

## What Do You Get When You Combine a Marine Biologist, a Video Game, and JMP?

The idea for this topic began with Dr. Anderson Mayfield's [discussion on the JMP User Community about a visualization of coral reef health](#). Dr. Mayfield is a marine biologist working as an Assistant Scientist at NOAA and the University of Miami. He studies reef corals and [presented in the Earth Day special version of JMP On Air](#). He is also presenting in this Discovery Summit on *Predicting the Fate of Reef Corals*.

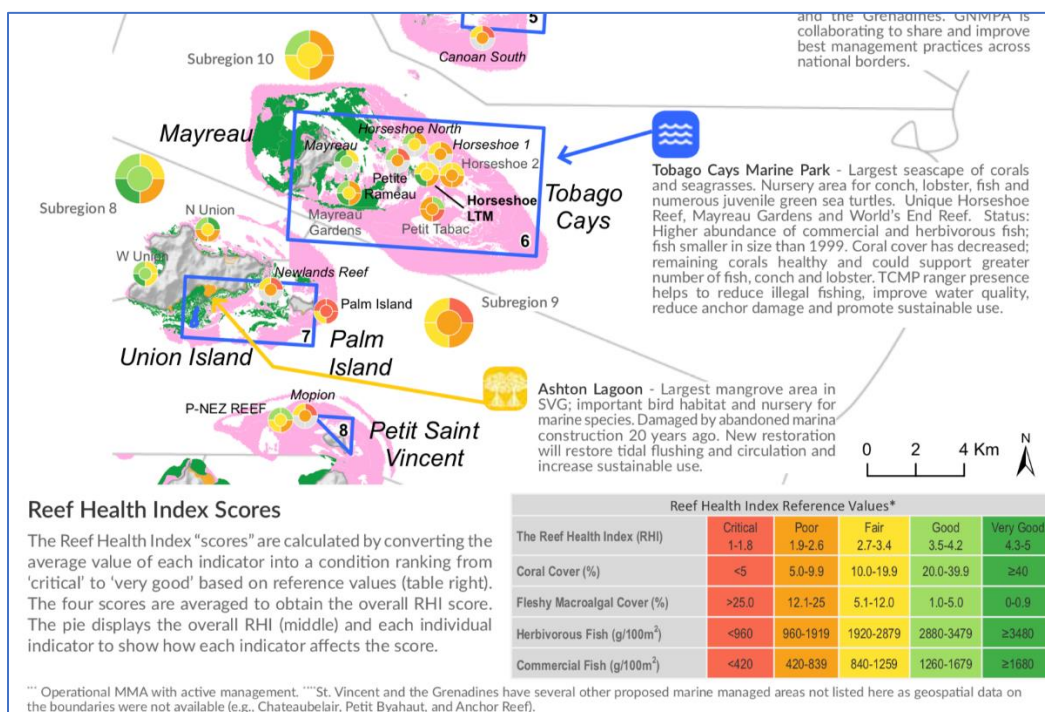


Figure 1 Reef Health Graphic

The circular graphics reminded me of the health monitors or player asset displays you see in video games hovering over the characters' heads as they move about the scene. Having some experience in video game engine development, I thought I could combine that experience with my knowledge of JMP to find a solution or two. This paper covers these solutions as well as a personal project where I combined JMP, web development, and another game engine technique to create a web-based 3D Scatterplot.

### Ways to build geo located health monitors in JMP

We essentially want to be able to pack multiple variables into a small geo-referenced space.

Solutions to this were discussed in the [JMP User Community post](#) and published to JMP Public. Click on images below to navigate to the specific JMP Public post.

- Using a custom graphic script:

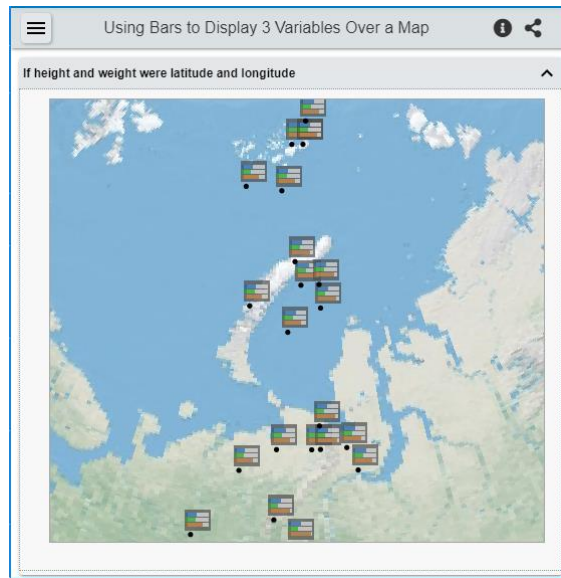


Figure 2 Custom graphics script example on JMP Public

- Using custom maps:

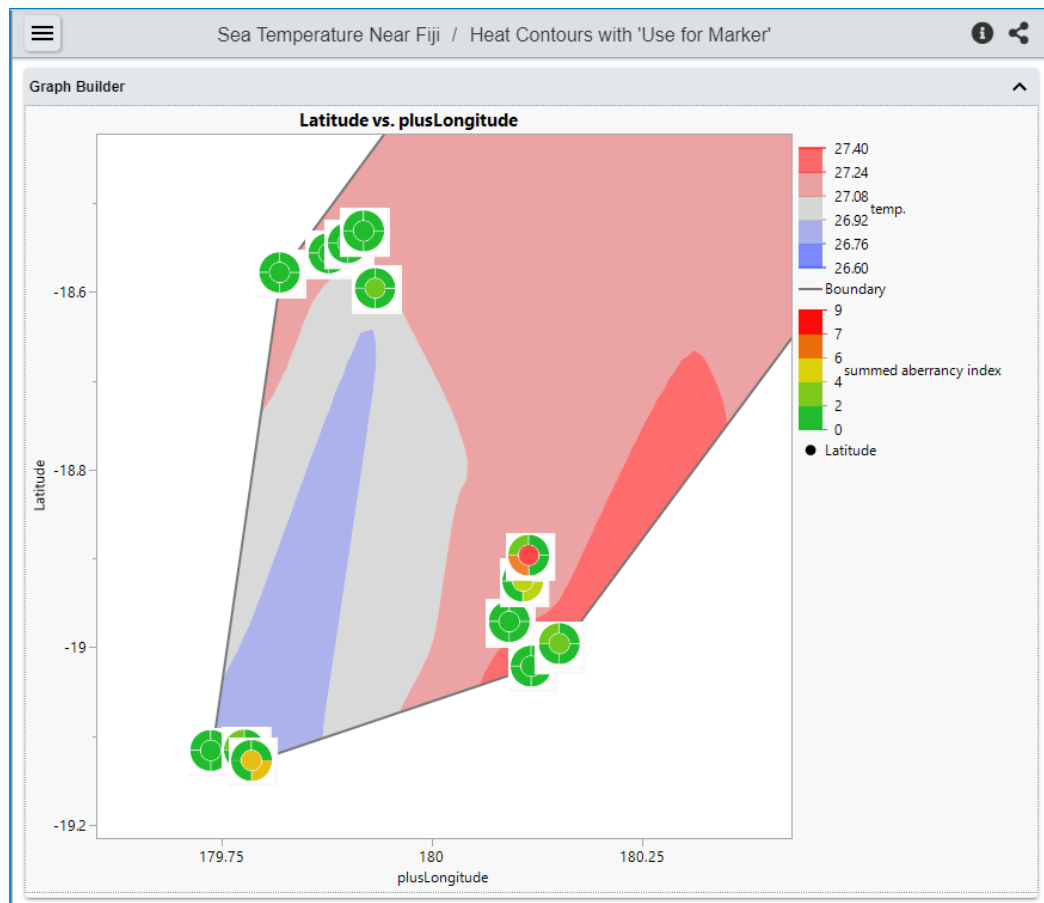


Figure 3 Custom map over contour on JMP Public

To illustrate these techniques, I built a completely fictitious data set of a small school of fish racing from the Great Barrier Reef to Sidney Australia. In the spirit of the Big Class sample data set, I called this data set *SmallSchool.jmp*.

		name	speed	strength	hunger	health	lat	long
1		Nemo	16	9	15	13.3	-24.5	154.1
2		Dory	9	9	15	11.0	-25.5	155.1
3		Marlin	11	9	15	11.7	-27.1	154.1
4		Gill	15	6	12	11.0	-27.5	155.7
5		Kathy	13	6	12	10.3	-29.2	154.5
6		Squirt	11	6	12	9.7	-29.6	156.1
7		Pearl	7	12	9	9.3	-30.5	154.1
8		Sheldon	9	12	9	10.0	-31	156.1
9		Crush	11	12	9	10.7	-26.5	156.5
10		Bubbles	13	12	9	11.3	-33.5	155.3
11		Jacques	15	12	6	11.0	-33.1	154.1
12		Deb	13	6	6	8.3	-31.5	155.1
13		Chum	11	6	6	7.7	-32	153.1
14		Bruce	15	6	16	12.3	-33.6	152.1

Figure 4 *SmallSchool.jmp*

The **Race** script in the table is a bivariate plot of **lat** by **long** to position the rows on the map.

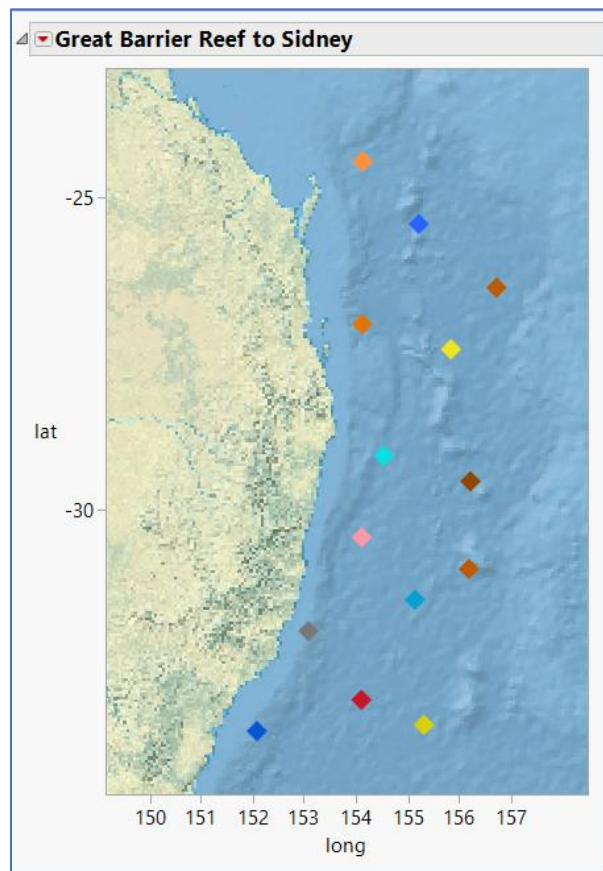


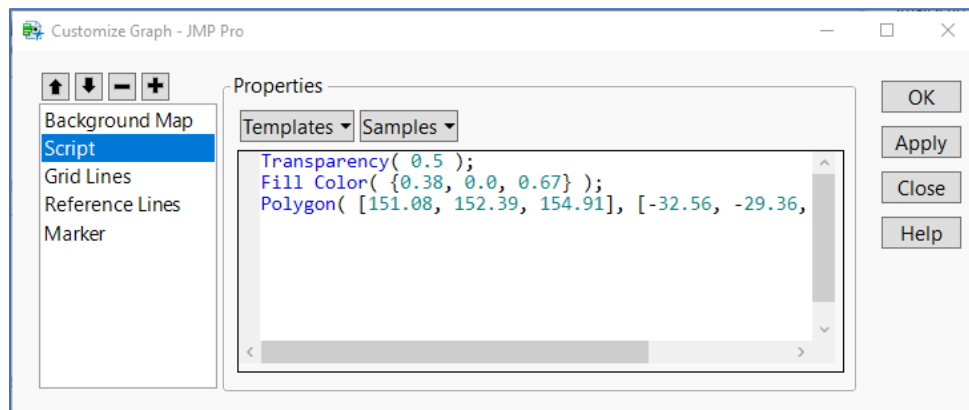
Figure 5 Bivariate plot

This data set and any other files mentioned in this paper are provided in the same JMP User Community post as this paper so you can use the steps provided below to follow along.

