Stat-JR LEAF Workflow Guide (1.0.4 beta release)

This documentation was written by **William Browne*, Richard Parker***, **Chris Charlton*, Danius Michaelides**** and **Luc Moreau****

*Centre for Multilevel Modelling, University of Bristol, UK

** Electronics and Computer Science, University of Southampton, UK.

June 2016

Citing Stat-JR:

Please cite Stat-JR as:

Charlton, C.M.J., Michaelides, D.T., Parker, R.M.A., Cameron, B., Szmaragd, C., Yang, H., Zhang, Z., Frazer, A.J., Goldstein, H., Jones, K., Leckie, G., Moreau, L. and Browne, W.J. (2016). *Stat-JR version 1.0.4.* Centre for Multilevel Modelling, University of Bristol & Electronics and Computer Science, University of Southampton, UK.

Stat-JR LEAF Workflow Guide (1.0.4 beta release)

ISBN: To be confirmed

Printed in the United Kingdom

Contents

The Stat-JR:LEAF workflow system: beta release

The Stat-JR software package was first released in 2012 as a beta version and in September 2013 as a fully-released piece of software (version 1.0.0). Since then there have been a number of updates and 2016 sees the release of Stat-JR 1.0.4: this features an additional workflow interface, LEAF (which stands for *Logging and Execution of Analysis Flows*), that is being distributed for the first time in beta form. This has been developed as part of our ESRC-funded project *"The use of interactive electronic-books in the teaching and application of modern quantitative methods in the social sciences"* (see http://www.bristol.ac.uk/cmm/research/ebooks/ for more information).

The Stat-JR:LEAF workflow system was primarily developed by **Danius Michaelides****, with additional input from **Luc Moreau**, Chris Charlton*, Richard Parker*** and **William Browne***.

To support the beta release of LEAF we have written this additional manual (to complement the existing *Beginner's Guide to Stat-JR's TREE interface*, the *Quick-start guide to the Stat-JR 1.0.4 TREE interface*, the *Advanced User's Guide to Stat-JR* and the *DEEP eBook Reader and Authoring Guide*). This guide is self-contained and does not assume that the reader is familiar with any of the other supporting guides, although each of those will provide further information regarding their corresponding topics.

This manual contains four main sections:

- In Section 1 we will introduce the TREE interface into Stat-JR and use the information we glean from that to create a simple workflow in the new workflow system.
- In Section 2 we will introduce some of the other work in our ESRC grant: namely investigating the development of a statistical analysis assistant.
- In Section 3 we will look at linking the workflow system to the training materials available in the LEMMA Multilevel Modelling Online Course (http://www.bristol.ac.uk/cmm/learning/online-course/).
- In Section 4 we will look at how we can export workflows into an eBook so that we can link the workflow system with the DEEP eBook interface, and also investigate, in greater depth, the system's logging features to "complete the loop" and produce a workflow from user interactions.

The materials in Section 1 to Section 3 are based on those used in teaching workshops and we are grateful to attendees at these workshops for their helpful comments that have greatly improved the final system.

* Centre for Multilevel Modelling, University of Bristol, UK

** Electronics and Computer Science, University of Southampton, UK.

Section 1 Getting Started with Stat-JR workflows

1.1 Overview

This manual is designed to introduce new users to Stat-JR and in particular to its new workflow interface LEAF (*Logging and Execution of Analysis Flows*). In order to introduce the workflow interface, we will first provide an overview of how to use the TREE (*Template Reading and Execution Environment*)[1] interface to Stat-JR and will briefly touch on certain aspects of the Python language (https://www.python.org/) in which large portions of Stat-JR is written.

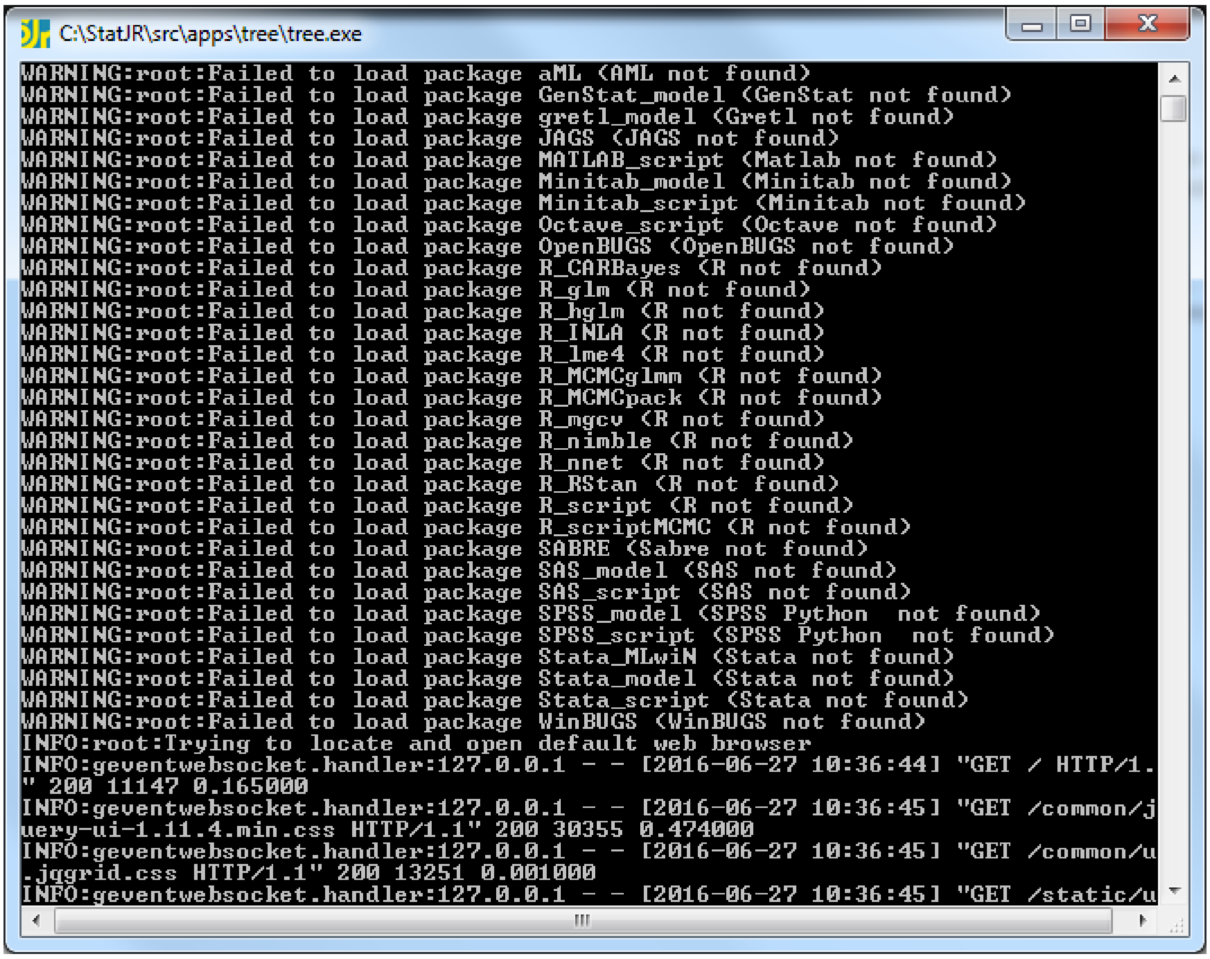
The main building block in Stat-JR is the *template*: a piece of code that performs operations one might associate with a (statistical) software package. For example, one template might draw a certain type of graph, whilst another might fit a particular statistical model, and so on. Templates are the common currency shared by the various Stat-JR interfaces – i.e. they are used in LEAF, TREE and DEEP (*Documents with Embedded Execution and Provenance*: Stat-JR's eBook-reading interface) – so it is important to have an understanding of how they work in order to use Stat-JR.

In order to perform its function appropriately, a template requires inputs from the user (just like a function call in *R* or *Stata*, for instance): for example they typically need to know which variables to use, and might need input concerning estimation options (for a model fit), plotting options (for a chart), etc. We will begin by illustrating this using the TREE interface.

1.2 Starting up TREE

To start we will fire-up Stat-JR TREE which we do via **All programs > Centre for Multilevel Modelling > StatJR - TREE**. When we do this we will find a command window appears which looks something like the following:

[1] For a more detailed introduction to TREE, see the *Beginner's Guide to Stat-JR's TREE interface.*

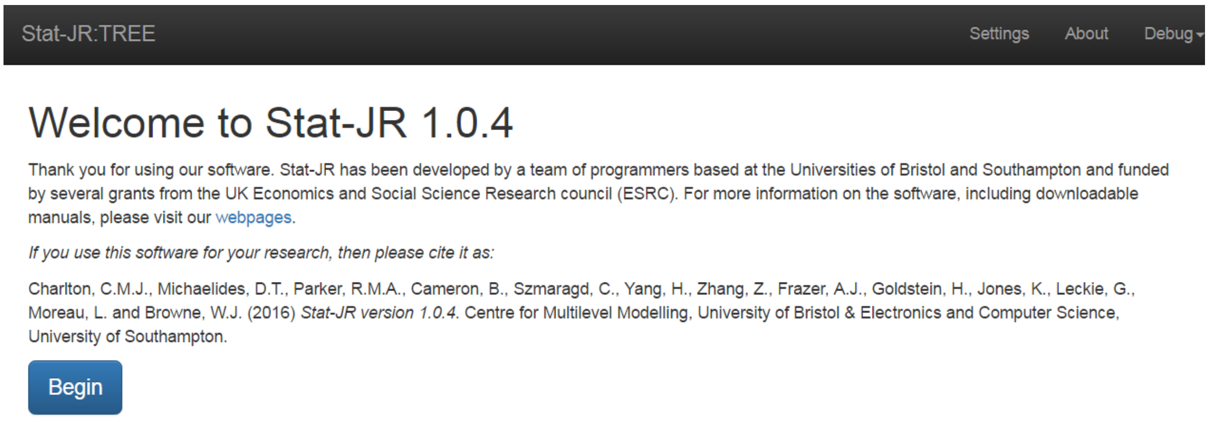


```
WARNING:root:Failed to load package aML (AML not found)
WARNING:root:Failed to load package GenStat_model (GenStat not found)
WARNING:root:Failed to load package gretl_model (Gretl not found)
WARNING:root:Failed to load package JAGS (JAGS not found)
WARNING:root:Failed to load package MATLAB_script (Matlab not found)
WARNING:root:Failed to load package Minitab_model (Minitab not found)
WARNING:root:Failed to load package Minitab_script (Minitab not found)
WARNING:root:Failed to load package Octave_script (Octave not found)
WARNING:root:Failed to load package OpenBUGS (OpenBUGS not found)
WARNING:root:Failed to load package R_CARBayes (R not found)
WARNING:root:Failed to load package R_glm (R not found)
WARNING:root:Failed to load package R_hglm (R not found)
WARNING:root:Failed to load package R_INLA (R not found)
WARNING:root:Failed to load package R_lme4 (R not found)
WARNING:root:Failed to load package R_MCMCglmm (R not found)
WARNING:root:Failed to load package R_MCMCpack (R not found)
WARNING:root:Failed to load package R_mgcv (R not found)
WARNING:root:Failed to load package R_nimble (R not found)
WARNING:root:Failed to load package R_nnet (R not found)
WARNING:root:Failed to load package R_RStan (R not found)
WARNING:root:Failed to load package R_script (R not found)
WARNING:root:Failed to load package R_scriptMCMC (R not found)
WARNING:root:Failed to load package SABRE (Sabre not found)
WARNING:root:Failed to load package SAS_model (SAS not found)
WARNING:root:Failed to load package SAS_script (SAS not found)
WARNING:root:Failed to load package SPSS_model (SPSS Python  not found)
WARNING:root:Failed to load package SPSS_script (SPSS Python  not found)
WARNING:root:Failed to load package Stata_MLwiN (Stata not found)
WARNING:root:Failed to load package Stata_model (Stata not found)
WARNING:root:Failed to load package Stata_script (Stata not found)
WARNING:root:Failed to load package WinBUGS (WinBUGS not found)
INFO:root:Trying to locate and open default web browser
INFO:geventwebsocket.handler:127.0.0.1 - - [2016-06-27 10:36:44] "GET / HTTP/1.
" 200 11147 0.165000
INFO:geventwebsocket.handler:127.0.0.1 - - [2016-06-27 10:36:45] "GET /common/j
uery-ui-1.11.4.min.css HTTP/1.1" 200 30355 0.474000
INFO:geventwebsocket.handler:127.0.0.1 - - [2016-06-27 10:36:45] "GET /common/u
.jqgrid.css HTTP/1.1" 200 13251 0.001000
INFO:geventwebsocket.handler:127.0.0.1 - - [2016-06-27 10:36:45] "GET /static/u
```

Figure 1

This command window will be where the software is actually running from and will contain debugging information, but the user interacts with the software via a web browser (although often Stat-JR will be running locally on the user's machine); this should open automatically after a few seconds, as follows[2]:

[2] Stat-JR works best with either Chrome or Firefox, so if the default browser on your machine is Internet Explorer it is best to open a different browser and copy the html path to it; this will be something like localhost:52228 (although the number will likely differ each time you run Stat-JR). You can change your default browser via **Settings** in the **Chrome** menu, or via **Options** > **General** in the **Firefox** menu (both menus are found in top-right of their respective browser windows).


