Advanced JMP Scripting: Enabling Consistent and Dynamic Analysis Using Multiple Modeling Platforms and Integrated Diagnostics

Author: Andrew Parker

Contributors: Robert Grube, Joshua Kendoll

Responsible Organization:

Supplier Management Finance – Joint Supply Cost Model Team

***Abstract* – As data accessibility increases, consistent and dynamic processes for data analysis have become increasingly critical. JMP Scripting Language is an essential tool in ensuring standards of analysis and creating supporting visual tools to enable multiple analysts to consistently repeat proven analytical methods in a common manner. Creating an advanced script can allow a team to integrate multiple platforms with diagnostic tools in order to transfer seamlessly between different types of related analysis.**

Building predictive cost models is a popular tool in manufacturing used for negotiations, design for cost, estimate check-fixtures, etc. These models are commonly built in the Fit Model platform using either simple regression or the Stepwise application. However, as more data is available, the analysis can grow more complex, requiring the use of the Nonlinear platform. Complex, proprietary formulas must be created by hand, as they do not exist in the JMP model library – a time-consuming and error-prone task. Proper scripting increases efficiency and reduces errors; a click of a button can transfer a model between platforms with ease and accuracy. Integrating diagnostic tools alongside these platforms allows for more time doing value-added analysis instead of tedious tasks.

This paper will provide information on how the integrated platform came to be created and how it is continuously being improved upon as is needed by the user community. First, however, the background of the team's purpose and process will be given to better aid in understanding the evolution of team's needs and how the script fulfills them.

I. Background

A. Business Framework

The group is the Joint Supply Cost Model (JSCM) team. It resides within the Supplier Management Finance (SMF) organization in the Boeing Commercial Aviation (BCA) business unit. The team works with similar organizations in the Boeing Defense, Space, and Security (BDS), Engineering Operations and Technologies (EOT), and Shared Services Group (SSG) business units.

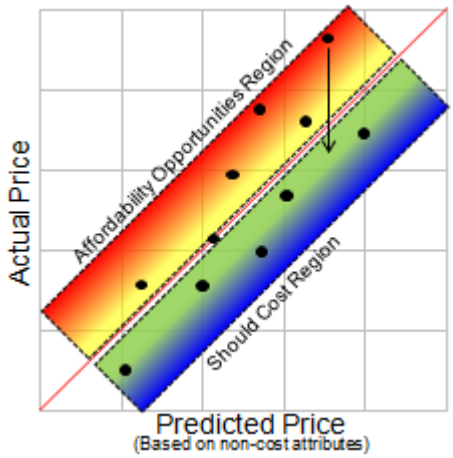


Figure 1. Theoretical Actual vs. Predicted Graph

This enterprise level effort focuses on building predictive models to help identify potential opportunities within the Boeing supply chain. Opportunities in this case are parts, products, or commodities which reside above the line of best fit (Fig. 1) which represents the market average for the commodity. The goal is to find parts which are above average in the marketplace and bring them down to a more competitive price. These models are utilized by finance analysts, engineers, and procurement agents for a multitude of purposes including negotiations, design-to-cost, procurement authority, and as a check-fixture against other estimating tools.

Overall, each model requires the engagement of engineers, finance, and a model focal to develop the framework of the model's definition, collect data, and then build and deploy the final product. This level of cross-functional effort requires a rigorous process to be followed from start to finish which is closely monitored by managers and directors. The expectation has been levied upon the team that it will complete the initial list of models by the end of second quarter in 2015.

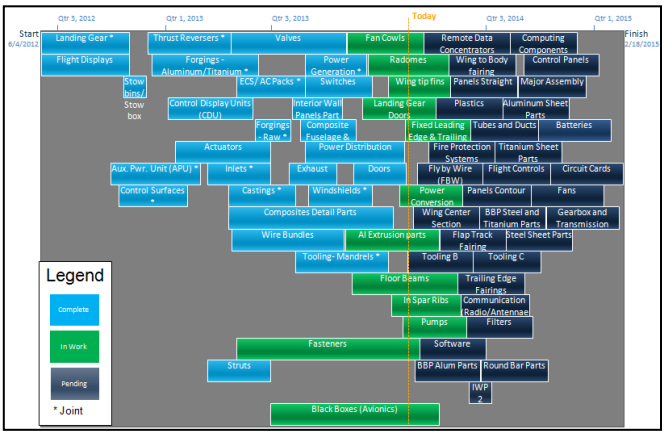


Figure 2. Project Status Visual Aid

Each box in figure 2 represents an individual commodity to undergo the JSCM process and the length of the box determines its relative duration. It also illustrates the build-up of projects in work and the need to adhere to timelines to complete all projects by the given deadline.

