



# JMP® Synergies: Using JMP® and JMP® Pro With Python and R

WHITE PAPER

## Table of Contents

<b>Introduction</b>	1
<b>JMP and Python</b>	1
Generating Python Code From a JMP Pro Model	1
The Basics of Connecting JMP and Python	3
A bit of extra fun with JMP – adding an image marker to our graph	5
Additional Simple JMP With Python Examples	6
<b>JMP and R</b>	7
The Basics of Connecting JMP and R	7
Additional Simple JMP With R Examples	11
Advanced Examples	11
Data Visualization With t-SNE and UMAP Add-In:	12
SVM Add-In:	14
The JMP to R Add-In Builder:	
Using the JMP to R Connection Without JSL	15
<b>Conclusion</b>	15

## Introduction

JMP is a standalone, full-featured data visualization and statistical analysis software package from SAS for the Windows and Mac desktop. JMP has the interactivity and dynamic linkage that makes data exploration exciting and insightful and contains many advanced analytical options, fully satisfying the needs of data explorers and analysts. Still, there may be occasions where you'll want (or need) to use JMP in conjunction with open source tools, like Python or R.

Consider the following examples:

- A corporate statistician prohibited from using non-validated software like R for primary analysis may nevertheless want to explore new methodology in an R package side-by-side with validated methods in JMP.
- A Python or R programmer may want to take advantage of the dynamic linkage, advanced visuals or powerful analytics available in JMP.
- A company's data analysis department may want to develop an add-in to JMP that allows some R or Python code to be run as an add-in within JMP, so that future users can interact with the GUI add-in without needing to write the R or Python code themselves.
- A statistical educator may want to introduce specific open source packages in the classroom but within a point-and-click environment that wouldn't require coding by the student.

Whatever your motivation for connecting open source (or other) tools with JMP software's GUI, this guide will help you to get started using the Python and R connections in JMP.<sup>1</sup>

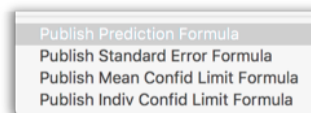
## JMP® and Python

### Generating Python Code From a JMP Pro Model

With JMP Pro you can generate Python code for predictive models that you create in JMP. This is useful when you would like to build, explore and compare models in JMP Pro, but deploy them in a different production environment. JMP Pro provides an easy way to construct multiple and compare competing models, and to even create a model average (ensemble) in the Formula Depot platform.



First, fit a model using tools under the Analyze menu in JMP. To save the model to the Formula Depot, use either the options "Publish Probability Formula" (for classification models) or "Publish Prediction Formula" (for prediction models). Generally, both can be found in the red triangle options for the model results or under "Save Columns."

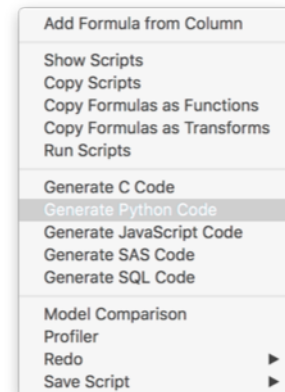


<sup>1</sup> JMP also connects to SAS and Matlab (in addition to R and Python), and JMP also generates C, JavaScript, SAS and SQL model scoring code (in addition to Python). To learn more about any of these JMP connections, and for even more Getting Started tips and examples, go to [jmp.com/support/help/en/15.1/#page/jmp/extending-jmp.shtml](http://jmp.com/support/help/en/15.1/#page/jmp/extending-jmp.shtml).

Once you select that key word “Publish” (or use the “Add Formula from Column” option), your model is sent to the Formula Depot. The Formula Depot is a platform that allows you to save one or many competing models, compare them, create ensemble models and deploy them in a variety of production environments.

“Generate Python Code” is an option under the red triangle for each individual model (i.e., this “Fit Least Squares” model for predicting **weight**) or under the Formula Depot red triangle for generating the code for the entire set of models saved in the Formula Depot window.

The resulting Python code, shown below, includes some required dependencies, which are called in the first section of the code. The copyright information follows, then some Python code to make the variable names and types, and finally the formula that will be used to predict the response variable for new observations with known values for the predictor variables (in this example, *age*, *height* and *sex*).



```

Fit_Least_Squares_weight

from __future__ import division
import jmp_score as jmp
from math import *
import numpy as np

"""
=====
Copyright(C) 2018 SAS Institute Inc.All rights reserved.

Notice:
The following permissions are granted provided that the
above copyright and this notice appear in the score code and
any related documentation. Permission to copy, modify
and distribute the score code generated using
JMP(R) software is limited to customers of SAS Institute Inc. ("SAS")
and successive third parties, all without any warranty, express or
implied, or any other obligation by SAS. SAS and all other SAS
Institute Inc. product and service names are registered
trademarks or trademarks of SAS Institute Inc. in the USA
and other countries. Except as contained in this notice,
the name of the SAS Institute Inc. and JMP shall not be used in
the advertising or promotion of products or services without
prior written authorization from SAS Institute Inc.
=====
"""

""" Python code generated by JMP v14.1.0 """

def getModelMetadata():
    return {"creator": u"Fit Least Squares", "modelName": u"", "predicted": u"weight",
            "table": u"Big Class.jmp", "version": u"14.1.0", "timestamp": u"2018-08-08T04:02:07Z"}

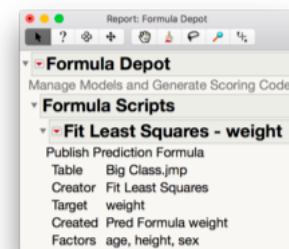
def getInputMetadata():
    return {
        u"age": "float",
        u"height": "float",
        u"sex": "str"
    }

def getOutputMetadata():
    return {
        u"Pred Formula weight": "float"
    }

def score(indata, outdata):
    _temp_0 = np.nan
    _temp_1 = np.nan

    if (indata[u"sex"] == u"F"):
        _temp_0 = 2.0492068900361
    elif (indata[u"sex"] == u"M"):
        _temp_0 = -2.0492068900361
    else:
        _temp_0 = np.nan
    if (jmp.numeq(indata[u"age"], 12)):
        _temp_1 = 0
    elif (jmp.numeq(indata[u"age"], 13)):
        _temp_1 = -13.6855931672148
    elif (jmp.numeq(indata[u"age"], 14)):
        _temp_1 = -25.8472610743993
    elif (jmp.numeq(indata[u"age"], 15)):
        _temp_1 = -19.7698493666905
    elif (jmp.numeq(indata[u"age"], 16)):
        _temp_1 = -10.1590212259529
    elif (jmp.numeq(indata[u"age"], 17)):
        _temp_1 = 2.52025614257322
    else:
        _temp_1 = np.nan
    outdata[u"Pred Formula weight"] = -176.033203126788 + 4.72294023921341 * indata[u"height"]
    + _temp_0 + _temp_1
    return outdata[u"Pred Formula weight"]

```



You can read more about this topic at [jmp.com/support/help/en/15.1/#page/jmp/formula-depot.shtml](http://jmp.com/support/help/en/15.1/#page/jmp/formula-depot.shtml).