## Custom Graphics Scripts

A graphics script can be added to any JMP graph. To see an example with the bivariate plot created by the **Race** script in *SmallSchool.jmp*,

1. Right-click and choose **Customize** from the context menu
2. Press the **+** button in the **Customize Graph** dialog shown below
3. Under the **Samples** drop down, choose **Polygon**
4. The script is editable. Set the second value (the green component) in **Fill Color** to 0.0



5. Figure 6 Customize Graph dialog

6. Click **Apply**. This will draw a semi-transparent triangle in the plot. The vertices are randomly generated, so you may end up with a sliver of a triangle as shown below:

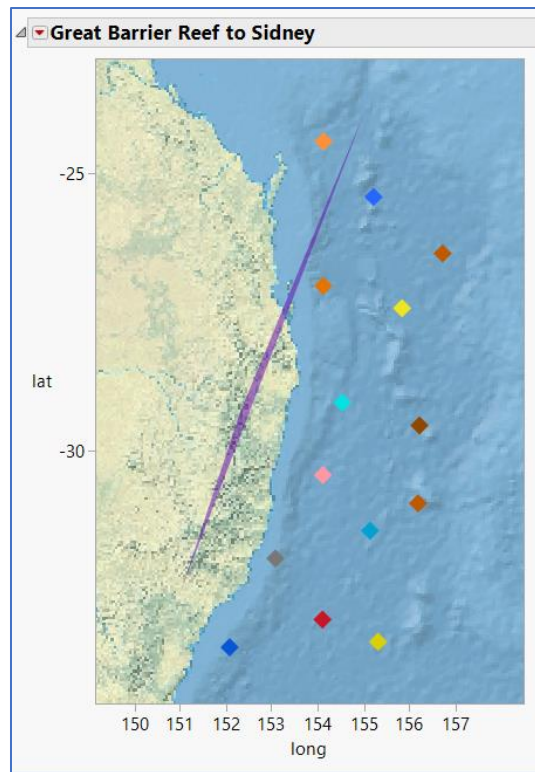


Figure 7 Sample Graphics Script

7. Click **OK**

Save the script to the **Script Window** to show where the **Add Graphics Script** is added.

1. Red triangle button > **Save Script > To Script Window**
2. Look for **Add Graphics Script**

```

27 Dispatch(
28     {},
29     "Bivar Plot",
30     FrameBox,
31     {Frame Size( 366, 551 ), Background Map( Images( "Detailed Earth" ) ),
32     Marker Size( 6 ), Add Graphics Script(
33         2,
34         Description( "" ),
35         Transparency( 0.5 );
36         Fill Color( {0.38, 0.0, 0.67} );
37         Polygon( [151.08, 152.39, 154.91], [-32.56, -29.36, -23.95] );
38     ), Grid Line Order( 3 ), Reference Line Order( 4 ),

```

Figure 8 Add Graphics Script function call

We want to replace this script with drawing commands for each point.

1. Run the **Race with Health Metric Bars** script to see the result we're after

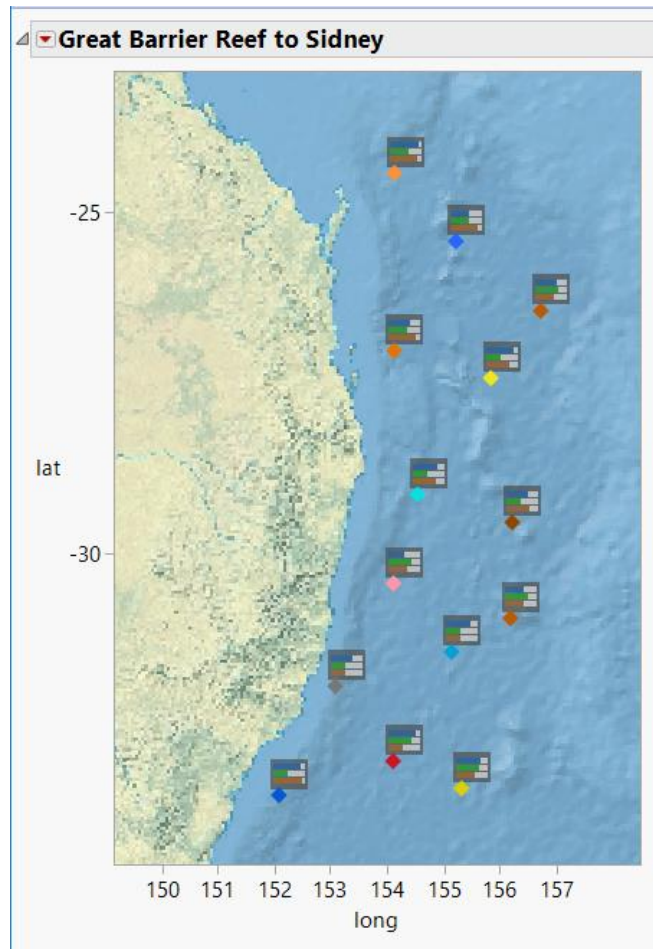


Figure 9 Health monitor bars

To view the script, right-click on **Run Race with Health Metric Bars** and choose **Edit**.

Here is the **Add Graphics Script** code:

```

Add Graphics Script(
  4,
  Description( "Bars" ),
  Transparency( 0.7 );
  For Each Row(
    // Draw health monitor.
    Pixel Origin( :long, :lat );
    // Draw monitor background.
    Pen Color( BACKGROUND_COLOR );
    Pen Size( 21 );
    yOffset = -15;
    Pixel Move To( -6, yOffset );
    Pixel Line To( 20, yOffset );
    // Health monitor bars.
    Pen Size( 4 );
    drawBar( -10, :speed,      COLOR1 );
    drawBar( -15, :strength,   COLOR2 );
    drawBar( -20, :hunger,     COLOR3 );
  );
)

```

Notice that it uses some constants and a function to draw each bar. These are defined at the top of the script:

```

// Constants
LIGHTNESS = 0.4;
SATURATION = 0.7;
// Bar colors defined using Hue, Lightness, and Saturation
// values between 0 and 1.
// If you think of the color wheel as a clock,
// the bar colors' hues will be at 1, 4 and 7 O'Clock.
COLOR1 = HLS Color( 1 / 12, LIGHTNESS, SATURATION );
COLOR2 = HLS Color( 4 / 12, LIGHTNESS, SATURATION );
COLOR3 = HLS Color( 7 / 12, LIGHTNESS, SATURATION );
// Bar colors defined using Red, Green, and Blue values
// between 0 and 1.
BACKGROUND_COLOR = RGB Color( 0.3, 0.3, 0.3 );
BAR_BACKGROUND_COLOR = RGB Color( 0.9, 0.9, 0.9 );
// Length and offset are in pixels.
BAR_BACKGROUND_LENGTH = 18;
BAR_X_OFFSET = -4;

// drawBar function draws a horizontal bar at a
// given height y in pixels from pixel origin,
// a given length and color. It also and fills
// in the background in the color specified by
// BAR_BACKGROUND_COLOR color upto the length
// specified by BAR_BACKGROUND_LENGTH.

drawBar = Function( { y, length, color },
    Pen Color( color );
    Pixel Move To( BAR_X_OFFSET, y );
    Pixel Line To( length, y );
    Pen Color( BAR_BACKGROUND_COLOR );
    //Pixel Move To( x, y );
    Pixel Line To( BAR_BACKGROUND_LENGTH, y );

```

With some effort and study of the [graphics functions available in JSL](#), you can get the look you need.

## Building a custom map

[JMP map shape files](#) have a -XY (coordinates) file and a -Name file. For shapes that define states of a country, the -Name file would contain the name of each state. Custom map files are no different.

The -XY file defines the coordinates of the shapes and the -Name file provides the names of the shapes.

Here's a minimal custom map example defining two square shapes named **weight** and **height**.



Shape ID	Part	X	Y
1	1	0	15
2	1	5	15
3	1	5	10
4	1	0	10
5	2	0	10
6	2	5	10
7	2	5	5
8	2	0	5

Figure 10 Tiny-XY.jmp

Shape ID	Name
1	weight
2	height

Figure 11 Tiny-Name.jmp

The rows don't need to be labeled in *Tiny-XY.jmp* but are labeled in this case so that you can run the **Show Points** script to see the shapes that are defined. The **Show Points** script is also only present for illustration.

1. Open *Tiny-XY.jmp* and run the **Show Points** script. It's a **Fit Y By X** plot. The arrows will not be visible, they were added to emphasize that the shapes are constructed in a clockwise direction.

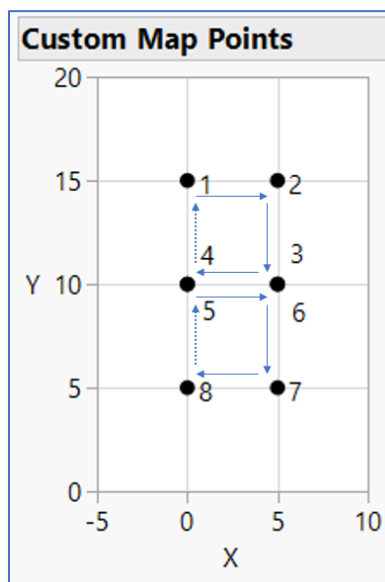


Figure 12 Tiny-XY points labeled

2. Open *Tiny-Name.jmp* and examine the column info for the **Name** column

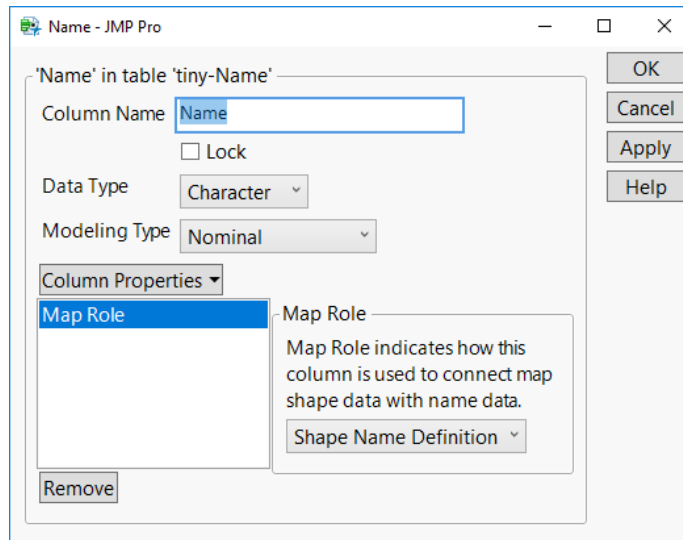


Figure 13 Tiny-Name.jmp Column Info

In the *-Name* file, the **Name** column should have a **Map Role** property with the **Map Role** set to **Shape Name Definition**.

