Color Hue
    2. Color Luminance
    3. Texture (e.g. Size, Orientation or Dot Density)

  Although hue attribute differences tend to be the most immediately distinguishable, schemes including red and green may cause difficulty for red-green color blind audience members.
- **Avoid Common Visual Pratfalls**. Both audience members and analysts are susceptible to limited memory for visual detail and limited attentional focus, known as Change Blindness and Inattentional Blindness respectively. Change Blindness can occur when users experience difficulty in making analytical decisions based on graphics on separate pages or in separate windows. Inattentional Blindness can occur when a user's full attention is fully deployed on a demanding task within a visualization, and they fail to notice other important features.

EXAMPLE VISUALIZATION

The basic framework followed by the authors to create visualizations is divided into three phases:

1. Consider the audience and purpose.
2. Organize and format the data to be visualized using SAS.
3. Build the visualization applet using Java and the Processing library.

This technique combines the strengths of both SAS and Java, while saving development time and processor time.

PRELIMINARY DESIGN CONSIDERATIONS

The presented example will begin by using SAS to explore the *iris* data set, located in the *sashelp* library. Before constructing the final product, it is important to consider audience and purpose, and to understand basic aspects of the data that will shape the size and dimensionality of the visualization. The audience for the example is expected to be somewhat technical and interested in creating more complex graphics. The purpose of the visualization is assumed to be exploring the *iris* data, and encouraging engagement with the proposed technique. Hence, the authors will attempt to create an infographic that will have an easily recognizable form to a technical audience, which will incorporate interactive functionality to foster exploration, and which will be aesthetically engaging while remaining true to the character of the *iris* data.

ORGANIZING AND FORMATTING SAMPLE DATA USING SAS

The *iris* data set contains measurements of petal and sepal length and width for three species. As the *iris* data set is small and pre-sorted `proc means` will be used to explore the data.

```
/* basic data exploration */

proc means data = sashelp.iris max min mean std;
       by species;
run;
```

```
------------------------------------- Iris Species=Setosa ------------------------------------
  Variable      Label                    Maximum        Minimum           Mean         Std Dev
----------------------------------------------------------------------------------------------
  SepalLength   Sepal Length (mm)     58.0000000     43.0000000     50.0600000       3.5248969
  SepalWidth    Sepal Width (mm)      44.0000000     23.0000000     34.1800000       3.8102440
  PetalLength   Petal Length (mm)     19.0000000     10.0000000     14.6400000       1.7351116
  PetalWidth    Petal Width (mm)       6.0000000      1.0000000      2.4400000       1.0720950

----------------------------------- Iris Species=Versicolor ----------------------------------
  Variable      Label                    Maximum        Minimum           Mean         Std Dev
----------------------------------------------------------------------------------------------
  SepalLength   Sepal Length (mm)     70.0000000     49.0000000     59.3600000       5.1617115
  SepalWidth    Sepal Width (mm)      34.0000000     20.0000000     27.7000000       3.1379832
  PetalLength   Petal Length (mm)     51.0000000     30.0000000     42.6000000       4.6991098
  PetalWidth    Petal Width (mm)      18.0000000     10.0000000     13.2600000       1.9775268

----------------------------------- Iris Species=Virginica -----------------------------------
  Variable      Label                    Maximum        Minimum           Mean         Std Dev
----------------------------------------------------------------------------------------------
  SepalLength   Sepal Length (mm)     79.0000000     49.0000000     65.8800000       6.3587959
  SepalWidth    Sepal Width (mm)      38.0000000     22.0000000     29.7400000       3.2249664
  PetalLength   Petal Length (mm)     70.0000000     45.0000000     55.5400000       5.5703662
  PetalWidth    Petal Width (mm)      25.0000000     14.0000000     20.2600000       2.7465006
```

Output 1. Output from a `proc means` statement on *sashelp.iris*.

For most audiences, petal width and length will be the most accessible measurements in the data set. Width and length measurements also lend themselves to intuitive visualization on the two dimensional x- and y- plane. Since the standard deviation values for both petal length and width constitute a fairly sizeable proportion of each variables respective mean, the spread of the values will yield an informative two-dimensional visualization on the original unit scale. Records that are tightly distributed around a single value must often be transformed or visualized from the perspective of another variable's dimension to create an informative graphic.

