# Practical 2: A statistical analysis assistant for conducting regression type analyses

## 2.1 Overview

In the first practical we introduced Stat-JR's TREE interface and its workflow system. By the end of the practical we had become familiar with blocks within a workflow that allow us to ask questions of the user, to perform some statistical operations via Stat-JR's template system, to output objects and to use outputs from one operation as inputs for another operation.

In this second practical we will introduce further blocks that will allow us to influence the route through a workflow, and also additional templates that contain some textual output conditional on the results. As well as covering the workflow system in more detail, our parallel aim in this practical is to think more about what people do when they want to fit models to a continuous response variable when they have an independent sample from the population, i.e. we are focussing here on linear modelling and the associated operations that go with it. As part of the eBook research grant we would like to create an automated system (a statistical analysis assistant) that will take a user's dataset and by asking him/her questions perform an appropriate statistical analysis of it (or at least offer the user useful help and guidance along the way). This practical will make a modest start in building one; we'll be some distance from achieving a generalisable statistical analysis assistant, but it will facilitate discussion about some of the possibilities and challenges one might encounter when trying to undertake such an endeavour, and it will also allow us to investigate further functionality in the workflow system.

## 2.2 Questions and a histogram

We will begin by simply creating a workflow that asks for some inputs and produces one plot. We have already encountered blocks that ask for a single and multiple variable input, and we will use those again here, but also introduce a third question block which asks for a dataset. This block is available from the **Input** menu, so either start up Stat-JR WF afresh or click on the **Clear** button to clear the current workflow, and set up the workflow as follows (note that currently a working dataset needs to be nominated at the outset before we can then reallocate it and hence we need the two *Select dataset* blocks here):

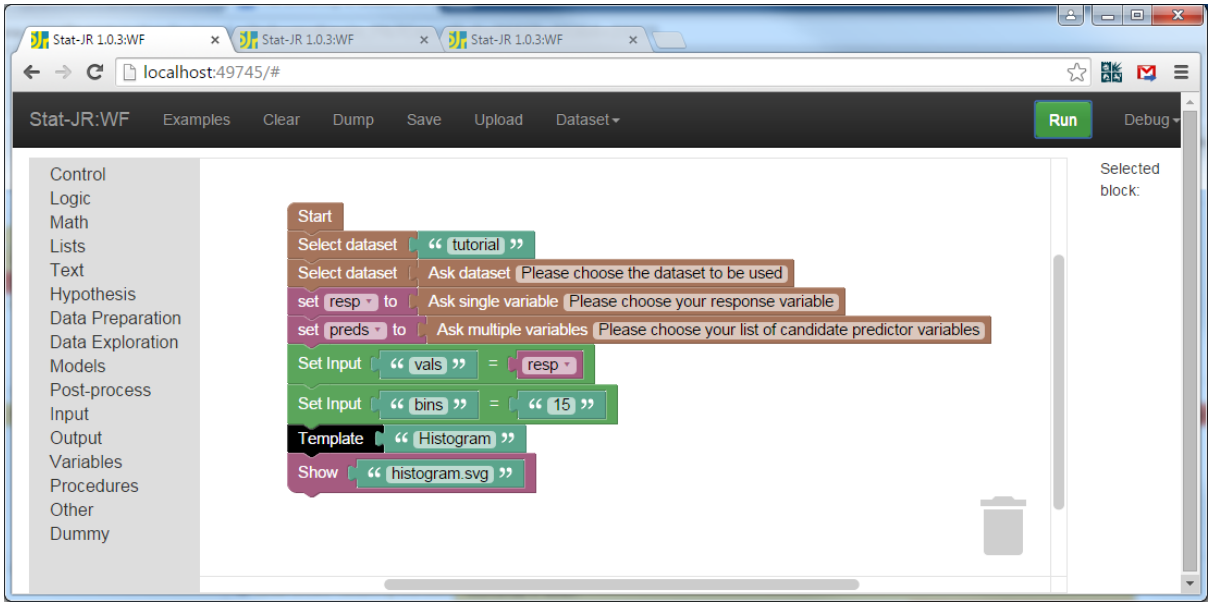



*Figure 1*

So here we have constructed blocks which first ask the user which dataset they would like to use, and then asks them to nominate their response and predictor variables of interest (assigning these

to the variables *resp* and *preds*, respectively). We then plug in their response variable as the values (*vals*) the *Histogram* template will plot (with 15 *bins*), and finally run the template and show the graph (*histogram.svg*). You'll notice that whilst we ask the user to nominate their predictor variables, we don't actually use these yet, but will do soon.

Save this workflow as *prac2_2.xml* and then **Run** it. In this example we are still using the *tutorial* dataset but you may like to try a different dataset yourself. Here is an example of the output:

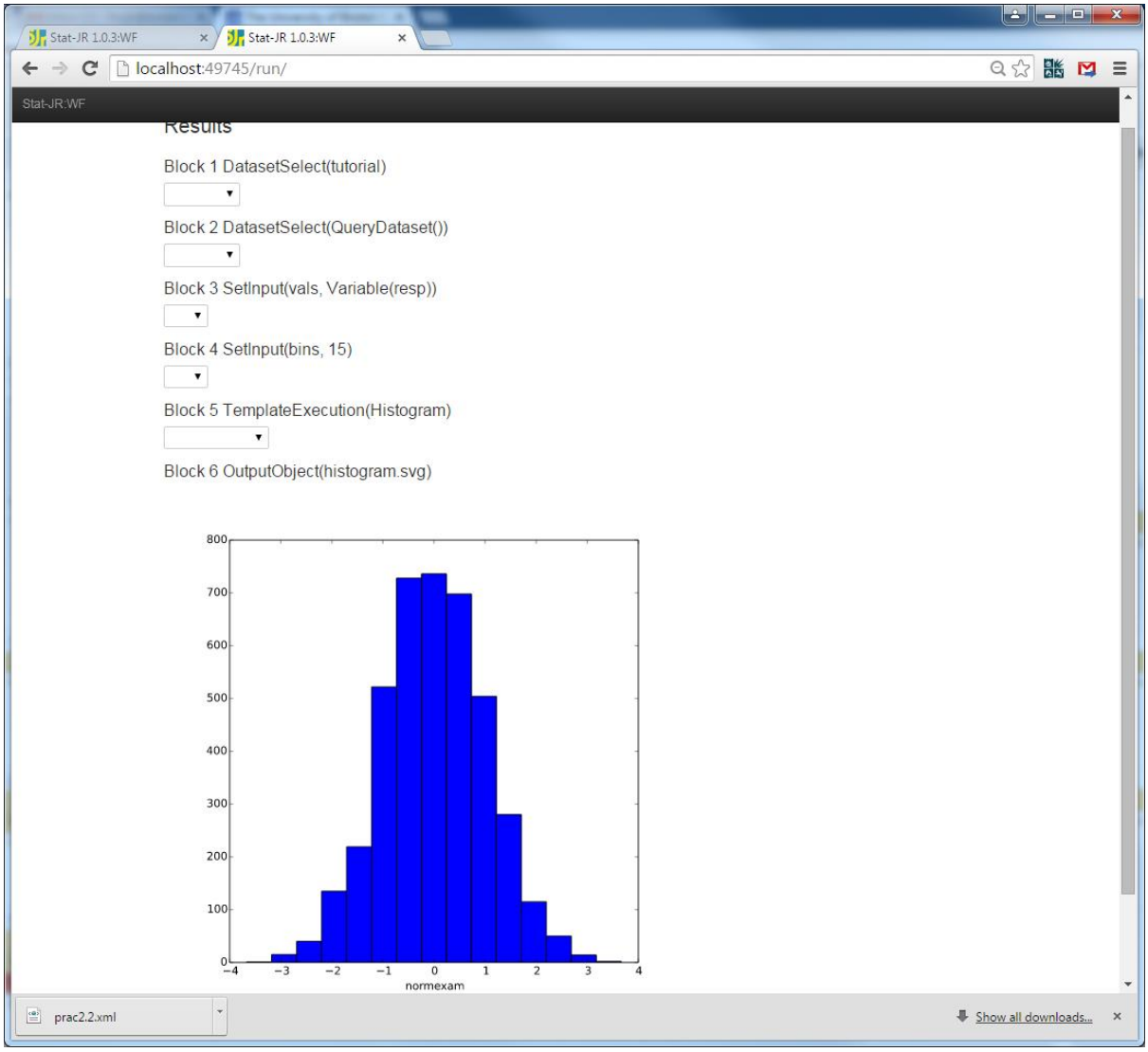



*Figure 2*

In this example we have chosen *normexam* as our response from the *tutorial* dataset (although answers to the questions vanish from the results page once given, so we can't see them here) and hence a histogram of *normexam* is returned.

## 2.3 Introducing the "for-do" block

We also asked for predictor variables, so we can do something with those as well: for example let's plot the response against each of them in turn. Here we face a situation we haven't previously encountered in that there are (likely to be) multiple predictors, so we need to introduce a new block which performs the same operation for each one. Such blocks are found in the **Control** list on the left hand side, and in this example the *for-do* block is a good choice:

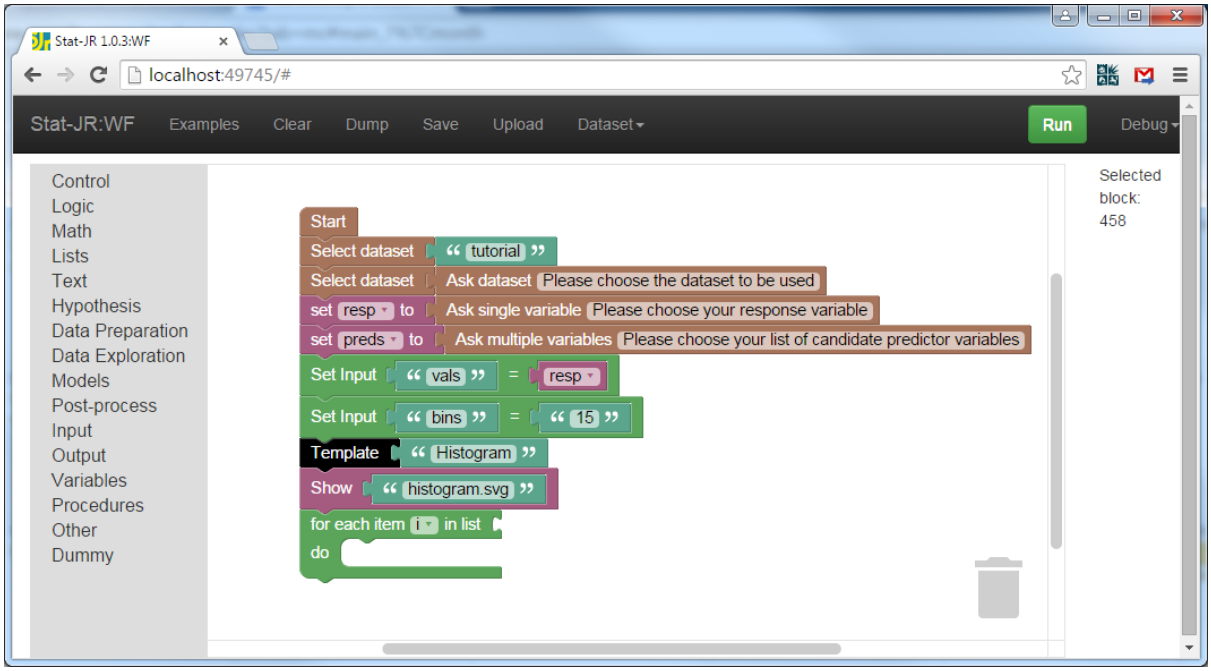



*Figure 3*

The *for-do* block has slots for two attached blocks (or sets of blocks) – the uppermost slot (to the right *of "…list")* requires a list containing elements to loop through, whilst the other slot, beneath, requires blocks defining what to do to each element of that list. The variable *i* will contain the value from the list at each pass through the *for* loop, and so can be used as an index to reference within the instructions.[1]

So what we want to do is to loop through the variables the user nominates as predictors, and for each one plot it against the user-nominated response variable (we can use the *XYPlot* template we used in the last practical), showing the relevant output (graph) for each. Have a go at doing this yourself, and then compare it to our worked example in Figure 35 of the Appendix (page 26).

How did you get on? Save your workflow as *pract2_3.xml*, and then **Run** it. In our example, below, we have chosen *normexam* as our response and the predictors *standlrt* and *avslrt*:

[1] Conventionally, *i* (perhaps an abbreviation of *index*, *iteration*, or *integer*) is used as the default counter in a control structure such as this, but you can change it to whatever name you like (although some care is needed if names have been used elsewhere).

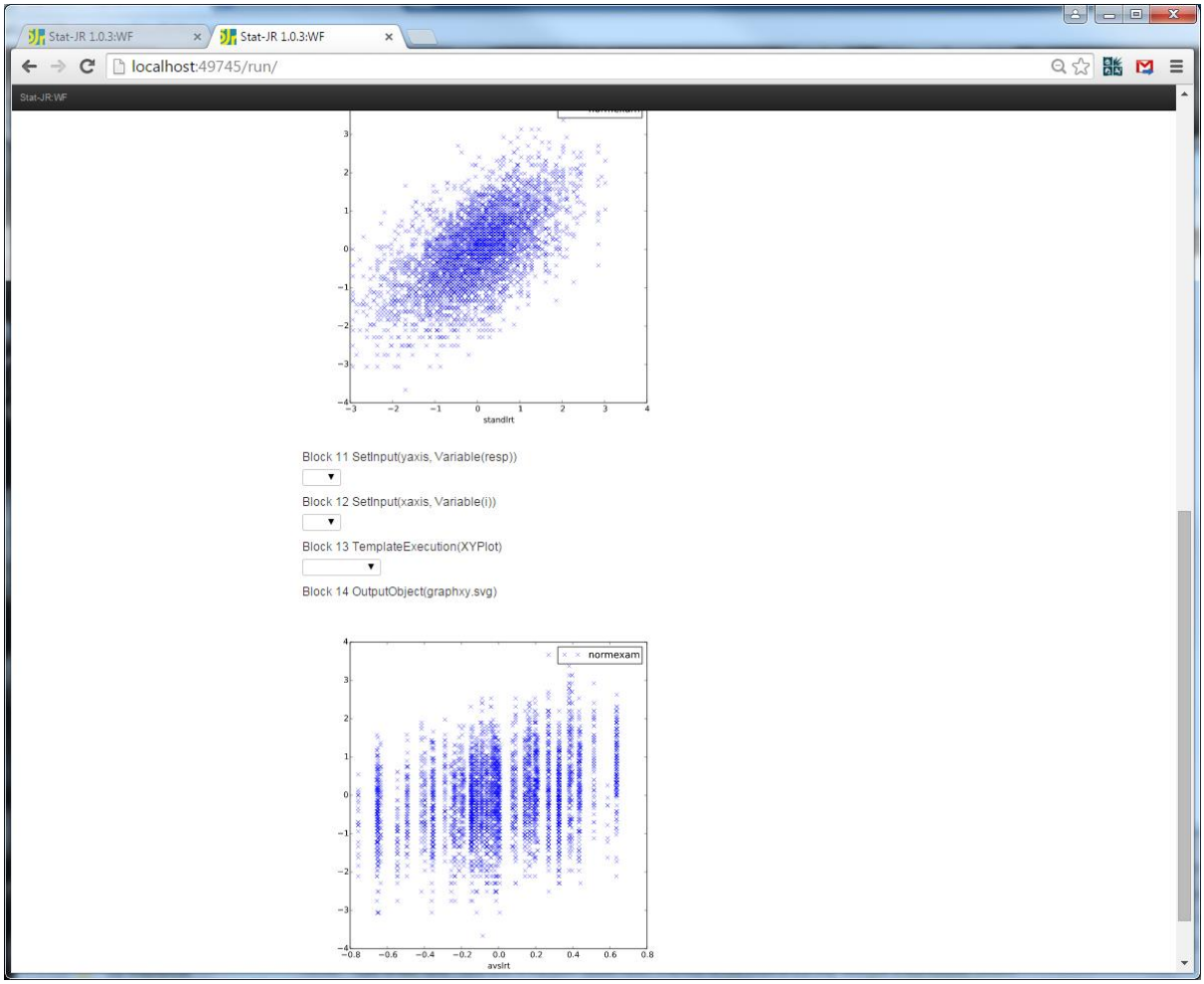



*Figure 4*

Here we see that both of the predictor variables we chose appear to have a positive relationship with the response (*normexam*), with *avslrt* plotted as discrete bands of points as this variable is constant for each school in our two level dataset.