Stat-JR:TREE

Settings About Debug

Welcome to Stat-JR 1.0.4

Thank you for using our software. Stat-JR has been developed by a team of programmers based at the Universities of Bristol and Southampton and funded by several grants from the UK Economics and Social Science Research council (ESRC). For more information on the software, including downloadable manuals, please visit our webpages.

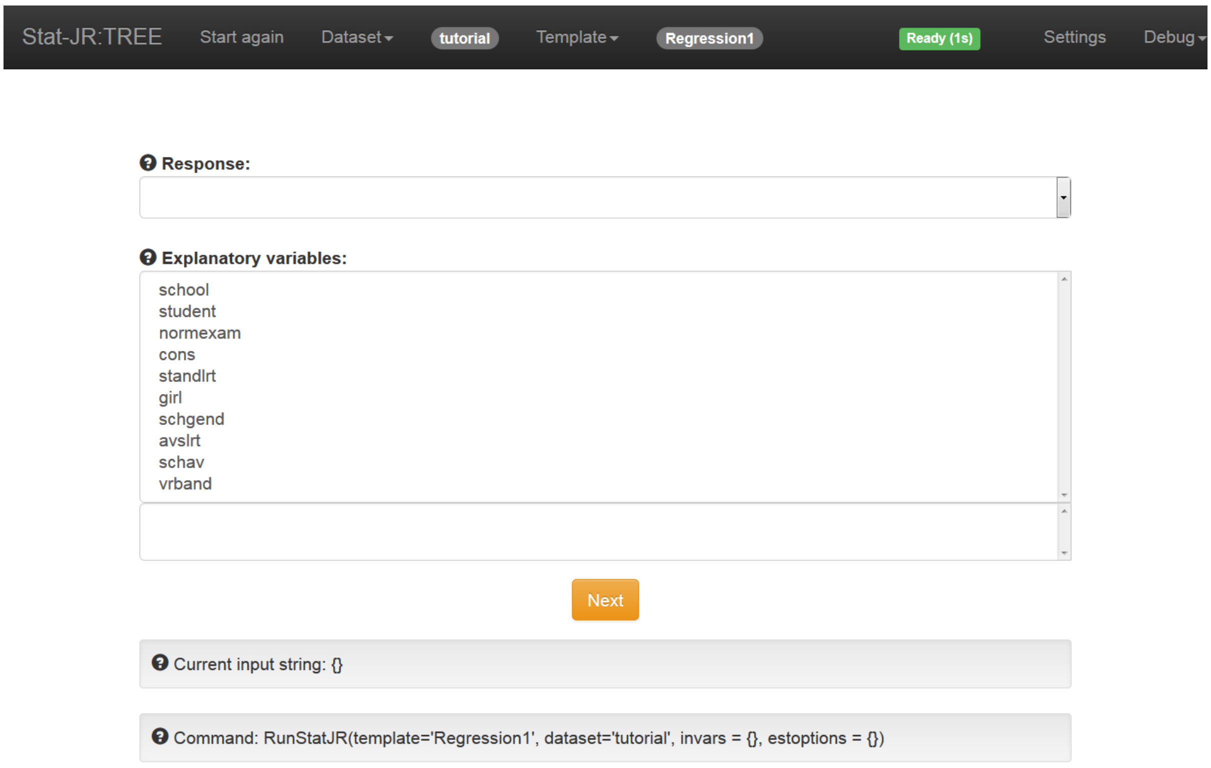
If you use this software for your research, then please cite it as:

Charlton, C.M.J., Michaelides, D.T., Parker, R.M.A., Cameron, B., Szmaragd, C., Yang, H., Zhang, Z., Frazer, A.J., Goldstein, H., Jones, K., Leckie, G., Moreau, L. and Browne, W.J. (2016) *Stat-JR version 1.0.4.* Centre for Multilevel Modelling, University of Bristol & Electronics and Computer Science, University of Southampton.

Begin

Figure 2

Now clicking on the **Begin** button will allow you to run the Stat-JR TREE software and the main screen will look as follows:


Stat-JR:TREE Start again Dataset tutorial Template Regression1 Ready (1s) Settings Debug

Response:

Explanatory variables:

school
student
normexam
cons
standlrt
girl
schgend
avslrt
schav
vrband

Next

Current input string: {}

Command: RunStatJR(template='Regression1', dataset='tutorial', invars = {}, estoptions = {})

Figure 3

The TREE interface allows the user to try out one template at a time, pairing it with one dataset, and you can see at the top of the screen pull-down menus headed **Dataset** and **Template**, and the names of the template and dataset currently selected by default (*tutorial* and *Regression1*). These pull-down menus allow you to change the template and dataset you are using (and also to view, edit and summarise the current dataset).

Below the black bar, in the central area of the window, you can see some of the inputs required for the currently selected template (*Regression1*), namely the **Response** and **Explanatory variables**, and you can further see that you are being offered variables from the default dataset (the *tutorial* dataset) as possible values for some of these inputs.

1.3 Using your own dataset

Below we will be working with one of the sample datasets provided with the Stat-JR package (one which you may be familiar with from MLwiN, namely the *tutorial* dataset). However, you might like to use your own dataset in certain sections (or try out both). The remainder of this section details how to import your dataset; if you don't have your own dataset, you can move onto Section 1.4.

Stat-JR works with datasets saved in Stata format, i.e. with a *.dta* extension. It looks for these in the*...\datasets* folder of the Stat-JR install, and also in a folder saved, by default, under your user name, e.g. *C:\Users\YourName\.statjr\datasets* (you can change the path via **Settings** in the black bar at the top of the browser window in the TREE interface).

1.3.1 If your dataset is already in .dta format

If your dataset is already in *.dta* format (see below), then you can upload it, in TREE, via (i) **Dataset > Upload** (menu options in the black bar at the top of the browser window), which will upload it into the temporary memory cache, or by (ii) saving your dataset in one of the *datasets* folders (as discussed above), and then selecting **Debug > Reload datasets** (again, accessible via the black bar at the top of the browser window).

In the case of option (i), the dataset will be available for use in the current session, but you then need to download it (as a *.dta* file) via **Dataset > Download** (e.g. saving it into the *C:\StatJR\datasets* or *C:\Users\YourName\.statjr\datasets* folders) for use in the future sessions too. In the case of option (ii), the dataset will be available in future sessions since it has been saved in one of the folders in which Stat-JR searches for datasets on start-up.

1.3.2 If your dataset is in .txt format

If, instead, you have your dataset saved as a *.txt* file, you can use Stat-JR's *LoadTextFile* template to save it into the temporary memory cache (the template *LoadTextFileMoreOptions* allows the user to specify more particulars, and can also handle string variables).

This dataset will be available for use in the current session, but you then need to download it (as a *.dta* file) via **Dataset > Download** (e.g. saving it into the *C:\StatJR\datasets* or *C:\Users\YourName\.statjr\datasets* folders) for use in the future sessions too.

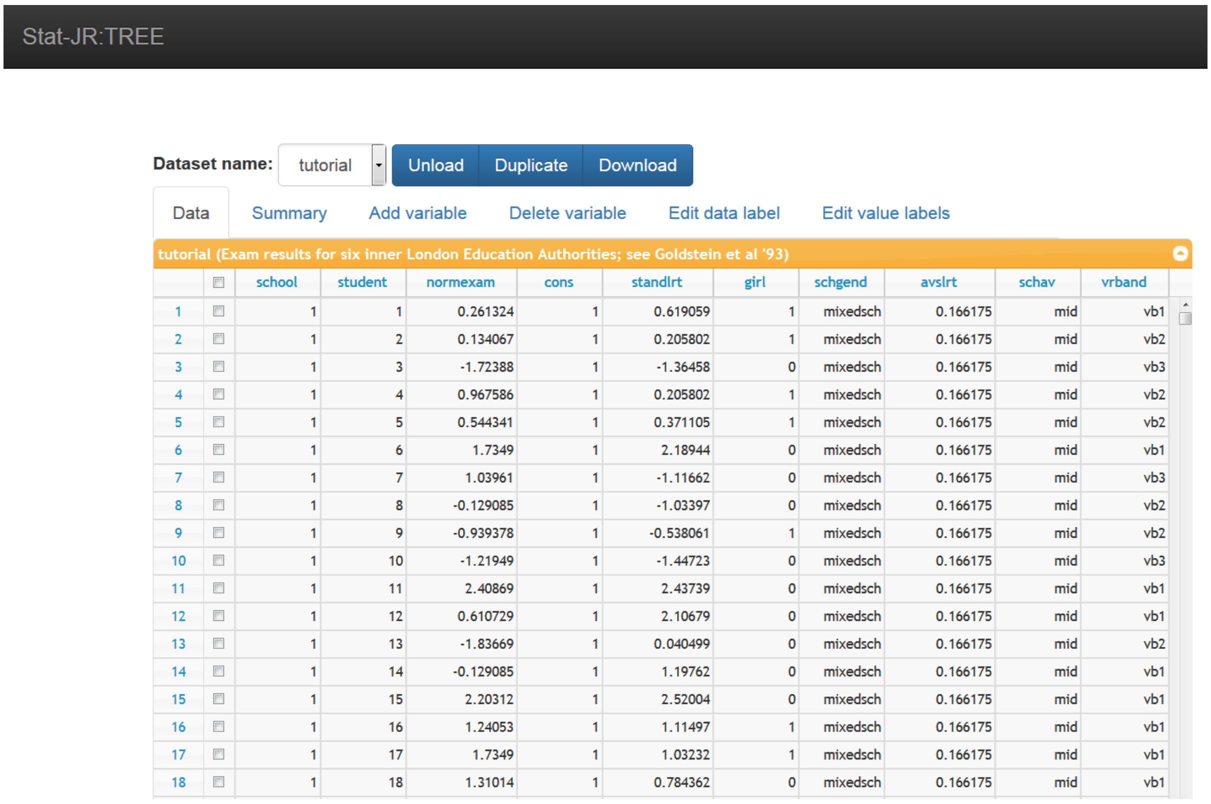
1.3.3 Converting your dataset to .dta format

Via the procedure described in Section 1.3.2 (and downloading), Stat-JR will save your *.txt* dataset as a *.dta* file, but you can also create *.dta* files via Stata, MLwiN and R (e.g. the foreign package in R).