Data must also be rescaled to fit the dimensions of the monitor or the final applet. From `proc means` output it can be seen that petal width values range from 1 mm to 25 mm, and that petal length values range from 10 mm to 70 mm. Since the dimensions of the visualization will be defined in pixels, it is a good idea to establish the size of the finished product and then use SAS to format the data to this size. Given that the petal width and length variables are known to roughly follow a normal distribution with no obvious outliers, any standard, screen-size applet such as 600 x 800 pixels (px), will suffice to display the data.

To build an easily understandable visualization, petal width will be displayed on the horizontal x-axis and petal length will be displayed on the vertical y-axis. Given the largest value of the petal width is 25mm, a multiplicative factor of 30 (25 X 30 = 750) can be used to rescale the width values to fit on the 800 px x-axis with a buffer between the minimum and maximum values on either horizontal edge of applet. By the same logic, a multiplicative factor of seven can be applied to fit the petal length variable (70 X 7 = 483) values to the 600 px y-axis. If a higher resolution is required, a much larger applet size should be utilized, and data values should be formatted accordingly.

```
/* scale and create variables to display */
data iris_draw;
       set sashelp.iris;
       pWidthX = petalWidth * 30;
       pLengthY = petalLength * 7;
       pLengthXSize = petalWidth;
       pWidthYSize = petalLength;
run;
```

With the dimensionality and size of the visualization established, the type of chart to display can be chosen. Though a simple scatter plot would adequately convey the petal width and length values, a bubble plot could utilize attribute size and color to add supplemental visual information and aesthetic appeal. Thus, the end product of this example will be an interactive bubble plot, with the petal width as the horizontal x-axis variable, and petal length as the vertical y-axis variable. Additionally, as in Figure 1, each bubble will correspond to a record with its color representing iris species. Each bubble will be further embellished using the original petal width and length values to define the horizontal and vertical size of the bubble. Although bubble size will be inconsistent with the x- and y- axes, this aesthetic enhancement should give audience members a relative, visual scale to compare the size of each iris species.