B. Process Framework

The JSCM process is schedule focused, well-defined, and repeated frequently. The benefit of such an approach is to provide assurance to our customers that the models produced are reliable and of a consistently high quality. There are 10-12 projects being worked simultaneously which may all be in different stages of the process.

The process follows a standard flow consisting of several key steps. Firstly, initial research ensures the modeler has an adequate knowledge in order to better communicate with the technical and financial community who are intimately involved with a product. Then the customer is engaged to help define the scope of the commodity's definition. This definition is then taken to the technical community who help to determine technical attributes which attribute to the overall cost of the product. Once the attributes are defined and the population is established, data collection commences. Finally, JMP is brought in to perform the data analysis with the goal of producing a model which encompasses data from across the enterprise. After a rigorous peer and management review, the results are then published for use.

II. Modeling Effort

This section will focus on the two most time consuming efforts in the modeling process and the issues concerning them.

A. Data Collection

This phase is the most time consuming step. Routinely, model focals are faced with collecting information from multiple databases or through a second party like an engineer collecting weight or performance information. Regardless of method or collector, many databases do not contain raw data and the collector must manually review documents to extract the raw data for each part.

Populations for these models can range from six to several thousand observations and the attributes collected generally range from 20 to 50 for smaller populations and five to 10 for larger ones. The default process is to collect a complete view of the total population. However, given the time-consuming nature of data collection, populations with over 100 unique observations are sampled. Uniqueness in the manufacturing environment can be difficult to determine due to contract decisions and reuse of technically identical parts on both the left and right hand sides of an aircraft. Individual part numbers must be checked for uniqueness in order to prevent attribute correlations from being unduly inflated or influenced.

B. Model Development

Once the population and attributes are collected from all business units, the dataset is checked for missing information and errors and then copied into JMP. From this point on there is a standardized method for progressing through data analysis.

Firstly, a univariate analysis is performed on each of the variables to check for convention/spelling issues (for categorical variables), distributions for potential transformation, proper variability within the data, and for data completeness. Secondly, we investigate the basic relationships between all of the potential variables. This includes using the Multivariate platform to check piecewise correlations visually and numerically. Graph Builder is also utilized to step up the bivariate analysis to start investigating potential interactions or extending a relationship to more than two variables. Finally, the Stepwise platform is utilized to investigate the potential variables and theorized interactions.

At this point there are several complicating factors concerning the analysis. The full extent of them will be addressed later; however the general concepts will be noted here. Nonlinear exponents are added to some variables which demonstrate a curvilinear relationship with the dependent variable. These would be non-standard exponents which depart from traditional square root, squared, or inverse relationships. A relationship may also be theorized to occur with the average annual quantity purchased and how that impacts cost. This theory has a simple logarithmic form or a more complex quantity discount formula[1]. Both the latter and the nonlinear exponents require fitting in the Nonlinear platform. In order to move from the Stepwise platform into the Nonlinear to investigate these improvements requires manual transfer of the desired attributes and the building of formula/parameters in between.

Additionally, every potential model must undergo diagnostic testing and analysis considering Studentized

[1] The formula is a Boeing proprietary method of applying EOQ theory into JMP. It allows for the parameterization and application of the general theory of amortization of fixed costs across variable costs to get unit costs. This is not in the Model Library.

residuals, hats, and Cook's Distance measures for influence and outliers. Correlations must be considered to determine the degree of multicollinearity present. Tests for statistically significant improvement if utilizing degrees of freedom to estimate exponential and quantity discount parameters must also be performed. These considerations are just some of the many tests and diagnostics performed regularly by a model focal. Analysis of these diagnostics and tests requires utilizing one platform's output to be analyzed in another.