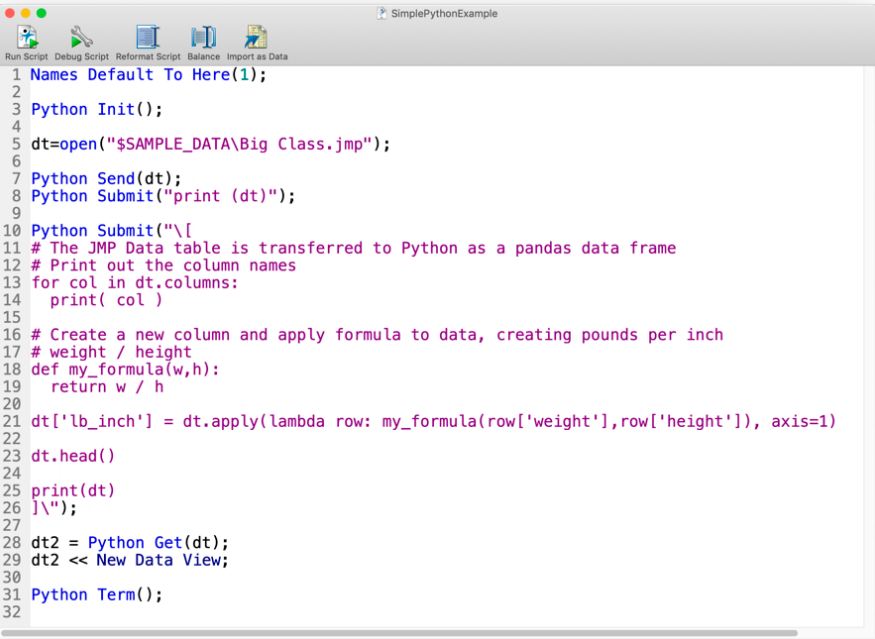
## The Basics of Connecting JMP and Python

In addition to generating Python model scoring code from within JMP, you can also open the connection between JMP and Python and send and receive data and results and actions (using a combination of JMP and Python code) between Python and JMP. (The JMP and R connection works in the same way.) In order to use the Python connection in JMP, you will use JMP Scripting Language (JSL) as a wrapper for your Python code.

Here is a simple example to illustrate. You will need Python<sup>2</sup> to be installed on your machine, but this is standard on most new computers.

- (1) Open the Python connection from JMP.
- (2) Perform an operation, like sending data from JMP to Python.
- (3) Perform another operation, such as submitting Python code directly to Python to create a new variable.
- (4) Perform yet another operation, such as bringing a result or data back from Python to JMP (perhaps visualize the result in JMP).
- (5) Close the Python connection.

Here is the code for this simple example. To follow along, you'll first need to open JMP. Next, go to File > New > New Script to open a new blank script window. Then type the following:



```

1 Names Default To Here(1);
2
3 Python Init();
4
5 dt=open("$SAMPLE_DATA\Big Class.jmp");
6
7 Python Send(dt);
8 Python Submit("print (dt)");
9
10 Python Submit("[
11 # The JMP Data table is transferred to Python as a pandas data frame
12 # Print out the column names
13 for col in dt.columns:
14     print( col )
15
16 # Create a new column and apply formula to data, creating pounds per inch
17 # weight / height
18 def my_formula(w,h):
19     return w / h
20
21 dt['lb_inch'] = dt.apply(lambda row: my_formula(row['weight'],row['height']), axis=1)
22
23 dt.head()
24
25 print(dt)
26 ]\");
27
28 dt2 = Python Get(dt);
29 dt2 << New Data View;
30
31 Python Term();
32

```

In the script above, there are nine pieces of JSL code (indicated in blue). The first line helps to ensure that any file or variable references we make will be related to this script. Line three of our script, "**Python Init();**", opens Python.<sup>3</sup>

<sup>2</sup> [python.org](https://python.org).

<sup>3</sup> The details of the Python installation and versioning can be found here: [jmp.com/support/help/en/15.1/#page/jmp/install-python.shtml#](https://jmp.com/support/help/en/15.1/#page/jmp/install-python.shtml#) and [jmp.com/support/help/en/15.1/#page/jmp/troubleshooting-the-jmp-python-integration.shtml#ww822804](https://jmp.com/support/help/en/15.1/#page/jmp/troubleshooting-the-jmp-python-integration.shtml#ww822804).

Line 5, `dt = Open("$SAMPLE_DATA/Big Class.jmp");` asks JMP to open a data set that is in the JMP sample data directory and assigns the name "dt" to it.<sup>4</sup> The data set in this example is called "Big Class.jmp."

Line 7, `Python Send(dt);` sends the "dt" data table from JMP to Python. The JMP data table will be transferred to Python as a Pandas data frame. Line 8, `Python Submit("print(dt)");`, submits some Python code to Python. The information inside the quotation marks is the Python script; the rest of the line is the JSL code wrapper. It is useful to monitor your code submissions and troubleshoot, if needed, by using the JMP log window. Go to the JMP menu bar and choose Window → Log to open the JMP log. Here you can see any internal Python error messages or results. For example, if we check the log after running this code up to this point, we will see the following Python output in the log:

```
dt=open("$SAMPLE_DATA\Big Class.jmp");
/*:
Data Table( "Big Class.jmp" )
//:*/
Python Send(dt);
Python Submit("print (dt)");
/*:
```

	name	age	sex	height	weight
0	KATIE	12	F	59	95
1	LOUISE	12	F	61	123
2	JANE	12	F	55	74
3	JACLYN	12	F	66	145
4	LILLIE	12	F	52	64

Lines 10-26 include a longer `Python Submit();` statement. This piece of Python code, embedded in the JSL code `Python Submit();` statement, will print the column names (in the log) and then create a new column made of the formula "lb\_inch = weight/height," before finally printing the new data table, which adds this new column to our old data table, in the log:

```
/*:
name
age
sex
height
weight
//:*/
Python Submit(
    print(
        name
        age
        sex
        height
        weight
    )
    ,
    Python(
        """
        name age sex height weight lb_inch
        0 KATIE 12 F 59 95 1.610169
        1 LOUISE 12 F 61 123 2.016393
        2 JANE 12 F 55 74 1.345455
        3 JACLYN 12 F 66 145 2.196970
        4 LILLIE 12 F 52 64 1.230769
        5 TIM 12 M 60 84 1.400000
        6 JAMES 12 M 61 128 2.098361
        """
    )
)
/*:
```

	name	age	sex	height	weight	lb_inch
0	KATIE	12	F	59	95	1.610169
1	LOUISE	12	F	61	123	2.016393
2	JANE	12	F	55	74	1.345455
3	JACLYN	12	F	66	145	2.196970
4	LILLIE	12	F	52	64	1.230769
5	TIM	12	M	60	84	1.400000
6	JAMES	12	M	61	128	2.098361

Lines 28 and 29 bring back the updated data table to the software's working memory and opens it as a new JMP data table.