**Shape name use** is not necessary for the *-XY* file, but it is necessary in the data set that uses the shapes.

To apply this custom map to *Big Class.jmp*, stack the **weight** and **height** columns.

1. **Tables > Stack ...**
2. Choose **height** and **weight**
3. Set **Output table name**. (*Big Class Stacked.jmp*)

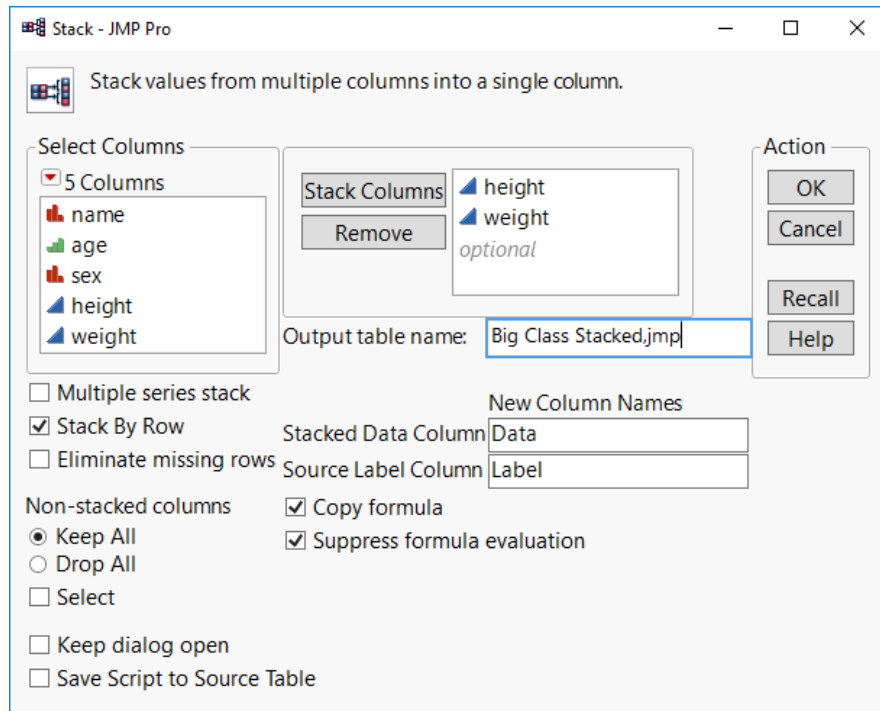


Figure 14 Stacking weight and height columns

4. Click **OK**.
5. Save the stacked column table **in the same directory as the map files**.

		name	age	sex	Label	Data
1		KATIE	12	F	height	59
2		KATIE	12	F	weight	95
3		LOUISE	12	F	height	61
4		LOUISE	12	F	weight	123
5		IANE	12	F	height	55

Figure 15 Big Class Stacked.jmp

The **height** and **weight** columns' data will be stacked in the **Data** column and there will be a column named **Label** that alternates between **height** and **weight**.

To link the **Data** to the shapes in the custom map files, we need to set the **Map Role** property on the **Label** column:

1. Right-click on the **Label** column and select **Column Properties > Map Role**
2. Choose **Shape Name Use**
3. Click the open table button to select *Tiny-Name.jmp*
4. Set **Shape definition column** to **Name**

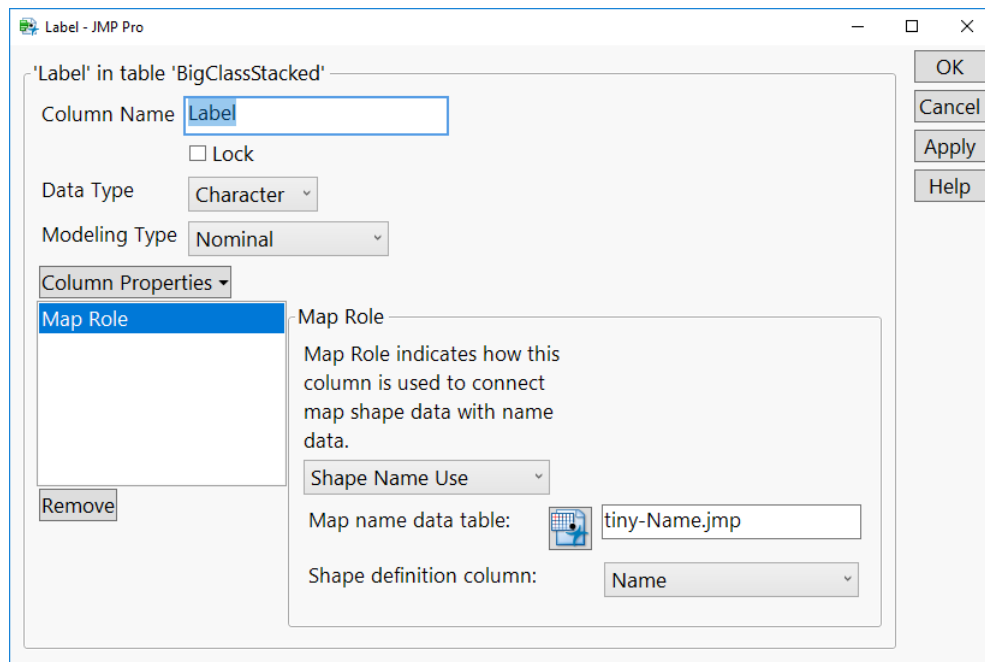


Figure 16 Label column's Shape Name Use Map Role

5. Click **OK**

You should see an asterisk (\*) next to the **Label** column in the **Columns** display (left).

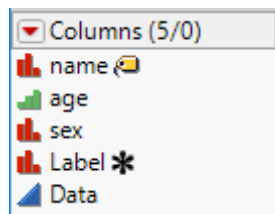


Figure 17 Columns display

To use the custom shape in a Graph Builder plot:

1. Start Graph Builder
2. Drag **Label** to the **Map Shape** role
3. Drag **Data** to the **Color** role

These red and blue box colors represent the mean of all the **weight** values(top) and **height** values(bottom).

This technique is best if the data values are on the same scale, which is not the case here, but this is just for illustration purposes.

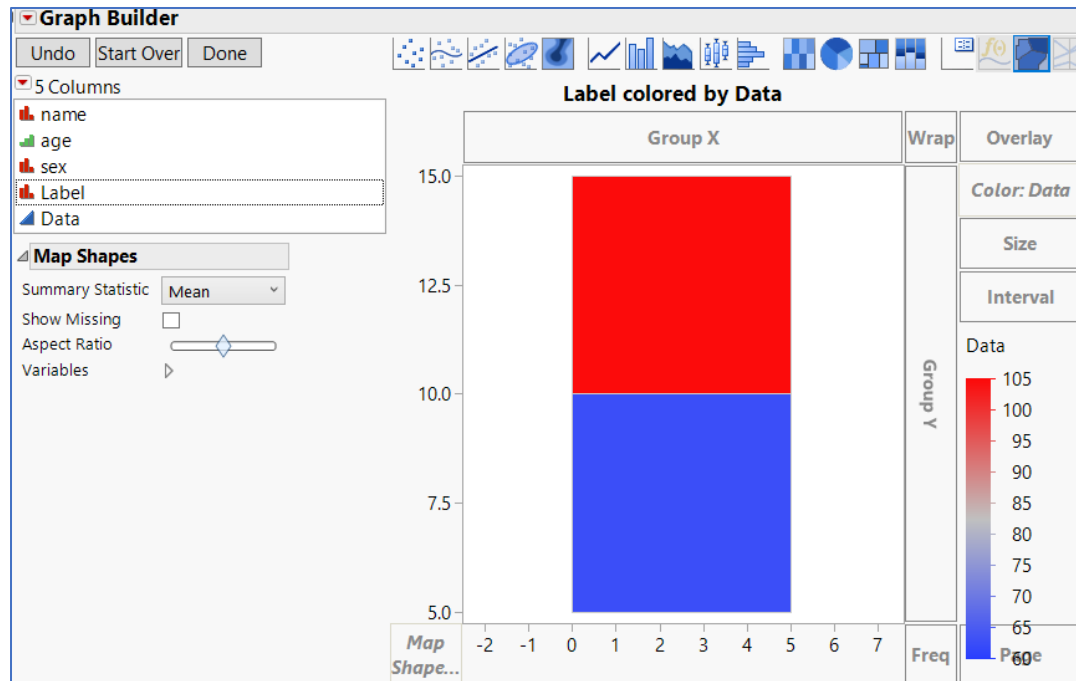


Figure 18 Using the custom map

To get one map for each student,

1. Drag **name** into the **Wrap** role.
2. Click **Done**

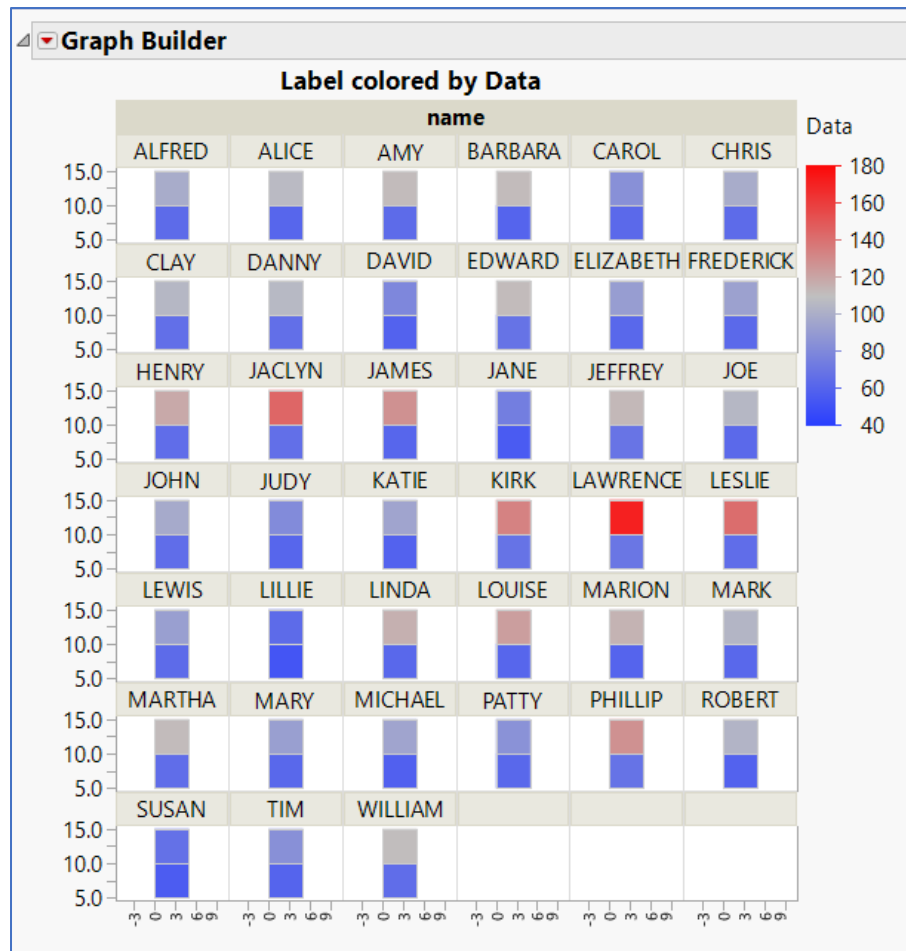


Figure 19 Tiny Custom Shapes applied to Big Class Stacked.jmp

For *Tiny-XY.jmp*, it was easy to manually enter the coordinates for the two shapes, but for a more complicated shape, like a shape that will allow us to display five variables, you can use the [Custom Map Creator add-in](#).