Overall, this process is extremely iterative, the movement between analysis platforms is a very laborious process and prone to errors. This step was originally allotted an amount of time which was later found to be very difficult to perform an adequate analysis within especially for newer analysts who are not as adept at navigating JMP.

C. Review, Approval, Distribution

Once a potential model has been vetted statistically, the proposed final model must be reviewed with the technical and finance stakeholders. This step ensures that the customers are confident in the model and will use it, reducing the concern of investing resources into the creation of a tool that will not be used. Much consideration is given to the final attributes which comprise the final product, and many times the strongest statistical model is passed on in favor of more technically sound attributes.

Once a technical review has been performed, the JSCM team comes together to perform a peer review. During this phase, the whole process as outlined to this point gets reviewed and questioned by representatives from any business unit which is represented in the model. The model definition, data collection, and final model must be critiqued in order to catch any problems prior to publishing. The peer review process also entails reviewing the tool for non-JMP users and the white paper documentation which gives a detailed background on the model.

Finally after a successful peer review, the model is reviewed with management. Ideally, initial reviews are thorough enough that the model only need be reviewed for compliance and given final approval before getting distributed. All of the final products should be completed and approved at this point.

Completed models are housed on a web-portal where the Excel based tool and supporting documentation can be accessed by all finance employees. Due to data sharing restrictions, engineers are subject to more controls which inhibits open access to physical files, but they do have access to the web-portal where they can see the tools available and request additional information.

III. Lean Efforts

A. General Improvements

The need to improve upon the modeling process is paramount to meeting the team's committed schedule. While the team is engaged in general lean efforts to reduce defects and improve speed and accuracy in the whole process, major improvements are still needed to resolve some of the recurring problems in the process.

Simple improvements like templates, guidebooks, checklists, file standards, and many others have continually been implemented to enhance productivity through reduced rework. These have netted many gains in efficiency and ease of communicating final products, but have not been able to provide a step-function improvement of schedule relief for the team.

As a result, bigger projects were undertaken. The largest effort involved the teams from BCA and BDS to critically analyze the current state process and attempt to solve the most critical problems which resulted in large improvements to the process. While this addressed inefficiencies in the process itself, the issue of improving productivity still had not been addressed and the schedule remained at risk.

B. Need – Improved Productivity

Through analysis of the JSCM process, the modeling effort was identified as having the most promise for productivity improvement. This decision was based on the modeling effort's iterative nature, consistently used diagnostics, and requirements for standard report generation.

Scripting was determined to be the most effective method for achieving those improvements. With scripting, the modeling process can automate most repeated functions, reports, and graphics and allow the modeler to focus on making critical decisions about the reports in lieu of creating them. This would ensure better first pass quality, greatly reduce overall cycle time spent in the step, and enable consistently standard results.

So an effort to build upon a previous team's[2] add-in (Grube's Integrated Stepwise Tool – GIST) was initiated. This script contained most of the necessary elements to improve the JSCM team's overall productivity, but did need to be built upon to cover all of the team's requirements. The following sections provide an integrated view of the overall process of development to improvement to provide a contiguous story.

IV. Solution – JSL Scripting

A. Itemize Needs

The first thing the team needed to do is identify the needs of the group: What are the essential functions and outputs required to perform a nearly complete analysis? This list is a collaborative effort of the whole team. Both subject matter experts and new modelers must contribute to the overall list of items which guide an analysis to prevent the new platform

[2] The previous team was started in 1998 under an organization called Global Partners and started using JMP in 2003. This team was disseminated into other organizations until a revival occurred for a similar type of analysis which evolved into the JSCM team under Supplier Management.

from becoming too oriented towards one type of user or the other.

Below is a breakdown by the eventual section of the items which were determined to be necessary:

Stepwise Platform – This platform would be the foundation of the new platform. However, the usability of the platform needed enhancement. The team decided to implement better tools to add and remove columns. They also requested better sorting functions than the 'sort by column' capability naturally in the platform.

Dynamic Diagnostics – Since the platform's primary function is to provide a faster way to analyze a model, eliminating the need to jump from platform to platform to retrieve information was necessary. The diagnostics which comprise the basic analysis of a potential model needed to include numeric and visual representations, but also needed to be aggregated into a simple display. Some diagnostics were determined to only require numeric representation and a check of an Actual vs. Predicted plot to verify. The numeric diagnostics included coefficient analysis, prediction vs. actual comparisons, maximum attribute correlation, and variance of attributes.