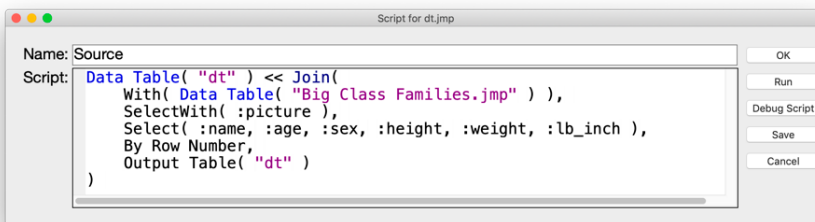
The final line of code, `Python Term();` terminates the connection to Python.

<sup>4</sup> Note that we used a slightly different line of JSL code in the R example to point to an open data file. This is another way to point to data, in this case to open a data file, using JSL.

## A bit of extra fun with JMP - adding an image marker to our graph

While the final section of this simple example does not make any further use of the JMP to Python connection, it does illustrate the potential benefits that using JMP with Python (or R or other connected software) can bring. Specifically, we can interactively work between JMP and Python, taking advantage of running prescribed Python code or special Python packages, then improving on our final analyses or visualizations by using the dynamic features of JMP.

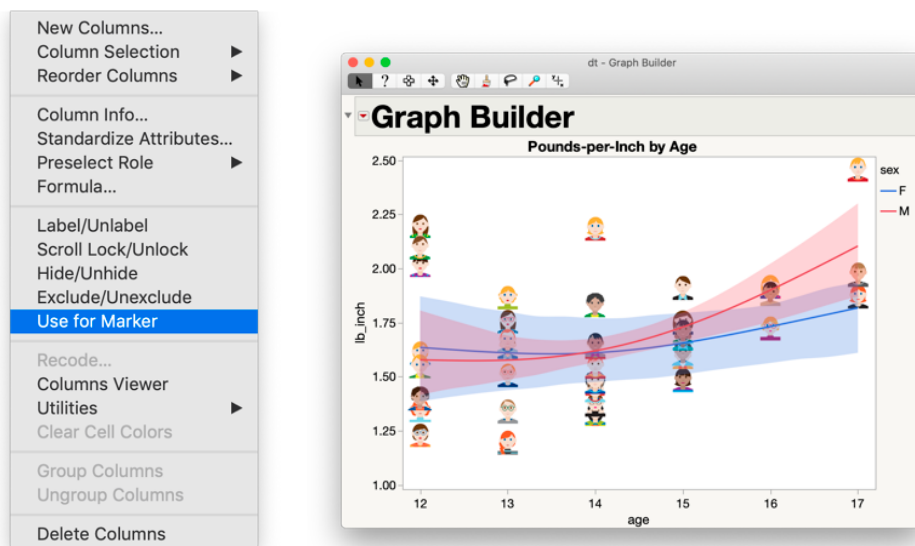
After bringing the "dt" data table back into JMP, for example, we can use some nice JMP visualization features to improve our graphics in just a few clicks or short JSL scripts. To illustrate, let's join our "dt" table with the Big Class Families sample data set, available from the Sample Data Library in JMP, to add a column of images to our new "dt." First, open Big Class Families (by clicking on it from Help > Sample Data Library > Big Class Families or running the code `"dt = Open("$SAMPLE_DATA/Big Class.jmp") ;"`) and use the Tables > Join (or the following script) to join the image column to our "dt" table.



These steps add a column called "picture" to our data set and import the student images from the Big Class Families data table into our new combined data table.

	picture	name	age	sex	height	weight	lb_inch
		ROBERT	17	M	70	172	2.46
		ALFRED					
		ALICE					
		AMY					
		BARBARA					
		34 others					
1		KATIE	12	F	59	95	1.6101...
2		LOUISE	12	F	61	123	2.0163...
3		JANE	12	F	55	74	1.3454...
4		JACLYN	12	F	66	145	2.1969...
5		LILLIE	12	F	52	64	1.2307...
6		TIM	12	M	60	84	1.4
7		JAMES	12	M	61	128	2.0983...
8		ROBERT	12	M	51	79	1.5490...
9		BARBARA	13	F	60	112	1.8666...

Now we can build a visualization of the new pounds-per-inch variable, using the images as markers in our plot. First, make the image column a marker by clicking on the “picture” column and going to Cols > Use for Marker.



In this simple example, we used the JMP to Python connection to open data in JMP, send it to Python, create a new column in Python and add the new column to the Python pandas data frame, and then bring back the updated data as a JMP data table, which we could analyze and further manipulate in JMP.

## Additional Simple JMP With Python Examples

You can find a few more simple examples of JMP with Python interactions in the JMP Help, under the Scripting Guide → Extending JMP → Working with Python, or by visiting [jmp.com/support/help/en/15.1/#page/jmp/additional-python-integration-examples.shtml](http://jmp.com/support/help/en/15.1/#page/jmp/additional-python-integration-examples.shtml).

Here you will find the code for:

- Sending a data table to Python.
- Creating objects in Python.
- Matrix operations in Python.



## JMP® and R

Connecting JMP with the open source software R<sup>5</sup> easily allows you to take advantage of R capabilities along with JMP software's dynamic and interactive approach to analytics and visualization. You can also create easy-to-use menu interfaces for R packages and combine elements of JMP and R. These are all possible when you have both JMP and R installed on your computer. The R functions to perform these connections are listed and described (with examples) in the Scripting Index (Help > Scripting Index > R Functions).

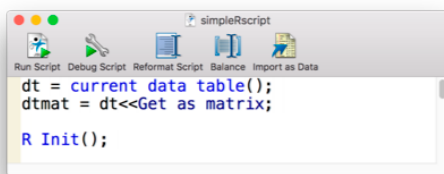
### The Basics of Connecting JMP and R

In order to use the R connection in JMP, you'll need to wrap your R code using the JMP Scripting Language (JSL).

As an example, let's move some data from JMP to R for an analysis and manipulation, then bring it back to JMP. The process is fairly straightforward. We'll write a little bit of JSL code to open R, send data from JMP to R, send the R code to R, and bring back data from R into JMP.

First, make sure that you have both R and JMP installed on the same machine.<sup>6</sup>

Next, open a scripting window in JMP from **File > New > New Script**.