With the **Custom Map Creator**, you can load an image and trace the shapes you need.

If you don't already have a picture of the shapes you want, you could create a grid in JMP or any drawing application that supports it and save it as an image. Then load it into the **Custom Map Creator** and trace out the shapes you need. Here's an example of five shapes within a square.

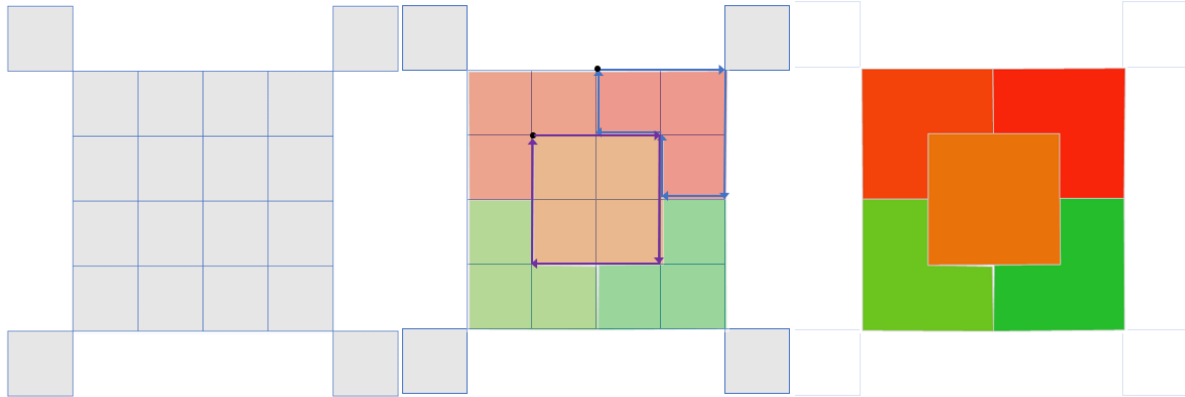


Figure 20 Grid, Tracing, and Shape test

The **Custom Map Creator** is great for organic shapes, but if you need to create points that line up perfectly or follow a perfectly circular arc, a script to generate the points is more suitable. This was the case for the complex pie shapes Dr. Mayfield was looking for.



Figure 21 Complex pie shape

One of the reasons we like to avoid circular shapes in data visualization is that it's harder for us to understand relative areas in circular shapes than in rectangular shapes, but in this case, all the outer shapes are the same size. The data are encoded in colors, not the size of the shapes.

Xan Gregg included a script with his solution to the JMP User Community post "[How can I make this polar plot in JMP?](#)". The script was flexible enough to create a ring with any number of wedges. I just needed to adjust it to add a center shape to create the script in *ComplexPieMaker.jsl*.

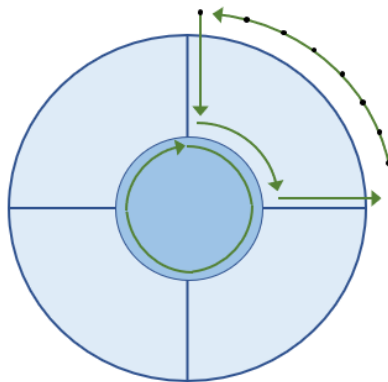


Figure 22 Complex pie drawing order

By changing the values of the variables in the script: **n shapes**; **isCentered**; **nc**; **inner radius**; and **outer radius**, you can generate the following shapes:

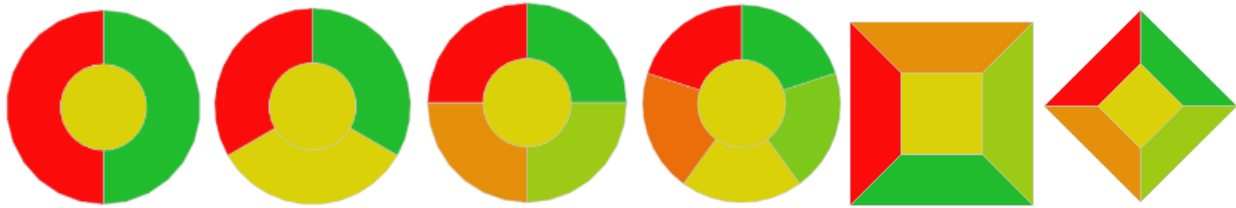


Figure 23 Shapes generated with ComplexPieMaker.jsl

As demonstrated with *Big Class*, to prepare the data for displaying within custom shapes, each area of the complex shape needs to be identified by a shape name. If you stack the columns with the individual data, the column names can be used as identifiers. If you do this, the shape file will also need to use the column names for each shape. This is OK if you are **only** going to use these columns whenever you want to use this shape file.

If you're likely to reuse this technique for different column names, you'll need to use more generic names in your shape file, like **shape1**, **shape2**, ... **shapeN** and add a column in your data set that maps the data name to the generic shape name. The Complex Pie Maker's shape names are **wedge1**, **wedge2**, ... **wedge<n shapes - 1>**, and **center**.

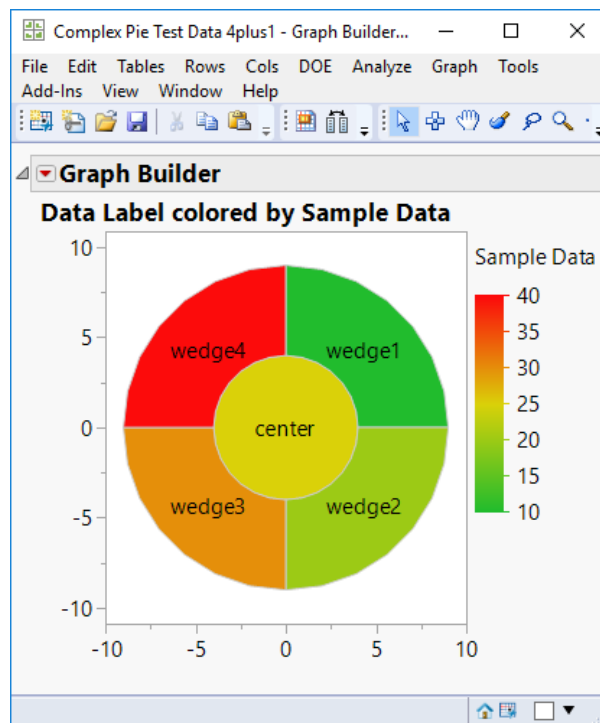


Figure 24 Using ComplexPie 4plus1

Another approach is to use a virtual join from your data set to a table that maps data column names to generic shape names. This approach will be used below.



*SmallSchool.jmp* contains three health measurements (**strength**, **speed**, **hunger**) and an overall **health** metric. The automatically generated shape files needed to display these four variables are *ComplexPie 3plus1-Name.jmp* and *ComplexPie 3plus1-XY.jmp*.

To use this custom map shapes pair with *SmallSchool.jmp*,

1. Stack **speed**, **strength**, **hunger** and **health** and output to *SmallSchoolStacked.jmp*.

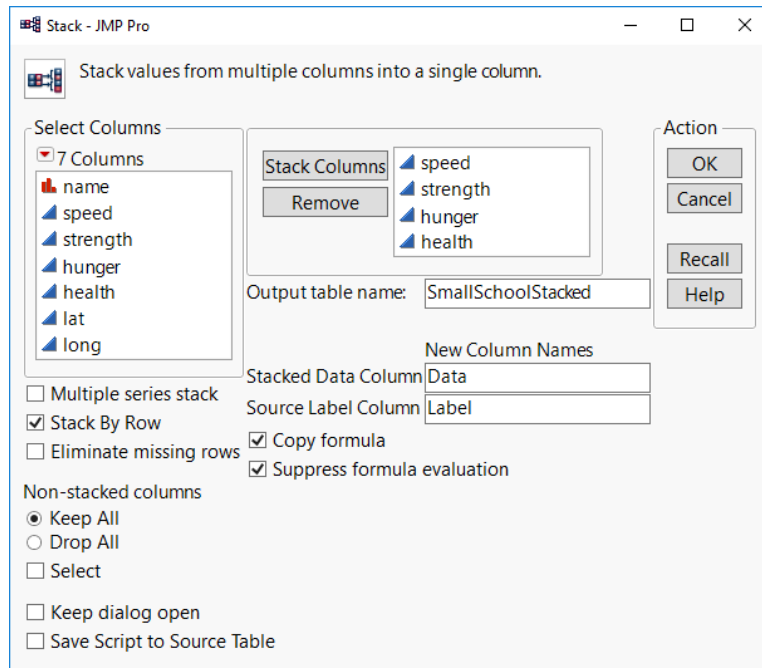
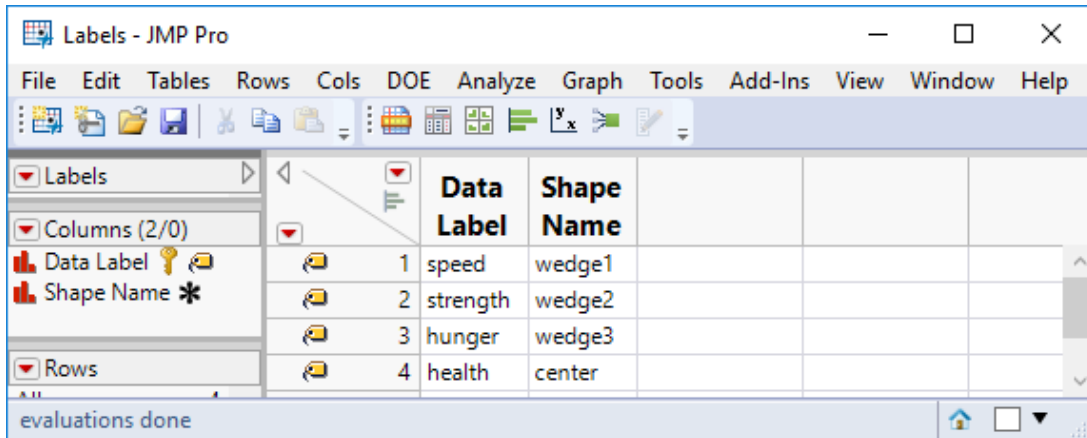


Figure 25 Stacking health measures columns


2. Create a table of column names (**Data Label**) to generic shape names (**Shape Name**)
3. Right-click on **Data Label** > **Link ID** (for use as virtual joined column)