1.4 Viewing the dataset

You can select your dataset of choice via **Dataset > Choose**, remembering to press the **Use** button once you have selected it from the list.

Once the dataset is selected, if we click on the **Dataset** menu and click on **View** we will get a second tab in our browser as shown:

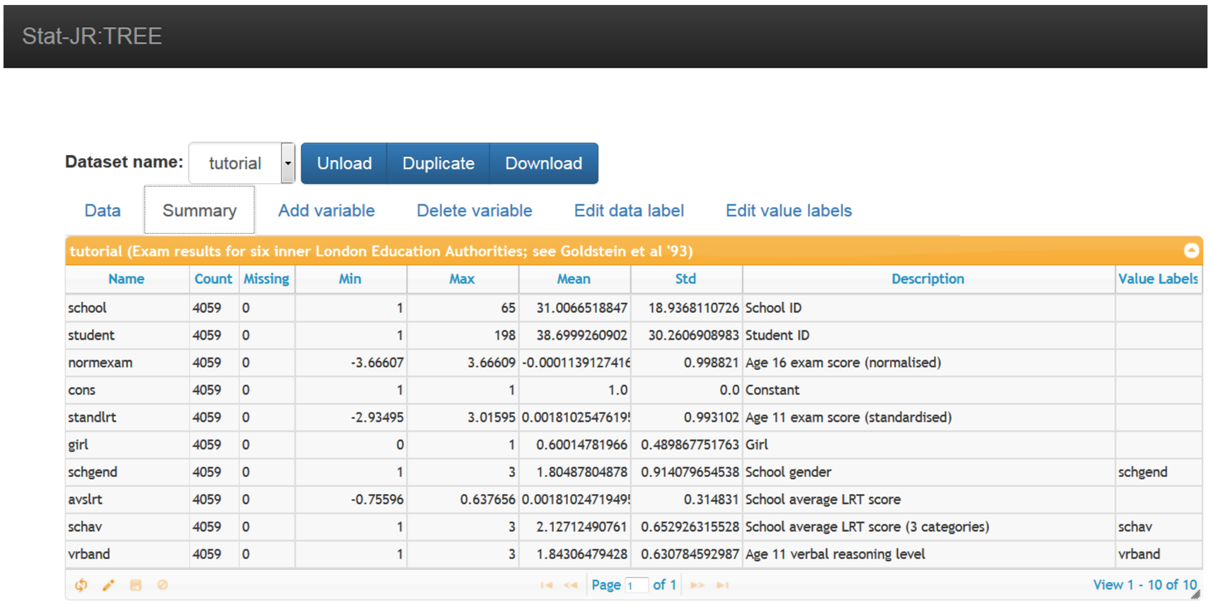


tutorial (Exam results for six inner London Education Authorities; see Goldstein et al '93)

		school	student	normexam	cons	standlrt	girl	schgend	avslrt	schav	vrband
1	☐	1	1	0.261324	1	0.619059	1	mixedsch	0.166175	mid	vb1
2	☐	1	2	0.134067	1	0.205802	1	mixedsch	0.166175	mid	vb2
3	☐	1	3	-1.72388	1	-1.36458	0	mixedsch	0.166175	mid	vb3
4	☐	1	4	0.967586	1	0.205802	1	mixedsch	0.166175	mid	vb2
5	☐	1	5	0.544341	1	0.371105	1	mixedsch	0.166175	mid	vb2
6	☐	1	6	1.7349	1	2.18944	0	mixedsch	0.166175	mid	vb1
7	☐	1	7	1.03961	1	-1.11662	0	mixedsch	0.166175	mid	vb3
8	☐	1	8	-0.129085	1	-1.03397	0	mixedsch	0.166175	mid	vb2
9	☐	1	9	-0.939378	1	-0.538061	1	mixedsch	0.166175	mid	vb2
10	☐	1	10	-1.21949	1	-1.44723	0	mixedsch	0.166175	mid	vb3
11	☐	1	11	2.40869	1	2.43739	0	mixedsch	0.166175	mid	vb1
12	☐	1	12	0.610729	1	2.10679	0	mixedsch	0.166175	mid	vb1
13	☐	1	13	-1.83669	1	0.040499	0	mixedsch	0.166175	mid	vb2
14	☐	1	14	-0.129085	1	1.19762	0	mixedsch	0.166175	mid	vb1
15	☐	1	15	2.20312	1	2.52004	0	mixedsch	0.166175	mid	vb1
16	☐	1	16	1.24053	1	1.11497	1	mixedsch	0.166175	mid	vb1
17	☐	1	17	1.7349	1	1.03232	1	mixedsch	0.166175	mid	vb1
18	☐	1	18	1.31014	1	0.784362	0	mixedsch	0.166175	mid	vb1

Figure 4

You can see the top few rows of the *tutorial* dataset, together with several tabs one could then click on. Clicking on **Summary**, for example, produces the following:



tutorial (Exam results for six inner London Education Authorities; see Goldstein et al '93)

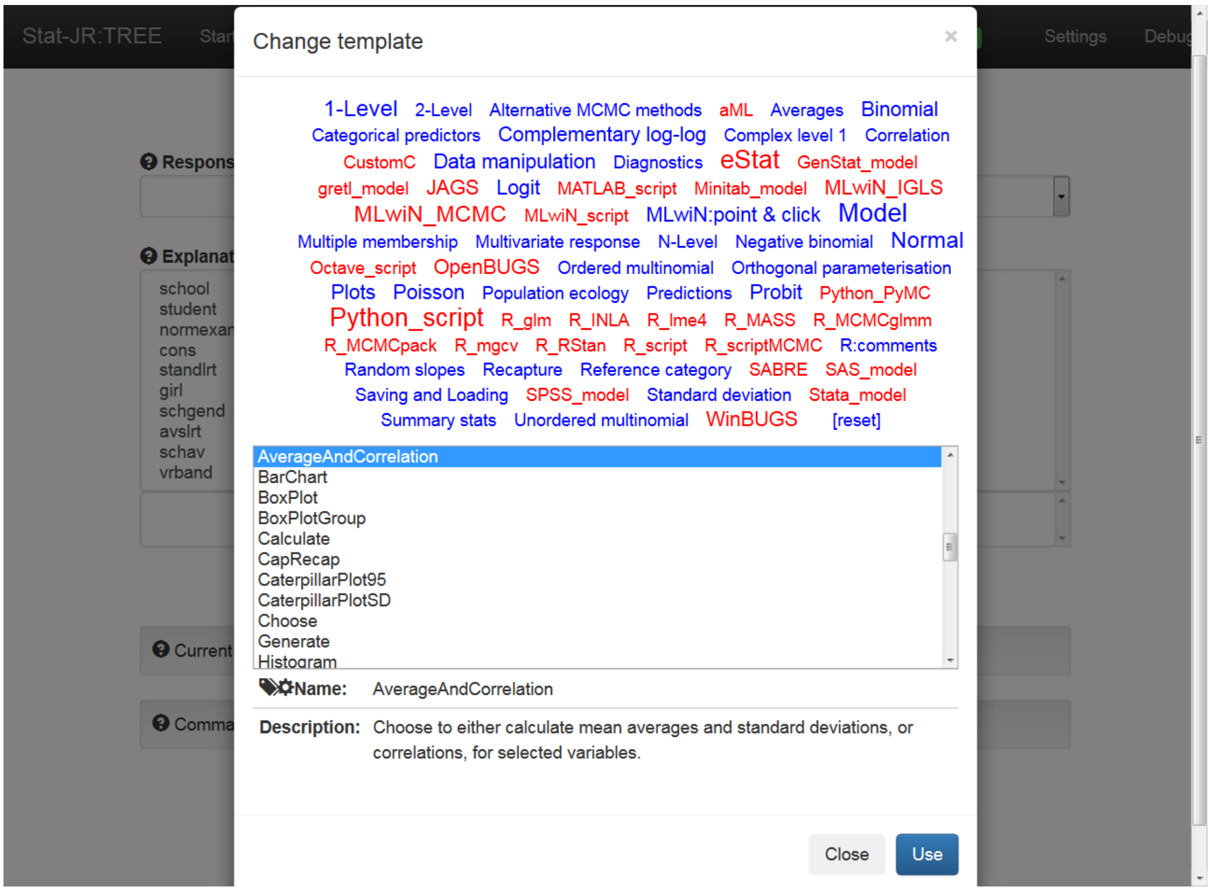
Name	Count	Missing	Min	Max	Mean	Std	Description	Value Labels
school	4059	0	1	65	31.0066518847	18.9368110726	School ID	
student	4059	0	1	198	38.6999260902	30.2606908983	Student ID	
normexam	4059	0	-3.66607	3.66609	-0.000113912741€	0.998821	Age 16 exam score (normalised)	
cons	4059	0	1	1	1.0	0.0	Constant	
standlrt	4059	0	-2.93495	3.01595	0.0018102547619!	0.993102	Age 11 exam score (standardised)	
girl	4059	0	0	1	0.60014781966	0.489867751763	Girl	
schgend	4059	0	1	3	1.80487804878	0.914079654538	School gender	schgend
avslrt	4059	0	-0.75596	0.637656	0.0018102471949!	0.314831	School average LRT score	
schav	4059	0	1	3	2.12712490761	0.652926315528	School average LRT score (3 categories)	schav
vrband	4059	0	1	3	1.84306479428	0.630784592987	Age 11 verbal reasoning level	vrband

Figure 5

This gives us, for each of our ten columns in the *tutorial* dataset, some basic statistics including the minimum, maximum, mean and standard deviation. In fact one of the first things one might do when presented with a dataset might be to produce summary statistics. The summary statistics we've just viewed are not actually produced via a template: this dataset summary table is just an in-house widget the TREE interface has to assist users with their exploratory data analysis. However, various summary statistics *can* be produced via templates, and we will do this ourselves as a means of illustrating both the TREE and workflow interfaces to Stat-JR.

Click on the first tab in the browser to return to the screen with the *Regression1* inputs showing. If you now choose the **Template** menu and click on **Choose,** a new window will appear that contains a list of templates (and a cloud of key terms to help pare down the list to those most relevant).