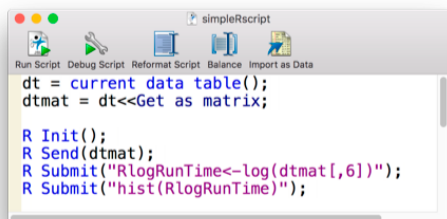


Let's also open a sample data set in JMP. Go to **Help > Sample Data Library** and open the **Fitness.jmp** data table. In order to send this to R, we need to use two simple lines of JSL. The first line essentially identifies the open data table, **Fitness.jmp**, and renames it "dt." The second line converts the newly named "dt" data table into a matrix format. Because R has slightly different data structuring options than JMP, the matrix format is a simple framework that works easily in both JMP and R. Next, we call R from JMP. If R is installed on the same machine, JMP will find it; you do not need to set up a connection or point JMP to the R installation. You only need to type the JSL line "**R Init() ;**". JMP will search for R on the machine, and then initialize (or open) it.

<sup>5</sup> [cran.r-project.org](http://cran.r-project.org).

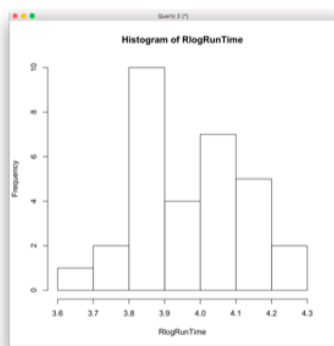
<sup>6</sup> The details of the R installation and versioning can be found here: [jmp.com/support/help/en/15.1/#page/jmp/installing-r.shtml](http://jmp.com/support/help/en/15.1/#page/jmp/installing-r.shtml).

Now we're ready to send information back and forth between JMP and R. The first line of the following code sends our matrix version of the data, called dtmat, to R. The next line sends R code to be performed in R.



The line `RlogRunTime<-log(dtmat[,6])` takes the sixth column of the input data matrix (the "Mile Run Time" from this Fitness data set) and log-transforms it. Those results will now be called "RlogRunTime" (because that's the name we assigned it in the R code snippet here).

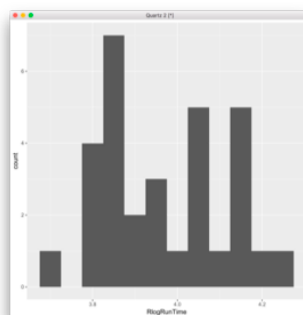
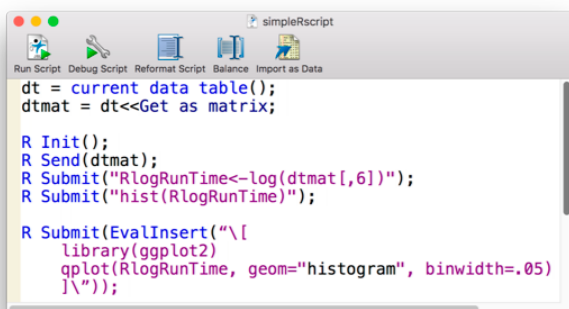
The line `R Submit("hist(RlogRunTime)")` generates a histogram in R, which will pop up on your desktop once you run this line of code.



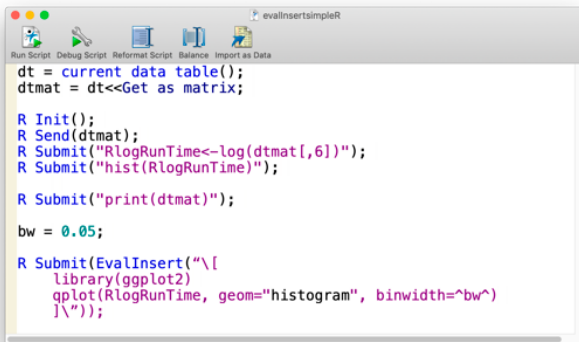
To run the code, select one or more lines (each line must end with ";") and click the "Run Script" button at the top of the script window:



If we have a multiple-line bit of R code to send through our JSL wrapper, we can use `R Submit(EvalInsert("\[____]\"))` in place of the simple `R Submit("____")`, as you'll observe in the following R code, where we use the `ggplot2` package in R to create a histogram:



This `EvalInsert()` is also generally helpful when you need to make substitutions or evaluate expressions inside a character string in JSL. For example, we could specify the bin width for the R histogram as a variable in the JSL script and then call that variable using the `EvalInsert()` code, like this:



```

dt = current data table();
dtmat = dt<<Get as matrix;

R Init();
R Send(dtmat);
R Submit("RlogRunTime<-log(dtmat[,6])");
R Submit("hist(RlogRunTime)");

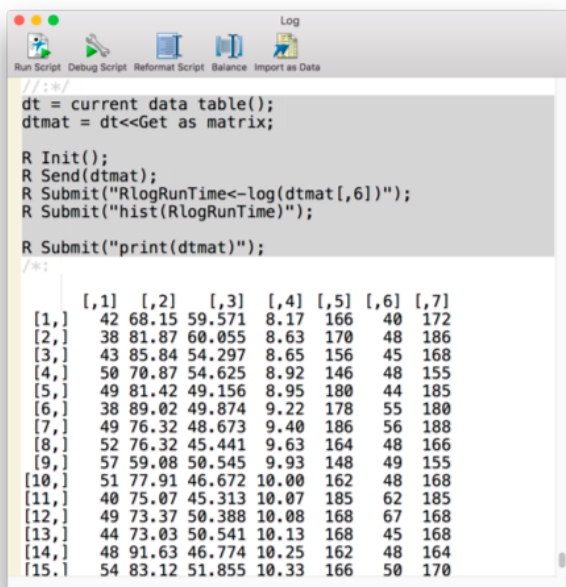
R Submit("print(dtmat)");

bw = 0.05;

R Submit(EvalInsert("\[
  library(ggplot2)
  qplot(RlogRunTime, geom='histogram', binwidth=^bw^
)\"]);

```

Monitor your code submissions and troubleshoot, if needed, by using the JMP log window. Go to the JMP menu bar and choose Window → Log to open the JMP log. Here you can see any internal R error messages or results.



```

// **
dt = current data table();
dtmat = dt<<Get as matrix;

R Init();
R Send(dtmat);
R Submit("RlogRunTime<-log(dtmat[,6])");
R Submit("hist(RlogRunTime)");

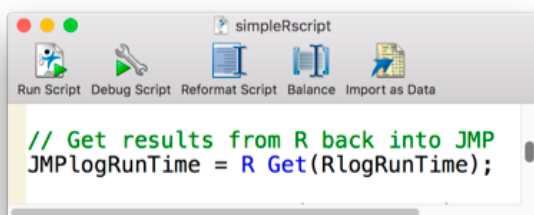
R Submit("print(dtmat)");
// **