## 2.4 Univariable models – creating an intercept

So we've started to visually investigate relationships in these plots and, as well as perhaps giving the user the option of different plot types (or of different settings for the plots we've used) we might want to allow them to explore cross-tabulations, or to examine the effect of transforming variables on their plotted distributions and relationships, and so on. We don't have time to explore all these options in this short example, other than to acknowledge they're all viable choices at this stage of an exploratory data analysis (and there may be many more options we have left out too; e.g. what would you do?) Instead, we'll jump into some models and run analyses with each predictor in turn, in what epidemiologists call univariable models. We can use the *Regression1* template that we used in the first practical. You will recall that it requires an intercept to be explicitly added as a constant in the list of predictors, and so as before we can generate a constant again, using the *Generate* template.

As we found in Practical 1, we will need a few blocks to generate a constant: four for the inputs, one to run the template, and another to extract the output of interest (the dataset with the new variable in it). In fact, to help further organise our workflow we can nest these into a *grouping* block (see the green block with *group description* written on it in the **Other** list on the left-hand side). This block

helps us to visually structure our workflow (identifying contiguous blocks all concerned with the same function), can be collapsed for brevity (by right-clicking on the *grouping* block and selecting **Collapse Block**), and allows easy duplication of all the blocks inside it (just by right-clicking the *grouping* block and choosing **Duplicate**), although it can't be called from elsewhere in the workflow (unlike *procedures*, which can; we will investigate these in the next practical).

Here we've nested our completed *Set Input* blocks and a black (run) *Template* block all inside a *grouping* block, together with a *Retrieve* block. Instead of plugging the *Retrieve* block straight into a *Select Dataset* block, though, we assign it to a variable (which we happen to call *modeldata*), and then, outside the *grouping* block, we plug this into the *Select dataset* block. As you can see, we have given the *grouping* block an appropriate name ("*Generate intercept*") to describe the function of the blocks within.

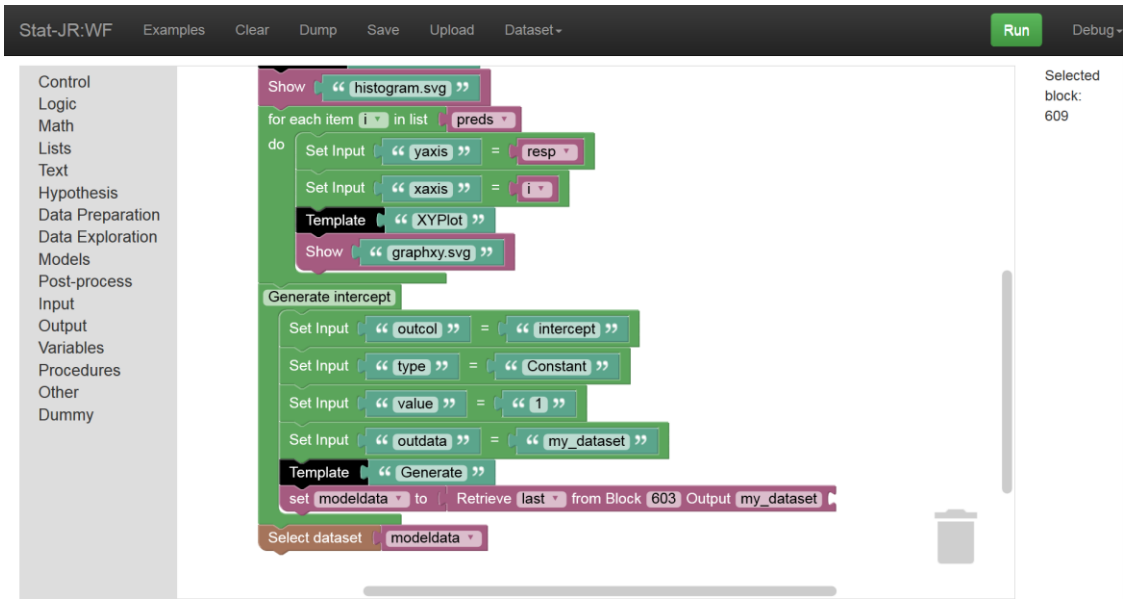



*Figure 5*

Here we've collapsed the block, helping to simplify what is becoming a busy workflow:

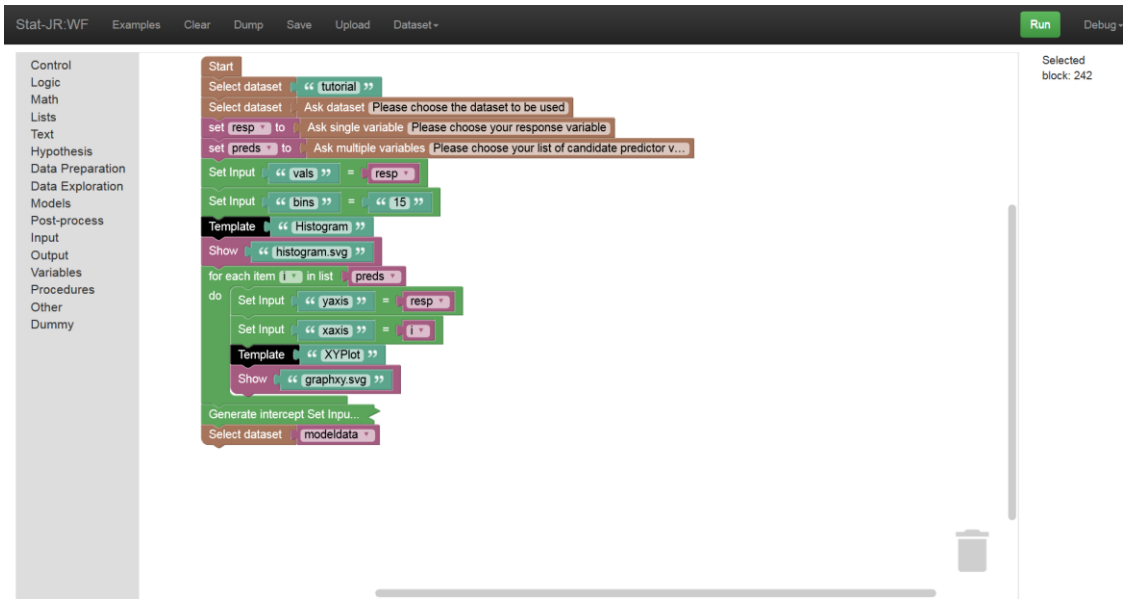



*Figure 6*

In fact we can further add *grouping* blocks around those generating the histogram of the response variable, and another around our *for-do* loop, as follows:

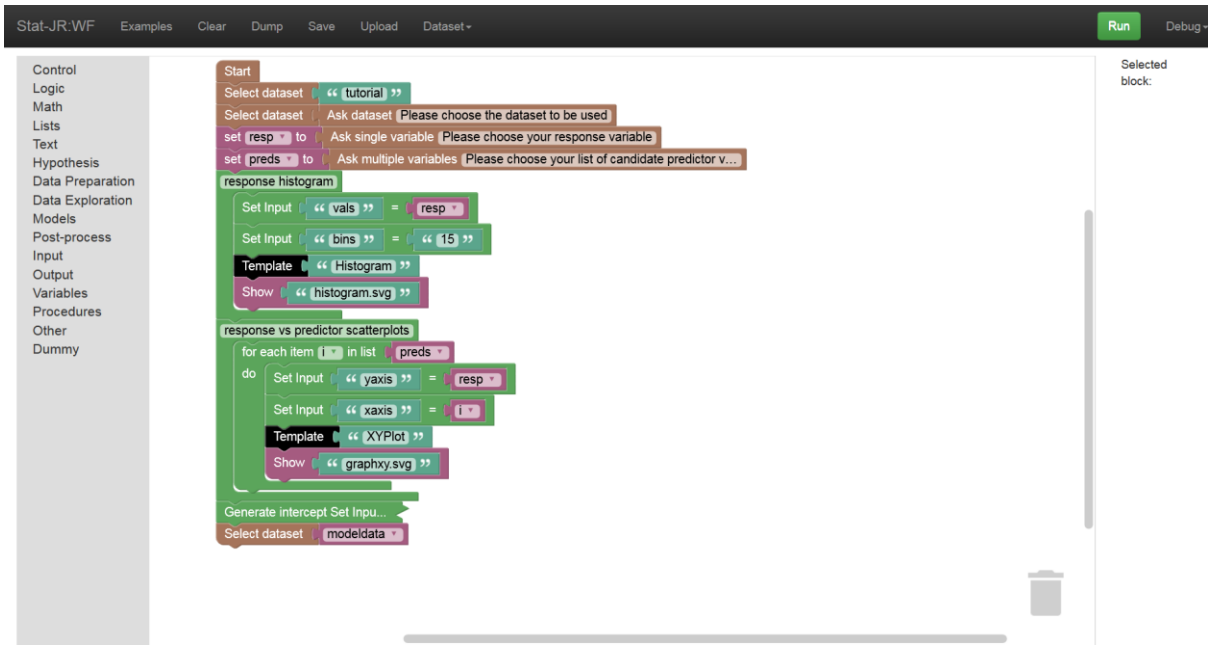



*Figure 7*

Collapsing those blocks effectively shows the workflow at a higher level of information:

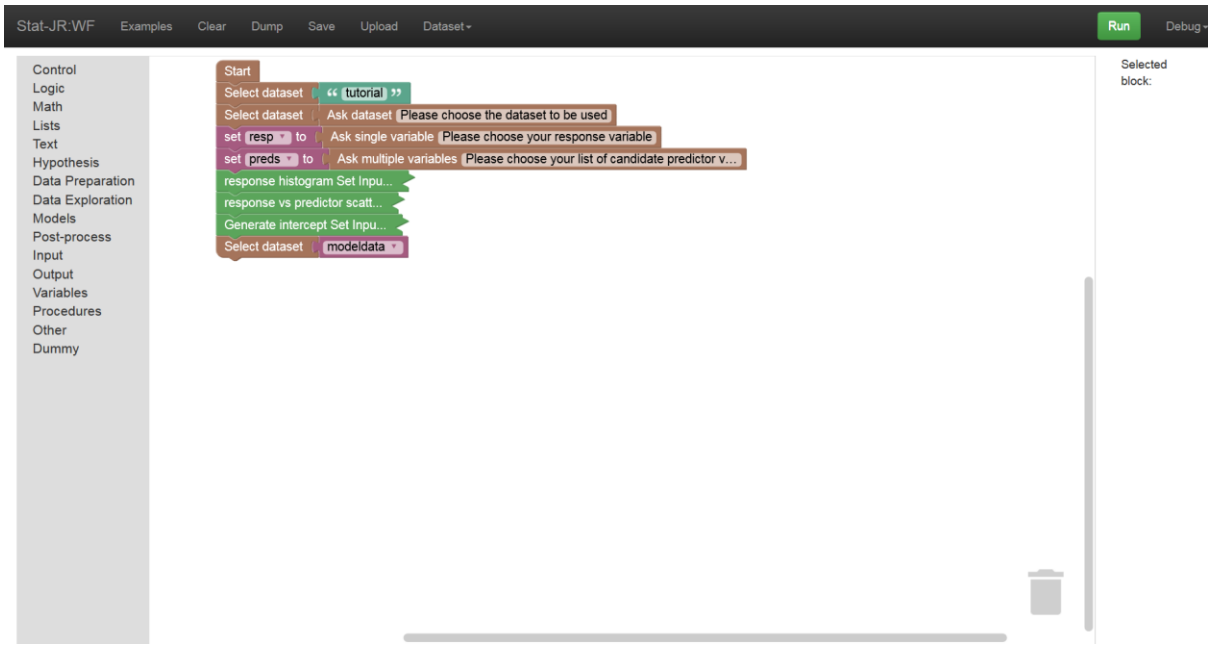



*Figure 8*

To complete the operation we will use a block we haven't yet investigated, namely the *Summary Statistics* block available in the **Data Exploration** block list. This block will produce summary statistics for the dataset and we can pull these out for display by adding a *Show* block for the *"table"* output, as shown below. In fact, the *Summary Statistics* block hard-wires the execution of a template called *SummaryStats*, with the inputs that template requires hardwired too (to include all the variables

contained in the current dataset); i.e. the same effect could be achieved by using *Set input* and *Template* blocks, as we've done previously.