Scroll down and select *AverageAndCorrelation* from the list and the screen will look as follows:

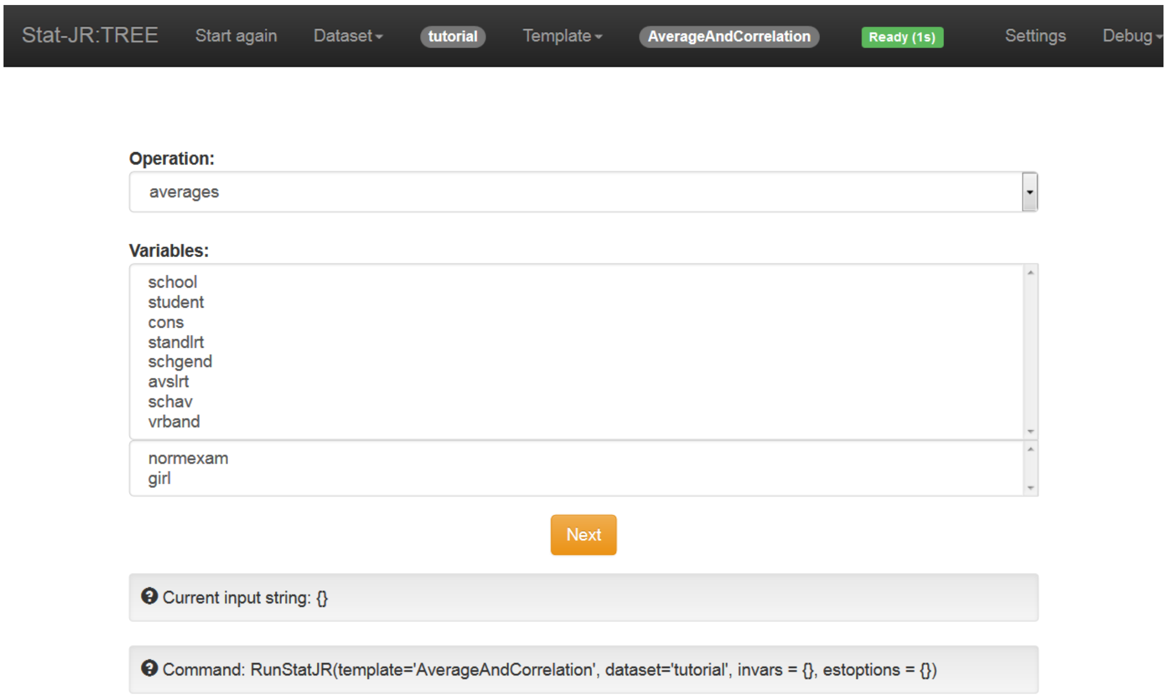

Stat-JR:TREE
Settings
Debug
Change template
Response
Explanat
Current
Comma
1-Level 2-Level Alternative MCMC methods aML Averages Binomial Categorical predictors Complementary log-log Complex level 1 Correlation CustomC Data manipulation Diagnostics eStat GenStat_model gretl_model JAGS Logit MATLAB_script Minitab_model MLwiN_IGLS MLwiN_MCMC MLwiN_script MLwiN:point & click Model Multiple membership Multivariate response N-Level Negative binomial Normal Octave_script OpenBUGS Ordered multinomial Orthogonal parameterisation Plots Poisson Population ecology Predictions Probit Python_PyMC Python_script R_glm R_INLA R_lme4 R_MASS R_MCMCglmm R_MCMCpack R_mgcv R_RStan R_script R_scriptMCMC R:comments Random slopes Recapture Reference category SABRE SAS_model Saving and Loading SPSS_model Standard deviation Stata_model Summary stats Unordered multinomial WinBUGS [reset]
AverageAndCorrelation
BarChart
BoxPlot
BoxPlotGroup
Calculate
CapRecap
CaterpillarPlot95
CaterpillarPlotSD
Choose
Generate
Histogram
Close
Use

Name: AverageAndCorrelation

Description: Choose to either calculate mean averages and standard deviations, or correlations, for selected variables.

Figure 6

If we next click on **Use** then the main screen will reappear, but this time asking for the inputs specific to this template. We can fill these in as follows (**Operation:** averages; **Variables:** normexam, girl; or variables from your own dataset if not using *tutorial*):



Operation:
averages

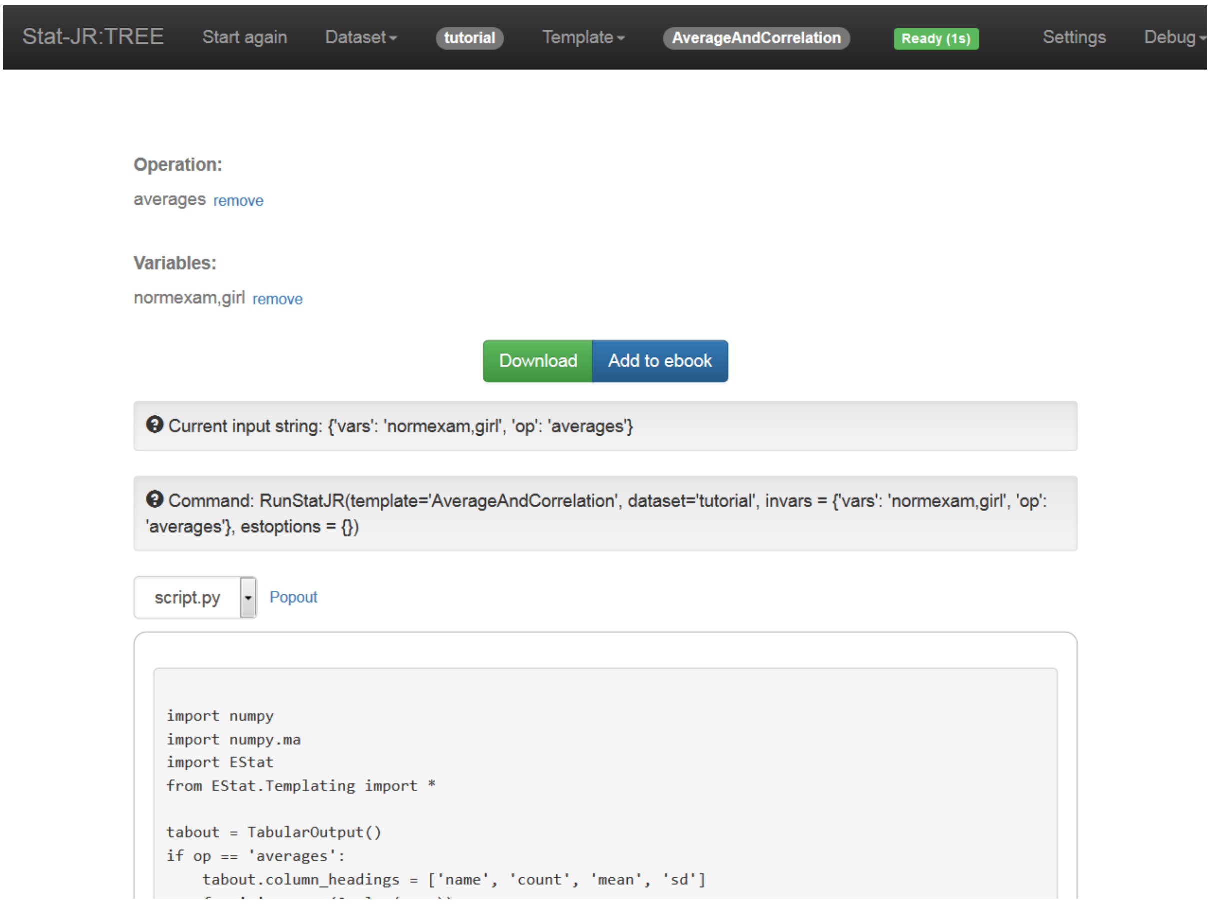
Variables:
school
student
cons
standlrt
schgend
avslrt
schav
vrband
normexam
girl

Current input string: {}

Command: RunStatJR(template='AverageAndCorrelation', dataset='tutorial', invars = {}, estoptions = {})

Figure 7

Here we have selected averages (as opposed to calculating correlations) and chosen two variables to work out averages for. If we then click on **Next** to confirm the inputs and **Run** to run the template, the screen will look as follows:



Operation:

averages remove

Variables:

normexam,girl remove

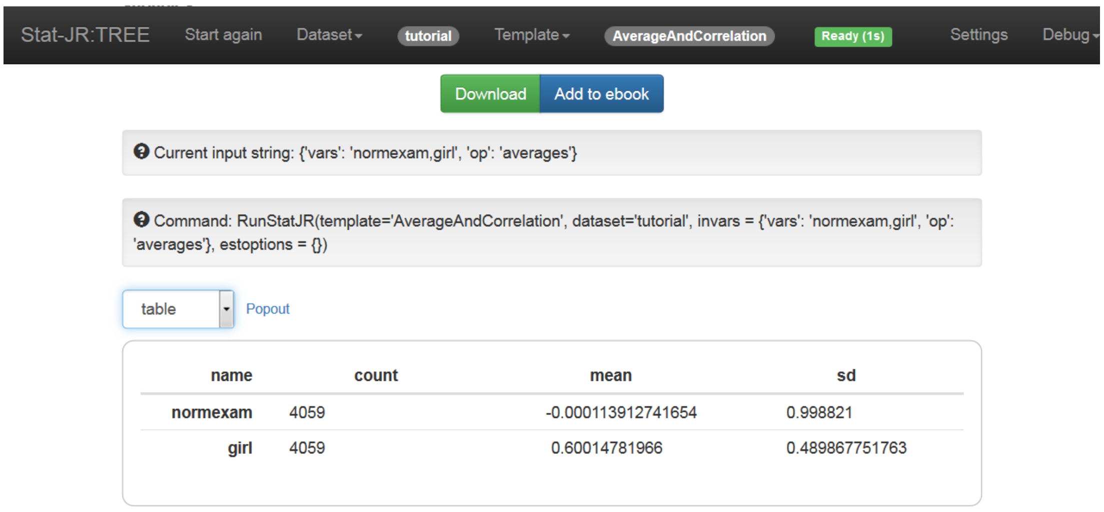
Current input string: {'vars': 'normexam,girl', 'op': 'averages'}

Command: RunStatJR(template='AverageAndCorrelation', dataset='tutorial', invars = {'vars': 'normexam,girl', 'op': 'averages'}, estoptions = {})

```
import numpy
import numpy.ma
import EStat
from EStat.Templating import *

tabout = TabularOutput()
if op == 'averages':
    tabout.column_headings = ['name', 'count', 'mean', 'sd']
```

Figure 8

At the bottom of the screen there is a results pane which displays whatever output object is selected in the pull-down list just above it. Here we see the Python script (*script.py*) that has been run to execute the template. If instead we pick the object *table* from the pull-down list of outputs then the screen looks as follows:



Current input string: {'vars': 'normexam,girl', 'op': 'averages'}

Command: RunStatJR(template='AverageAndCorrelation', dataset='tutorial', invars = {'vars': 'normexam,girl', 'op': 'averages'}, estoptions = {})

name	count	mean	sd
normexam	4059	-0.000113912741654	0.998821
girl	4059	0.60014781966	0.489867751763

Figure 9

So here we have done something really rather simple which is to execute a template that has taken the two variables we chose and worked out their means and standard deviations; these should correspond to those we have already seen in the **Dataset Summary** screen we looked at earlier.

We will shortly use this template in the workflow version of Stat-JR to create a workflow that performs the same averaging operation. For this we need to pay attention to the names of the inputs, which you can see in the grey *Current input string*[3] box and again in the *Command* box below (which is how one would run this template with these inputs in the Python command driven version of Stat-JR).

As this implies, the templates are written such that the input questions asked of the user in the browser window (in this example, **Operation** and **Variables**) might be different to the name the template actually assigns to those input objects in the background (in this example, *op* and *vars*, respectively). This simply allows the input questions posed of the user to be more expansive than the underlying assigned names, which may be shorter to spare the coder's fingers and allow for coding efficiency. We'll have a look at the template itself in a moment to illustrate how this distinction is realised in its code.