```

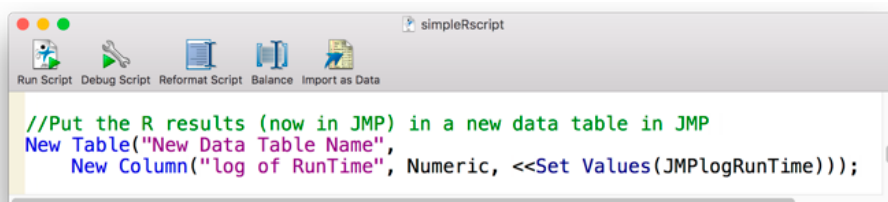
	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]
[1,]	42	68.15	59.571	8.17	166	40	172
[2,]	38	81.87	60.055	8.63	170	48	186
[3,]	43	85.84	54.297	8.65	156	45	168
[4,]	50	70.87	54.625	8.92	146	48	155
[5,]	49	81.42	49.156	8.95	180	44	185
[6,]	38	89.02	49.874	9.22	178	55	180
[7,]	49	76.32	48.673	9.40	186	56	188
[8,]	52	76.32	45.441	9.63	164	48	166
[9,]	57	59.08	50.545	9.93	148	49	155
[10,]	51	77.91	46.672	10.00	162	48	168
[11,]	40	75.07	45.313	10.07	185	62	185
[12,]	49	73.37	50.388	10.08	168	67	168
[13,]	44	73.03	50.541	10.13	168	45	168
[14,]	48	91.63	46.774	10.25	162	48	164
[15,]	54	83.12	51.855	10.33	166	50	170

Of course, we can also create this histogram in JMP using the Distribution platform. We can choose to use JMP functions in two ways, as follows.

First, we need to bring the **"RlogRunTime"** back from R to JMP. This section of code takes the newly transformed data, called **"RlogRunTime"** in R, and brings it back to JMP, with the new name **"JMPlogRunTime."**



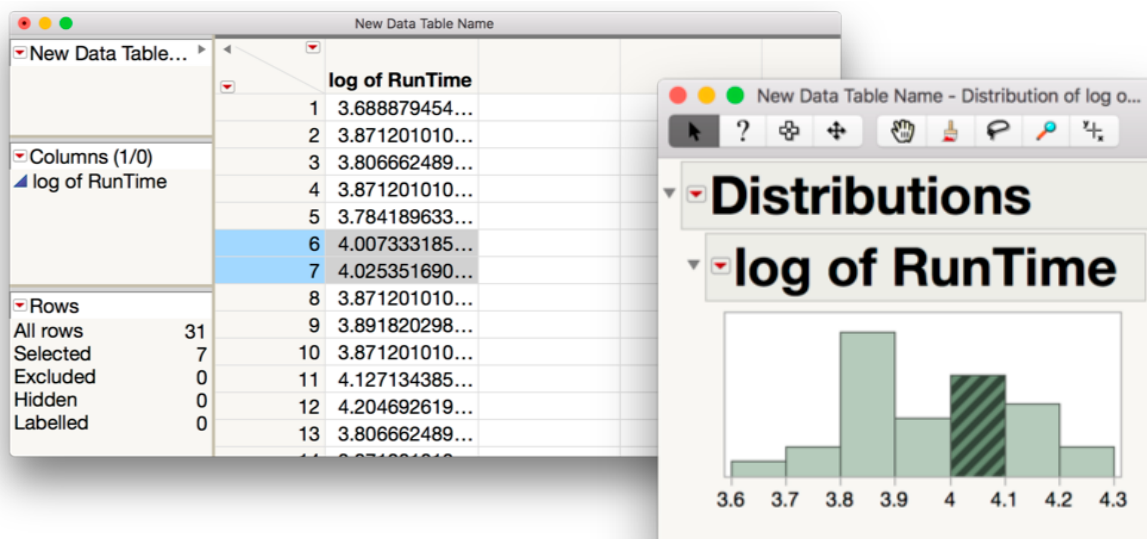
We also need to move this new JMP data from the internal memory to a new data table in JMP so that we can save it and do further operations on it. The code below creates a new JMP data table with one column, composed of the data from **"JMPlogRunTime."**



We can then choose to use further JMP functions in either of two ways, as follows.

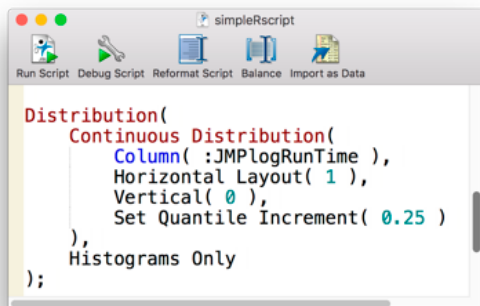
- (1) Operating on the new JMP data table, use JMP functions from the point-and-click menus.

In our example, we can use the Analyze > Distribution option to create an interactive histogram in JMP.



(2) Use JSL to continue working in a script window to call those same point-and-click JMP functions.

If you are familiar with JSL, you can write more JMP actions directly in your script window. If you are unfamiliar with the JSL that corresponds to your point-and-click actions, you can easily generate that JSL by using point-and-click to create the actions once. Then click the red triangle from the results window and choose Copy Script to Script Window. To update the script for a new data table, edit the JSL script to apply it to the appropriate variable(s).



## Additional Simple JMP With R Examples

You can find more simple examples of how JMP and R work together by searching “R” in the JMP Help menu and under the Scripting Guide → Extending JMP → Working with R.

In the JMP Help you will find code for:

- Sending a Data Table to R.
- Creating Objects in R.
- Using R Functions and Graphics.
- Simple Matrix Addition in R.
- Getting Bootstrapped Confidence Intervals in R.

## Advanced Examples

Once you understand the basics around using the JMP interfaces with R and Python, you can extend this in interesting and interactive ways.

Here are links to a few examples:

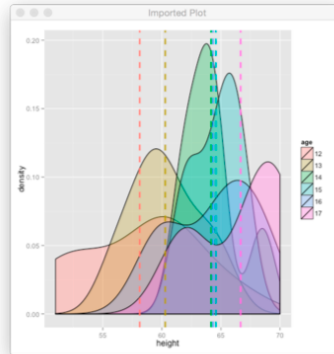
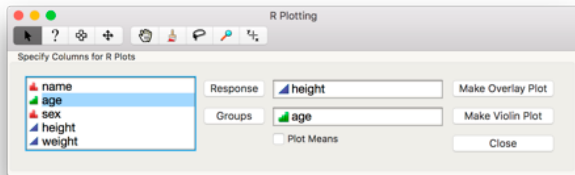
### JMP to GGPlot Interactive Caller

This script utilizes the JMP connection to R to create smooth-curve density plots<sup>7</sup> with the R package ggplot2,<sup>8</sup> broken apart by the levels of a grouping variable.