Correlations		Predictions		Diagnostics	
Max X Corr.:	0.159697	Min Prediction:	71.699	N:	48
Attribute 1:	Non-Linear Attribute-X	Min Actual:	16.77	Mean pred:	266.44
Attribute 2:	Technical Attribute 3	Y's < Min Y-hat:	11	RMSE:	93.93
Condition:	1.175	Pred's < 0:	0	Coeff of Var:	0.352527
Min Non-Modal % :	0.375 = 18 obs.	Coeff's < 0:	0	Correlational r^2:	84.6
Attribute:	Technical Attribute 3	Y:Y-hat max	4.356	Mean observed:	266.44

Figure 3. General Summary Information & Diagnostics

For more complex diagnostics, such as Studentized residuals, hats values, and Cook's Distances it was necessary to provide a better visualization of the numeric outputs to allow insight and proper comparisons. Two items were included for this; something to measure and organize violators and another to visually represent the relationship between Studentized residuals and hats values with Cook's Distances.

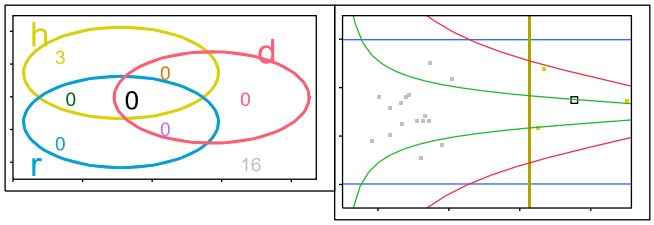


Figure 4. Leverage Summary and Leverage Plot

Figure 4 shows the result of the organization. A Venn diagram summarizing counts of threshold violators (which can be set anywhere) and a two dimensional plot which utilizes the relationship between residuals, hats, and Cook's Distance for individual points (Equation 1) to analyze a lot of information in a simple manner.

$$D_i = \frac{e_i^2}{p * MSE} * \frac{h_{ii}}{(1-h_{ii})^2}$$

Equation 1– Cook's Distance

Nonlinear Platform – This need requires integrating a second platform alongside the first. The Nonlinear platform needed to be able to capture exponents of an attribute and to build a formula to handle the Boeing quantity discount parameterization. This would then be fed into the Nonlinear platform and loaded into a platform which would provide much of the same diagnostics mentioned before. Communication with the Stepwise platform was necessary to draw in the correct variables and to return modified exponents. It would also need to overcome the limits of the Nonlinear platform's reporting of goodness of fit measures. The need for model comparison would also be a critical feature in comparing improvements using Nonlinear regression against the degrees of freedom used, namely the General Linear Test (Equation 2).

$$F^* = \frac{\frac{SSE_{red} - SSE_{full}}{d.f.(SSE_{red}) - d.f.(SSE_{full})}}{\frac{SSE_{full}}{d.f.(SSE_{full})}}$$

Equation 2. General Linear Test

B. *Build & Implementation*

Once the team finalized the requirements, the build of the architecture could begin. The simplest approach was a white-board layout of the graphic user interface (GUI). The layout could be set simply by using dry-erase pens or sticky notes so the structure can be modified easily. It is essential to finalize the structure prior to the coding effort. When needing to use Report and Get as Matrix type commands (See Appdx A)[3] the target must be specified from the tree structure created in the platform layout. Changing the structure after creating targets may cause an overall change in the tree structure requiring time consuming investigation and retargeting.

Once a GUI has been finalized (See Appdx B), the object oriented style of JSL script can be utilized in conjunction with its ability to manage procedural[4] design to make the daunting task of programming each requirement into a much more manageable structured effort. As a reminder, the essence of object oriented programming is that pre-made objects such as graphics, data tables, and data columns have predetermined properties and attributes which can be accessed via script or by other objects. Procedural design provides the user a framework to essentially create objects or to create abilities for objects. In this new platform, there is very little code not part of a procedure (typically a function), this code is usually the establishment of global variables or some read in scripts (See Appdx A).

The advantage of this style of code is that the buildup of code can be done in steps. For a complex GUI window, the

[3] There are multiple targeting techniques in JSL now; however, the same problem can occur when directly targeting box type and number.