4. Add a **Map Role** column property to **Shape Name**
  - a. **Shape Name Use**
  - b. Use the table selector to select *ComplexPie 3plus1-Name.jmp*
  - c. Set Shape definition column to **Label**
  - d. Click **OK**
5. Save as *Labels.jmp*



	Data Label	Shape Name
1	speed	wedge1
2	strength	wedge2
3	hunger	wedge3
4	health	center

Figure 26 Labels.jmp

6. In *SmallSchoolStacked.jmp*, right-click on the **Label** column and set **Link Reference** to *Labels.jmp*
7. Build a graph for each fish:
  - a. Start Graph Builder
  - b. Drag **Shape Name[Label]** into the **Map Shape** role
  - c. Drag **Data** into the **Color** role
  - d. Notice that the hover label shows **Shape Name[Label]**
  - e. **Optional step:** Suppress **Shape Name[Label]** from the hover label
    - i. Right-click > **Hover Label** > **Hover Label Editor...**
    - ii. Click on **Gridlet**
    - iii. Under **Delete** tab, set **Type** = **Column**
    - iv. **Target** > **Shape Name[Label]**
    - v. Click **Add**
    - vi. Click **OK**
    - vii. Hover over the graph and notice how the hover label no longer shows **Shape Name[Label]**
  - f. Right-click on the legend and choose **Gradient ...** to change gradient as desired
    - i. Choose the Green Yellow Red gradient 
    - ii. **Reverse Colors** so that better health is green
    - iii. To get one of these per fish, drag the **name** column into the **Wrap** role

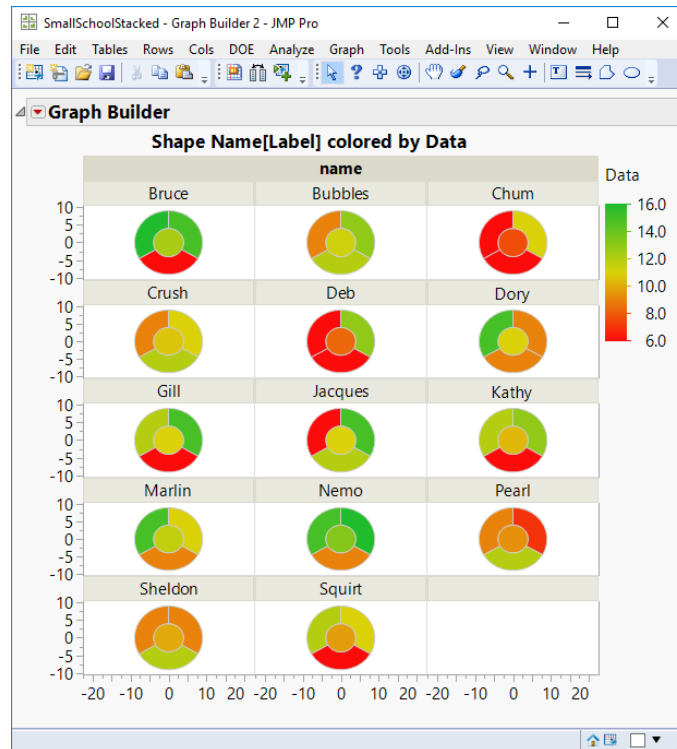


Figure 27 Health metrics per fish

This graph may be useful as is, but we'll need to do more with this graph to be able to place the individual graphics on a map.

**Make into Data Table** will build a table of images we can use to add to hover labels or use for markers.

1. Squeeze the graph horizontally or stretch vertically to get square cells so the graphics fill the cells.

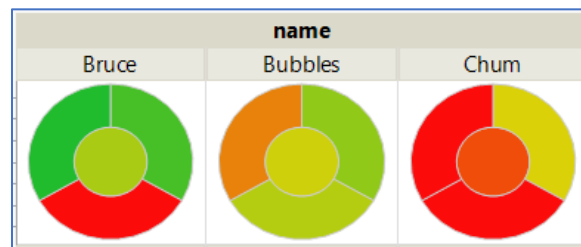


Figure 28 Square cells

2. From the red triangle menu, choose **Make into Data Table**. This will create a table with **name** and **Graph** columns shown below.
3. Right click on the **name** column and set **Link ID** (for use as virtual joined column)
4. Save as *HealthImages.jmp*

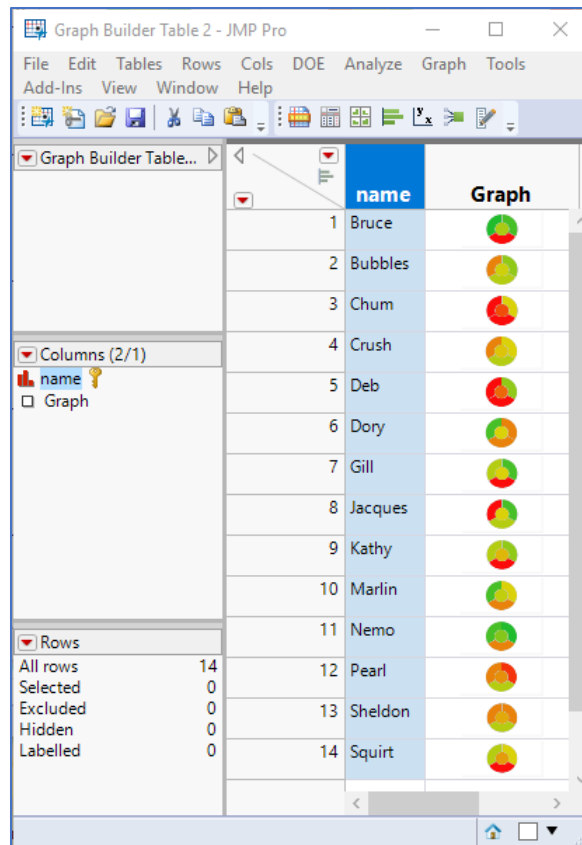


Figure 29 HealthImages.jmp

To add these images to hover labels for points on the map,

1. Open *SmallShool.jmp*. (not the stacked version)
2. Right-click on the **name** column and set **Link Reference** to *HealthImages.jmp*
3. Notice **Graph[name]** added to the **Columns** list:

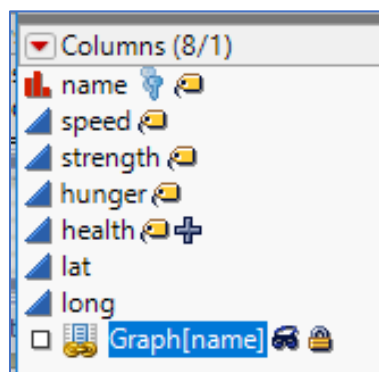


Figure 30 Virtually joined health images

4. Label this column – right-click and choose **Label/Unlabel**

To Test this:

1. Run the **Race** script
2. Hover over any fish
3. Need some to stay in place? Pin the hover label with the pin in the top right of the hover label

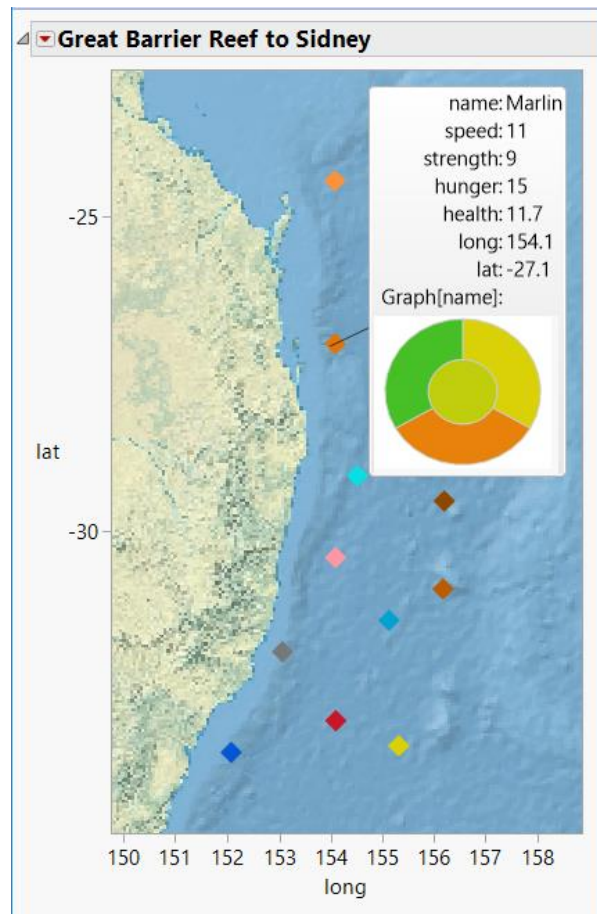


Figure 31 Pinned hover label with image

That's OK, but we want to see health indicators for all the fish at the same time. That's easily done with **Use for Marker**, but we have a minor issue with our health images in *HealthImages.jmp*. There's a white border around the images that we don't want. To remove it we can set the alpha (opacity) to 0 for gray and white pixels using this script:

```
dt = CurrentDataTable();
For Each Row(
  {r, g, b} = dt:Graph[] << getpixels( "rgb" );
  a = !AND(r == g, g == b);
  dt:Graph[] << setpixels( "rgba", {r, g, b, a} );
);
```

If our script only modified the alpha for white pixels, we would have been left with a gray border. Fortunately, we're using a color gradient that doesn't include any shades of gray. Otherwise, whole

sections would become transparent. This script is added to the provided *HealthImages.jmp* file with the name **Make Transparent**.

To set **Use for Marker**, right-click on **Graph[name]** in the **Columns** list and choose **Use for Marker**.