<sup>7</sup> Note that this type of histogram view is also a native option in the JMP Graph Builder, starting in version 15, so the use of R in this example is for illustration rather than necessity.

<sup>8</sup> See [cran.r-project.org/web/packages/ggplot2/index.html](http://cran.r-project.org/web/packages/ggplot2/index.html).

To execute, choose a data file and then open and run this script, which opens a dialog box filled with the columns from the open data table. By interacting with this prompt, the resulting R plot is created.



To use this script, you do not need to write the R or JSL code. However, if you want to create something similar, the underlying JSL (containing snippets of R code) is shown here (downloadable from the link to this example):

```

/* Plot Functions with R v5
Zellian L. Parriss Ph.D.
2020
*/

// Expressions List
LoadExprs = expr(
  name,
  age,
  sex,
  height,
  weight
);

// MakePlot = expr(
  yy = year <- Get Items;
  dt = dt <- Get Items;
  //Name = R JMP Name to R Name yy[1];
  //Name = R JMP Name to R Name dt[1];
  dt = Current Data Table();
  //Based on checkbox, either plot with or without mean lines
  if (cb <- Get == 1, DensityWithMeans, DensityNoMeans);
);

MakePlot = expr(
  yy = year <- Get Items;
  dt = dt <- Get Items;
  //Name = R JMP Name to R Name yy[1];
  //Name = R JMP Name to R Name dt[1];
  //Print to console
  dt = Current Data Table();
  R Submit[["sample.violinplot("Name" = "Name",data=dt, col = "lightgray")"]];
);

```

```

DensityWithMeans = expr(
  //Calculate means for y based on grouping variable provided
  R Submit[["cat <- dt[yield, summarize, "Name"=meanmean("Name")]"]];
);

//Generate Plot
R Submit[["cat <- dt[yield, summarize, "Name"=meanmean("Name")]"];
//Plot
ggplot(dt, aes(x="Name", fill="Name")) + geom_density(alpha=.2) +
  geom_vline(data=dt, aes(xintercept="Name", y=0, colour="Name"),
    linetype="dashed", size=1);
];

DensityNoMeans = expr(
  R Submit[["cat <- dt[yield, summarize, "Name"=meanmean("Name")]"];
  //Plot
ggplot(dt, aes(x="Name", fill="Name")) + geom_density(alpha=.2);
];

CapturePlot = expr(
  JMP_Plot = R Get Graphics(png);
  New Window("Imported Plot", Picture Box(JMP_Plot));
);

RunClose = expr(
  JMain <- Close Window; // close window
  R Terminate; //Close R connection
);

```

```

LoadExprs; //load all the expressions that reside at the end of the file
JSL; //Start R Connection and load libraries

// Prompt for data table if none is open
if(IsEmpty(Current Data Table()), dt = Open(), dt = Current Data Table());

//Draw Window and Create Fields and Buttons
JMain = New Window( //Draw New Window and name JMain
  "R Plotting", //Window Title
  Panel Box("Specify Columns for R Plots", //Draws visible box
    //List Box //Visible box for layout - glues horizontally
    cList = Col List Box All, //Column listbox, with name cList so we can access later
    // Draw First Lineup Box (available, provides arrangement) for Selecting Columns
    LBox1 = Col List Box All, //With 2 columns indicated, each pair will open a row
    //First Row
    Button Box("Response", yvar <- setItems(cList <- Get Selected)),
    yvar = Col List Box All, //With 1 column indicated, each pair will open a row
    //Second Row
    Button Box("Groups", xvar <- setItems(cList <- Get Selected)),
    xvar = Col List Box All, //With 1 column indicated, each pair will open a row
    //Third Row
    Spacer Box(10, 10),
    cb = Check Box("Plot Means", <=set(0)),
  ),
  // Draw Second Lineup Box for Buttons
  LBox2 = Col List Box All,
  Button Box("Make Overlay Plot", MakeOverlayPlot),
  Button Box("Make Violin Plot", MakeViolinPlot),
  Button Box("Import Plot to JMP", CapturePlot),
  Button Box("Close", RunClose)
);
//Close Lineup Box
//Close R List Box
//Close Panel Box
//Close New Window

```

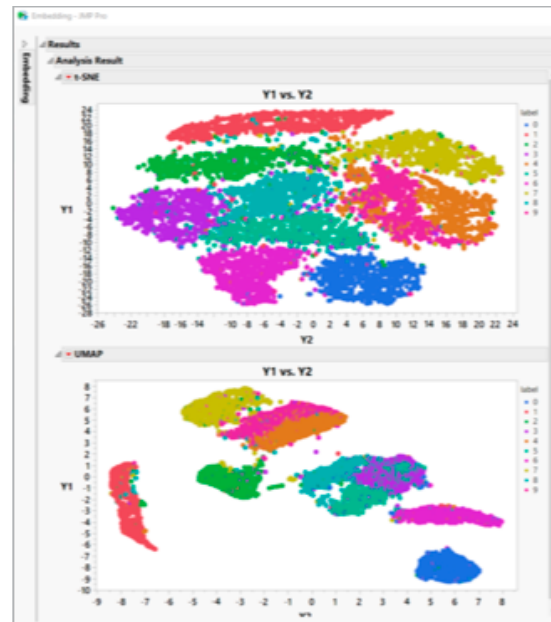
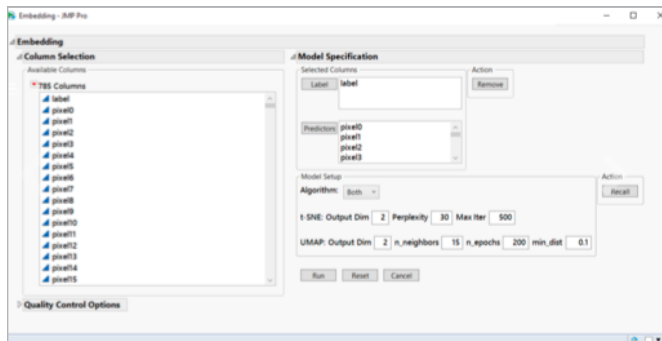
To learn more, and to download this add-in, visit the JMP Community at [community.jmp.com/docs/DOC-6472](https://community.jmp.com/docs/DOC-6472).

## Data Visualization With t-SNE and UMAP Add-In:

This script, which makes use of the t-SNE and UMAP packages<sup>9</sup> in R, is made even more user-friendly when packaged in a JMP add-in.

The t-Distributed Stochastic Neighbor Embedding (t-SNE) and Uniform Manifold Approximation and Projection (UMAP) are nonlinear dimension-reduction and visualization algorithms becoming popular in fields like image processing, text mining and genomics. This add-in provides a user-friendly interface to these two R packages, enabling data table navigation, data quality control, sparsity handling, intuitive parameterization and interactive results interpretation.