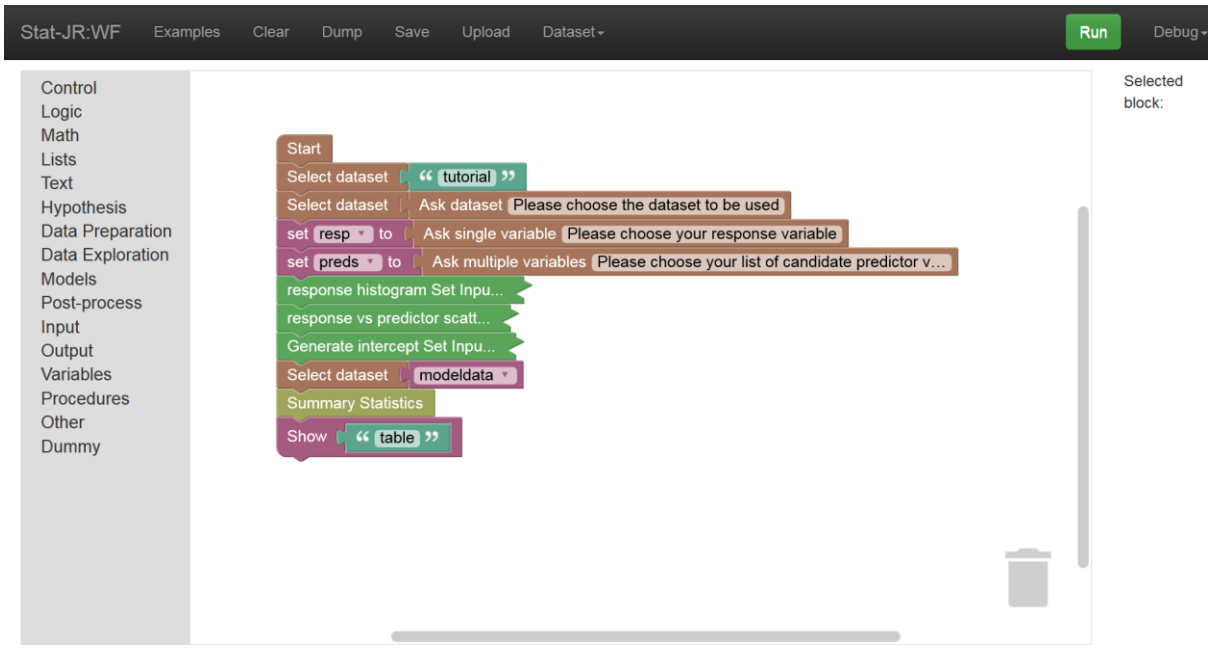
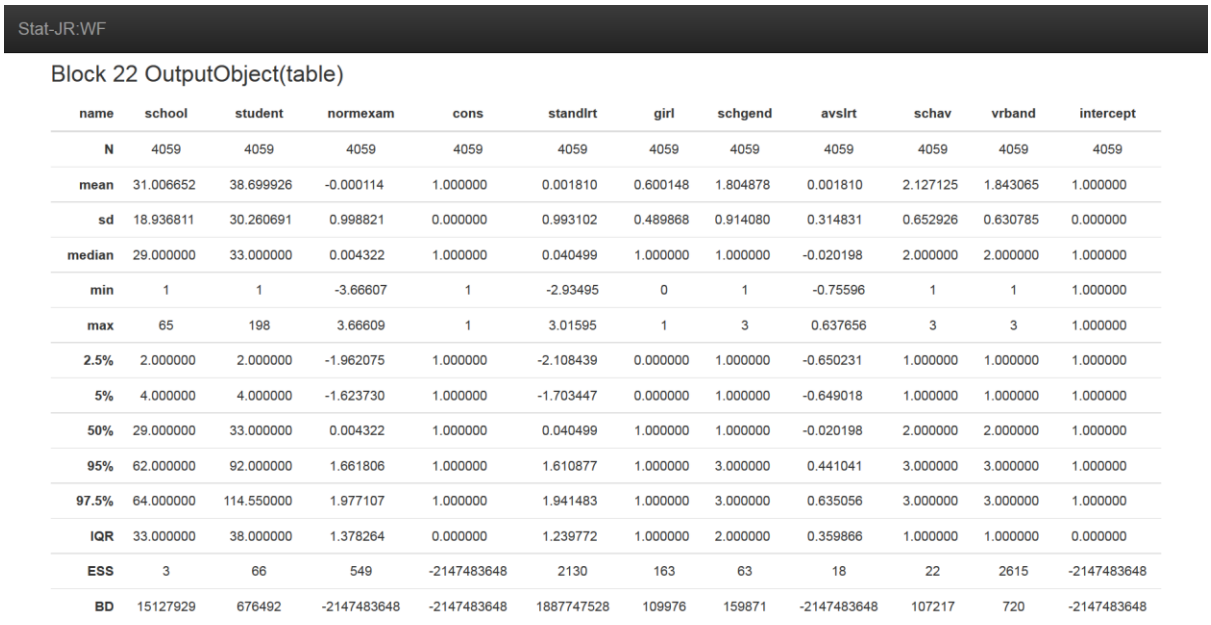



*Figure 9*

Running the workflow should still produce plots and finally the summary table thus (for *tutorial*):




| name | school | student | normexam | cons | standlrt | girl | schgend | avslrt | schav | vrband | intercept |
|---|---|---|---|---|---|---|---|---|---|---|---|
| N | 4059 | 4059 | 4059 | 4059 | 4059 | 4059 | 4059 | 4059 | 4059 | 4059 | 4059 |
| mean | 31.006652 | 38.699926 | -0.000114 | 1.000000 | 0.001810 | 0.600148 | 1.804878 | 0.001810 | 2.127125 | 1.843065 | 1.000000 |
| sd | 18.936811 | 30.260691 | 0.998821 | 0.000000 | 0.993102 | 0.489868 | 0.914080 | 0.314831 | 0.652926 | 0.630785 | 0.000000 |
| median | 29.000000 | 33.000000 | 0.004322 | 1.000000 | 0.040499 | 1.000000 | 1.000000 | -0.020198 | 2.000000 | 2.000000 | 1.000000 |
| min | 1 | 1 | -3.66607 | 1 | -2.93495 | 0 | 1 | -0.75596 | 1 | 1 | 1.000000 |
| max | 65 | 198 | 3.66609 | 1 | 3.01595 | 1 | 3 | 0.637656 | 3 | 3 | 1.000000 |
| 2.5% | 2.000000 | 2.000000 | -1.962075 | 1.000000 | -2.108439 | 0.000000 | 1.000000 | -0.650231 | 1.000000 | 1.000000 | 1.000000 |
| 5% | 4.000000 | 4.000000 | -1.623730 | 1.000000 | -1.703447 | 0.000000 | 1.000000 | -0.649018 | 1.000000 | 1.000000 | 1.000000 |
| 50% | 29.000000 | 33.000000 | 0.004322 | 1.000000 | 0.040499 | 1.000000 | 1.000000 | -0.020198 | 2.000000 | 2.000000 | 1.000000 |
| 95% | 62.000000 | 92.000000 | 1.661806 | 1.000000 | 1.610877 | 1.000000 | 3.000000 | 0.441041 | 3.000000 | 3.000000 | 1.000000 |
| 97.5% | 64.000000 | 114.550000 | 1.977107 | 1.000000 | 1.941483 | 1.000000 | 3.000000 | 0.635056 | 3.000000 | 3.000000 | 1.000000 |
| IQR | 33.000000 | 38.000000 | 1.378264 | 0.000000 | 1.239772 | 1.000000 | 2.000000 | 0.359866 | 1.000000 | 1.000000 | 0.000000 |
| ESS | 3 | 66 | 549 | -2147483648 | 2130 | 163 | 63 | 18 | 22 | 2615 | -2147483648 |
| BD | 15127929 | 676492 | -2147483648 | -2147483648 | 1887747528 | 109976 | 159871 | -2147483648 | 107217 | 720 | -2147483648 |

*Figure 10*

You will see that this has allowed us to confirm that we indeed have a new column labelled *intercept* in the dataset we've selected, and so the workflow operated as anticipated. You may also notice that the *Summary Statistics* block (via the *SummaryStats* template) produces a table with some rows which are better suited to an MCMC chain (such as the ESS (effective sample size) and BD (Brooks-Draper) diagnostics), so might be something to change in future lest one confuses the user. The

display precision may also benefit from adjustment too (too many decimal places may be distracting to the user when they are trying to quickly survey the data for patterns).

Save your workflow as *prac2_4.xml*.

## 2.5 Univariable Models – running the models

We will next perform the actual model fitting by looping through the list of predictors. Note, in this short example, we are assuming that all predictors will be treated as continuous rather than categorical variables and thus be included in the model in their current form rather than as a series of dummy variables. Recall that in the first practical we fitted a regression model and so many of the input blocks will be the same as we used there. To begin we will take the current workflow and add a grouping block which we will label as *"univariate model fitting"* thus:

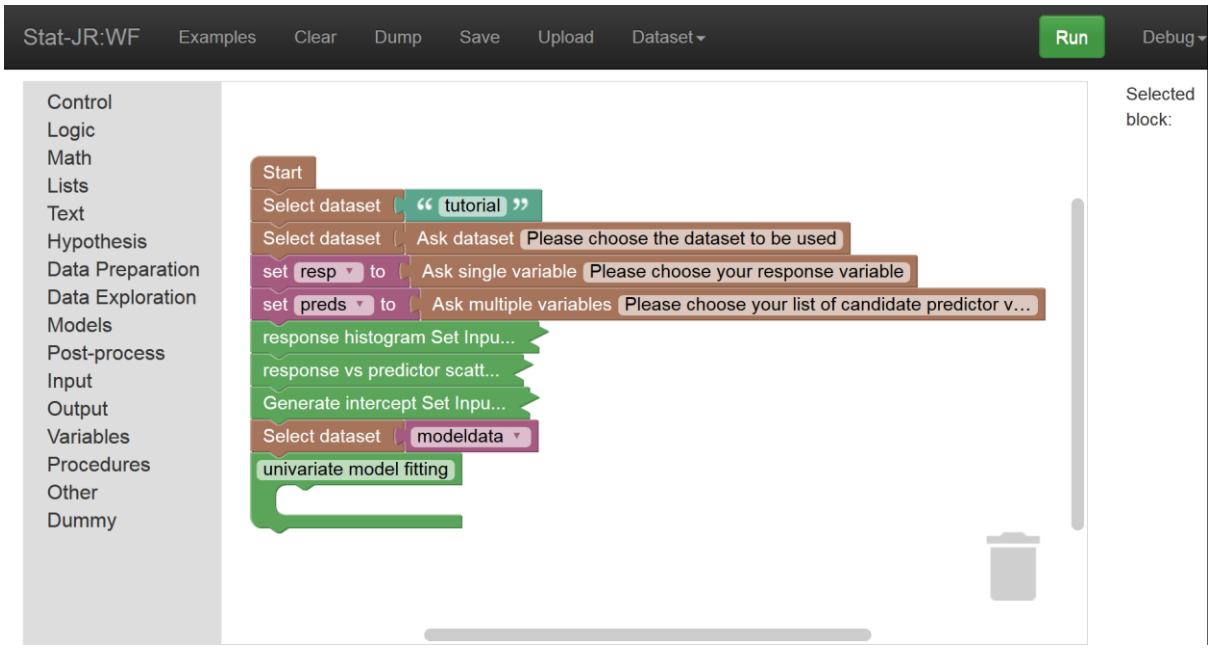



*Figure 11*

We will now need to loop over the predictors and so we will add a *for-do* loop block inside the *grouping* block and begin filling in the inputs required for a regression as shown below:

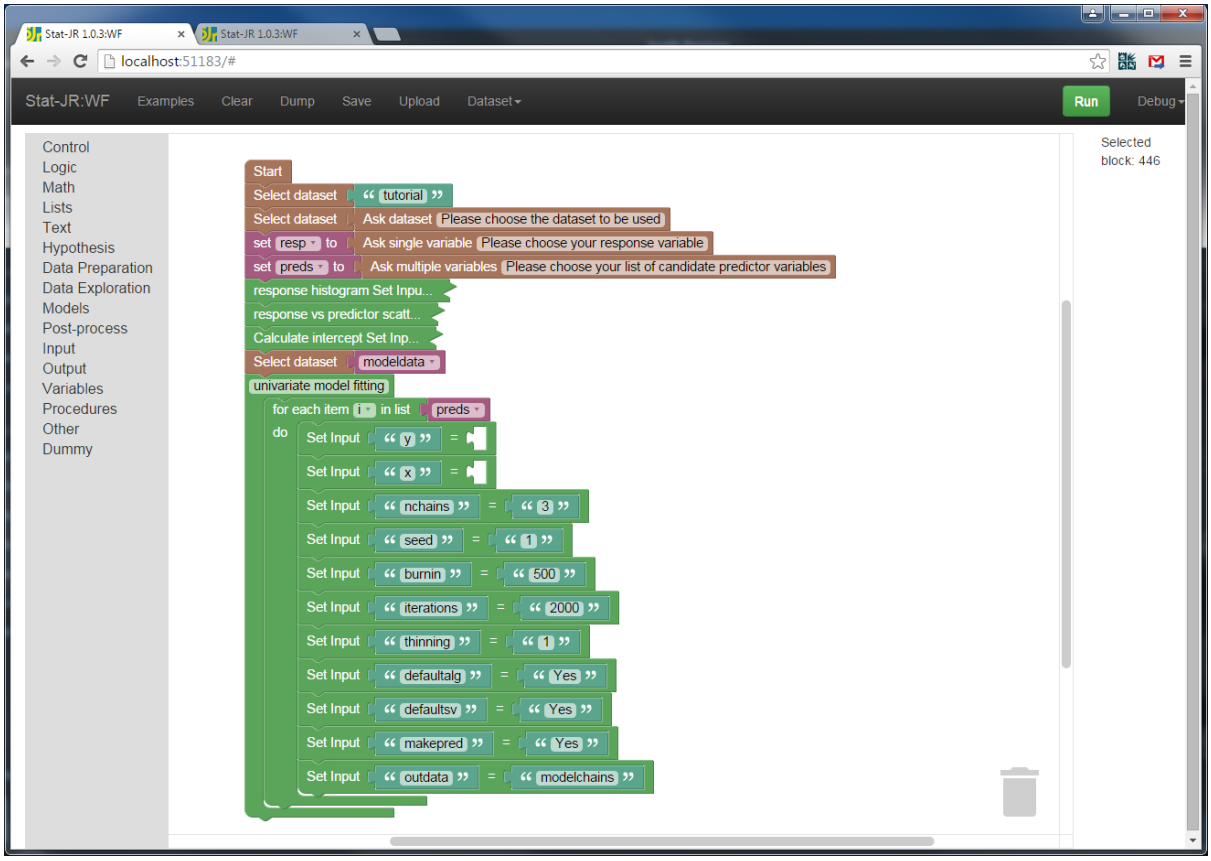



*Figure 12*

These inputs are just a reiteration of what we chose in the last practical for this template, although we need to add the *y* and *x* variables. For *y* we simply choose the *resp* variable nominated (by the user) earlier in the workflow but for *x* we need to introduce a new block, namely the *create list* block (available from the **Lists** menu to the left). We will use this block to create a list of names for the *x* variables. Here you can reduce the number of items that the *create list* block expects by clicking on the blue button in the *create list* block and dragging out one of the items from within the block as illustrated below:

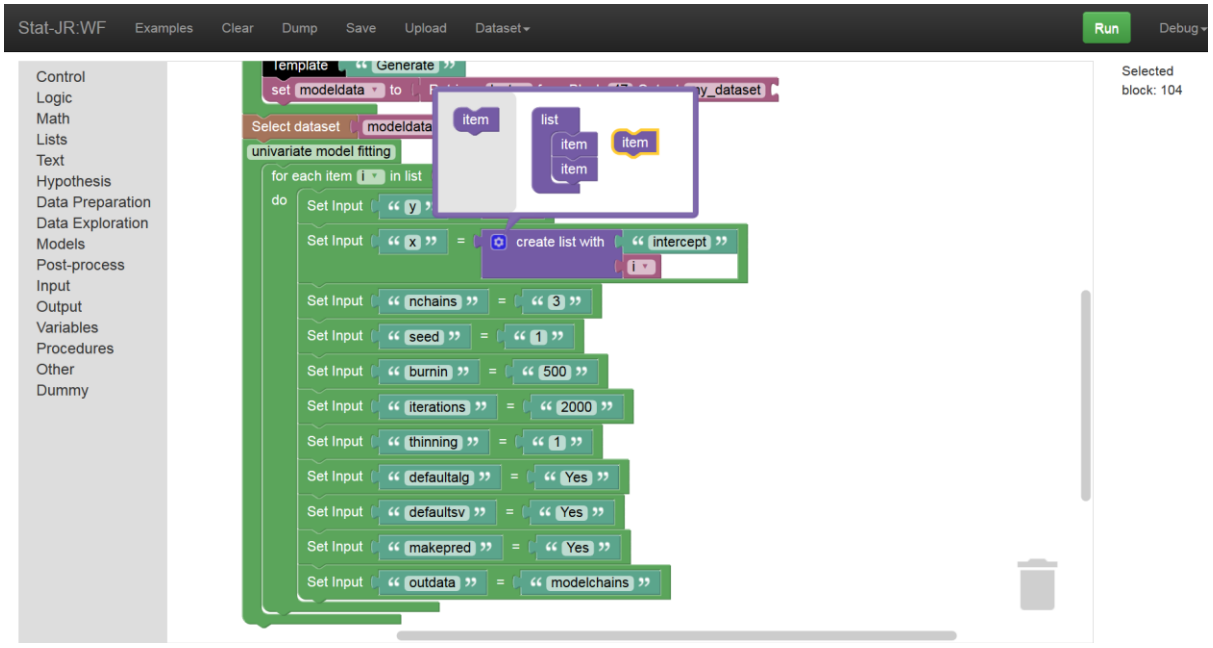



*Figure 13*

You'll see we are creating a list that includes the variable *intercept* and whatever is the current predictor (as indexed by *i*) as we loop through the list. We next need to add the template block and we will also add a *Show* block for the *ModelResults* thus:

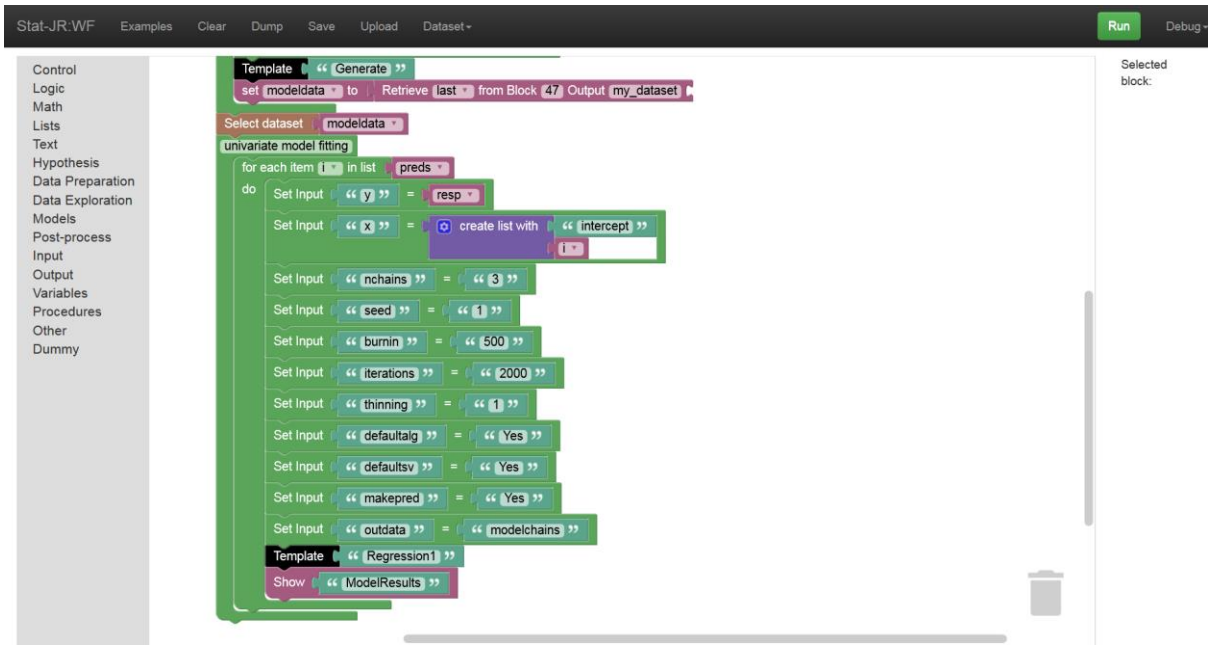
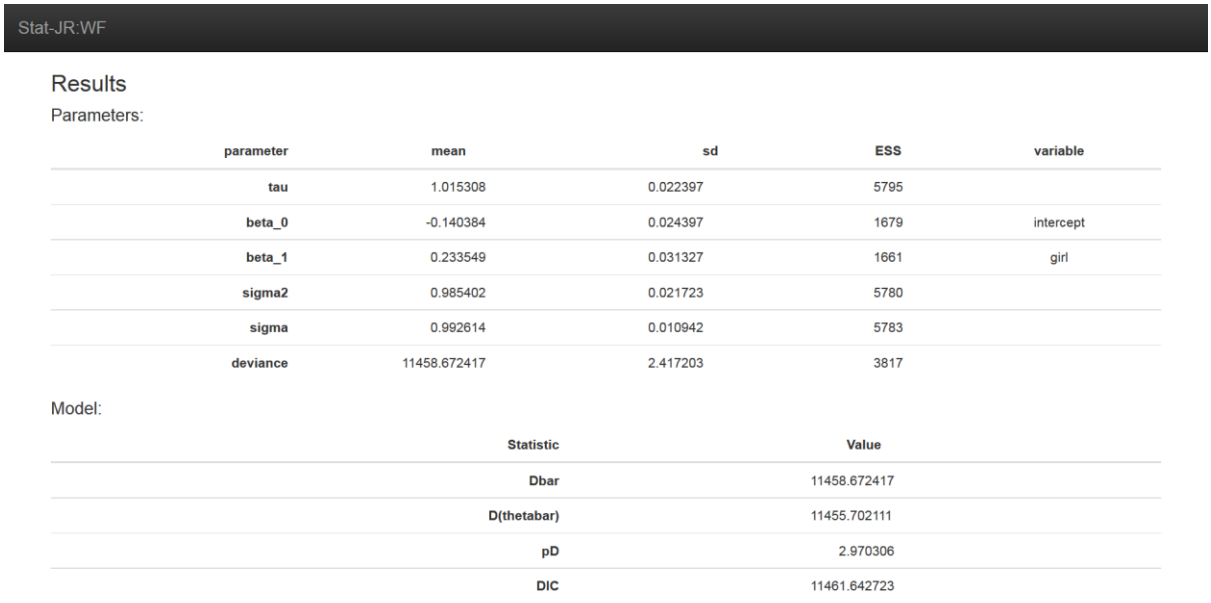



*Figure 14*

If we now save this workflow as *prac2_5.xml* we can then run it. Note we will be fitting an MCMC model for each predictor so it may take a while to run. Below are the last outputs for the *tutorial* dataset with response *normexam* and predictors *standlrt* and *girl*.


Stat-JR:WF

Results

Parameters:

| parameter | mean | sd | ESS | variable |
|---|---|---|---|---|
| tau | 1.015308 | 0.022397 | 5795 | |
| beta_0 | -0.140384 | 0.024397 | 1679 | intercept |
| beta_1 | 0.233549 | 0.031327 | 1661 | girl |
| sigma2 | 0.985402 | 0.021723 | 5780 | |
| sigma | 0.992614 | 0.010942 | 5783 | |
| deviance | 11458.672417 | 2.417203 | 3817 | |

Model:

| Statistic | Value |
|---|---|
| Dbar | 11458.672417 |
| D(thetabar) | 11455.702111 |
| pD | 2.970306 |
| DIC | 11461.642723 |

*Figure 15*

Here we see the results for a model with just *girl* as the predictor (*normexam* is the response variable), and higher up were the results for a model with *standlrt* as the sole predictor instead. By viewing the *beta_1* parameters in the model fit with *girl* we can see that the mean estimate of its effect is far larger than its standard deviation (sd) and so there is a significant effect of gender on exam score.

## 2.6 Interrogating the outputs

It would be good to automate this description (of significance) or even simply to construct a table from the model results that includes the significance of the predictors. Looking down the list of outputted objects (e.g. via the pull-down list in the penultimate block of the workflow output window), we can see that the model parameter estimates are also returned as a *.dta* file, with the name *modelparameters.dta*, and so we can work with this dataset.

To do this we can add to the workflow within the *for-do* loop:

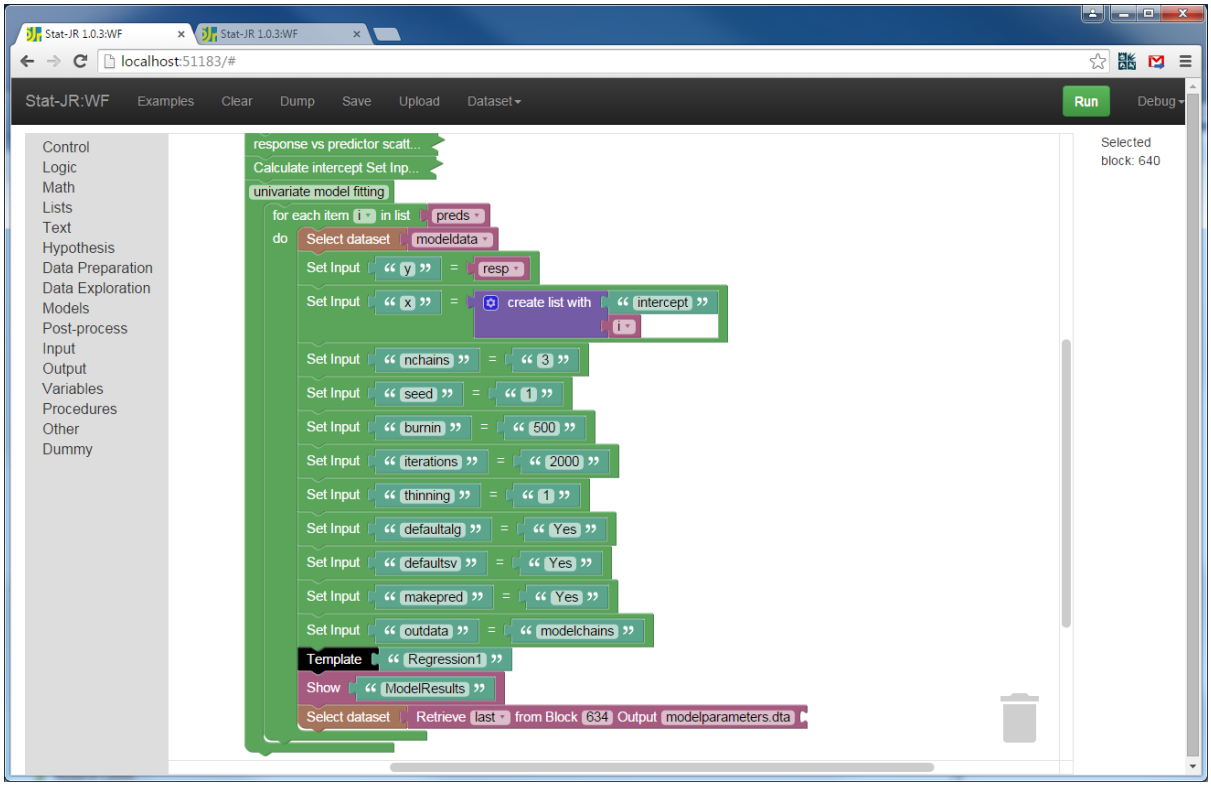



*Figure 16*

Here we've included another *Retrieve* statement, this time to pluck out the *modelparameters.dta* dataset.

We will next append an additional column to this dataset that contains the ratio of the mean estimate to its standard deviation which we will give the name *zscore* (since, for the fixed parameters, this will have an approximate normal distribution). To do this we will use the *Calculate* template which adds a variable to the working dataset based on an expression defined by the user.[2]

Going back to TREE (or opening it, if it is closed), your last execution may still be the model run via the *Regression1* template, but if not you can run one (the specifics of the model you fit don't matter so much here, we're just running one to demonstrate use of the *Calculate* template in post-processing the results). Having run a model, change the working dataset to *modelparameters.dta*, and the template to *Calculate.* You'll see from the template description that the expression it evaluates is based on *numexpr* syntax. *numexpr* is a Python package "for the fast evaluation of array expressions elementwise" – there's a hyperlink in the template description which takes you to a supporting website describing the operators and functions it supports. We need to ensure that the z-score is positive, so can use the *abs* function to return the absolute value of the expression *mean/sd* (dividing the columns of interest from our selected dataset); our whole expression therefore is *abs(mean/sd)*. The screenshot below shows our inputs, and also the outputted dataset (which we've chosen to call *zscore_table*) in the results pane at the bottom:

[2] NB: there is also an in-built *Calculate* **block** in the workflow system which simply calls the Calculate **template**, although it currently outputs a dataset with the name *a* which is perhaps a little opaque for the user, so here we use the template directly instead.