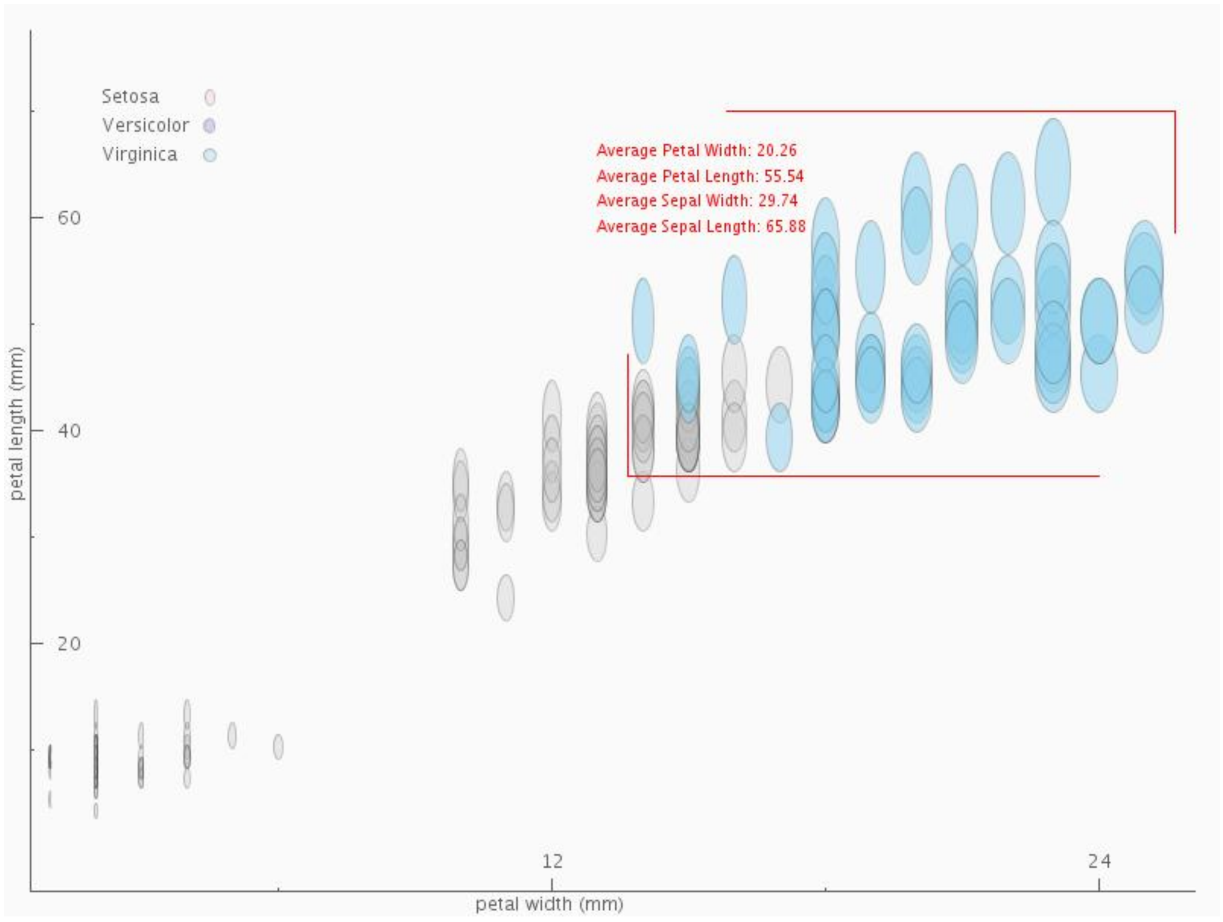

Figure 1. Fisher's Iris data set plotted with petal width on the x-axis and petal length on the y-axis. The red text and highlighted selection at upper right illustrates a mouse-click interaction to display additional information.

SERVING FORMATTED DATA TO A PROCESSING APPLET

As the data has been organized and formatted for easy visualization, it now must be made available to Java apps. A SAS/SHARE server supports database-like read and write access to shared SAS data sets through JDBC. Once a SAS library has been shared using the SAS/SHARE server it can be easily accessed locally, over Local Area Network (LAN) or over the web.

```
/* move formatted iris set into SAS/SHARE server library */
libname sasshare 'C:\Path\To\Shared\SAS\Libraries';

data sasshare.iris_draw;
       set iris_draw;
run;

/* share all info within the library over SAS/SHARE server */
proc operate serverid=__8551;
       allocate library sasshare 'C:\Path\To\Shared\SAS\Libraries';
quit;
```

Although a SAS/SHARE server is probably the most convenient and stable way to share SAS data sets with other applications, several other mechanisms are available. For instance, using SAS/ACCESS® to write data sets to a third-party database and using JDBC to retrieve that data, or instantiating Java Objects directly from SAS data set records using the Javaobj construct in Base SAS 9.2. (See Snyder, Ryan and Hall, Patrick. “A Guide for Connecting Java to SAS® Data Sets.” SAS Global Forum 2012. Cary, NC: SAS Institute, for a more thorough discussion of moving data between Java and SAS.)

CONSTRUCTING THE PROCESSING APPLET (PApplet)

Before proceeding with further details of the example, it should be mentioned that the authors use the Eclipse Integrated Development Environment (IDE) for creating Java classes. Eclipse is popular, highly functional, and available free of charge from the Eclipse foundation at http://www.eclipse.org. Detailed instructions for using the Processing library in Eclipse are available at http://www.processing.org. Also, the context of this white paper only allows for discussion of example Java code snippets. In the presented samples, many necessary blocks of code are omitted and some underlying complexities are masked by using a ShareConnection class written by the authors. The full code for the example class, along with the ShareConnection class, is available online at https://sites.google.com/site/stamb2012/.

STRUCTURE OF THE PApplet

The example Java class built to display the SAS data draws its functionality from the Processing library and the Java SQL package. This Java class will be referred to as the "painter" class. The painter class must extend the Processing PApplet class and import the Java packages java.sql and processing.core.papplet to have access to the shared data and the Processing graphics methods. All JDBC connections require a client driver Java Archive (jar) to be available in the Java project classpath. Connecting to a SAS/SHARE server requires at least the sas.intrnet.javatools.jar and sas.core.jar client driver jars. These jars may be found in your local SASHOME directory or downloaded from SAS. The binary Processing core.jar must be added to the painter class Java Build Path as well. The core.jar archive is included in the Processing download. Instructions for adding core.jar to the painter class build path in Eclipse are available on Processing’s website.