So using TREE is a useful way to test out a template and find the names of the inputs it requires, and the names of the output objects too (via the pull-down list above the results pane); i.e. we now know:

- The name of the template: *AverageAndCorrelation*
- The inputs it requires:
    - *op*, which we assigned the value *averages*
    - *vars,* which we assigned the value *normexam, girl*
- The name of the template's output most relevant to us: *table*

As well as gleaning a template's required inputs by running the template in TREE, however, you can also retrieve that information by looking at the code in the template file itself. In the Stat-JR directory from which you ran TREE, you will see there is a subdirectory called *templates*. In this subdirectory there will be a Python file for each template; for example *AverageAndCorrelation.py* contains the Python code for the template we've just run. If you open this file you will see the Python code as shown below:

```
# Copyright (c) 2013, University of Bristol and University of Southampton.

from EStat.Templating import Template

class TemplateAverages(Template):
    'Choose to either calculate mean averages and standard deviations, or correlations, for selected variables.'

    __version__ = '1.0.0'

    tags = [ 'Summary stats', 'Correlation', 'Averages', 'Standard deviation' ]
    engines = ['Python_script']

    inputs = '''
op = Text('Operation: ', ['averages', 'correlation'])
vars = DataMatrix('Variables: ')
'''

    pythonscript = '''
```

[3] The input string allows the user to specify all the inputs directly, via the **Set Inputs** option in the **Template** pull down list, without having to point-and-click through the list as we have done. If you click on **Template > Set Inputs** you will see this input string reproduced in the **Input string** box; clicking on the **Use** button populates the inputs with these values, which obviously will have no effect here, but it would if you first changed a value, or indeed used the inputs from a previously-run template execution, as selected from the **History** box above.

```
import numpy
import numpy.ma
import EStat
from EStat.Templating import *

tabout = TabularOutput()
if op == 'averages':
    tabout.column_headings = ['name', 'count', 'mean', 'sd']
    for i in range(0, len(vars)):
        var = datafile.variables[vars[i]]['data']
        tabout.add_row(vars[i], [len(var), var.mean(), var.std()])

if op == 'correlation':
    invars = numpy.ma.row_stack([datafile.variables[var]['data'] for var in vars])
    corrs = numpy.corrcoef(invars)
    tabout.column_headings = ['name']
    for j in range(0, len(vars)):
        tabout.column_headings.append(vars[j])

    for i in range(0, len(vars)):
        row = []
        for j in range(0, len(vars)):
            row.append(corrs[i, j])
        tabout.add_row(vars[i], row)

outputs['table'] = tabout
'''
```

Here you can see that the template code is structured such that it includes an *inputs* section where you can see both the prompts asked of the user (**Operation** and **Variables**) and, importantly, the names the template assigns to the values provided by the user to those prompts (*op* and *vars*, respectively; all highlighted in yellow); i.e. the latter names are the same as those appearing in the *Current input string* box in TREE. You can also see why we were offered a choice of *averages* or *correlation* as values for *op*, since these are coded as the options to be presented to the user.

Below that you will find a section of the code called *pythonscript*; this contains the Python code executed once the inputs defined in the section above have all been completed (i.e. had values assigned to them) by the user (you can see that the objects *op* and *vars* are used in this section, so the template cannot run to completion unless the user has provided values for them). On the last line of this section you can see the output name of interest (*table*; again highlighted in yellow), which is one of the outputs which appeared in TREE.

So either of these methods (via TREE, and via the template code itself) can be used to uncover the information needed by a workflow in order for it to execute the operation we have just performed in TREE. Having gleaned this information, we could 'manually' start building up such a workflow from scratch in the LEAF (workflow) interface, but TREE can help us make a start by providing blocks corresponding to our choice of input values, dataset and template. Back in the browser window you will see a **Make workflow** button just below where you specified the inputs, above the *Current input string* grey box. If you press this button a box will open entitled **Save history**. If the template execution described above is the only one you've conducted in the current Stat-JR session then the **Only include last run** box can remain unticked (otherwise tick as appropriate if you have run other template executions beforehand).

Press the **Workflow** button; you now have a few options with regard to the choice of directory in which to save it. You can simply choose any directory of your choice and then request LEAF upload it from wherever you have saved it, or you can save it into one of the two directories in which LEAF automatically looks for workflows. By default these two directories are (a) a folder under your user name, e.g. *C:\Users\YourName\.statjr\workflows*, and (b) a folder under the Stat-JR install, e.g. *C:\StatJR\workflows*[4]. To complicate matters a little further, workflows need to be saved in a subdirectory of these root folders to be automatically accessible from LEAF: e.g. if the workflow you are saving is called *my_workflow.xml*, then:

C:\Users\YourName\.statjr\workflows\my_workflow.xml

C:\StatJR\workflows\my_workflow.xml

…won't work (i.e. your workflow will not be automatically accessible from LEAF), whereas:

C:\Users\YourName\.statjr\workflows\My new workflows\my_workflow.xml

C:\StatJR\workflows\My new workflows \my_workflow.xml

…will work (i.e. your workflow will appear be automatically accessible from LEAF, and will appear under "My new workflows").

1.5 Opening Stat-JR:LEAF

We will now open LEAF: the workflow interface to Stat-JR; you can open this via **All programs > Centre for Multilevel Modelling > StatJR - LEAF**. This will fire-up another command window which will contain debugging commands and another web browser window for the workflow version of Stat-JR, as shown below:

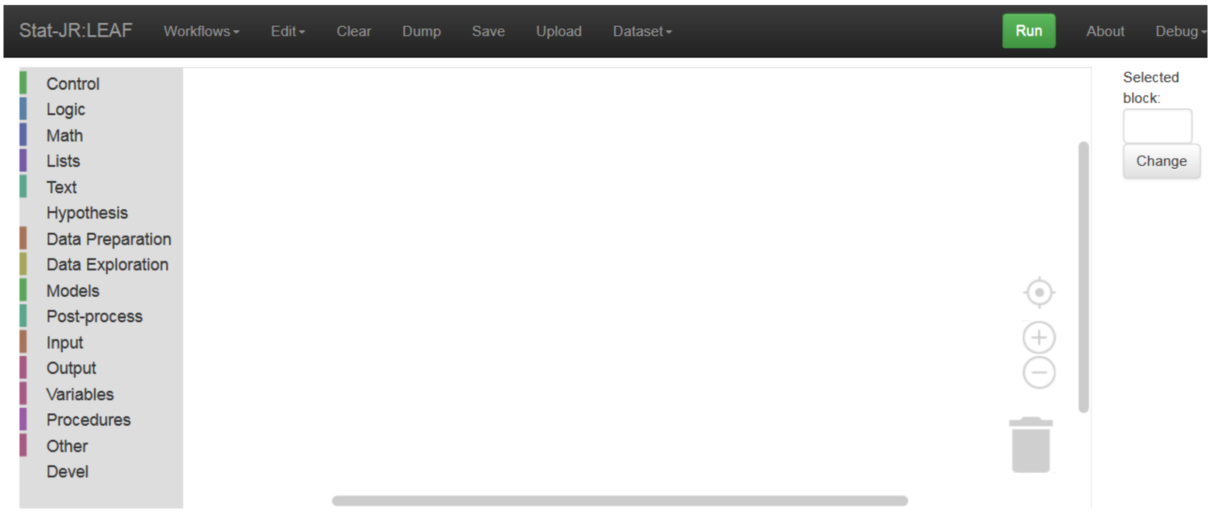


Figure 10

Stat-JR's LEAF system is still using Python as the code in the background but the web interface is using a program called *Blockly* (developed by Google; https://developers.google.com/blockly/; https://blockly-games.appspot.com/); this is a visual programming system that involves using blocks to represent operations, and has been used by a variety of applications as an aid to help people learn to code.

We will first open the workflow we have just made in TREE. Depending on where you saved it you can either do this via the **Upload** link in the black bar at the top of the LEAF interface, or – if you

[4] The distinction between these folders is that workflows saved in the Stat-JR install directory (e.g. *C:\StatJR\workflows*) will (usually) be available to all users, whereas those saved under your own user name (e.g. *C:\Users\YourName\.statjr\workflows*) will be available just to you.

have saved it to one of the directories in which LEAF automatically looks for workflows on start-up – via the **Workflows** pull-down list.[5] Note that, whichever method you choose, you are asked to make a choice as to whether you would like to **(Up)Load** or **Import** the workflow.

If you **(Up)Load** a workflow, this will clear any workflow currently displayed in the LEAF interface, replacing it by the workflow you are bringing in.

If instead you choose to **Import** a workflow, this will keep the workflow currently displayed in the LEAF interface and bring the imported workflow into the same workspace (you may need to move blocks around to see them both). This can be useful if you want to add blocks from one to the other.

Having opened the workflow we saved in TREE, our screen will look as follows (to make things a little clearer we've increased the size of the blocks here by pressing the **+** button just above the bin symbol towards the bottom-right corner):

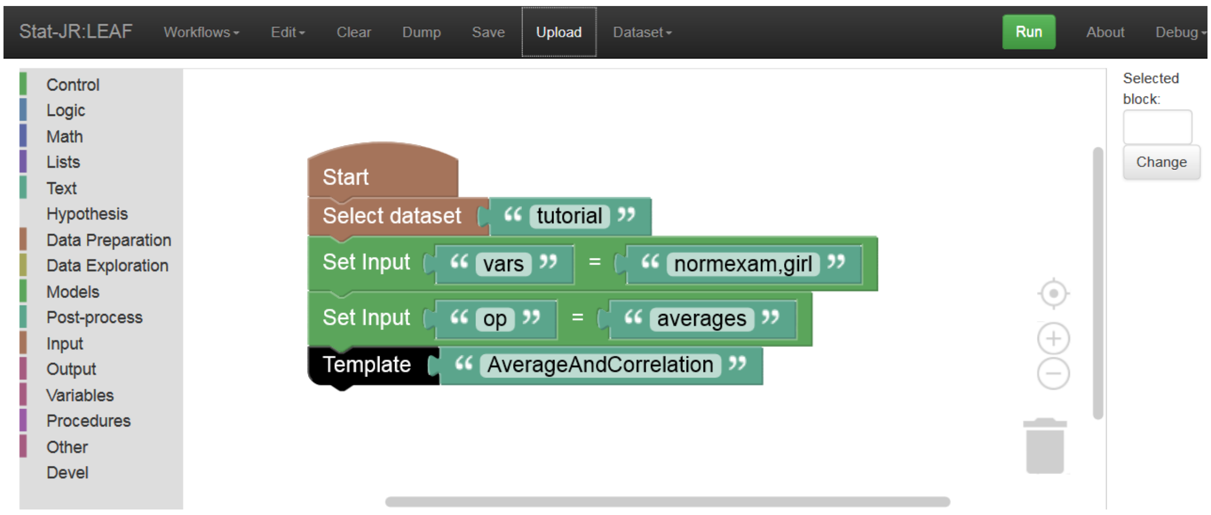


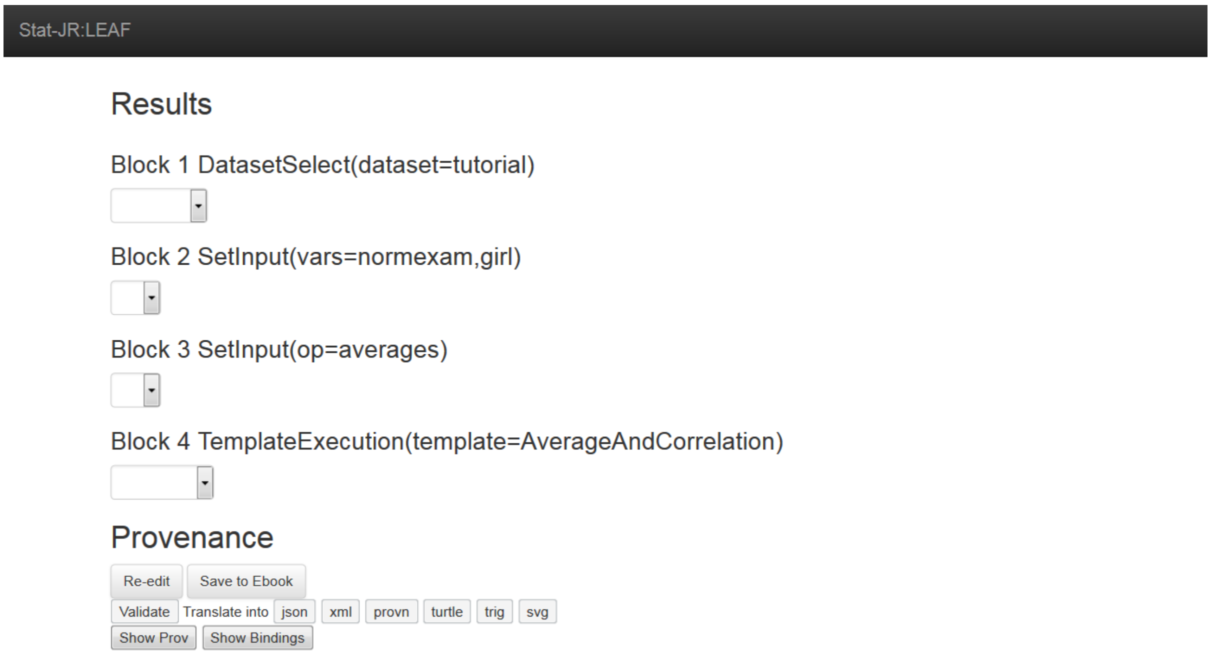
Figure 11

These blocks represent our earlier choices in the TREE interface as a workflow, and we could have instead constructed it by choosing blocks from the menu on the left-hand side and dragging them into the central area.