[4] This combination is sometimes classified as a dynamic programming language.

best starting point is to lay out the box structure. H List Box, V List Box, Outline Box, Button Box, and Text Box (See Appdx A) are the starting tools to figure out where the eventual code or functions will be placed. This is another opportunity to draw out the box structure using a physical layout; denoting "H" and "V" boxes to split a window with "O", "B", and "T" boxes denoting content or outputs.

Once this window object has been created, the task of populating each requirement becomes a simplified process of implementing pieces of code. In the case of the JSCM team, the initial starting point was implementing the Stepwise platform. This required building a Column Dialog (See Appdx A) which collects the response variable and the candidate independent attributes. Once collected, a script was built to pass the selected list of column objects into one continuous string. This string combined with non-parsed text to round out the necessary inputs to allow parsing into a new line of code which could evaluate into a Stepwise platform containing the information collected from the Column Dialog.

This new add-in can be run as-is and, after the prompt for column inputs, the GUI code would build a working Stepwise platform with some surrounding boxes and buttons. From this point, additional scripting can be done to set certain aspects of the Stepwise object like hiding unnecessary information, changing Stopping Rule and Prob to Enter/Leave properties, and even automatically running if desired. This update could then be released which would incrementally improve the add-in for the team. While this example is a bit trivial in content, the capability of rolling out continual updates is essential in development. Eventually more functions will be built and added to buttons and display boxes, however the development of each function does not prevent the team from using an 'incomplete' add-in.

Scripting within JMP enables a programmer to utilize the existing add-ins script/functions while building and testing code independent of the main body of code. This is done through the use of scripting windows. With multiple script windows, the main body of code can be compiled into JMP. Once that is done, those global variables can be used in any other script window. For example, when a dataset is opened and GIST is run, the code extracts the selected independent variable columns and associated data. When the 'max X correlation' calculations needed to be implemented, this required building a correlation matrix and then cycling through to find the largest absolute value (non-intercept) and track the associated independent variable names. This was not easily scripted, but with GIST having been run in the background, the global variables for the vector of independent variable names and data were accessible. This new piece of code was built in a separate temporary JMP scripting window. Scripting and debugging could be done without potentially overwriting code or creating an issue with another analyst attempting to load an incomplete GIST code. Some functions, like the quantity discount functionality, can take months to properly implement, but, with the object oriented approach and the procedural design, progressive updates are worked and released without having to stop use of the add-in.

Another aspect of JSL scripting is the ability to program multiple JSL files to be compiled together at the same time (See Appdx A). This enables multiple files to be worked at the same time allowing an experienced programmer to build more complex functions and objects while a novice scripter works in another area to improve the appearance. Multiple files allow for faster rollouts of improvements and better organization of scripts.

For the team's usage, these files are housed on a common server and are routinely accessed by 10-12 users multiple times a day. JMP doesn't maintain direct links to scripting files. The file is accessed, read, compiled, and the functions and processes are then stored for future use. This common source ensures configuration control for the programming team and enables simple separation of development files from production files.

With the development of GIST's original layout complete, the JSCM team is capable of performing many tasks without having to manually jump between platforms to perform diagnostic analysis on potential models. Enhancements beyond the standard commands for the Stepwise platform enabled more control when sorting and adding/removing columns. The integration of the Nonlinear platform and the auto build of the standard exponential and quantity discounting formulas dramatically reduced errors and time spent hand creating the parameter-variables and assigning the correct seed values. These improvements provided huge gains in productivity and quality and with incremental releases were able to provide improvements even before all requirements had been implemented.

V. Conclusion

This paper outlined a business team which was under pressure to fulfill a schedule and needed help to alleviate the risk which had been building around failing to meet the final deadline. Utilizing some tools, the process became more efficient and standardized, however the need remained for a vast improvement in productivity.

Investigation of the team's process identified the modeling effort itself to be the most time consuming and have the most room for improvement. JMP scripting was identified as the easiest means to achieve improved standards and productivity from this part of the process.

Once this improvement was targeted, a scripting team was developed to implement the team's ideas and requirements into a cohesive and dynamic platform. This development process followed a simple format of overall design to methodical implementation of JSL scripting.