The painter class is required to have two methods, `setup()` and `draw()` that are called by the executable Processing core library once the applet is started. The `setup()` method is called only once upon starting the applet. In this example, `setup()` is used to set the size of the applet, set several Processing options, connect to the SAS/SHARE server, and query necessary information from the *sasshare.iris_draw* and *sashelp.iris* data sets. The `draw()` method is called a default rate of 30 times per second after `setup()` is called initially. It is used to write data and other objects to the screen or file rendering. Since `draw()` is called constantly during the execution of the applet, it is easy to create animation with the Processing apps. The basic form of the example painter class will follow the code snippet below:

```
package processing.sample;

import processing.core.PApplet;
import java.sql.*;

public class IrisPainter extends PApplet {
…
       public void setup() {
       …
       }

       public void draw() {
       …
       }
}
```

THE setup() METHOD

In the example painter class, the `setup()` method is used to set the size of the applet, set several Processing options, and connect to the SAS/SHARE server. The size of the PApplet display is set using the Processing `size(int, int)` method. The `size(int, int)` method takes two integer arguments, a horizontal and vertical pixel measure in that order. The `size(int, int)` method should generally be called before any other methods in `setup()`.Two additional Processing options are set in the example `setup()` method. The `smooth()` method invokes Processing's anti-aliasing capabilities, but can decrease performance. The `frameRate(int)` method changes the number of times `draw()` is called per second.

The connection to the SAS/SHARE server is also established in the `setup()` method. The SAS/SHARE connection requires four basic pieces information, passed to it as Java Strings: the address of the server, the port of the server, a username, and a password. Using SQL-like queries, pertinent information is retrieved from the *sasshare.iris_draw* and *sashelp.iris* data sets and stored into Java ArrayLists and Vectors, which are similar to standard arrays. Java Strings comprised of information from the *sashelp.iris* data set that are used for interactive selection of the three species groups are also created at this time. The Java `try/catch` statements are necessary for applet compilation. They also allow the applet to fail gracefully and display pertinent information in the case of a failed connection. Note that all necessary information is gathered from the data set only once at the beginning of execution in the `setup()` method, and connections are closed directly after data retrieval.

```
…
//setup method - called once prior to draw when applet is executed
public void setup() {

       size(xSize, ySize);//define screen size
       smooth();
       //anti-aliasing
       frameRate(frameRate); //set animation speed

       //gather information from database resources into strings and collections
       //calls to database made only once at beginning of applet execution
       try {
              shareConnection = ShareConnection.makeConnection("localhost", "8551",
              "username", "p@$$w0rd");

              //collect information regarding petal length in width into collections
              displayed
              stmtDraw = shareConnection.createStatement();
              rsDraw = stmtDraw.executeQuery("select species, pwidthx, plengthy,
              plengthxsize, pwidthysize from sasshare.iris_draw");

              while (rsDraw.next()) {

                     speciesList.add(rsDraw.getString("species"));
                     petalWidthXs.add(rsDraw.getFloat("pwidthx"));
                     petalLengthYs.add(rsDraw.getFloat("plengthy"));
                     petalLengths.add(rsDraw.getFloat("plengthxsize"));
```

```
                petalWidths.add(rsDraw.getFloat("pwidthysize"));

            }

            rsDraw.close();

            //gather information for Setosa species into string for use later
            stmtMouseSetosa = ShareConnection.createStatement();
            rsMouseSetosa = stmtMouseSetosa.executeQuery("select avg(PetalWidth) as
            avg_p_width, avg(PetalLength) as avg_p_length, avg(SepalWidth) as
            avg_s_width, avg(SepalLength) as avg_s_length from sashelp.iris where
            species = 'Setosa'");
            setosaText = generateMouseText(rsMouseSetosa);
            rsMouseSetosa.close();

            …

        }
        catch (Exception e) {//if database is unavailable, just close
            e.printStackTrace();
            System.exit(-1);
        }

}
…
```

THE draw() METHOD

The `draw()` method is called after the `setup()` method and is called multiple times per second, as determined by the `frameRate(int)` method throughout the duration of applet execution. It is typically used to set image-specific Processing options and to write data and other objects to the screen or rendering file. In the painter class, `draw()` is used as a central switch for the user interaction functionality and calls one of several methods to ensure the visualization appears with the correct selection at all times. Regardless of user data selection, the author-defined `drawFlowerBubble(String, int, int, int, int)` method draws each visual data attribute to the screen.