First we have a *Start* block, whose simple purpose is to indicate the start of the workflow (you will find this in the **Control** menu). Then we have a *Select dataset* block (**Data Preparation** menu), with a text block (**Text** menu) attached to the right of it indicating we wish to select the *tutorial* dataset. Next we have two *Set Input* blocks (**Models** menu) which specify that the inputs *"vars"* and *"op"* have the values *"normexam,girl"* and *"averages"*, respectively. As we saw earlier, *"vars"* and *"op"* are the inputs for which the *AverageAndCorrelation* template needs values in order to run to completion. Finally we have a *Template* block (**Devel** menu) with a text block to the right of it indicating we wish to run the *AverageAndCorrelation* template.

If you press the **Run** button, towards the top right-hand corner, a new tab will open in the browser, and after a short time the following content will appear:

[5] NB If you have saved it to one of these folders *during* a session of LEAF, then you can refresh the list accessible via the **Workflows** pull-down by pressing **Debug > Reload workflows**.


Stat-JR:LEAF

Results

Block 1 DatasetSelect(dataset=tutorial)

Block 2 SetInput(vars=normexam,girl)

Block 3 SetInput(op=averages)

Block 4 TemplateExecution(template=AverageAndCorrelation)

Provenance

Figure 12

The current output from workflows is a little crude: essentially we get a list starting with “Block 1” and numbered through to “Block 4”, corresponding to the four blocks (counting vertically, downstream from the *Start* block) in the workflow. If we click on the pull-down list just below Block 4 (the *Template* block) we can see the outputs from the template execution; e.g. selecting *table* displays the output we saw earlier in the TREE interface, containing selected summary statistics for the variables *normexam* and *girl*.

Returning to the browser tab containing the workflow blocks, we can request that this table is displayed automatically by using a *Show* block from the **Output** menu. We need to do this manually as the **Make workflow** tool in the TREE interface currently does not have the facility to specify which output to show. So, click on **Output** in the left-hand menu to find the *Show* block:

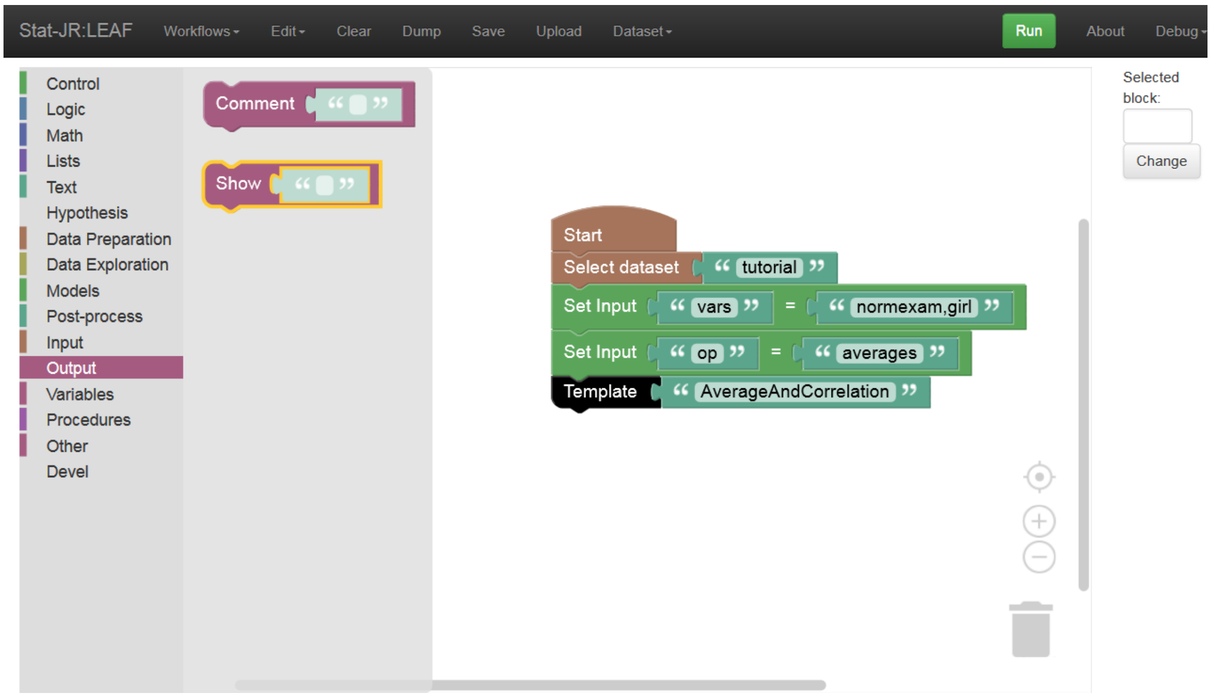


Figure 13

Having located the *Show* block, place your cursor over it and, holding down the left mouse button, drag it into the central workflow area. You will see there is a groove into which it can fit under the *Template* block: if you attach it, it should join to it with a satisfying clicking noise (if your speakers are on), and visually 'snap' into place to look as follows:

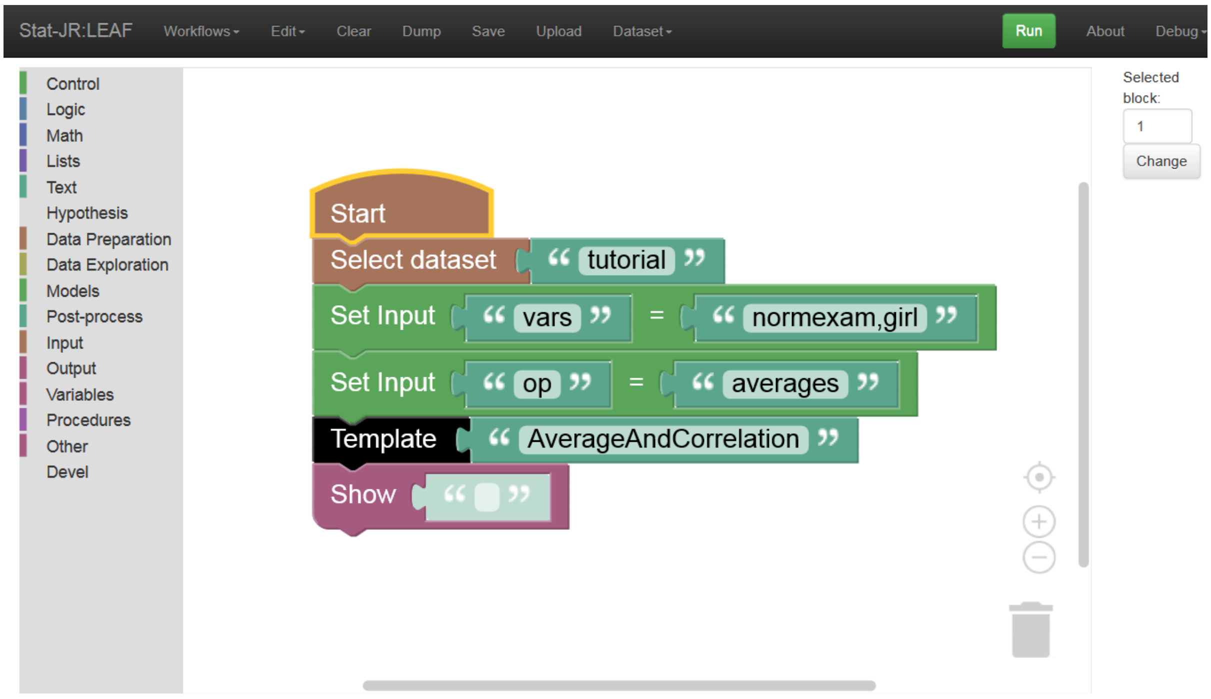


Figure 14

Here you can see, in a hollow towards the right of the *Show* block, a faint 'shadow block': this is a prompt, or placeholder, to save the user pulling in a block separately from the palette of blocks on the left-hand side. It's not actually an active block until we decide to type something in it, so let's go ahead and type the output we want to display, *table*, as follows:

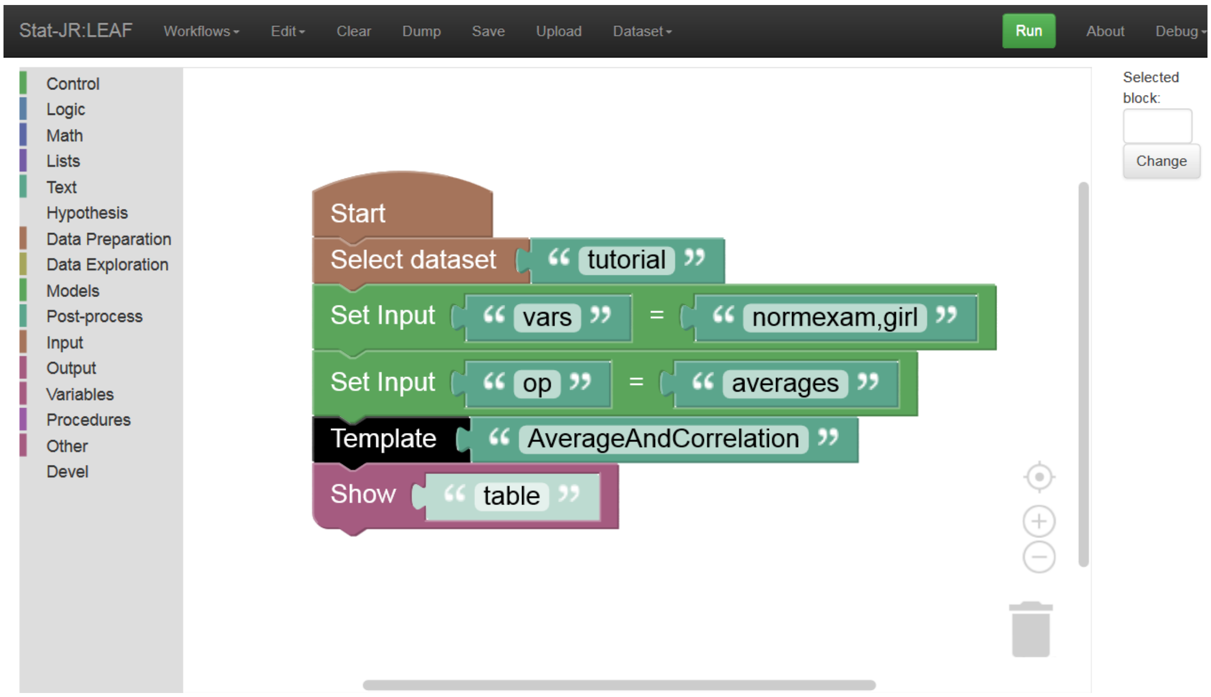
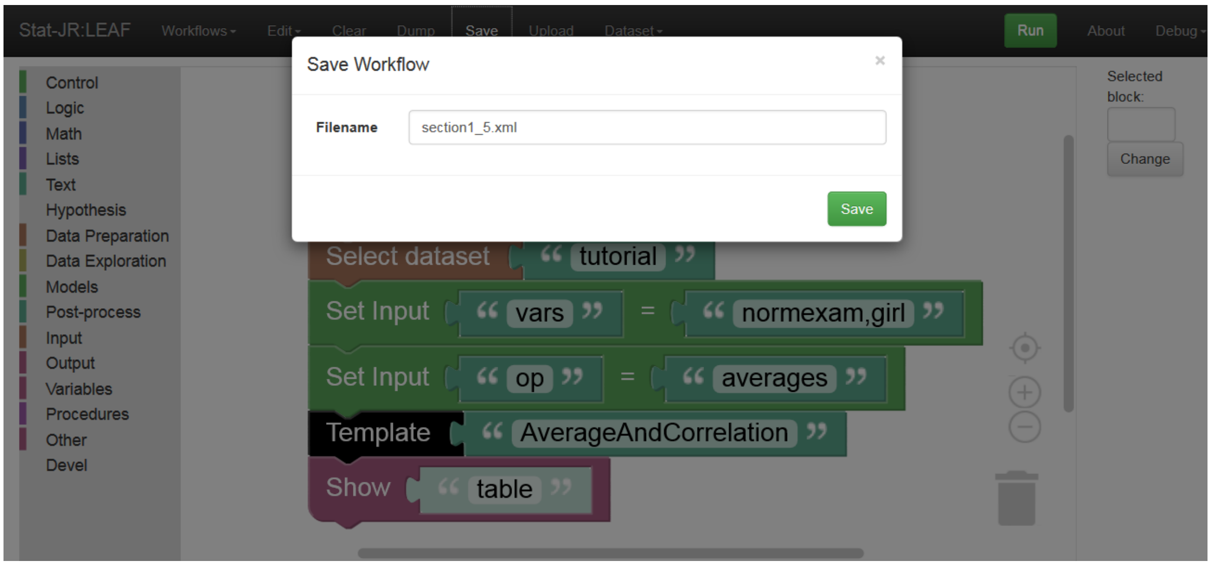