The capabilities of JSL scripting resulted in an add-in which was able to start improving the analytical speed and precision of the team from the first release. Using the object-oriented framework and the ability to create unique functions and objects independent of the main program the team was able to continuously release updates with new capabilities and diagnostics. The overall result was a huge increase in productivity, standardization, first-pass quality, and immensely reduced flow time.

ACKNOWLEDGMENT

I would like to acknowledge the original architect for which GIST is named after, Robert Grube. His abilities and vision combined with the innate capabilities of JMP to create a truly amazing script and add-in. The continuous improvements I have been implementing are almost insignificant when compared to initial product.

I would also like to thank Josh Kendoll. He has been a counterpart in modeling and JMP scripting for several years and has been able to act as a counterbalance to my statistical opinions and comments to create better products for our company.

Finally, I would like to thank the Joint Supply Cost Modeling team as a whole for their dedication and support in the scripting endeavor as well as their successes in producing models which are driving tangible business results.

BIOGRAPHY

Andrew Parker has been with The Boeing Company for six years. Currently with the BCA Joint Supply Cost Models team in Everett, Washington, his current assignment is to be a statistical consultant for SM Finance and work as a model focal within the group. His background is predominately from 787 Program Estimating and 787 PSE activities. Andrew is completing his Masters of Applied Statistics from Colorado State University (expected 2015) and has completed a BS in Mathematics and a BS in Economics from the University of Puget Sound.

APPENDIX A: CODE EXAMPLES

In the body of the text there is reference to code which may be clear to some programmers, but not others. This appendix is a repository of the code examples which are shared to give the reader insight into the topic at hand. Any one code line is annotated from above and the each specific example is a continuous piece of code from top to bottom. Examples are generally not related to one another.

Section IV, Subsection B – Tree Structure Referencing:

```
//Pulls a report for the object
nlplatrep = nlplat<<REPORT;
//Looks through report's tree structure
nlparamvals = nlplatrep[2][3][1][1][2]<<get as matrix;
nlparamnames = {};
//Tree structure in a for loop
for(i = 1, i<=nrow(nlparamvals),i++,
 nlparamnames = insert(nlparamnames, nlplatrep[2][3][1][1][1][i])
);
```

Section IV, Subsection B – Sample H/V List Buildup:

```
//Create a window object
nlqtywin = New Window( "NLQty",
      //Split window into columns
      H List Box(
            //Split first column into rows
            V List Box(
                  //Split the first row into columns. Here are three button boxes
                  aligned horizontally in those columns before closing the row.
                  H List Box(
                        Button Box( "<< GIST (no update)",//Code goes here.),
                        Button Box( "Adjust z's in columns",//Code goes here.),
                        Button Box( "... & NORM Y method",//Code goes here.)
                  ),
                  //Similar to above, but with the second row
                  H List Box(
                        Button Box( "Save Leverage Columns",//Code goes here.),
                        Button Box( "Lev Diagnostic",//Code goes here.),
                        Button Box( "ResetGoResetGoResetGo",//Code goes here.),
                  ),
                  //Once again with the third row, but only two buttons. The
                  dimensions of a box is determined by the longest row/column of a
                  child box
                  H List Box(
                        Button Box( "View Improvement",//Code goes here.),
                        Button box("Summary of Moments",//Code goes here.)
                  ),
                  //Here we have closed off three rows in the first column, the
                  following runs a function which was built to add in a Nonlinear
                  platform and it follows the three button rows as a fourth row, but
                  we don't use the H List Box because this row is only one object. The
                  comma following the code closes off the V List Box and finishes the
                  first (and only at this point) column. We would need to close out
                  the H List Box and the New Window to use at this point.
                  Eval(parse(nlscript)),
```

Section IV, Subsection B – Sample Column Dialog:

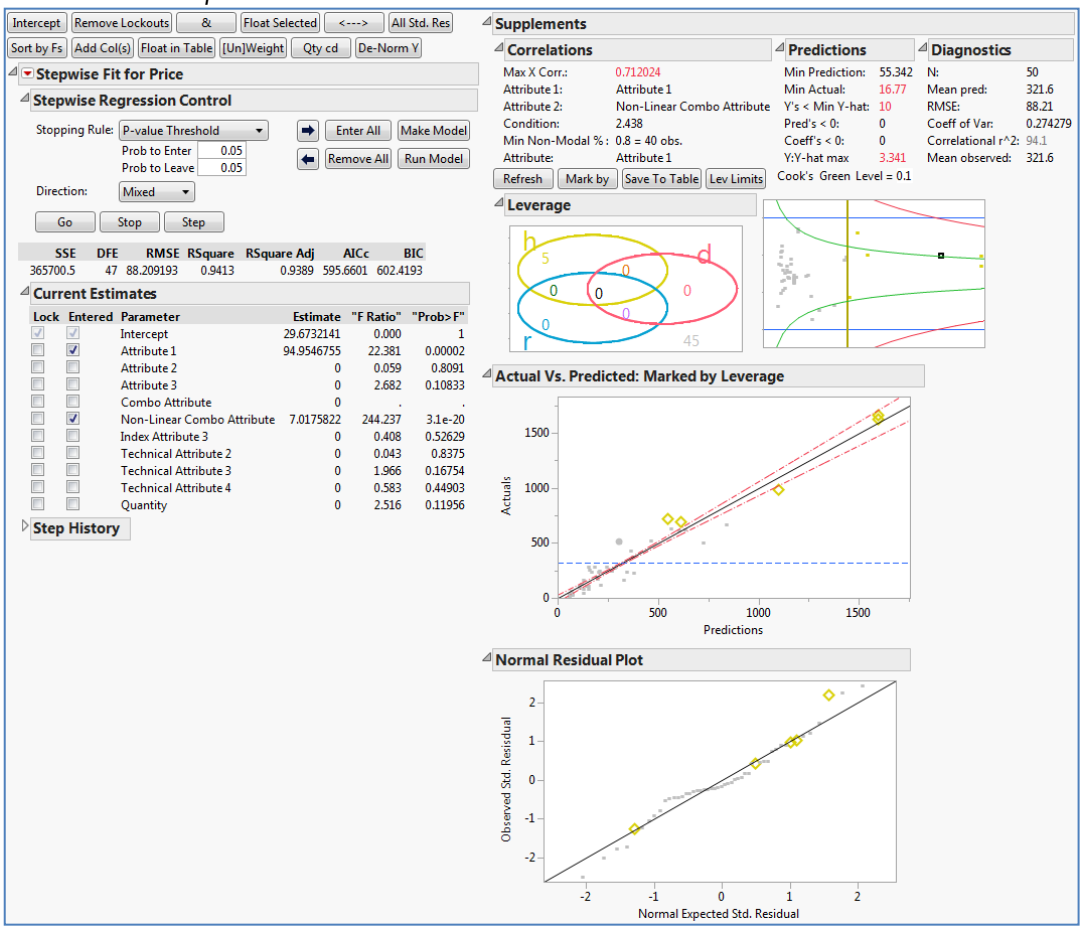
```
//Create a Column Dialog object and grabs response variable column.
user = Column Dialog(ys = Collist("Dependant Variable"),
      //Grab independent attributes
      useratts =Collist("Candidate Attributes"),
      //Grab weighting variable
      weightcolumn =Collist("Weight (Optional)"),
      //Create radio button data to check for holes in data
      vlist("Any Missing Data?",
            cleandata = radio buttons("Yes", "No"))
      );
```

Section IV, Subsection B – Example Read in Code:

```
//Sets the file target when your script files are located
set filesearchpath("C:\Users\ad846c\Documents\GIST Files-Dev");
//Build an array of script names (names of the files themselves)
Includelist = {
"GIST Graphics",
//… several more files …
"GIST IQPLAT"
};
//Create for loop to read in each of the arrayed items in the target file. This also
//includes an error catch using pass and throws a response depicting which one failed.
for(i= 1, i<=nitems(includelist),i++,
      pass = 1;
      //Builds string which is the whole file name (could be included in array)
      openstring = includelist[i] || ".jsl";
      //Attempts to read file and include into compiled code.
      try(include(openstring),pass = 0);
      //Throw an error message if failed.
      if(pass == 0, dialog("Failed to find file:"," ",openstring," ", button("Ok")));
);
```

APPENDIX B: GIST GUI

GIST – Linear Platform



GIST – Nonlinear Platform (Not converged – Coding also in work)