In `drawFlowerBubble(String, int, int, int, int)`, data is moved from the Java Collections initiated in `setup()` into Java Iterators, to allow the applet to cycle through and display data records. Color is assigned to each visible bubble representing a data record according to the value of the SAS `species` variable using the `fill(float, float, float, float)` method with three ordered RGB arguments followed by an opacity argument. The Processing `ellipse(float, float, float, float)` method sets the x- and y- coordinates, length, and width respectively for each bubble. Bubble x- and y- coordinates are set by the pre-scaled SAS `pWidthX` and `pLengthY` variables, and bubble size is set by the raw petal length and width SAS variables, `pWidthXSize` and `pLengthYSize`. The y-coordinate variable, `pLengthY`, is subtracted from the `ySize` variable within the `ellipse(int, int, int, int)` method call to ensure that the y-coordinate of the bubble corresponds to the intuitive notion of the y-axis increasing from the bottom left, as opposed to the Processing convention of the y-axis increasing from the top left. An intermediate method sets the background applet shade and greyscale shade of shape outlines prior to calling `drawFlowerBubble(String, int, int, int, int)`.

```
…
//central method for drawing, allows each group species to be drawn separately
public void drawFlowerBubble(String species, int r, int g, int b, int alpha) {

        StringBuffer s = new StringBuffer("");

        //pull collections into iterators
        Iterator<String> speciesIter = speciesList.iterator();
        Iterator<Float> petalWidthXsIter = petalWidthXs.iterator();
        Iterator<Float> petalLengthYsIter = petalLengthYs.iterator();
        Iterator<Float> petalLengthsIter = petalLengths.iterator();
        Iterator<Float> petalWidthsIter = petalWidths.iterator();


        //cycle through iterators and draw when s matches species from method call
        while(speciesIter.hasNext()) {

                s.append(speciesIter.next());

                if (species.equalsIgnoreCase(s.toString().trim())) {
                        fill(r, g, b, alpha); //color and opacity for Setosa species
                        ellipse(petalWidthXsIter.next(), ySize - petalLengthYsIter.next(),
                        petalLengthsIter.next(), petalWidthsIter.next()); //draw ellipse
                        at intuitive y value instead of screen y value
                }
                else {
                        petalWidthXsIter.next();
                        petalLengthYsIter.next();
                        petalLengthsIter.next();
                        petalWidthsIter.next();
                }
                s.setLength(0); //reset s to blank
        }
}
…
```

INCLUDING USER INTERACTION

One of the more exciting aspects of using Processing to display SAS data is the ease of adding user interaction features into visualization apps. It provides methods to easily register user input from either the mouse or keyboard, and the object-oriented (OO) nature of the Java and Processing environment allows the developer to craft fitting responses. In this example, the `keyPressed()` and `mousePressed()` methods are used to save a static image file and highlight a selected species.

The `keyPressed()` method is invoked whenever a user presses a key during the execution of a Processing applet and records the actual key pressed by the user. A simple logical test can be performed to determine this key, and the developer can design a suitable response. Here the *s* key is used to *save* a static frame from the animated visualization. When the user strikes the *s* key, the `keyPressed()` method calls the `save(String)` method directly. The `save(String)` method saves a file whose path, name and type are defined by the String argument. Processing supports TIFF, TARGA, JPEG, and PNG image formats.

The `mousePressed()` method is called whenever a mouse button is clicked. When Processing detects input from a mouse button, `mousePressed()` checks whether the click occurred within one of three specified species x- y-ranges or outside of all ranges. If a click is registered within a species range, the value of a global Java Enum `DrawMode` is set accordingly. `draw()` then recognizes the changed value of `DrawMode`, and alters the visualization's appearance to reflect user selection of a given species group. A selected group will be highlighted by decreasing the transparency of its bubbles' RGB colors, assigning all other species' bubbles a grey hue, outlining the group in bold red and displaying additional textual information about the species. In Figure 1 above, the Virginica species is highlighted. If the user clicks outside of the three species ranges, `mousePressed()` and `draw()` reset the applet to display no selected species groups.

```
…
//called whenever a key is pressed
public void keyPressed() {

        if (key == 's')
                save(outFileName); //save when "s" is pressed

}

//called whenever a mouse button is pressed
public void mousePressed() {

        //if a mouse click is within a certain range, change the draw mode, i.e. bring
        a certain species group into focus
        if((mouseX >=  mouseXRangeMinSetosa && mouseX <= mouseXRangeMaxSetosa) &&
        (mouseY >= mouseYRangeMinSetosa && mouseY <= mouseYRangeMaxSetosa)) {
                drawMode = DrawMode.SETOSA;
        }
        else if((mouseX >=  mouseXRangeMinVersicolor && mouseX <=
        mouseXRangeMaxVersicolor) && (mouseY >= mouseYRangeMinVersicolor && mouseY <=
        mouseYRangeMaxVersicolor)) {
                drawMode = DrawMode.VERSICOLOR;
        }
        else if((mouseX >=  mouseXRangeMinVirginica && mouseX <=
        mouseXRangeMaxVirginica) && (mouseY >= mouseYRangeMinVirginica && mouseY <=
        mouseYRangeMaxVirginica)) {
                drawMode = DrawMode.VIRGINICA;
        }
        else
                drawMode = DrawMode.NORMAL;
}
…
```