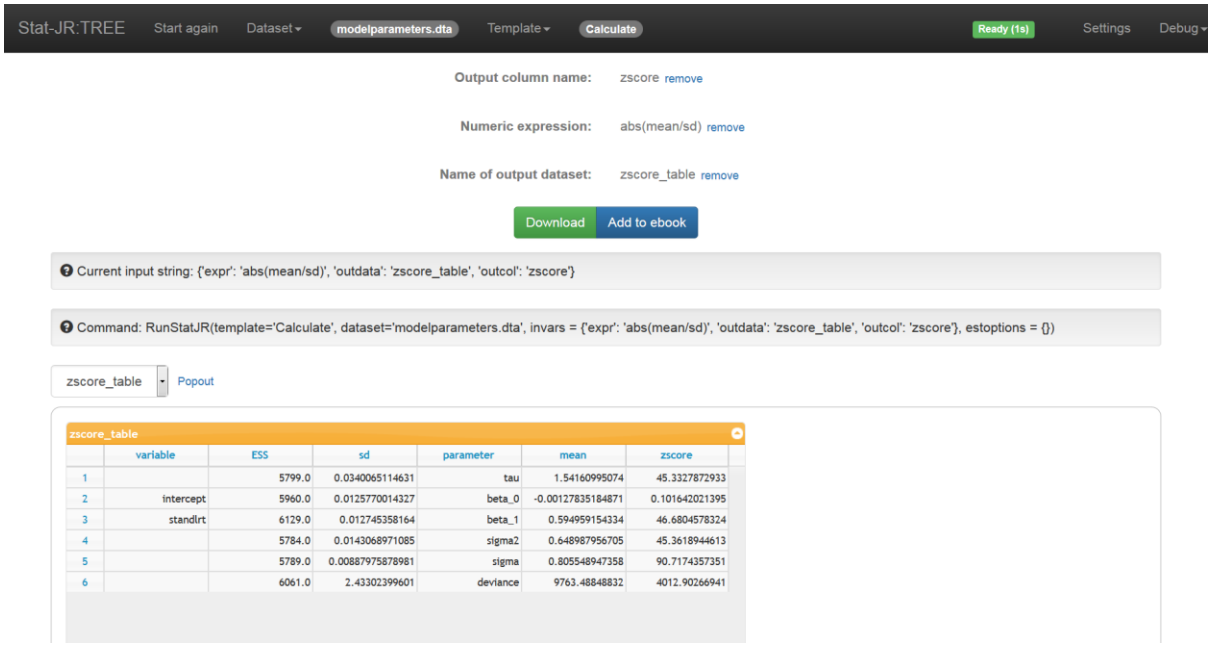



*Figure 17*

So let's now convert this into workflow blocks:

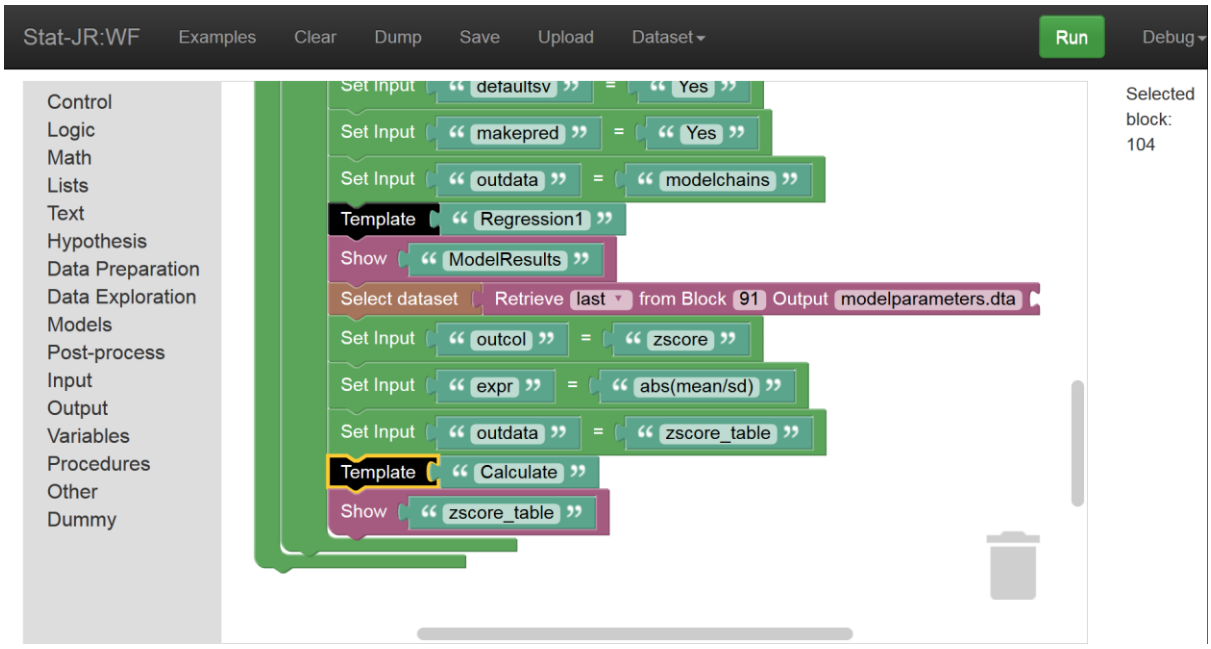



*Figure 18*

Press **Run** to check we've set this up correctly. Does the output make sense? In our example we chose *standlrt, girl* as our predictor variables; once it had been running for a few seconds it strangely asked us to nominate our **Response** and **Explanatory variables** for the *Regression1* template – behaviour we weren't expecting:

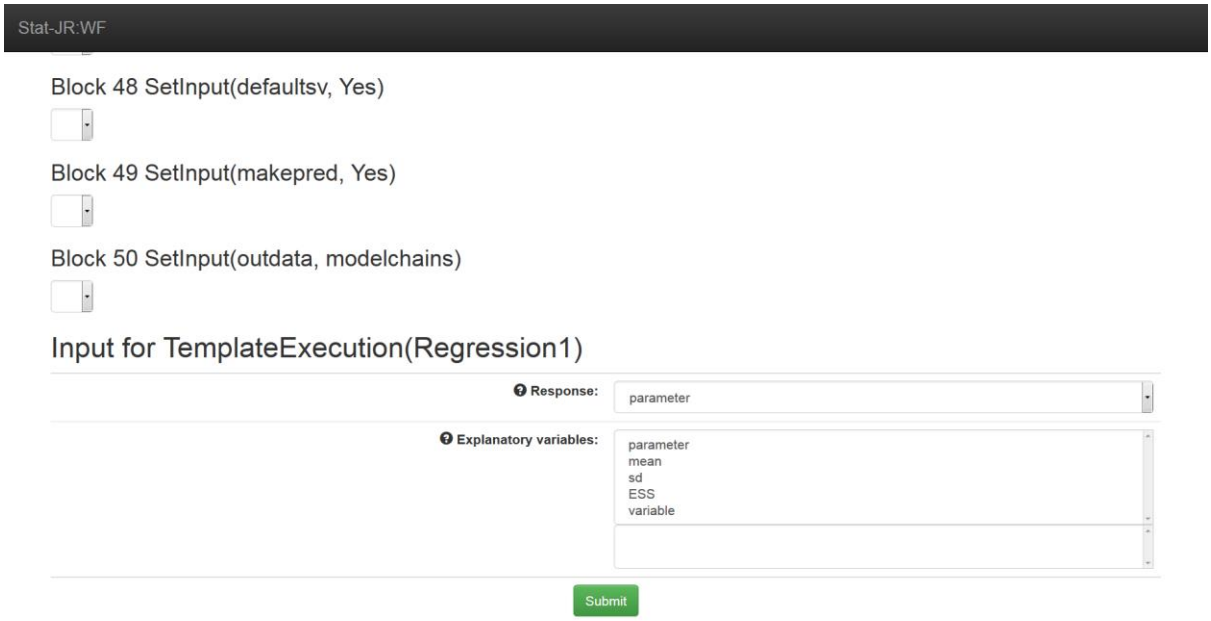



*Figure 19*

So it's behaving as if it *doesn't* have the inputs it needs to run the *Regression1* template (hence it's asking the user for them). Let's look back at the workflow; can you work out what might have happened?

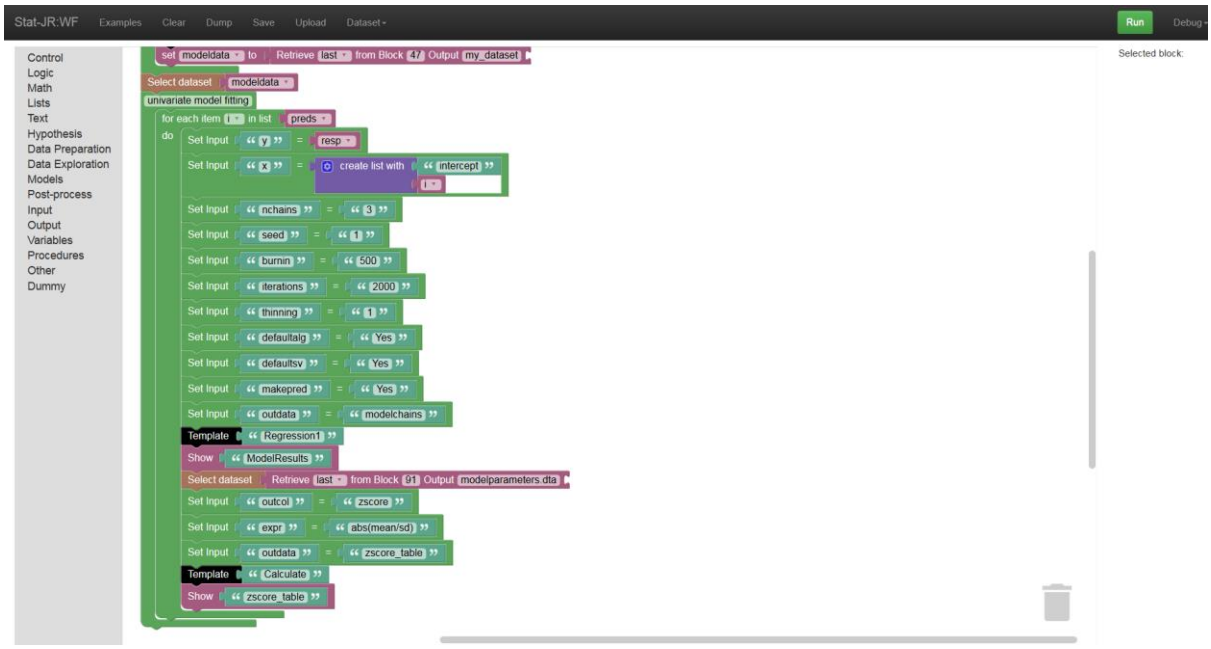



*Figure 20*

Inspecting our workflow indicates that before the group of *univariate model fitting* blocks start, we select our working dataset (*modeldata*): this is the dataset from which the inputs for the *Regression1* template are to be drawn. However, within the model-fitting loop we change the working dataset away from this one, and instead select one of the datasets outputted by the *Regression1* template, *modelparameters.dta*, as the working dataset to use when calculating the z-scores. When the loop starts over with the second predictor variable, then (assuming the user has chosen >1 predictor), the working dataset is *modelparameters.dta*, which has a whole different set of columns from the one

our inputs (as written by us) within the loop are expecting. Looking back at the workflow outputs tab, this makes sense: it's asking us to choose variables for the **Response** and **Explanatory variables** which are from the outputted *modelparameters.dta* dataset, and further up we can see that it has actually fitted one model successfully: when it first passed through the loop; the problems began when it swept through for a second time.

Let's remedy this by moving the *Select dataset: modeldata* blocks so that they appear *within* the loop. That way, the working dataset will be changed to the one the *Regression1* template inputs are expecting each time the loop begins again:

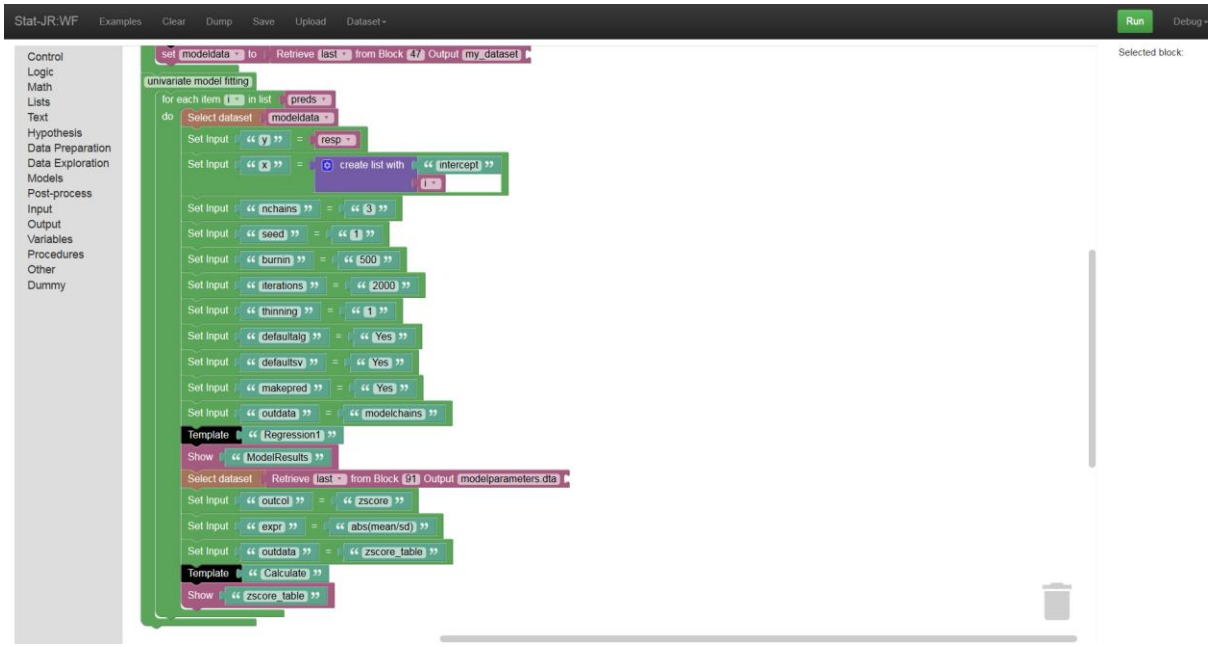
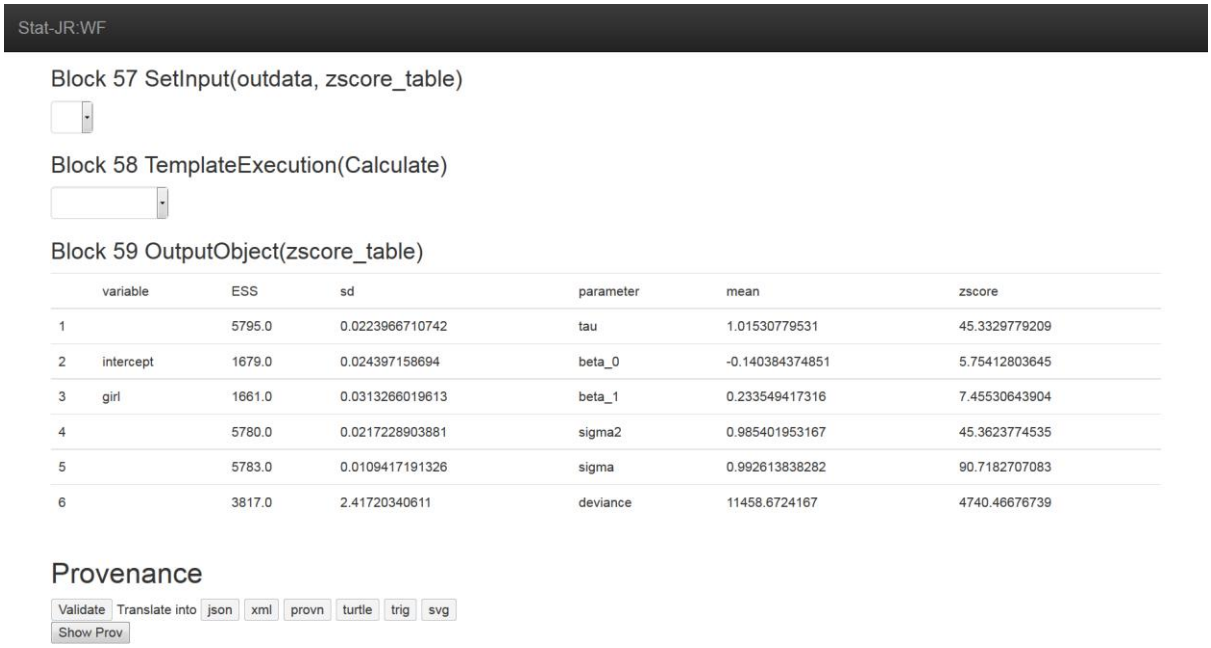