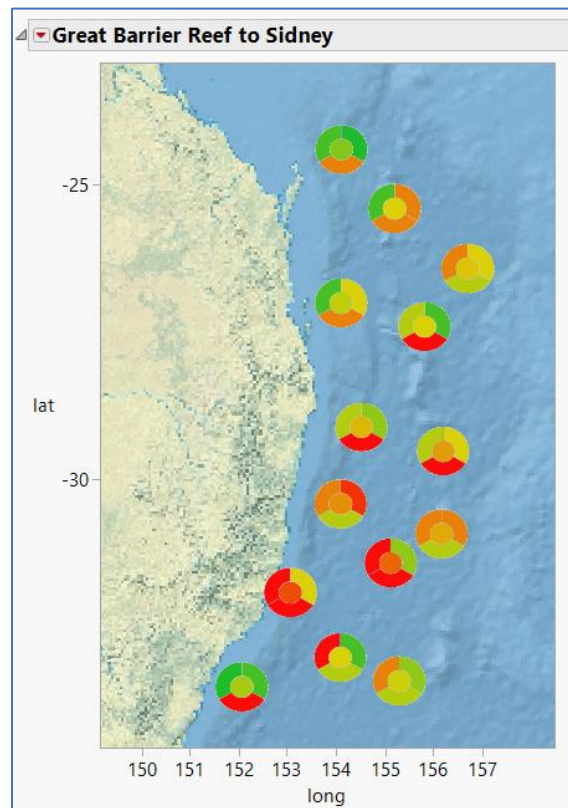


Figure 32 Use for marker

When you zoom in or out, you can adjust the image sizes by adjusting the marker size (right-click > **Graph > Marker Size**). When the images are small, you can use the hover label to see the underlying data and a larger health monitor image:

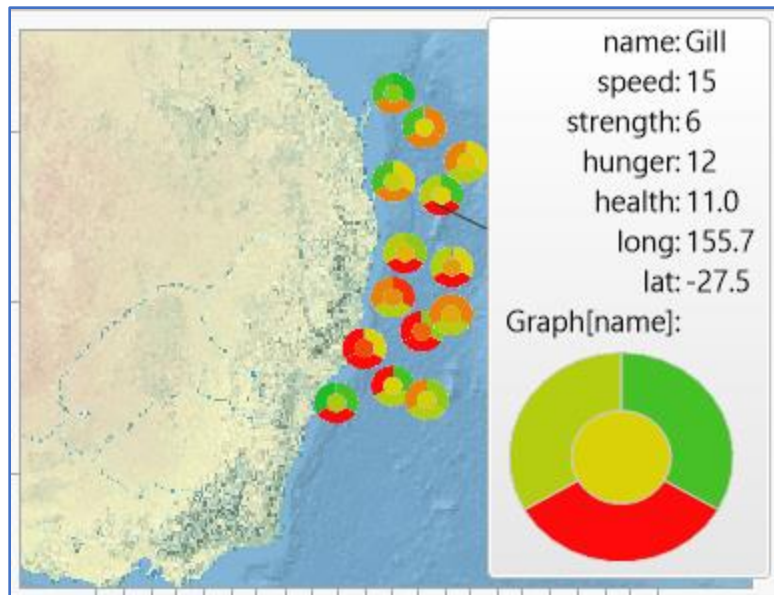


Figure 33 Smaller marker size and image in hover label

In the [JMP Public graph example with coral health monitors](#), there's also a contour showing ocean temperatures. Here's how that can be done with *SmallSchool* if we add some ocean temperature measures.

1. Add temperature data to the *SmallSchool.jmp* (*SmallSchoolOceanTemps.jmp*)
  - a. I used fictitious, possibly unrealistic data for the sake of demonstration.
2. We don't have health indicator images for these data, so mark these rows hidden
3. Start Graph Builder
4. Set "Detailed Earth" as the background map.
5. Drag **lat** into the **Y** role and **long** into the **X** role
6. Click on the **Points** and **Smoother** element buttons to remove them
7. Click on the **Contour** element to add the contour to the graph
8. Drag **temp** to the **Color** role
9. Adjust **Contour** settings:

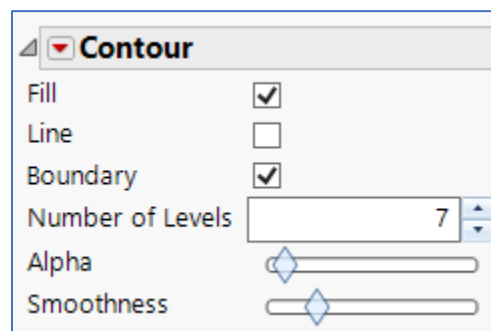


Figure 34 Contour settings

10. Right-click on the gradient legend and set **Transparency** to 0.6
11. Shift-click the **Points** Element button to add **Points** back on top of the contour

12. To add the **health** legend, drag the **health** column to left side of the color role

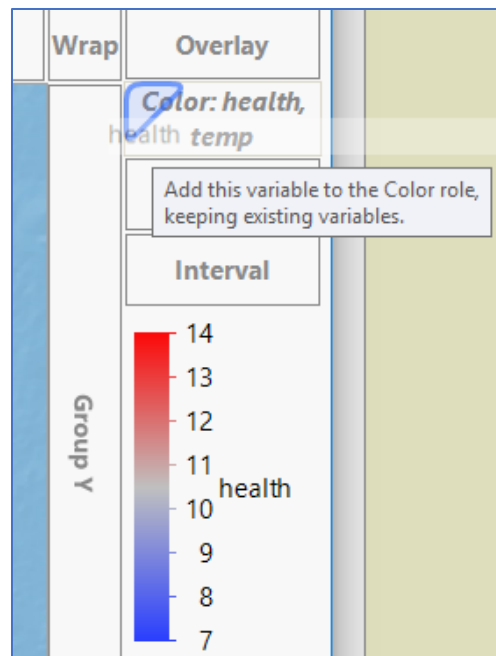


Figure 35 Adding a color role for points

13. We don't want the health color role to affect the contour, so expand the **Variables** section under the **Contour** element settings and uncheck **Color health**:

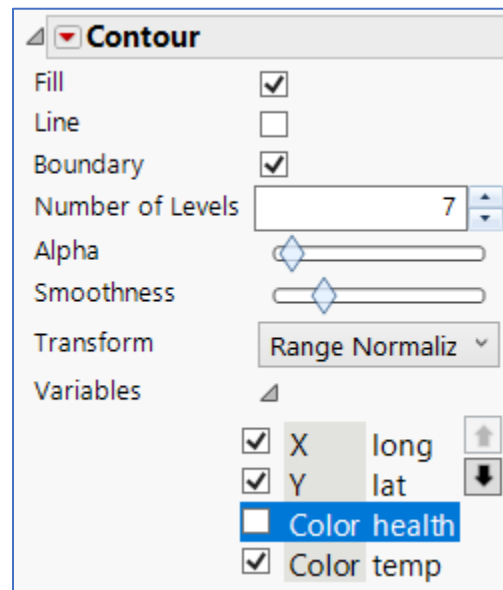



Figure 36 Disable health color role for contour

14. Edit the **health** gradient legend
  - a. Right-click on the legend and choose **Gradient**
  - b. Pick the **Green Yellow Red** color theme 
  - c. Set **Number Of Labels** = 4, **Minimum** = 5, and **Maximum** = 20



- d. Check **Reverse Colors**
- e. Click **OK**

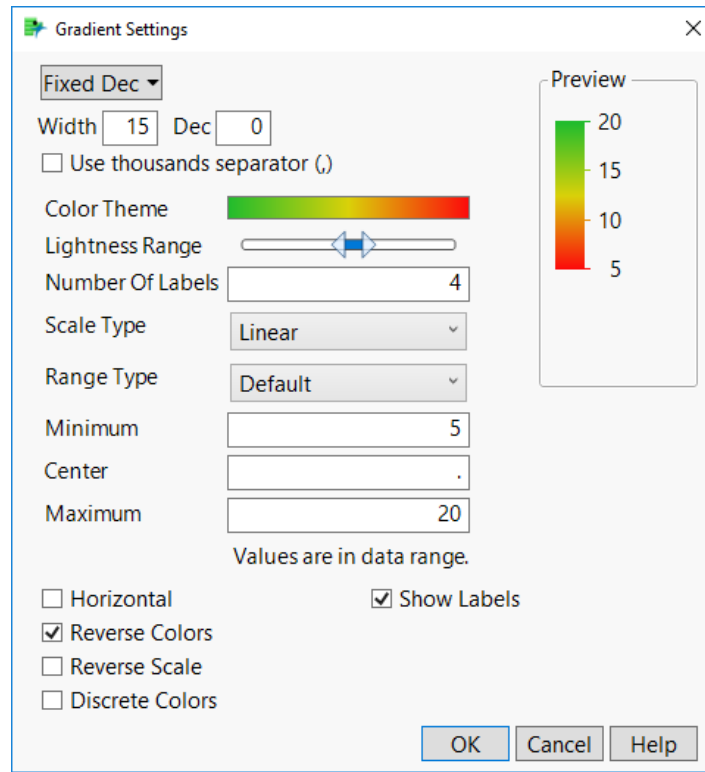


Figure 37 Gradient settings

15. Bring the health gradient legend to the top and remove **lat** and **Boundary** from the legend
  - a. From the red triangle menu choose **Legend Settings...**
  - b. Select **health** and use the up arrow to move it to the top
  - c. Uncheck **lat** and **Boundary**

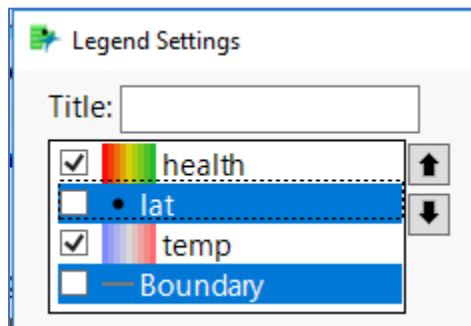


Figure 38 Legend settings

- d. Click **OK**
16. Click Graph Builder's **Done** button
17. Change Title from "Graph Builder" to "Health and Sea Temperature" (Double click & type this title)
18. Remove title: "lat vs. long" (Double click & Delete)

19. Zoom in as desired and set marker size as desired (for example, 10)

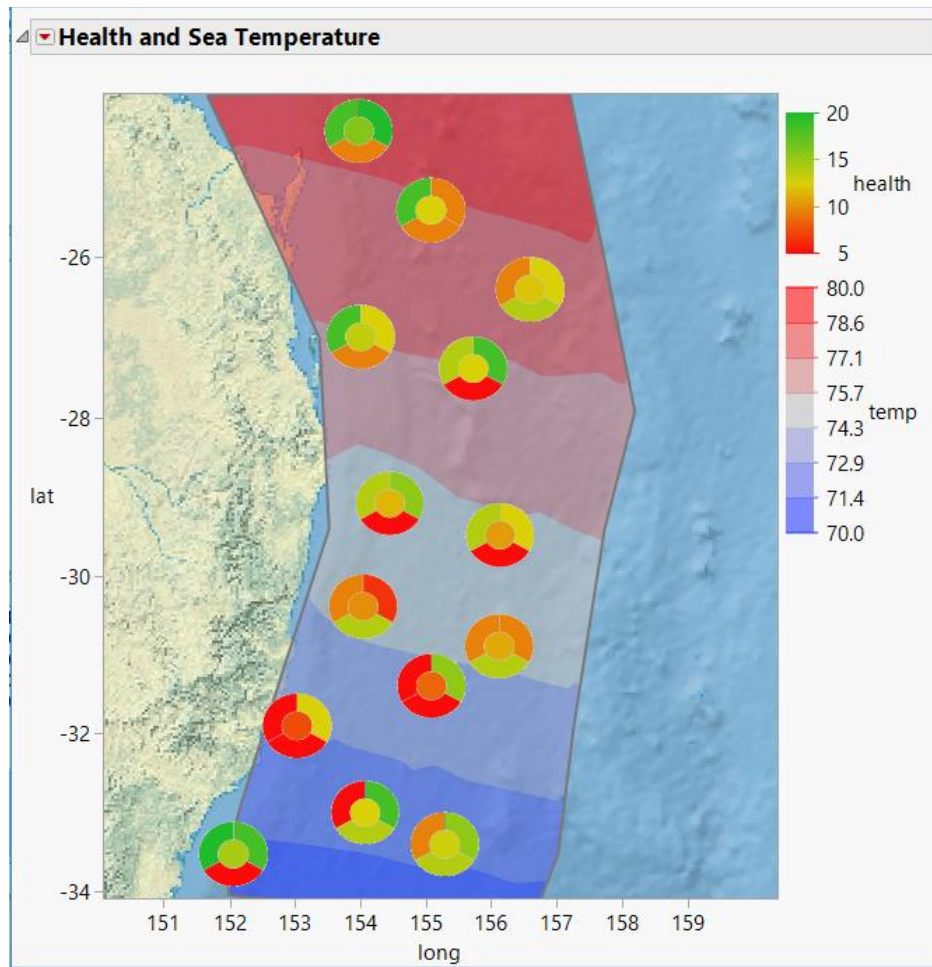


Figure 39 Contour with health indicators

It would be nice to add a labeled version of the health indicator to let the viewer know what each section represents.

