Build a Tailored Report using Messages

Ingredients:

- Report objects
- Platform objects
- Enhanced Log,
- Scripting Index

Sample Data Tables: Tablet Production
Difficulty –Medium

JMP Report windows combine two separate but related objects: the underlying analysis platform and the window in which the results are displayed. In this first recipe we will look at the simplest case, where there is no grouping condition. While much of this can be easily implemented directly through hard coding, it is helpful to understand how it can be done using JSL object messages. This will let us generalize to situations where column names are unknown until runtime.

Steps:

1. Open the sample data table Tablet Production. Run Analyze > Multivariate Methods Multivariate with all the continuous variables except Disso. Save the script to a script window. We will use this script to build the output created interactively and given in the next script.
2. Rerun the analysis. In the hotspot to the left of Multivariate select Simple Statistics > Univariate Simple Statistics and Color Maps > Color Map on Correlations. In the report window under the hotspot to the left of Scatterplot Matrix, turn off Show Points and select Matrix Options > Heat Map, Show Historgrams > Horizontal. Turn on Show Correlations and Nonpar Density. Close the outline box at the top titled Correlations and shorten the Color Map on Correlations outline box name to Color Map. Resize the color map. Save the script to a script window.
3. Examining the two scripts, we can see that the selections from the hotspot appear as arguments to `Multivariate`. These options can be implemented via messaging the platform. In most cases the argument names will be identical to the message names. To be certain, open the Scripting Index and select Multivariate from the leftmost list. The first two options appear in the list to the right. If you select the subcategory Scatterplot Matrix Message you can find the remaining three messages.
4. The remaining messages are found in the `SendToReport` argument. They are messages sent to report window objects. More details will be given below.
5. Add these two lines to the top of the script.
   ```
   Names Default to Here(1);
   tblRef = Current Data Table();
   ```
   This scopes the variables to the script window and creates a reference to the last selected (or opened) table.
6. Explicitly send the Multivariate message to the current table. The operation returns a pointer to the Platform object associated with the analysis.
   ```
   multivarPlt = tblRef << Multivariate(
   .
   ```

.
.

Run this script to see what the default output looks like.

7. Messages associated with hotspot items from the topmost outline box can be sent directly to the platform.

```
multivarPlt << Color Map on Correlations(1);
multivarPlt << Univariate Simple Statistics(1);
```

8. Hotspot items from embedded outline box require a slightly different approach. It will be similar to how we will deal with visual items from the report.

   a. Create a reference to the scriptable object associated with the hotspot.

```
scatterplotObj = Report(multivarPlt)["Scatterplot Matrix"] << Get Scriptable Object;
```

   `Report` accesses the report tied to the analysis platform. The outline box is referenced using its name. This works as long as it is the first outline box with this name and the names doesn't change over time.

   b. Message the Scatterplot object reference

```
scatterplotObj << Show Points(0);
scatterplotObj << Heat Map(1);
scatterplotObj << Show Correlations(1);
scatterplotObj << Nonpar Density(1);
scatterplotObj << Horizontal(1);
```

9. Changes to non-hotspot items appear in the `SendToReport` argument. There is limited documentation on `SendToReport` and it is common for superfluous code to appear. For example, removing all `Dispatch` elements with `Multiv Scatter Plot` as the second argument does not change the report window. In general, nearly all changes can be made via messaging the appropriate report window element. `Dispatch` can help finding elements, identifying their object type, and the message to be sent. The three remaining `Dispatch` elements

```
Dispatch( {}, "Correlations", OutlineBox, {Close( 1 )} ),
Dispatch(
   {},
   "Color Map on Correlations",
   OutlineBox,
   {Set Title( "Color Map" )}
),
Dispatch(
   {"Color Map on Correlations"},
   "Color Map",
   FrameBox,
   {Frame Size( 402, 345 )}
)
```

can be turned into the messages

```
Report(multivarPlt)[Outline Box("Correlations")] << Close( 1 );
Report(multivarPlt)[Outline Box("Color Map on Correlations")] << Set Title( "Color Map" )
Report(multivarPlt)[Outline Box("Color Map"), Frame Box(1)] << Frame Size(400,345)
```

Here, if the first argument is an empty list the visual element is contained in the top-level outline box (Multivariate). Otherwise, the outline box hierarchy is given. The second argument names the display element, this is only useful for outline boxes, where the name is given. The third argument identifies the object type and the fourth, the message. Only outline boxes can be referenced by their name. Other objects must be located by their position among all other like objects in the outline. In this situation, since there is only one frame box in the entire outline, the task becomes easy.

Hints for Success:

- Outline box names are very consistent across JMP versions. Using their name is a robust way to access items in a report window.
- To access objects associated with lower-level outline boxes, use the `Get Scriptable Object` message on the outline box containing the hotspot associated with the object. If there is no hotspot, reference the object type using the title of the outline box. In the saved script, look for the `Dispatch` argument with the outline box title. It should contain object type information.