*Figure 21*

Pressing **Run** this time results in the workflow working as anticipated, as the end of the outputs window indicates:




| | variable | ESS | sd | parameter | mean | zscore |
|---|---|---|---|---|---|---|
| 1 | | 5795.0 | 0.0223966710742 | tau | 1.01530779531 | 45.3329779209 |
| 2 | intercept | 1679.0 | 0.024397158694 | beta_0 | -0.140384374851 | 5.75412803645 |
| 3 | girl | 1661.0 | 0.0313266019613 | beta_1 | 0.233549417316 | 7.45530643904 |
| 4 | | 5780.0 | 0.0217228903881 | sigma2 | 0.985401953167 | 45.3623774535 |
| 5 | | 5783.0 | 0.0109417191326 | sigma | 0.992613838282 | 90.7182707083 |
| 6 | | 3817.0 | 2.41720340611 | deviance | 11458.6724167 | 4740.46676739 |

*Figure 22*

As all our 'univariable' models are just fitting the intercept and one predictor during each run through the loop, then we know that the important number in the table indicating whether we have a significant predictor is in row 3 of the last column. We can interrogate individual entries in a table by extracting them into variables. We will do this here as indicated in the bottom of the workflow below:

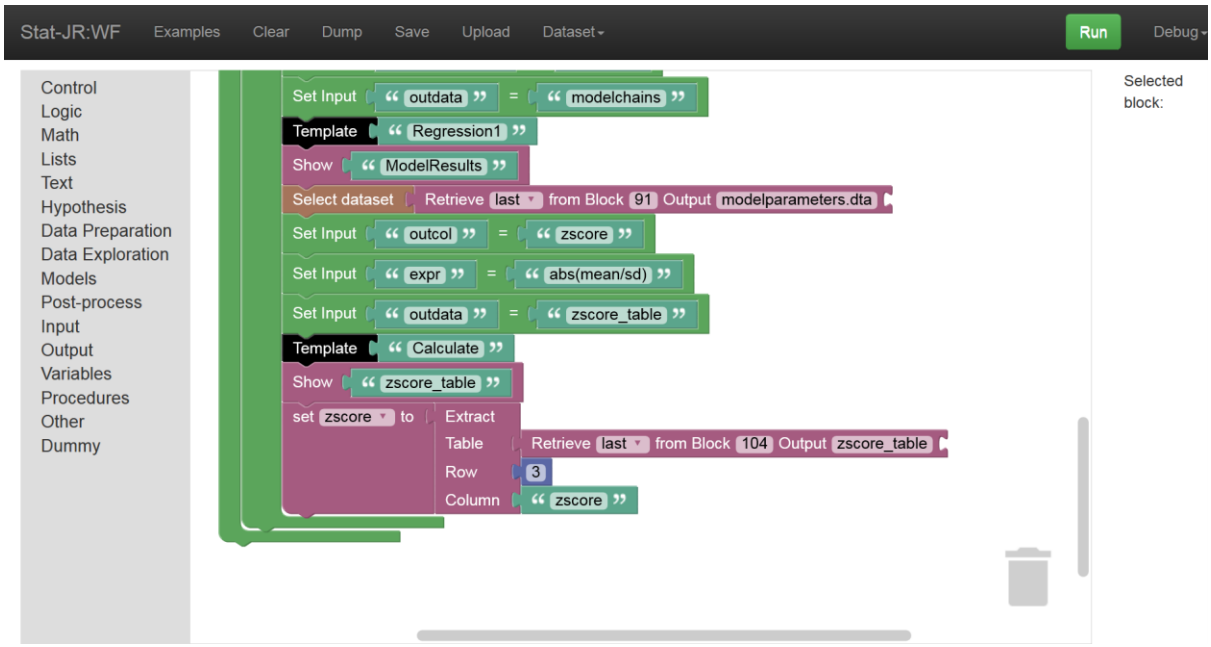



*Figure 23*

So here we've defined an item called *zscore* as comprising the value of whatever is in row 3 of the column headed *"zscore"* of the table we constructed with the z-scores in it (*zscore_table*).[3]

We now want to make some form of decision based on the value of our *zscore* item and for the purposes of this example we will do something simple, namely indicate in the output that the predictor is significant if the *zscore* is greater than 1.96. We can do this by using some further new blocks – an *if* block that is available from the **Control** block list, a light blue *comparison* block available from the **Logic** block list and *Comment* blocks available from the **Output** block list.

We will begin by simply grabbing the three blocks and arranging thus:

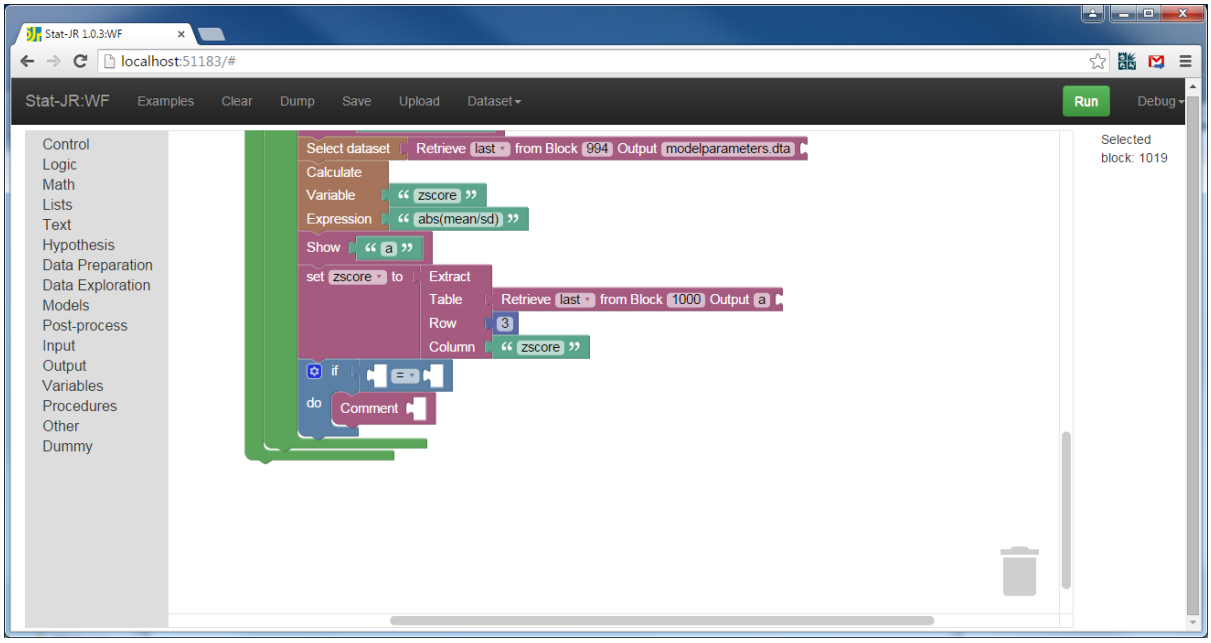



*Figure 24*

By default, the *if-do* block can do something if the *if* statement is evaluated as true, but it isn't giving us the opportunity to state what we wish to happen if the *if* statement is evaluated as false. However, we can modify the block to suit our requirements by clicking on the blue symbol on the *if* block to expose structural changes one can make thus:

[3] NB: the blue block is retrieved from the **Math** block list, and the *Extract* block is from the **Other** block list.

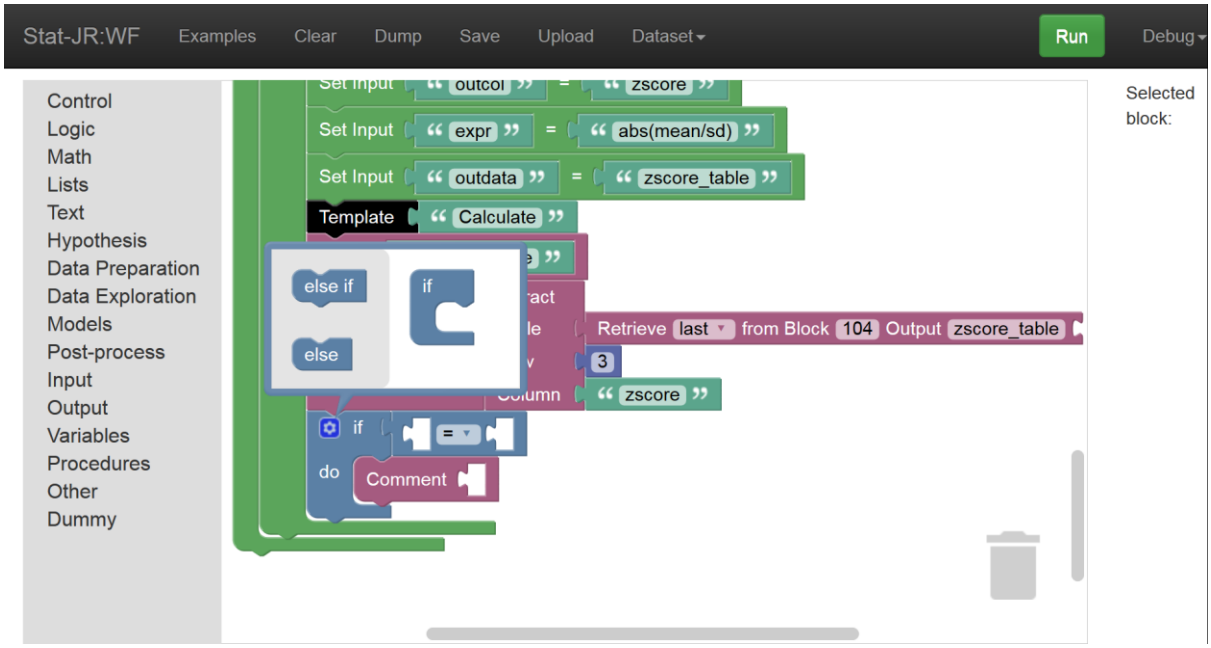



*Figure 25*

If you drag the *else* into the *if* then we can add an alternative if the condition tested is evaluated as false:

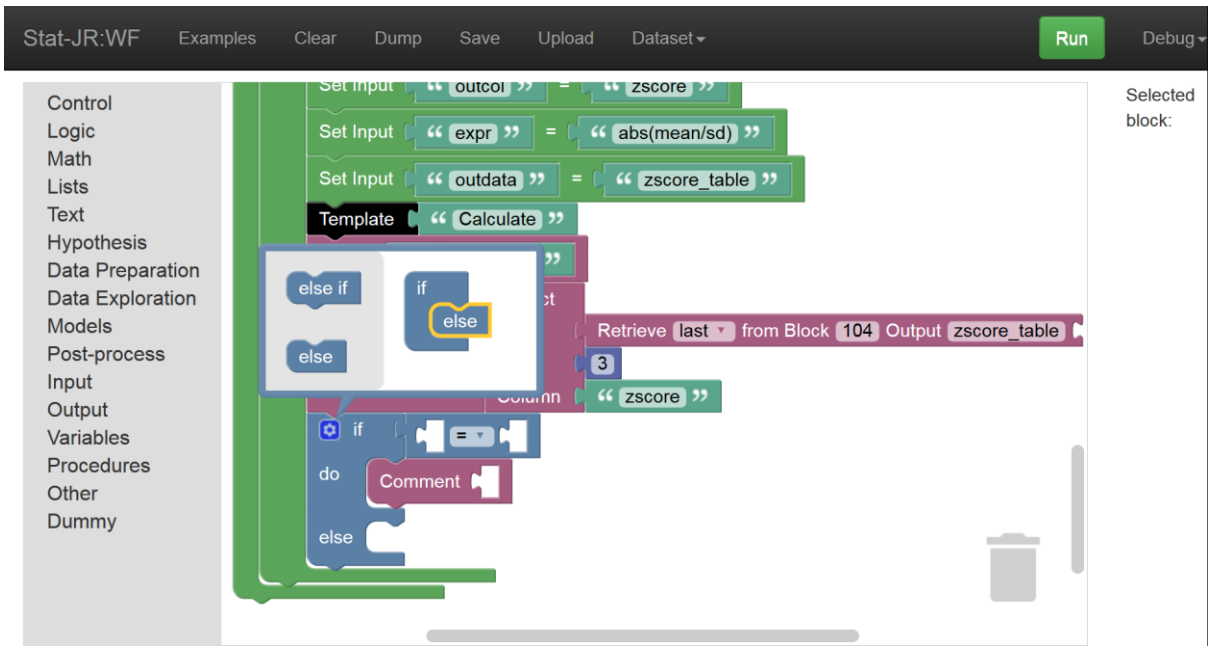



*Figure 26*

As you do so you can see the *if-do* block turning into an *if-do-else* block. Clicking on the blue symbol will return control to the main workflow and now we can fill in the gaps in the blocks. Firstly we need to define our conditional statement as *zscore > 1.96* by using an appropriate combination of blocks, and then instruct the *if-do-else* block what to do when it returns 'true', and what to do when it returns 'false'. The *Comment* blocks are simply used to send a string to the output. We could use these to simply say this variable is significant or not depending on the result of the evaluated

expression, but it's helpful to the user if we include the predictor's name as well; to do this we can use the *create text* block to create a text string that contains the current predictor name used in this iteration of the loop (remember to include a space at the end of "…variable " and the start of " does not…"):