1. With the file we used to map data labels to shape names (*Labels.jmp*),
  - a. Label the **Shape Name** column
  - b. Label all the rows
2. Start Graph Builder
  - a. Drag **Shape Name** to the **Map Shape** role
  - b. Click **Done**
3. Adjust the size of the image and move the labels if desired.
  - a. We're going to want to capture just the graph, so it would make it easier if we remove the axis ticks.
  - b. In the **Axis Settings** dialog for each axis, disable
    - i. **Automatic Tick Marks**
    - ii. **Major and Minor Tick Marks** and **Labels**

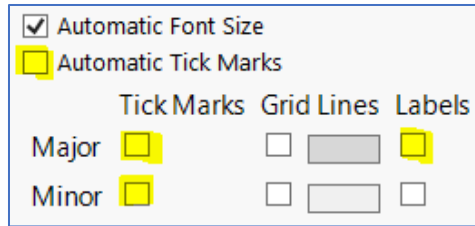


Figure 40 Removing the axis in Axis Settings

- c. If you also remove the title and rename “Graph Builder” to “Health Legend”, you’ll be left with:



Figure 41 Health legend graph

- d. Use the **Selection** tool (fat plus toolbar button) to select just the graph
- e. Right-click over the selected graph and choose **Copy**
- f. Paste the clipboard into an image editor (Paint on Window/Photos on Mac)
- g. Save to disk. (*PieLegend.PNG*)
- h. Drag the file from your file browser onto the **Health and Sea Temperature** graph. It will be placed in the background initially, so you will need to bring it to the front.
  - i. Right-click and choose **Customize...**
  - ii. Click the down arrow to move the **Picture** to the bottom of the drawing order.
  - iii. If desired, set the **Transparency** of the picture to 0.8 to dim the legend and let some of the underlying graph show through. You can use **Apply** before closing the dialog to preview it.
  - iv. Click **OK**
- i. Drag the legend image to the top right corner.

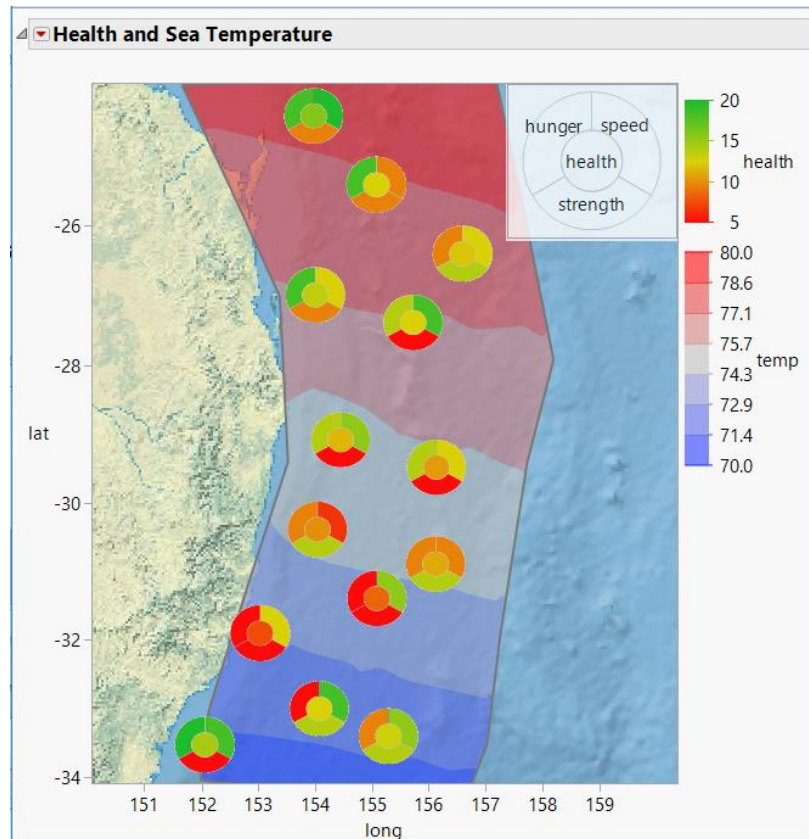


Figure 42 Shape legend

The complex pie maker script *ComplexPieMaker.jsl* and custom maps are included with this paper so you can try this with your own data.

## Web 3D Scatterplot

The web version of a 3D scatterplot described here is one of my personal visualization projects. It is also inspired by video game technology and JMP. It is not a supported JMP feature but may be used as an example of how to export data from JMP in a format usable by a third-party application.

Here is JMP's Scatterplot 3D launch dialog with *Diamonds Data.jmp*:

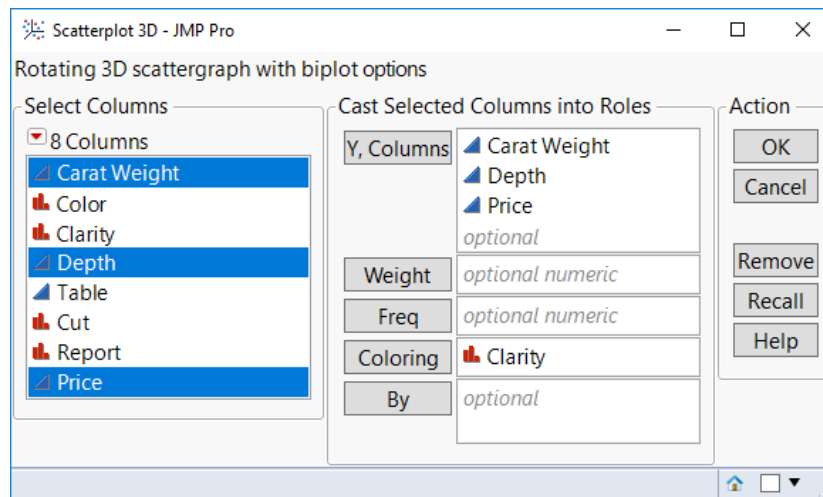


Figure 43 JMP's Scatterplot 3D launch dialog

It will produce:

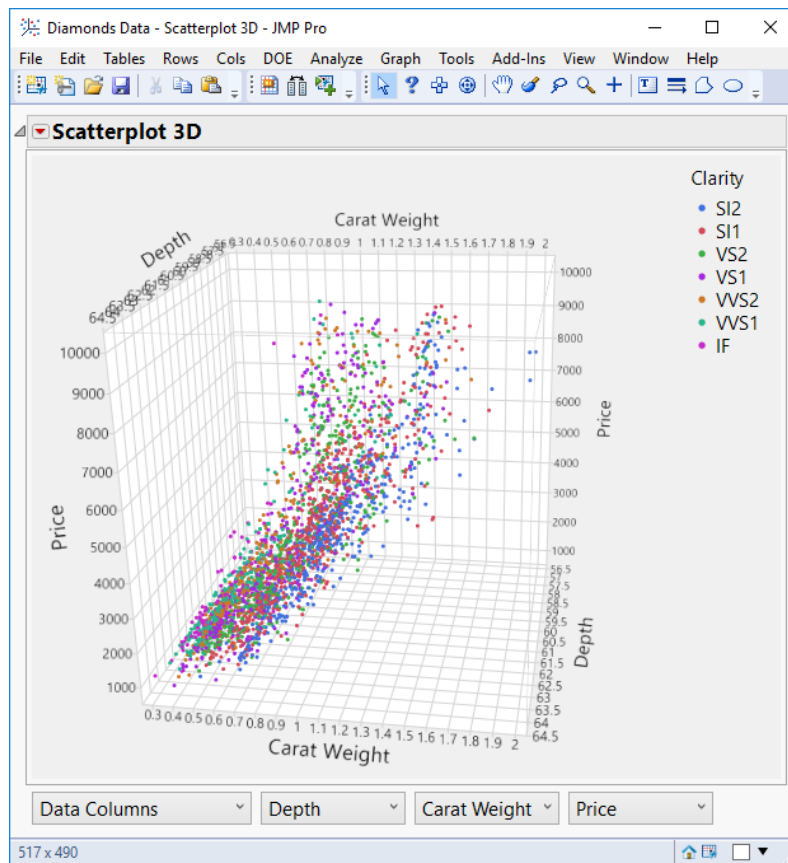


Figure 44 Scatterplot 3D with Diamonds Data

Here is a screen shot of the [Web 3D Scatterplot](#) of *Diamonds Data.jmp* displaying the **Carat Weight**, **Depth**, and **Price** columns colored by the **Clarity** column:

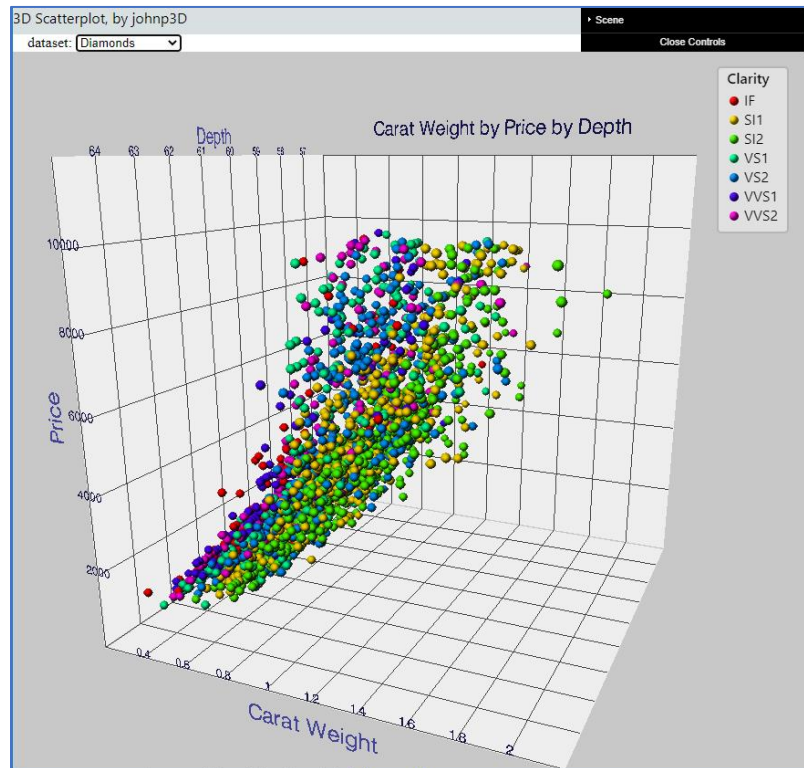


Figure 45 Web 3D Scatterplot

I built the application using open source JavaScript Libraries.

- [Three.JS](#) for 3D drawing
- [OrbitControls.js](#) for zooming, panning and rotating the scene
- [dat.gui.js](#) for scene customization user interface (top right).