Figure 15

At this point it would be good to save our modified workflow, so click on **Save** and specify a name (we will name it after this section of the manual, and choose *section1_05.xml*) thus:

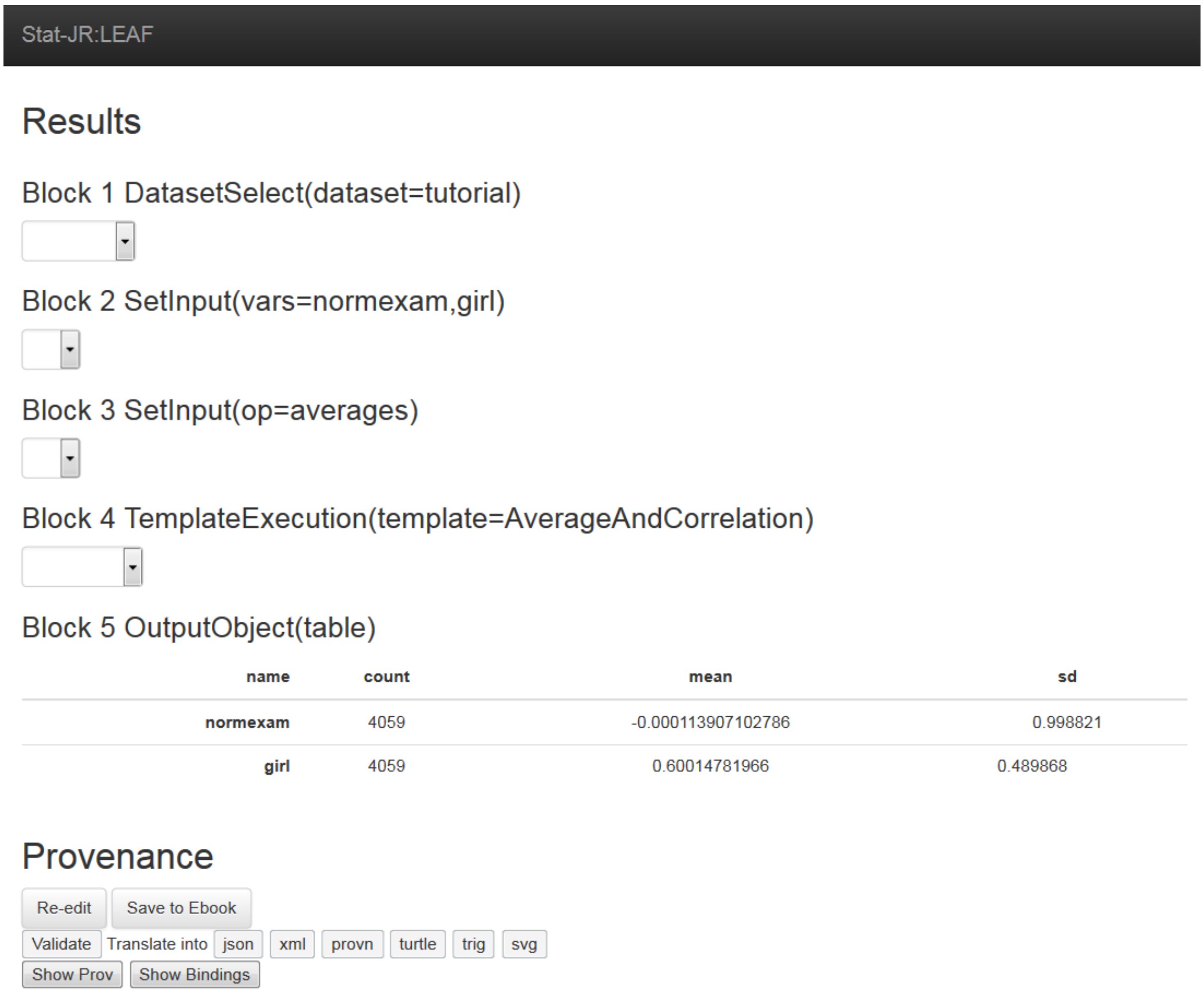

Stat-JR:LEAF
Save Workflow
×
Save
Run
Select dataset " tutorial "
Set Input " vars " = " normexam,girl "
Set Input " op " = " averages "
Template " AverageAndCorrelation "
Show " table "

Filename: section1_5.xml

Figure 16

You will be asked for a directory, so store this file somewhere you know where to find it!

If you press **Run**, a separate tab will again open, but this time displaying the *table* towards the end:



Results

Block 1 DatasetSelect(dataset=tutorial)

Block 2 SetInput(vars=normexam,girl)

Block 3 SetInput(op=averages)

Block 4 TemplateExecution(template=AverageAndCorrelation)

Block 5 OutputObject(table)

name	count	mean	sd
normexam	4059	-0.000113907102786	0.998821
girl	4059	0.60014781966	0.489868

Provenance

Figure 17

1.6 Making our workflow interactive

As things stand we have what is effectively a log of what we did in TREE and for which there is no interactivity. Next we will show how we can make the workflow interactive by asking the user which variables they want to use to calculate the averages.

We will firstly do this rather crudely: click on the first tab to return to the workflow creation screen. Next we will remove the *Set Input* block for *"vars"* from the workflow; there's no need to delete it: we can simply set it to one side of the workflow as shown below. The workflow system doesn't currently have a separate place to store fragments of workflow; however, only those blocks that are contiguous with the *Start* block will be executed by the **Run** button, so effectively we've rendered these inactive by removing them from the workflow stream: i.e. we're simply storing them to the side for now:

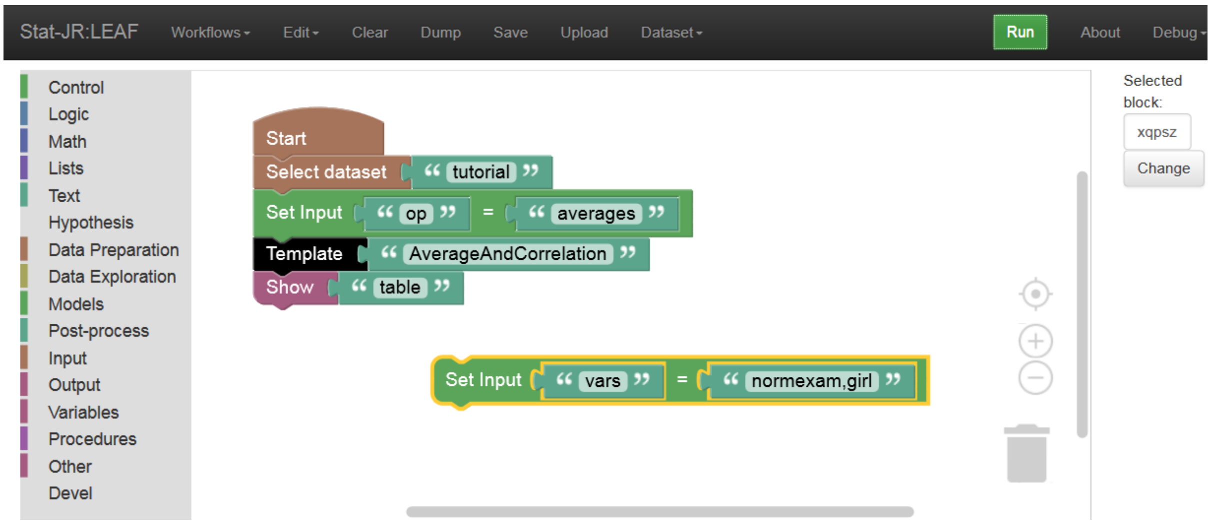
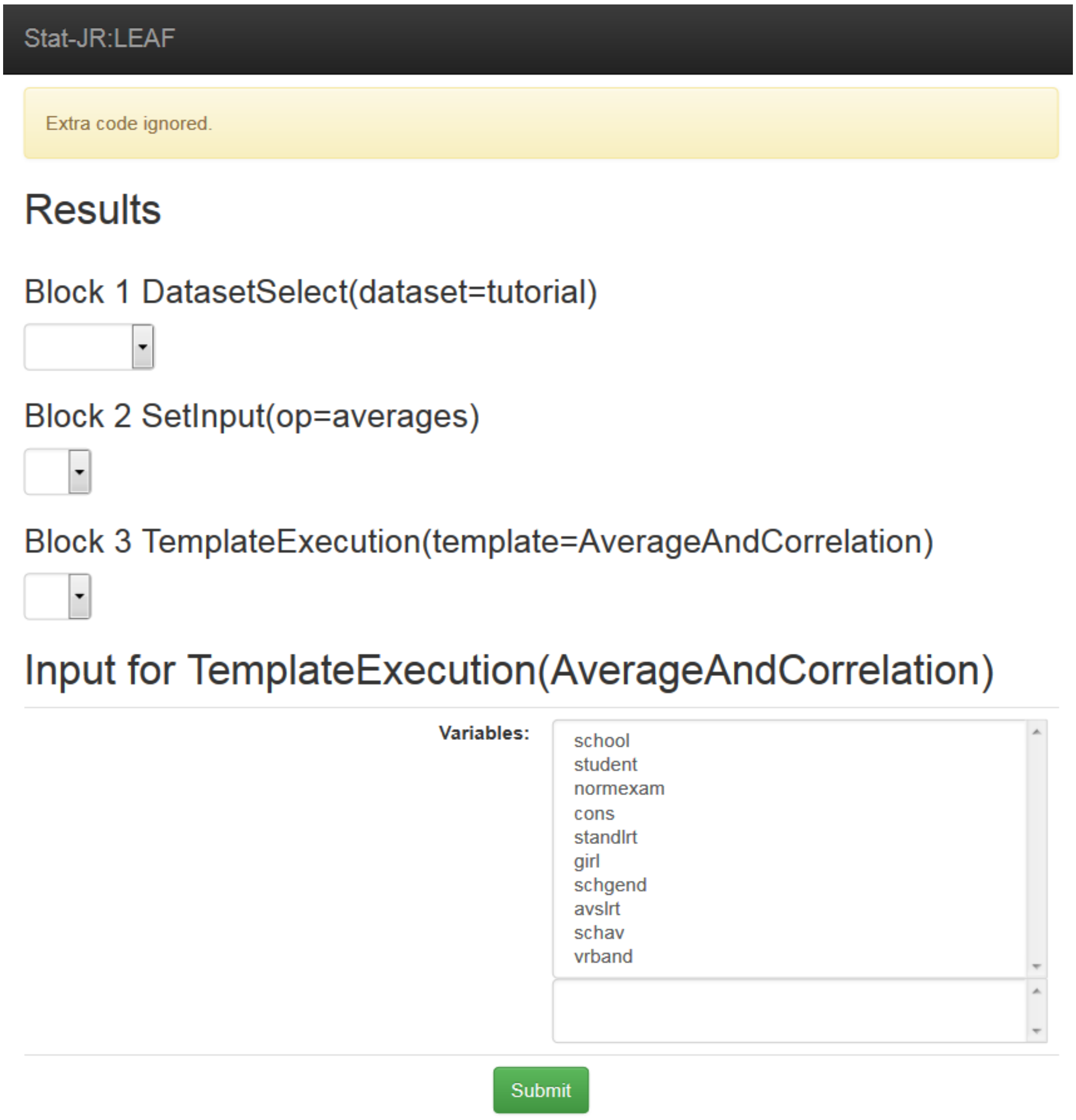


Figure 18

This time, after we click on **Run**, two aspects of the output are notable. Firstly there is a statement at the top indicating “Extra code ignored”. This simply means that it has detected the *Set Input* block we removed from the workflow and set to one side. Secondly, we are prompted for the outstanding input values the template needs before it can run to completion:


Stat-JR:LEAF

Extra code ignored.