<sup>9</sup> See [cran.r-project.org/web/packages/Rtsne/](https://cran.r-project.org/web/packages/Rtsne/) and [cran.r-project.org/web/packages/umap/](https://cran.r-project.org/web/packages/umap/).



When creating an add-in like this, you simply need one user to develop the R or Python code, plus the JSL code that wraps around it. This user would then wrap it all up in a JMP add-in using the JMP Add-In Builder.<sup>10</sup> The resulting .jmpaddin file can either be emailed to other users or posted on the web.<sup>11</sup> This process allows subsequent users to run the add-in without needing to write, view or edit the code – instead, interacting with the easy-to-use point-and-click display of JMP.

The code that we are not seeing, which is running in the background, begins like this (R code is in purple):

```

Name: Embedding
Description: This is an add-in that provide access to t-SNE and UMAP R packages.
It has enables basic quality control, sparsity handling, paramater specification,
and result interpretations.
Author: Meijian Guan
Version: v1.2 3/14/2019
Changelog:
v1.2: Fixed a bug for Rtsne package where too many columns would cause stack overflow.
v1.1: Fixed a bug that could cause "Issues found in R..." error message. Fixed a bug when Both algorithms are selected and no label is selected.
v1: Initial version

Names Default To Here(1):
include("JSL_Utils.jsl");
if(not(is(windows),_slash_="/" ));
_addinPath = Get Path Variable("$ADDIN_HOME\com.jmp.embedding");
if(not(is(windows),_addinPath = Convert File Path(_addinPath,windows)));

//Clear Log();
//close all Data Tables, NoSave();
//namespace("here")<-remove(namespace("here"))<-getkeys();

//label=labelY;
//label=();
//data4R=getDataR;
//algrth=algr;
//algrth="Both";
//data2R=InDataPrep(inData,predictor,labelY);

//mtx2R=mtx2R<-Get as Matrix;
//mtx4R=Sparse SVD(mtx2R,20);
//svd=mtx4R[1];
//data4R=asTable(svd0,<<invisible);
//dim(data4R);
//A function to talk communicate with R: send script & data to R, get result table back.
talk2Rfunction(inDataR,label,algrth,dim=2,perplexity=2,iter=500,n_comp=2,n_neib=15,n_epch=200,dist=0.1),
(Default Locals),

labelText="";
if(nitems(label)>1,
  labelText=eval insert("["
    labelText=eval insert(names(inData4R)) %>% label
    labelText=as.List(labels)
  ]\n");
  labelText="";
);
Print(algrth|" is selected!");
Rtext=eval insert("\n
data4R=asTable(svd0,<<invisible)
label=unlist(label)
cat("label is ",label,"\n")
#print(length(label))
#cat("\n")

#remove duplicated observations from both datasets
inData4R=inData4R[!duplicated(data4R[,names(data4R)]),] #allow excluding multiple labels
#head(data4R)
cat("dim of inData4R is: ",dim(inData4R),"\n")
labelText=
cat("Ready for Run","\n")

if(algrth=="t-SNE"){
  cat("We are running t-SNE","\n")
  library("Rtsne")
  inData4R=asTable(svd0,<<invisible)
  cat("dim of inData4R is: ",dim(inData4R),"\n")
  tsne <- Rtsne(inData4R, dims = "dim", perplexity="perplexity", verbose=TRUE, max_iter = "iter",pca=F)
  outputY=tsne$Y
  head(outputY)
}else if(algrth=="UMAP"){
  cat("We are running UMAP","\n")
  library("umap")
  inData4R=asTable(svd0,<<invisible)
  cat("dim of inData4R is: ",dim(inData4R),"\n")
  umap <- umap::umap(inData4R, min_dist="min_dist", n_neighbors="n_neighbors", n_epochs="n_epochs",
    outputY=umap$embedding
  )
  head(outputY)
}

```

<sup>10</sup> To learn more about how to create a JMP add-in from your JSL script, see [jmp.com/content/dam/jmp/documents/en/academic/learning-library/01-add-in-builder.pdf](https://jmp.com/content/dam/jmp/documents/en/academic/learning-library/01-add-in-builder.pdf).

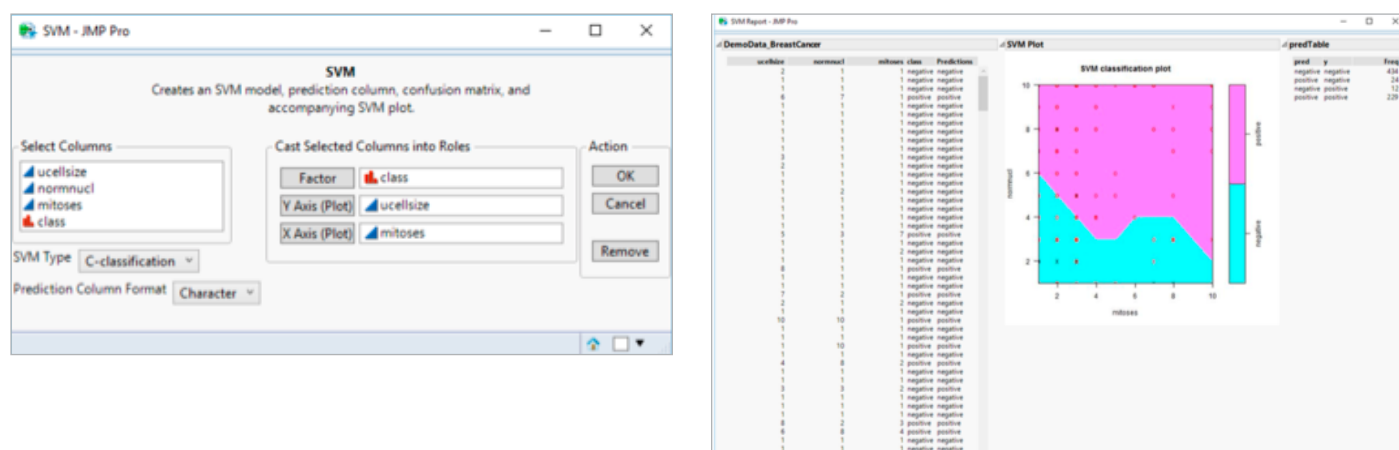
<sup>11</sup> Post it in our JMP User Community at [community.jmp.com](https://community.jmp.com) so that other users can find it.

Once written, subsequent users won't need to see or interact with this code at all. This feature allows you to write it once and wrap it in an add-in. Now you have a point-and-click interface to call R or Python functions.

To learn more, and to download this add-in, visit the JMP Community at [community.jmp.com/t5/JMP-Add-Ins/Data-visualization-with-t-SNE-and-UMAP/ta-p/177969](https://community.jmp.com/t5/JMP-Add-Ins/Data-visualization-with-t-SNE-and-UMAP/ta-p/177969).