Processing provides the additional methods `keyReleased()` and `keyTyped()` to register, process and respond to keyboard input. Further methods and functionality to record mouse input are available as well. Processing can easily distinguish between left, middle, and right mouse clicks. The library also allows for mouse drags, enabling data-generated objects to be moved on-screen by the user.

SAMPLE VISUALIZATIONS

The authors have developed several visualizations for use in analysis and to showcase the capabilities of the presented technique. Further examples with SAS and Java code are provided at https://sites.google.com/site/stamb2012/ for interested readers.

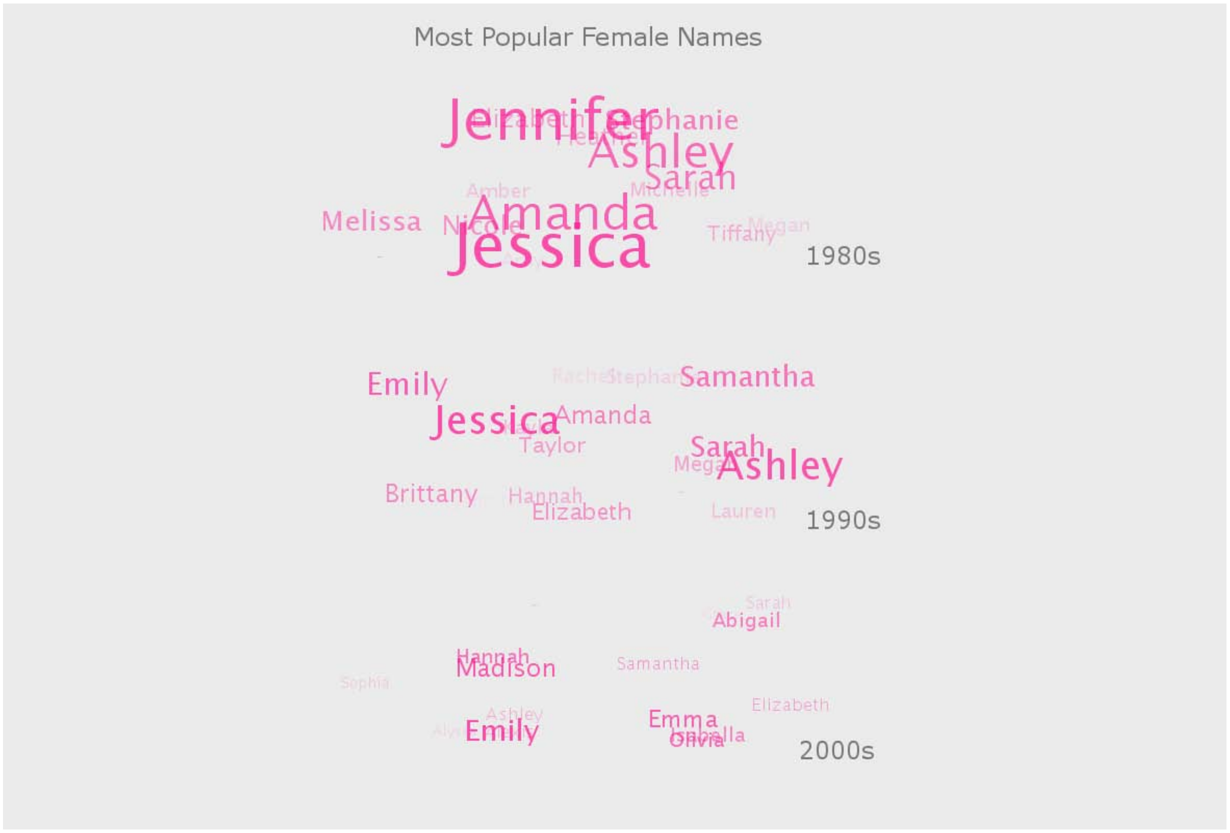

Figure 2. Most popular female names at birth from 1980 to 2000. Size and opacity are proportional to popularity.