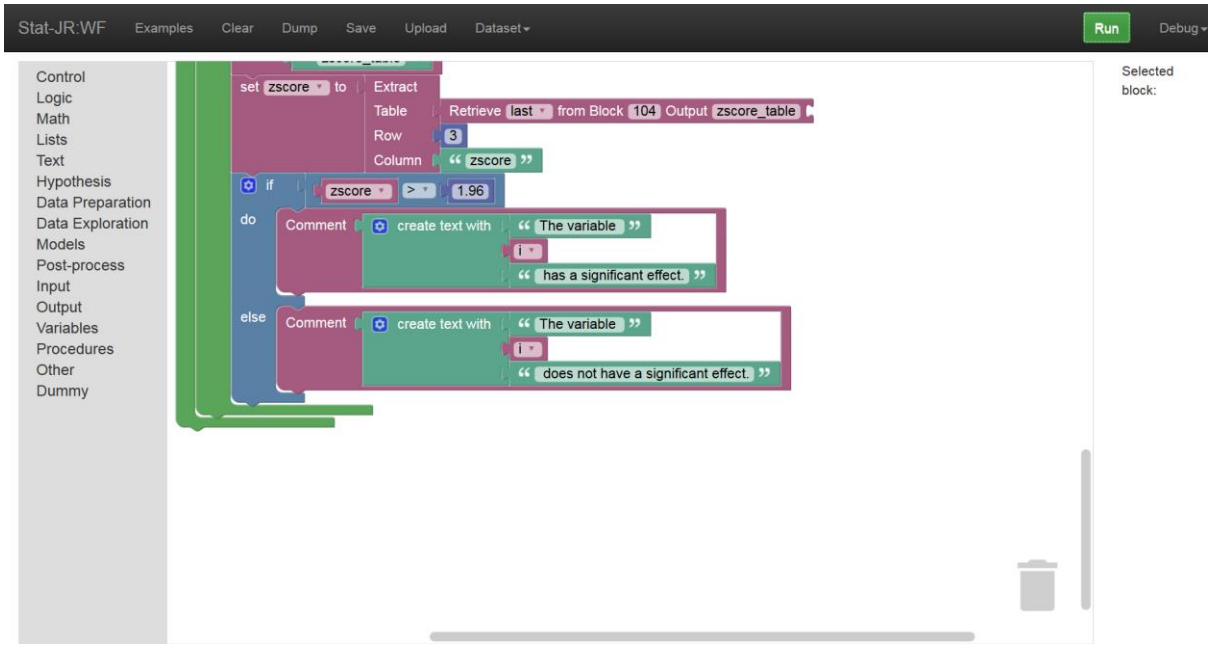
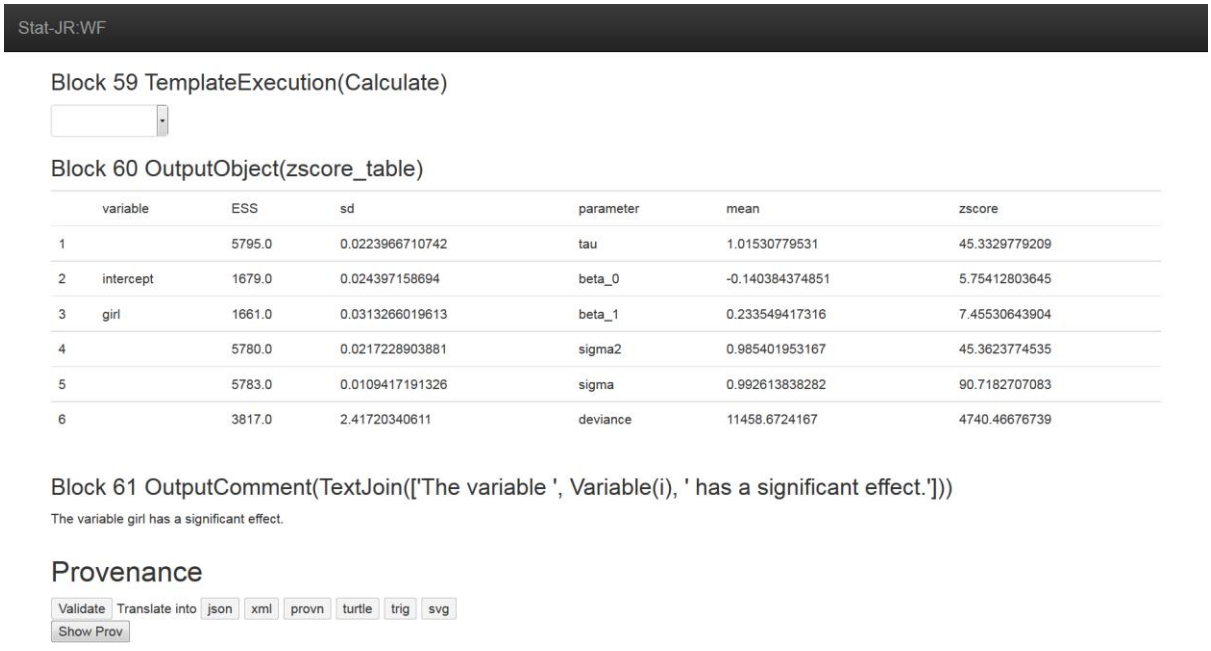



*Figure 27*

If we save this workflow as *prac2_6.xml* and then run it we can see it in action. Here again I am using the *tutorial* dataset with response *normexam* and predictors *standlrt* and *girl*:




| | variable | ESS | sd | parameter | mean | zscore |
|---|---|---|---|---|---|---|
| 1 | | 5795.0 | 0.0223966710742 | tau | 1.01530779531 | 45.3329779209 |
| 2 | intercept | 1679.0 | 0.024397158694 | beta_0 | -0.140384374851 | 5.75412803645 |
| 3 | girl | 1661.0 | 0.0313266019613 | beta_1 | 0.233549417316 | 7.45530643904 |
| 4 | | 5780.0 | 0.0217228903881 | sigma2 | 0.985401953167 | 45.3623774535 |
| 5 | | 5783.0 | 0.0109417191326 | sigma | 0.992613838282 | 90.7182707083 |
| 6 | | 3817.0 | 2.41720340611 | deviance | 11458.6724167 | 4740.46676739 |

*Figure 28*

So here we see the textual output indicating that *girl* has a significant effect on *normexam*. We can use the *if-do* block to perform more advanced operations like running different templates depending on the result of the evaluated statement, and we can also nest *if* statements to create more complex structures too.

You should now hopefully have an idea of how, through conditional blocks and comment blocks, we can produce a system that can give feedback and take the user through the workflow in different ways.

## 2.7 Templates that do their own interrogation

We will finish this practical by introducing a couple of templates that have some built-in interrogation of their outputs. Firstly we will replace the *Histogram* template that we used towards the start of our workflow with a template by the name of *Histskew* which gives textual feedback about the shape of the variable.

In addition, whilst we haven't greatly dwelt on the fact that we are fitting our model using MCMC estimation, we can pay our choice of method a little more attention here by checking whether we have run our MCMC chains for long enough. Indeed, we have created a template called *MCMCExplanation* which aims to do precisely that.

## 2.8 Checking for skewness

Let's begin by replacing the current *Histogram* template in the workflow. If we continue from the workflow as stands we will need to **Expand** the block entitled *response histogram* (to make life easier we can also **Collapse** the *univariate model fitting* block). The workflow will then look as follows:


*Figure 29*

The *Histskew* template actually has the same inputs as the *Histogram* template, so we only need to change the name in the *Template* block and then add the additional outputs as follows:

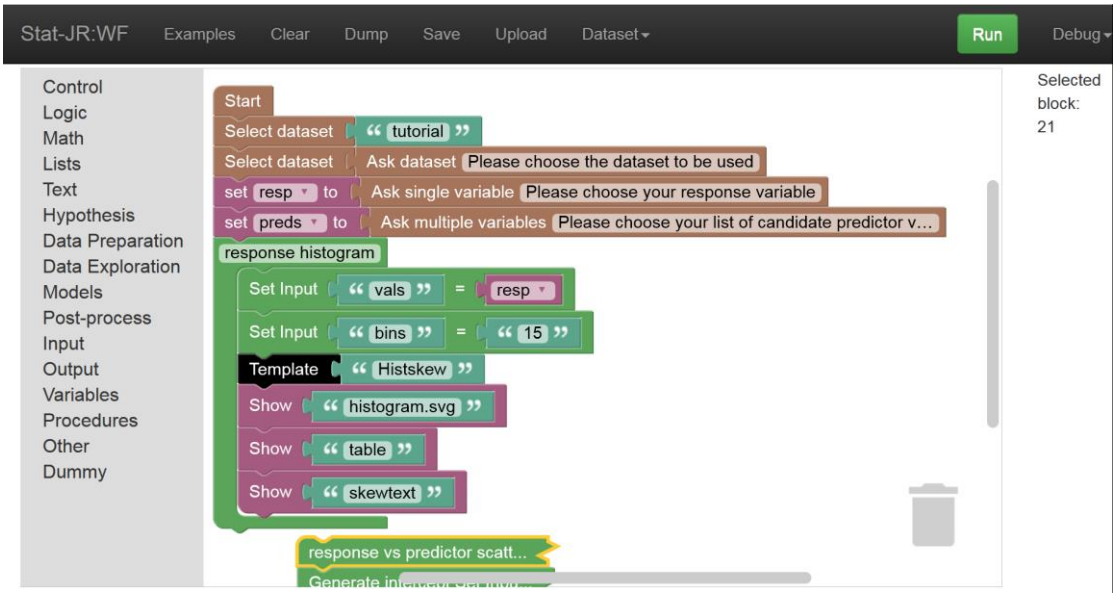



*Figure 30*

Of course, we could have discovered the names of the outputs we required by running the *Histskew* template in TREE, but for brevity we've done that for you. We've also detached (but not deleted) the section of workflow downstream of the *response histogram* group of blocks; this is simply so our outputs of interest are returned more quickly without having to first wait for all the models to fit. Running this workflow we see the following outputs:

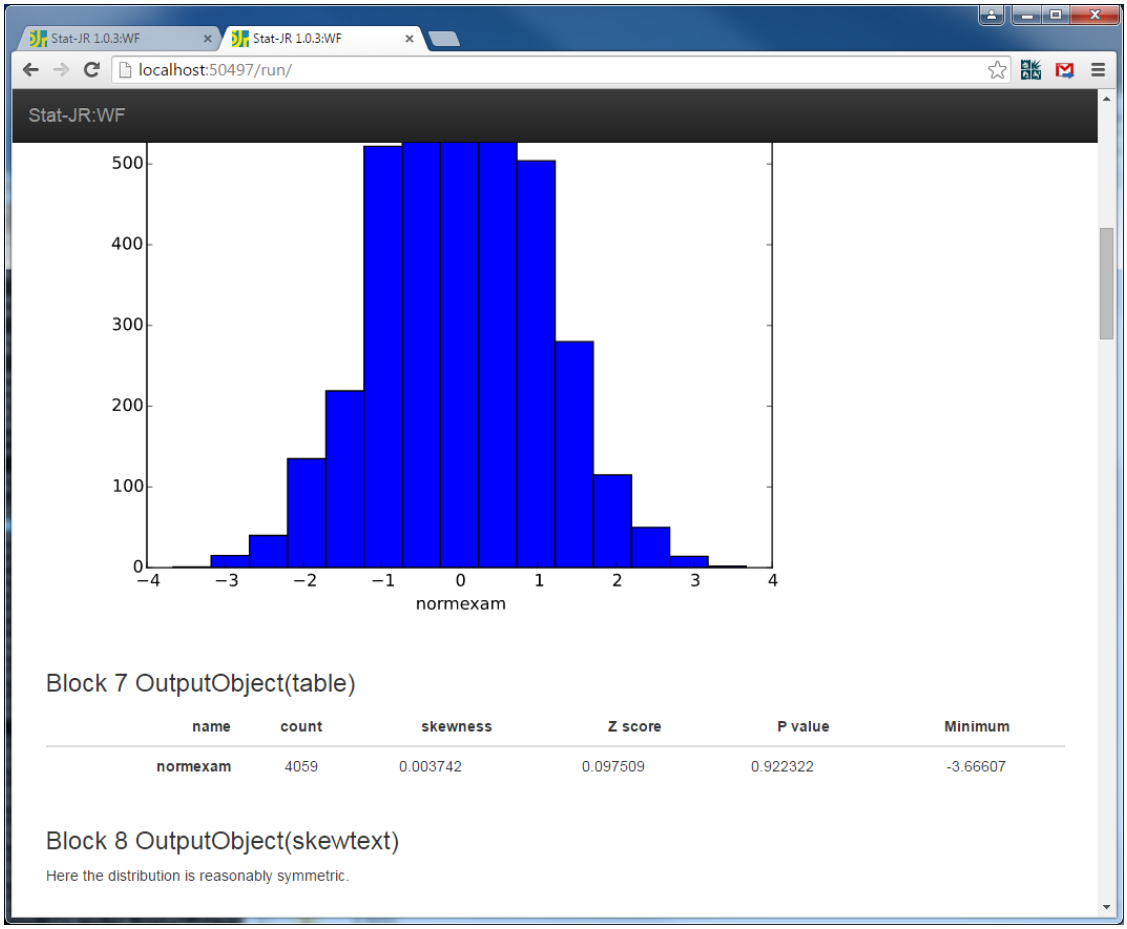


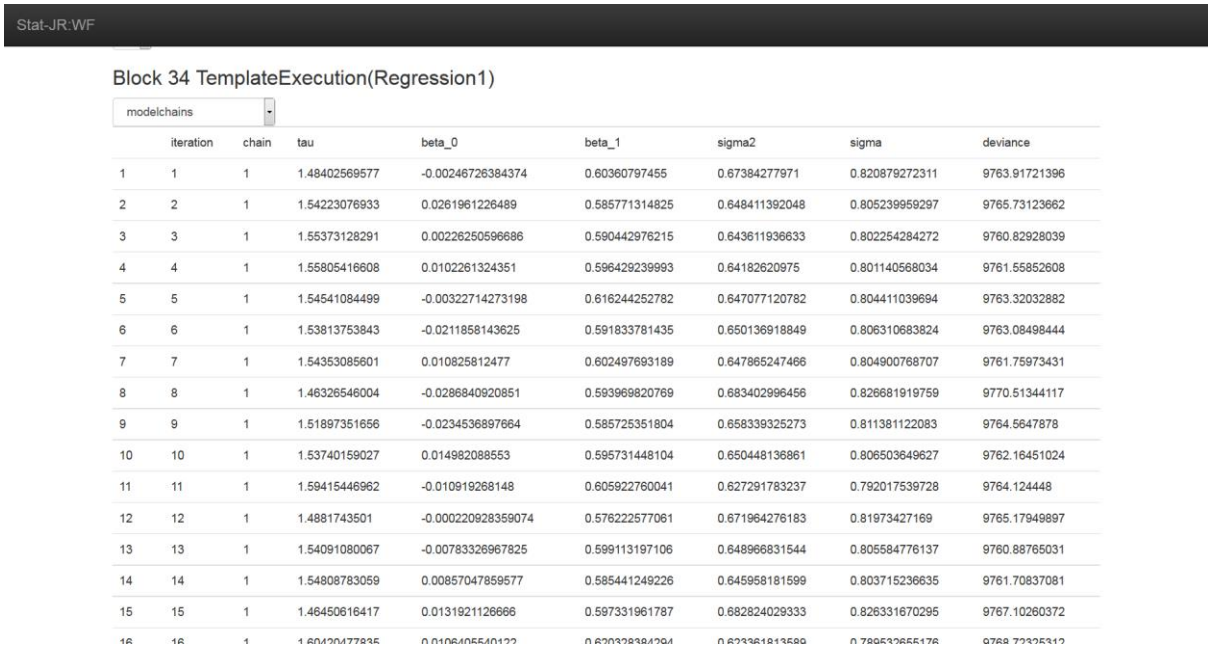

*Figure 31*

So the *Histskew* template simply works out the skewness of the column of numbers (given in the *table* along with its significance) and based on this statistic provides a textual output providing some (hopefully appropriate) information about the distribution. We could use this in a statistical analysis assistant to potentially suggest fitting a transformed response variable to make normality of the residuals more likely if the data are very skewed (whilst remembering it is the residuals not the response that is assumed normally distributed; see an example template referred to in Section 2.10 which performs some transformations).

Attach the latter part of the workflow back onto the former part of it, and save it as *prac2_8.xml* before continuing.