### What Is Three.JS?

Three.js is an open source JavaScript library that supports 3D Drawing using WebGL which is a 3D Canvas Drawing API supported by all modern web browsers. Web GL can be used directly, but Three.js provides objects that encapsulate some of the boiler plate code needed to build a 3D Web application with Web GL. It also provides examples and documentation.

### What are particle systems?

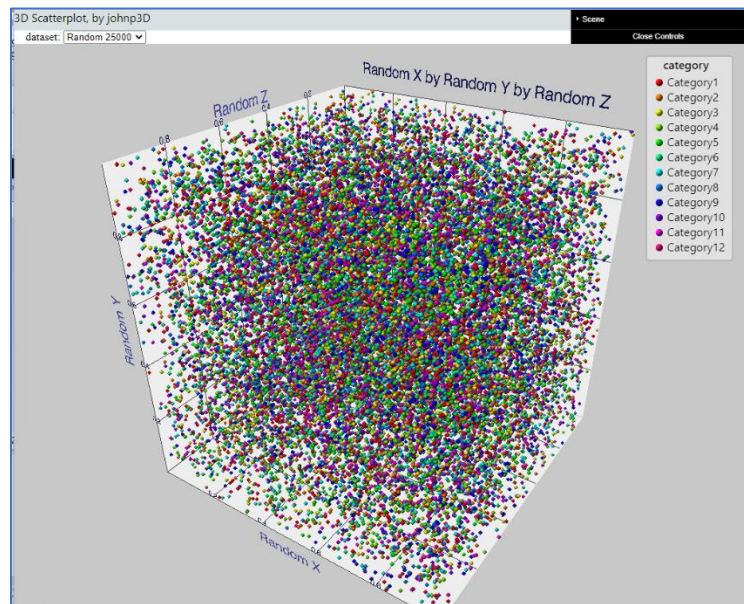
Particle systems are used by games, simulations and visualizations to provide effects such as fire, smoke, fireworks, and clouds. Usually, they provide dynamic behavior allowing the particles to move over time.

They often use textures or sprites for 3D points in a scene to add more detail in an efficient way. In some cases, it's desirable to have the textures rotate as the scene rotates to constantly face the front.



If the geometry for each of the spheres in this scene were generated and shaded with a light source, this 3D Scatterplot wouldn't rotate smoothly in a web browser running on a low powered device. With the help of particles systems, we can use a shaded gray sphere image texture for each point blended with a color to represent the category or "Color By" variable.

When I first started building this Web 3D Scatterplot application, I used random data to see how well it would perform with medium to large data sets.



*Figure 46 3D Scatterplot with 25000 randomly place points*

The point of this discussion is to describe how to export data from JMP in a format that can be consumed by or integrated into another application. When doing so, the first concern is the format required by the target application.

There are many available data sets online in various formats that can be read by an open source JavaScript library, but I wanted to use a column-based format like JMP datasets and make it easier to read into my program.

A popular web format is JSON (JavaScript Object Notation), because it's easy to read into a JavaScript program. I could have used JSON, but since JavaScript code is interpreted rather than compiled, I decided to just use JavaScript objects. JavaScript objects are something like Associative Arrays in JSL.

Here's a very simple example:

```
// x, y, z, color category
let simpleData = [
  { name: "X data", min: 1, max: 10,
    values: [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ] },
  { name: "Y data", min: 10, max: 100,
    values: [ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ] },
  { name: "Z data", min: -0.1, max: .7,
    values: [ -0.1, 0, .1, .3, .2, .4, .6, .5, .8, .7 ] },
  { name: "category", names: [ "category 1", "category 2", "category 3" ],
    values: [ 0, 0, 0, 0, 1, 1, 1, 2, 2, 2 ] },
];
```

I found the famous iris data set in CSV(comma-separated values) and wrapped it in some JavaScript code to make it easier to read:

```
let irisColumnNames = ["sepal_length", "sepal_width", "petal_length",
  "petal_width", "species"];
let irisCategories = ["setosa", "versicolor", "virginica"];
let irisCSV = [
  [5.1, 3.5, 1.4, 0.2, 0],
  [4.9, 3.0, 1.4, 0.2, 0],
  . . .,
  [5.9, 3.0, 5.1, 1.8, 2]
];
```

Rather than manually converting this to my column-based format, I wrote some JavaScript code to convert it.

Although this worked, I really didn't want to do this for every data set I wanted to use. JMP is good at importing data in many formats and converting them into a column-based format.

So, I built a JSL script (in *Export\_XYZC\_to\_JS.jsl*) to choose columns from the current dataset and export them in JavaScript objects compatible with my 3D Scatterplot program.

I hope by describing and sharing this script, you'll be able to use it as a starting point for your exporting need even if the format you need is very different.

First, here's how you would use the script.

1. Open *Diamonds Data.jmp*



2. Open and run *Export\_XYZC\_to\_JS.jsl*
3. Pick 3 continuous columns for **XYZ**
4. Pick a categorical column for **Color**

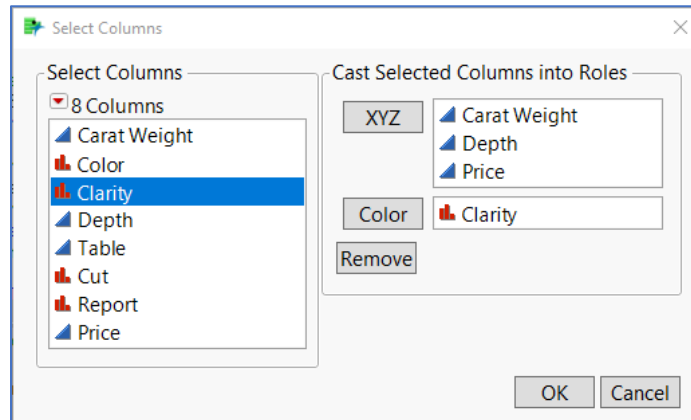


Figure 47 Select Columns dialog

5. Click **OK**
6. The log will display the location of the generated file.
7. Open the file in a text editor:

```
let simpleData = [
  { name : "Carat Weight", min: 0.3, max: 2.02, values: [0.3, 0.44, 0.31, . . .
```

The script performs the following tasks.

1. Launches the **Select Columns** dialog
2. Builds up the JavaScript code into a string
3. Save the string into a text file

Let's look at the script:

```
dlg = Column Dialog(
  xyz = Col List( "XYZ", Min Col( 3 ), Max Col( 3 ), Data Type( "Numeric" )),
  clr = Col List( "Color", Max Col( 1 ), Modeling Type({ "Ordinal", "Nominal" })))
);
```

We ask for exactly 3 Numeric columns and up to 1 ordinal or nominal column for color.

Here is example output for a small continuous column:

```
{ name: "X data", min: 1, max: 10, values: [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ] },
```

This is code builds the **name**, **min**, **max**, and **values** parts of this line and concatenates them:

```

nameString = "name : \!" || ( col << GetName() ) || "\!", ";
minString = "min: " || char( colMin( col ) ) || ", ";
maxString = "max: " || char( colMax( col ) ) || ", ";
valuesString = "values: " || char( col << getValues );
colString = " { " || nameString || minString || maxString || valuesString || "
},\!n";

```

Here's example output for a categorical column:

```

{ name: "meal", names: [ "breakfast", "lunch", "supper" ],
  values: [ 0, 2, 1, 2, 1, 0, 2, 2, 0, 1 ] },

```

This code builds the **names** array of the output:

```

x = col << getAsMatrix;
aa = associativeArray( x );
keys = aa << getKeys; // List of category names.
keyString = "[";
n = N Items( keys );
For( i = 1, i <= n, i++,
    if ( i == n,
        Insert Into ( keyString, "\!" || char( keys[ i ] ) || "\!" ),
        Insert Into ( keyString, "\!" || char( keys[ i ] ) || "\!", " " );
    );
);

```

The following code builds the **values** array for the line. Notice that the values are 0-based indices into the **names** array in JavaScript while JSL array indices are 1-based.

```

indicesString = "[";
For Each Row(
    s = col[ Row() ];
    index = Loc( keys, s )[ 1 ]; // 1-based indexing.
    Insert Into( indicesString, char( index - 1 ) || ", " ), // 0-based
indexing.
);
Insert Into( indicesString, "]" );

```

These lines concatenate the color column's JavaScript object members:

```
nameString = "name : \!" || ( col << GetName()) || "\!", ";
catString = "names: " || keyString || ", ";
valuesString = "values: " || indicesString || ", ";
colString = " { " || nameString || catString || valuesString || " },\!n";
```

Finally, these lines build up the output file name and save the code in the file.

```
dataTableName = ( dt << Get Name());
jsFilename = Get Path Variable( "TEMP" ) || dataTableName || ".js";
saveTextFile( jsFilename, jsString );
```

To learn JSL Scripting, I recommend:

- Saving reports to the **Script Window** and examining the generated code
- Looking up functions in the **Scripting Index**
- Reading the **Scripting Guide**
- Searching for JSL solutions in the [JMP User Community](#)'s blogs, discussions, and [JSL Cookbook](#)

## Summary

JMP brings people together from different industries and often cross pollinates ideas. There's no limit to the number of ways you can combine ideas from different industries, and past experiences to discover new useful techniques in JMP. I enjoyed bringing together old skills with new skills to create this paper.

Don't be afraid to experiment with JMP and JSL. If you discover something new and worth sharing, please do so.

## References:

JMP User Community post inspiring this topic:

<https://community.jmp.com/t5/Discussions/pie-graphs-heat-maps/td-p/221475>

Dr. Anderson Mayfield's JMP On Air Earth Day Presentation:

<https://community.jmp.com/t5/JMP-On-Air/Exploring-the-Reef-Coral-Response-to-Changing-Environments/ta-p/258762>

Custom Graphics Script on JMP Public:

<https://public.jmp.com/packages/Using-Bars-to-Display-3-Variables-Over-a/js-p/5d605bce3560ce0d2cafd48b>

Custom Maps over Contour on JMP Public:

<https://public.jmp.com/packages/Sea-Temperature-Near-Fiji/js-p/5d64597acf28b80f980812ba>

JMP Custom Map files:

<https://www.jmp.com/support/help/en/15.2/index.shtml#page/jmp/custom-map-files.shtml>

Nascif's example with bar graphs as markers and using virtual join:

<https://community.jmp.com/t5/JSL-Cookbook/Geographic-Maps-with-Graphs-as-Markers/ta-p/53782>

Custom Map Creator:

[community.jmp.com/t5/JMP-Add-Ins/Custom-Map-Creator/ta-p/21479](https://community.jmp.com/t5/JMP-Add-Ins/Custom-Map-Creator/ta-p/21479)

JSL Graphics functions:

<https://www.jmp.com/support/help/en/15.2/#page/jmp/graphics-functions.shtml>

Three.js 3D Graphics JavaScript open source library:

<https://threejs.org/>

Inspiration for *SmallSchool.jmp*:

[https://pixar.fandom.com/wiki/Category:Finding\\_Nemo\\_Characters](https://pixar.fandom.com/wiki/Category:Finding_Nemo_Characters)

<https://www.seaturtlestatus.org/articles/2010/takin-a-ride-on-the-eac-across-the-southern-pacific-ocean>

Editing Hover Labels:

<https://www.jmp.com/support/help/en/15.2/#page/jmp/edit-hover-labels.shtml#>