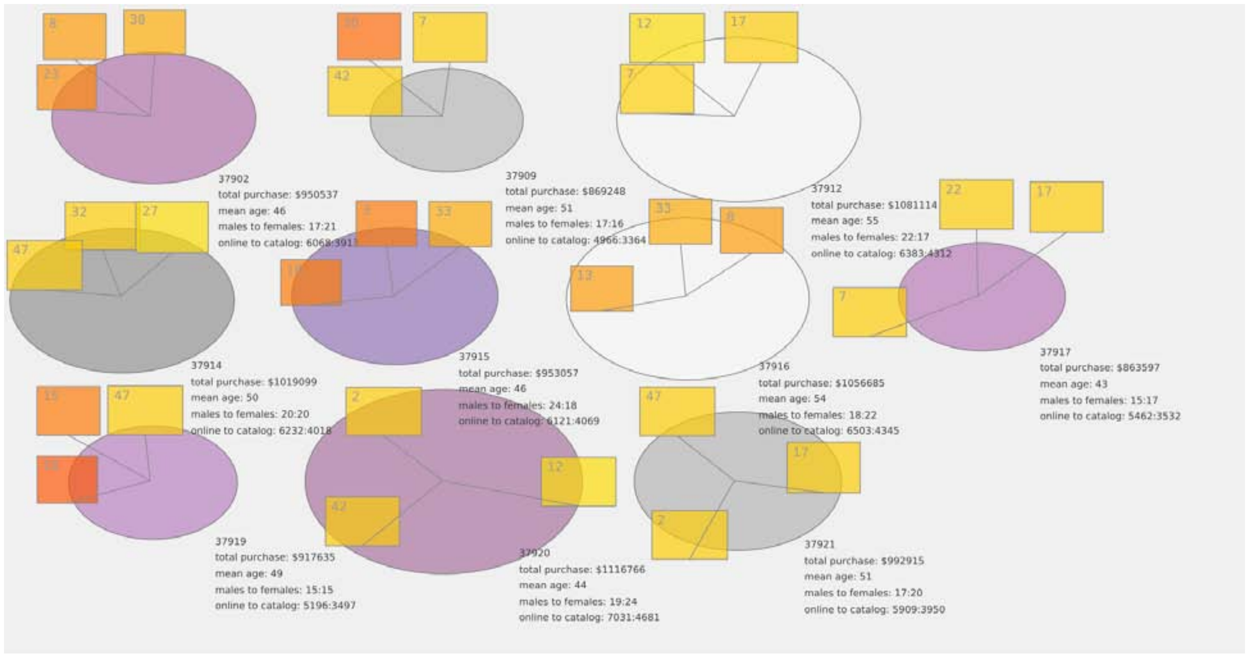

Figure 3. Approximately 100,000 simulated transactional sales records compiled into a bubble map by customer ZIP code, with ellipses representing the aggregate revenue, proportion of online sales, predominate gender, and average age of customers. Rectangular data attributes represent the aggregate sales, proportion of online sales, and profit contribution of the top three selling products in each ZIP code.