SVM Add-In:

This add-in calls the svm function from the e1071 R package.<sup>12</sup> The user of the SVM Add-In interacts with this dialog box to get the resulting SVM<sup>13</sup> classification report:



What the user does not see is the code running in the background:

```
library(e1071)
library(gridExtra)

theText <- paste("x <- subset(dt, select=", colnames(factor)[1], ")")
eval(parse(text=theText))
y <- factor
svm_model <- svm(x, y, type=svmType)
modelText <- paste("second.svm <- svm(", colnames(factor)[1], " ~ ., data = dt)")
eval(parse(text=modelText))

summary(svm_model)
summary(second.svm)
pred <- predict(svm_model, x)
pred <- as.vector(pred)

if(is.list(y)) {y <- unlist(y)}
predTable <- table(pred,y)
predTable <- as.data.frame(predTable)

eval(parse(text=paste(colnames(y_var)[1], "<- as.vector(as.matrix(y_var))"))
eval(parse(text=paste(colnames(x_var)[1], "<- as.vector(as.matrix(x_var))"))

max1 <- eval(parse(text=paste("max(", colnames(y_var)[1], ")"))
max2 <- eval(parse(text=paste("max(", colnames(x_var)[1], ")"))
gridSize <- eval(parse(text=paste("max(c(", max1, ",", max2, ")"))

eval(parse(text=paste("plot(second.svm, dt, ", colnames(y_var)[1], " ~ ",
colnames(x_var)[1], ", fill=TRUE, grid=gridSize)"))
```

Creating this add-in is even easier using the JMP to R Add-In Builder (see the next example). The JMP to R Add-In Builder allows you to skip the JSL coding entirely and simply interact with some dialog boxes, dropping in R or Python code in the prompt box.

<sup>12</sup> See [cran.r-project.org/web/packages/e1071/index.html](http://cran.r-project.org/web/packages/e1071/index.html).

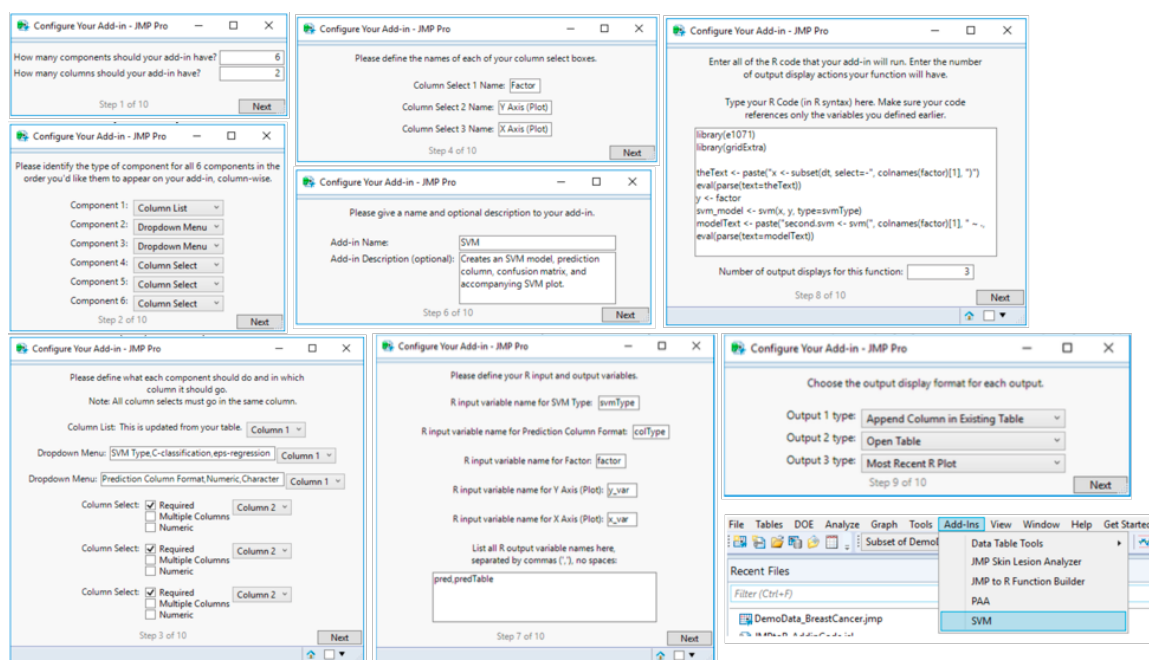
<sup>13</sup> JMP Pro added SVM to the predictive modeling suite in version 15, so, again, this example is illustrative of the power of the connection to R but is not needed for JMP Pro 15 users.



## The JMP to R Add-In Builder: Using the JMP to R Connection Without JSL

This tool allows users to define and use custom JMP add-ins that execute R functions. To use this interface builder, you first need to develop the R code that you want to send to R, but you do NOT need to write any JSL code. Similarly, end users of your new point-and-click interface will not need to use or see that R code – they will just see the easy-to-use, point-and-click interface. (Note: This add-in only works on Windows OS, not on Mac.)

This add-in allows you to skip the JSL part of using the JMP with R connection. Instead, you'll interact with a few dialog prompts to create the boxes and buttons and prompts that you want to include in your add-in. Most steps are shown here:



The SVM Add-In example above was created using this JMP to R Add-In Builder.

To learn more, and to download this JMP to R Add-In Builder add-in, visit the JMP Community at [community.jmp.com/t5/JMP-Add-Ins/The-JMP-to-R-Add-In-Builder/ta-p/43879](https://community.jmp.com/t5/JMP-Add-Ins/The-JMP-to-R-Add-In-Builder/ta-p/43879).

## Conclusion

In this guide, we introduced the Python scoring-code generation, the Python in JMP scripting, and the R in JMP scripting. We also provided links to further examples, including some advanced R examples that make use of the Add-In Builder feature in JMP. For more information on any of these topics, see the footnotes and links provided throughout this paper. If you have questions, need help implementing any of these techniques or just want to share a discovery, post to our JMP User Community at [community.jmp.com](https://community.jmp.com).

## About SAS and JMP

SAS, the world's leader in analytics, created JMP (pronounced "jump") in 1989 to empower scientists, engineers and other data analysts to explore and analyze data visually and interactively. Since then, JMP has grown from a single product into a family of statistical discovery tools, each one tailored to meet specific needs. John Sall, SAS co-founder and Executive Vice President, leads the JMP business unit.



SAS Institute Inc. World Headquarters

+1 919 677 8000

JMP is a software solution from SAS. To learn more about SAS, visit [sas.com](http://sas.com)

For JMP sales in the US and Canada, call 877 594 6567 or go to [jmp.com](http://jmp.com)

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. 111094\_G118201.0120