## 2.9 MCMC Explanation template

We will next add some extra blocks to the *univariate model fitting* group of blocks and so let's expand this group, and collapse the *response histogram* group. To provide feedback on the MCMC chains, we need to select them from amongst the output of the *Regression1* template and then feed them into an appropriate operation. The *Regression1* template produces an output object called *modelchains*. To illustrate its structure we've selected it from the pull-down list of outputs after running the workflow in the screenshot below:




| | iteration | chain | tau | beta_0 | beta_1 | sigma2 | sigma | deviance |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1.48402569577 | -0.00246726384374 | 0.60360797455 | 0.67384277971 | 0.820879272311 | 9763.91721396 |
| 2 | 2 | 1 | 1.54223076933 | 0.0261961226489 | 0.585771314825 | 0.648411392048 | 0.805239959297 | 9765.73123662 |
| 3 | 3 | 1 | 1.55373128291 | 0.00226250596686 | 0.590442976215 | 0.643611936633 | 0.802254284272 | 9760.82928039 |
| 4 | 4 | 1 | 1.55805416608 | 0.0102261324351 | 0.596429239993 | 0.64182620975 | 0.801140568034 | 9761.55852608 |
| 5 | 5 | 1 | 1.54541084499 | -0.00322714273198 | 0.616244252782 | 0.647077120782 | 0.804411039694 | 9763.32032882 |
| 6 | 6 | 1 | 1.53813753843 | -0.0211858143625 | 0.591833781435 | 0.650136918849 | 0.806310683824 | 9763.08498444 |
| 7 | 7 | 1 | 1.54353085601 | 0.010825812477 | 0.602497693189 | 0.647865247466 | 0.804900768707 | 9761.75973431 |
| 8 | 8 | 1 | 1.46326546004 | -0.0286840920851 | 0.593969820769 | 0.683402996456 | 0.826681919759 | 9770.51344117 |
| 9 | 9 | 1 | 1.51897351656 | -0.0234536897664 | 0.585725351804 | 0.658339325273 | 0.811381122083 | 9764.5647878 |
| 10 | 10 | 1 | 1.53740159027 | 0.014982088553 | 0.595731448104 | 0.650448136861 | 0.806503649627 | 9762.16451024 |
| 11 | 11 | 1 | 1.59415446962 | -0.010919268148 | 0.605922760041 | 0.627291783237 | 0.792017539728 | 9764.124448 |
| 12 | 12 | 1 | 1.4881743501 | -0.000220928359074 | 0.576222577061 | 0.671964276183 | 0.81973427169 | 9765.17949897 |
| 13 | 13 | 1 | 1.54091080067 | -0.00783326967825 | 0.599113197106 | 0.648966831544 | 0.805584776137 | 9760.88765031 |
| 14 | 14 | 1 | 1.54808783059 | 0.00857047859577 | 0.585441249226 | 0.645958181599 | 0.803715236635 | 9761.70837081 |
| 15 | 15 | 1 | 1.46450616417 | 0.0131921126666 | 0.597331961787 | 0.682824029333 | 0.826331670295 | 9767.10260372 |
| 16 | 16 | 1 | 1.60420477835 | 0.0106405540122 | 0.620328384294 | 0.623361813589 | 0.789532655176 | 9768.72325312 |

*Figure 32*

As you can see, there is a column for each parameter, with rows corresponding to the value at each *iteration* of each *chain* (chains 2 and 3 appear further down), so here we have our chains.

We will add some MCMC explanations after the model fit, so let's insert some blocks under the *Show: ModelResults* block, changing our working dataset to *modelchains*. We will run the template *MCMCExplanation* which requires one input only (*incol*), namely an MCMC chain. The output we are interested in is called *mcmctext*:

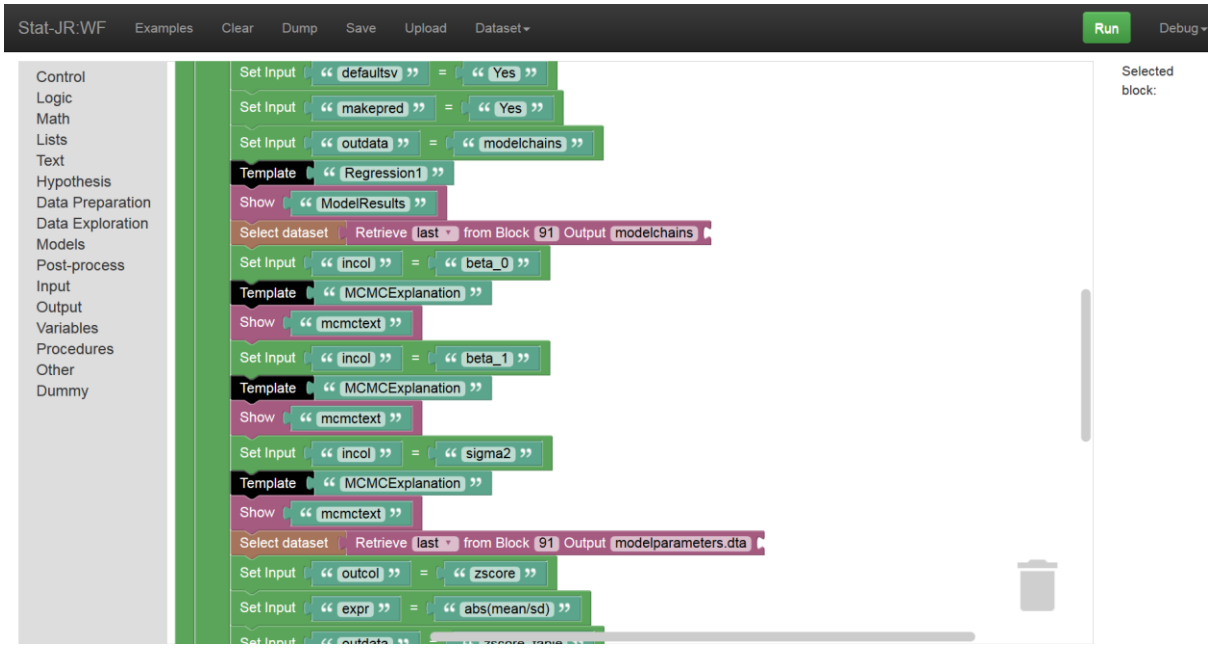



*Figure 33*

So here we have changed the dataset (to *modelchains*) and then repeated three steps of setting the *incol* input, running the *MCMCExplanation* template and showing the resulting object of interest (*mcmctext*). We do this for the intercept (*beta_0*) the slope (*beta_1*) and the residual variance (*sigma2*; as an exercise, perhaps at the end of the practical, you may like to try converting this to a loop). Save this workflow as *prac2_9.xml*.

If we **Run** it you will see it creates lots of output, including the *mcmctext* object we have just added, e.g.:

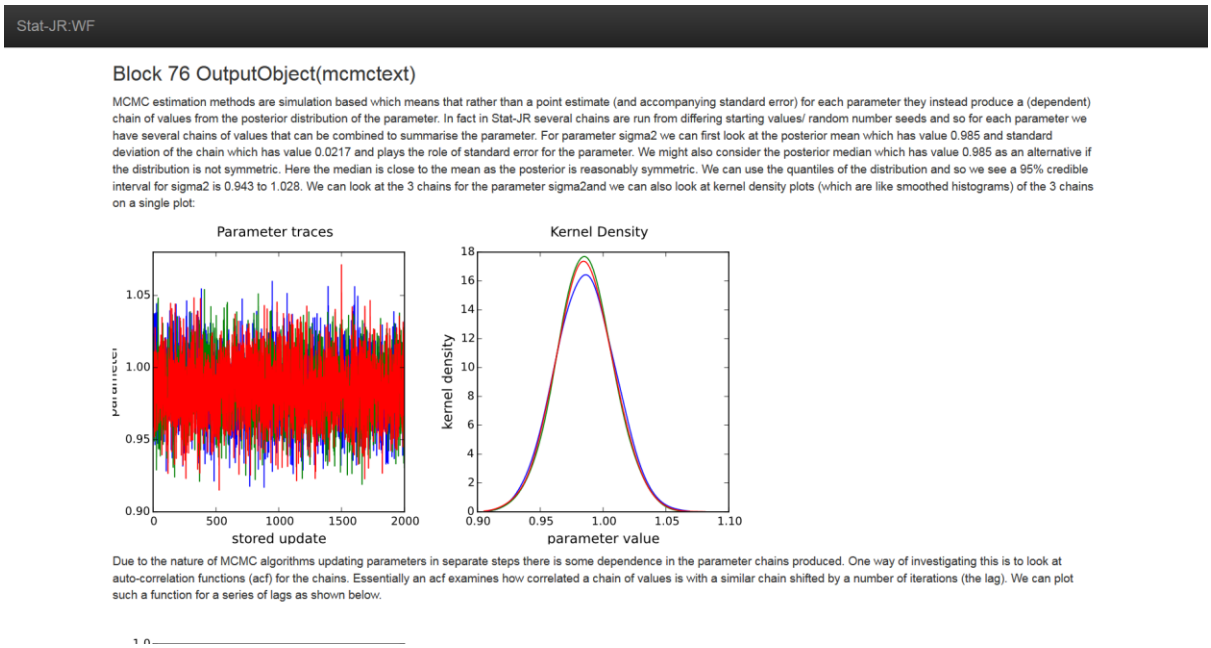



*Figure 34*

Within this template we have written code to interrogate the chains that come out of the MCMC algorithm, and have used the results to tailor output in the style of a short report. The hard work's done in the template here, and this is an alternative to coding such interrogations within a workflow, albeit one which is somewhat less transparent as the user would need to go into the template code if they wished to change what is written, or query the algorithms. We might also like to allow the user to decide whether to run the chain for longer based on the diagnostics, and so this would require interaction within the workflow.

## 2.10 Other workflows and templates to explore

In this practical we've briefly touched on some of the things a user may wish to do if planning to conduct linear regression with their data. We explored a very constrained number of options: e.g. we plotted a histogram, but of course a user may wish to inspect a variable's distribution via a different sort of plot, such as a densityplot, or even a box-and-whisker plot (or maybe all three), and they may wish to have more control over plot settings for each of these too; i.e. even when considering the visual inspection of a univariable distribution we can, of course, think of a number of blocks, templates, and inputs our workflow could incorporate. We also noted earlier that the user might be interested in exploring bivariate distributions via alternative means too, such as via cross-tabulations, etc. To this end, you may be interested in a draft workflow, called *EDA_for_linear_regression.xml* (EDA standing for exploratory data analysis), which produces a slightly wider variety of plots and tables for a continuous response variable, and continuous or categorical predictor variables. You can give it a go by loading it via the **Examples** button at the top of the workflow screen.

As well as expanding the number of methods via which users can explore their data, having a degree of interactivity might be helpful too: perhaps the user immediately sees the number of bins they have chosen (or have been chosen for them in default settings) in the histogram is not the best choice, and would like to immediately change it *in situ*, or to click on the plot to identify an outlying point: functionality which is not currently realised in the workflow system.

We've also produced a draft template called *Transformation* which you may be interested in looking at. This is somewhat like the *MCMCExplanation* template you used above, in that it produces a textual output describing to the user how it has performed the requested transformations, along with tables and graphs which compare, for example, the original variable to a log- and square-root-transformation. This isn't currently used in one of the example workflows, but you can use it via TREE.

## 2.11 What have we covered?

In this practical we have made a tentative start to building a 'statistical analysis assistant', in doing so encountering some new functionality, such as:

- *grouping* blocks;
- control structures, such as *if-do* blocks;
- modifying the functionality of blocks (e.g. changing *if-do* to *if-do-else*);
- selecting elements from tables;
- returning textual output from a workflow;
- investigating templates which have their own textual outputs.

In the third practical we will return to teaching-focussed examples and show how we are updating the LEMMA training materials to be used within a workflow format.

## 2.12 Appendix

From Section 2.3, here's how we set-up our loop plotting the response against each of the predictors in turn:

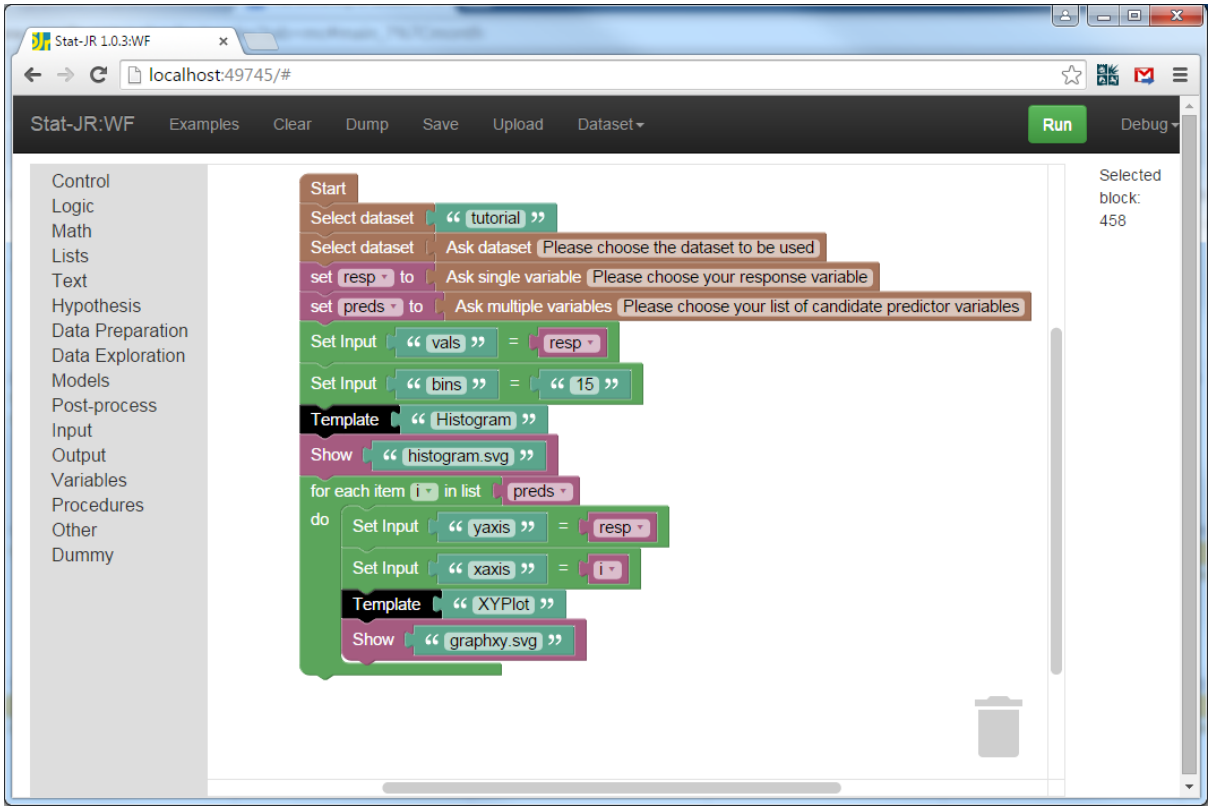



*Figure 35*

So for each predictor variable in *preds*, the four blocks in the *"do"* section of the *for-do* block will be run. The first block assigns the user-nominated variable *response* as the variable to be plotted on the y-axis, whilst the second block sets the variable currently indexed in our list of predictors (*preds*) as the variable to be the plotted on the x-axis. Finally, the *XYPlot* template is run (with these two inputs), and the output object of interest is plotted.

Save this workflow as *prac2_3.xml*.