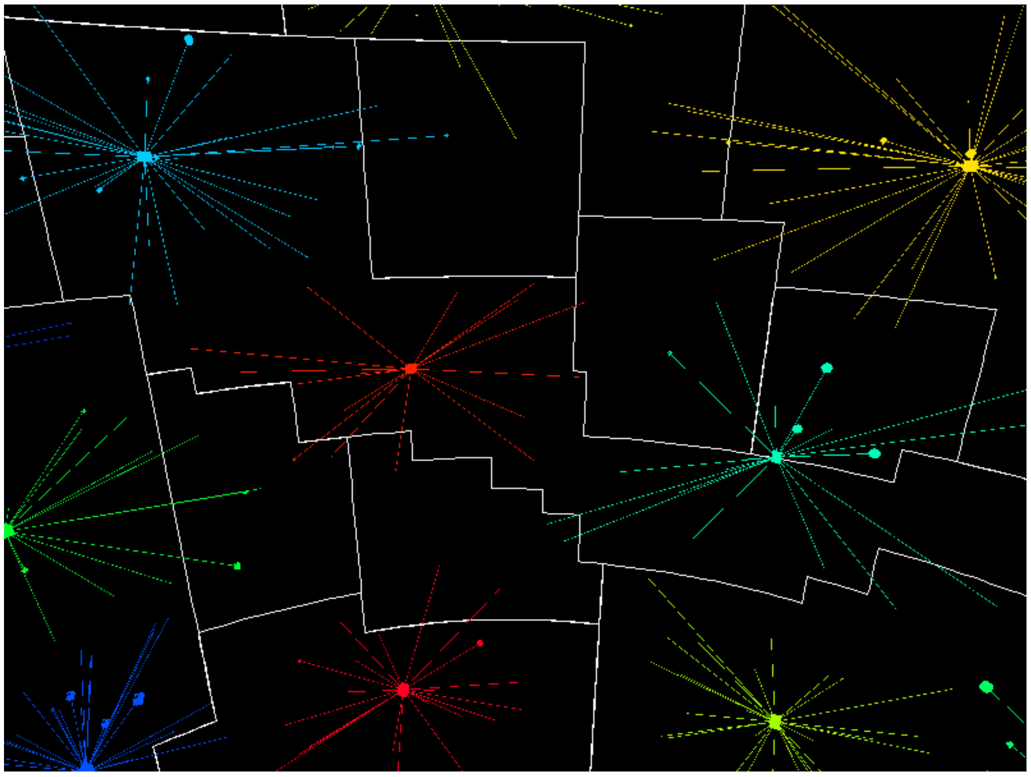

Figure 4. Output from a k-means clustering of Southern Hemisphere constellations. Cluster centroids are drawn as large colored cubes, with colored lines drawn to each clustered star. k-means clustering was implemented using `proc fastclus` in SAS 9.2.

CONCLUSION

The presented example utilizes a small, sorted and well understood data set. While the *iris* data provides a convenient platform from which to convey the idea of leveraging the complementary strengths of SAS and Java to create an interactive infographic app, the true benefits of integrating Processing with SAS 9.2 are realized when confronting large, noisy data sets. When combined with judicious consideration of a given visualization's audience and purpose, the customizability and scalability of this technique allow it to be applied in circumstances that push the limits of pre-packaged software.

REFERENCES

- Wikimedia Foundation, Inc. "List of information graphics software." Wikipedia. February 15, 2012. Available at http://en.wikipedia.org/wiki/List_of_information_graphics_software.
- Wikimedia Foundation, Inc. "Iris flower data set." Wikipedia. February 26, 2012. Available at http://en.wikipedia.org/wiki/Iris_flower_data_set.
- Healey, Christopher. "Introduction to Graphs, Maps, and Visualization." July 15, 2011. Available at http://www.csc.ncsu.edu/faculty/healey/iaa-11/IAA-Graphs-Maps.pdf.
- Pirrelo, Chuck. "Effective Visualization Techniques for Data Discovery and Analysis." *SAS Global Forum* 2010. Cary, NC: SAS Institute. Available at http://support.sas.com/resources/papers/proceedings10/235-2010.pdf.
- Fry, Ben and Reas, Casey et al. "Processing". Processing. September 1, 2011. Available at http://processing.org.

ACKNOWLEDGMENTS

- Ryan Snyder for continuing to hammer out the technical details of this technique, and spearheading development of the website.
- IAA Faculty and Staff for encouraging the authors to pursue this paper and providing advice, resources and proofreading.
- Associate Professor Christopher Healey, NCSU Department of Computer Science, for conducting several workshops on visual perception and visualizations.
- Professor Emeritus Henry E. Schaffer, NCSU Department of Genetics, instructor for the IAA courses AA 500 and AA 501, for providing the simulated transactional sales data used to build Figures 3.

RECOMMENDED READING

- Snyder, Ryan and Hall, Patrick. "A Guide for Connecting Java to SAS® Data Sets." *SAS Global Forum* 2012. Cary, NC: SAS Institute.
- Snyder, Ryan and Hall, Patrick. "Data Visualization Using SAS® Processing." March 7, 2012. Available at https://sites.google.com/site/stamb2012/.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Patrick Hall
jpatrickhall@gmail.com
http://www.linkedin.com/in/jpatrickhall

Ryan Snyder
ryancsnyder@gmail.com
http://www.linkedin.com/in/ryancsnyder

Institute for Advanced Analytics
920 Main Campus Drive, Suite 530
Raleigh, NC 27606
http://analytics.ncsu.edu/