# Using a Dialog Box to Create a Tailored Report

Ingredients:

- Dialogs
- User Input
- Expression Handling

**Sample Data Tables** – Tablet Production
**Difficulty** – Hard
**Video Length** – 5:07

This takes the code from Build a Tailored Report using Messages and generalizes it so it can be used with two or more continuous columns from any data table. A non-modal dialog box is created so columns can be selected by the user.

Steps:

1. Start with `Names Default to Here(1)` and create a variable referencing the current data table.
2. The code below creates a non-modal dialog box. Non-modal dialogs are more flexible than modal dialogs but a bit more complicated to work with. Any operations tied to dialog box actions, such as button clicks, must be created in such a way that they are only executed when the action takes place. We will use the `Expr` function for this.

```
dialog = New Window("Select Columns",
  tbWarning = Text Box("",<<Set Font Style(Bold)),
  H List Box(
    Panel Box(
      "Select one or more inputs",
      inputBox = Col List Box(All,<<Modeling Type({"Continuous"}))
    )
  ),
  H List Box(
    Button Box(
      "OK",
      inputList = inputBox<<Get Selected;
      Eval(runOK);
    ),
    Button Box("Cancel",dialog<<Close Window())
  )
);
```

   a. The first argument to `New Window` is the text string giving the widow title. It is required. The remaining arguments are components that provide information (`Text Box`), allow user input (`Button Box`, `Col List Box`), or organize visual components (`H List Box`, `Panel Box`).
   b. `Text Box` is used to hold a warning message if the user selects fewer than two columns. Initially, it is left blank.
   c. By default, items are arranged vertically, stacked one on the next. Elements in `H List Box` in are arranged horizontal. `Panel Box` draws a box around its contents and can only take two arguments, a title and the element to be contained within the panel box.

d. `Col List Box` is used to populate a list box with columns from the current data table. The `All` argument indicates that all the columns should be used. This is need otherwise no items will appear. The argument `<<Modeling Type({“Continuous”})` controls the Modeling Type of the columns appearing in the list box. Note that << must appear before `Modeling Type`. It takes a single argument, a list, which must be text strings corresponding to the names of JMP Modeling Types. In this case only one Modeling Type is given, but multiple Modeling Types can be used if needed.

e. Button boxes take two arguments, a title and an optional script. Scripts can contain multiple semicolon separated statements. The script is evaluated when the button is clicked. The OK Button Box has two statements. The first gets the user selected items from the `Col List Box` named `inputBox` and the second executes the code stored in the expression `runOK`.

f. The Cancel button is added to let users dismiss the dialog without taking further action.

3. Create a variable runOK to hold the expression to be executed when the OK button is clicked

```
runOK = Expr();
```

The code that checks `inputList` and builds the report will be contained inside the `Expr` function.

4. Start by checking to see if there are at least two items in `inputList`. If not, add a warning to the dialog.

```
If(N Items(inputList) < 2,
     tbWarning << Set Text("You must select 2 or more columns");
,//ELSE
```

5. The remainder of the `If` statement, executed when two or more columns have been selected, generates the report. We don’t know how many columns have been selected or their names, so hard coding is not an option. We will build the `Multivariate` platform message with expression handling starting with the argument holding the column names:

```
yExpr = Expr(Y());
For Each({colName},inputList,
  Insert Into(yExpr, Column(tblRef,colName))
);
```

The first line creates the variable to hold the argument expression and the remaining code inserts the column references into it. We may be tempted to insert the unquoted column name into the argument using `Parse`. This will work until we encounter a column name containing a special character function name, when it will failure during platform message evaluation.

6. Next, we want to insert the expression above into an expression holding the platform message. This can be a bit tricky since two things need to happen. First, we want to insert the unevaluated contents of `yExpr`. We will use `Name Expr` for this. Second, we want to message the current data table `tblRef` directly and retrieve the pointer to the platform object that is returned. This done using `Substitute` to build the code and `Eval` to execute it.

```
Eval(Substitute(
  Expr(
    multivarPlt = tblRef << Multivariate(
      inCols,
      Estimation Method( "Row-wise" ),
      Scatterplot Matrix( 1 )
      )
    ),
    Expr(inCols),Name Expr(yExpr)
  ));
```

When `Substitute` is used for expression handling it takes an odd number of arguments. The first corresponds to the expression into which substitutions are made. `Substitute` evaluates its first argument, so we'll need the `Expr` function to treat it as an expression. The remaining pair of arguments gives the expression within the first argument to be replaced (`inCols`) and the value to use, `Name Expr(yExpr)`. `Name Expr` is used because we want the contents of `yExpr` as an expression, not its evaluated value.

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.