Results

Block 1 DatasetSelect(dataset=tutorial)

Block 2 SetInput(op=averages)

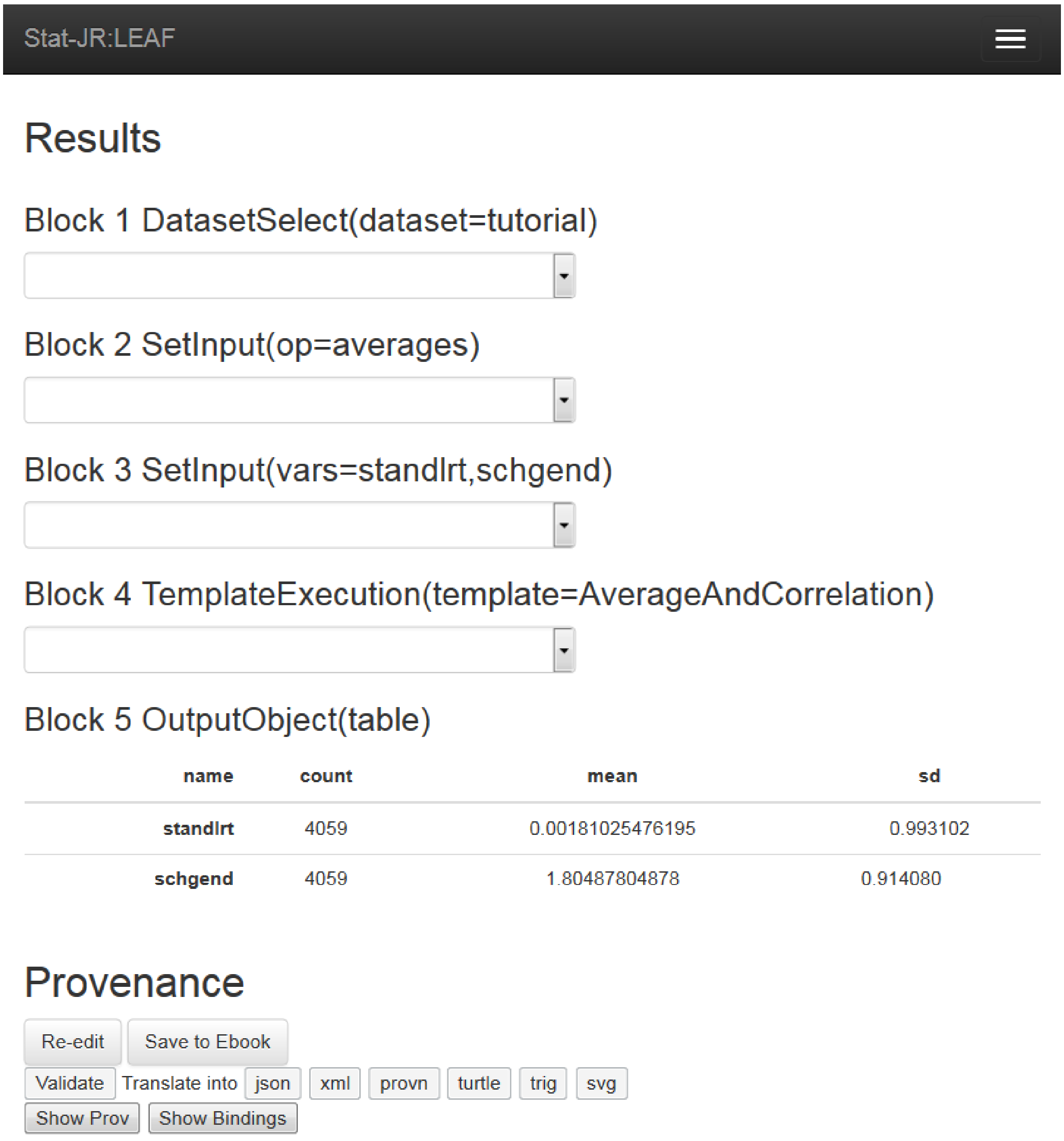
Block 3 TemplateExecution(template=AverageAndCorrelation)

Input for TemplateExecution(AverageAndCorrelation)

Variables: school, student, normexam, cons, standlrt, girl, schgend, avslrt, schav, vrband

Figure 19

If we click on *standlrt* and *schgend* (or variables of your choice) and then **Submit** then the workflow will execute and look as follows:


Stat-JR:LEAF

Results

Block 1 DatasetSelect(dataset=tutorial)

Block 2 SetInput(op=averages)

Block 3 SetInput(vars=standlrt,schgend)

Block 4 TemplateExecution(template=AverageAndCorrelation)

Block 5 OutputObject(table)

name	count	mean	sd
standlrt	4059	0.00181025476195	0.993102
schgend	4059	1.80487804878	0.914080

Provenance

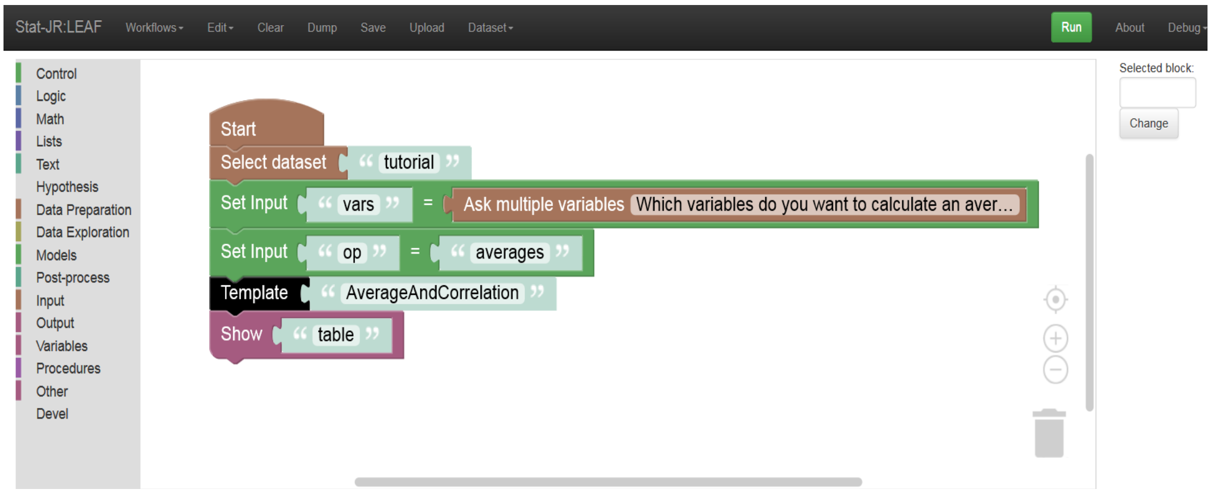
Figure 20

And thus we have created a workflow that will ask the user for variables (from the *tutorial* dataset, in our example) and then produce their means and standard deviations.

1.7 Adding question blocks

If we want to change how we *ask* for an input – i.e. the prompt presented to the user – from within the workflow (*cf.* changing the code in the template itself) then instead of removing the *Set Input* block from the workflow, we can instead reinstate it but this time with the addition of a question block. So, move the *Set Input* block back in, and remove the text block in which the input values were 'hard-wired' (you can select it and press the *Delete* button on your keyboard, or right-click and select 'Delete Block', or finally you can drag it to the bin in the bottom right corner - the bin will open and if you let go of the mouse button it will swallow the blocks!)

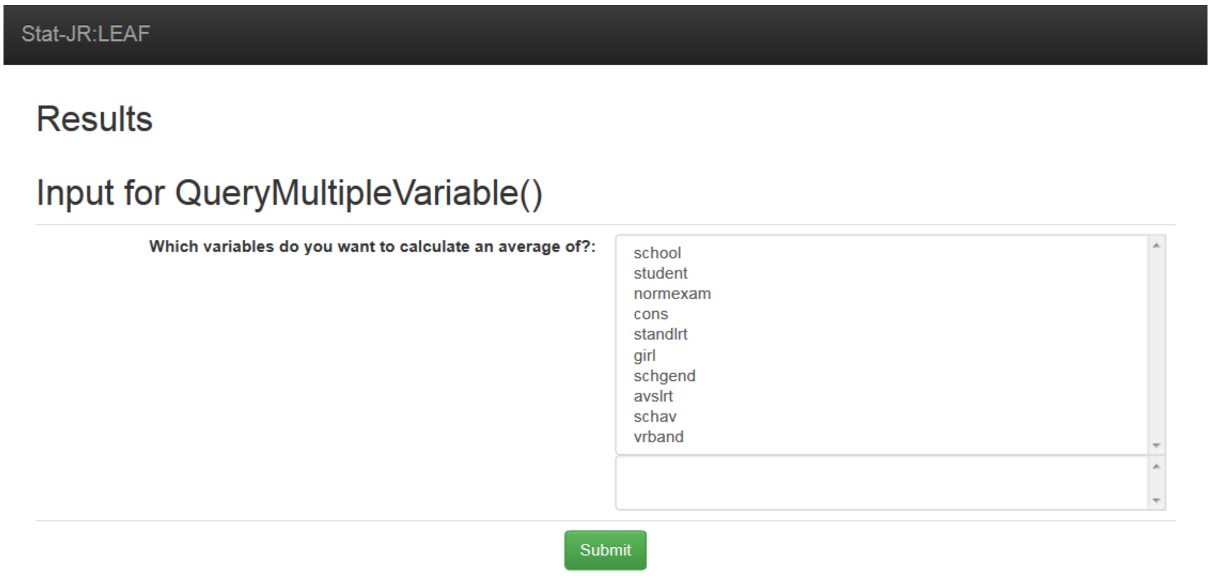
Then, from the **Input** list of blocks select the *Ask multiple variables*[6] block from the list and drag it to fill the hole we left in the *Set Input* block. You will see that *the Ask multiple variables* block has a blank box in which you can type your question (truncated here, but we asked *"Which variables do you want to calculate an average of?"*):



Start
Select dataset "tutorial"
Set Input "vars" = Ask multiple variables Which variables do you want to calculate an aver...
Set Input "op" = "averages"
Template "AverageAndCorrelation"
Show "table"

Figure 21

Running the workflow will then prompt the user with this question, as we see below:



Results

Input for QueryMultipleVariable()

Which variables do you want to calculate an average of?:

school
student
normexam
cons
standlrt
girl
schgend
avslrt
schav
vrband

Submit

Figure 22

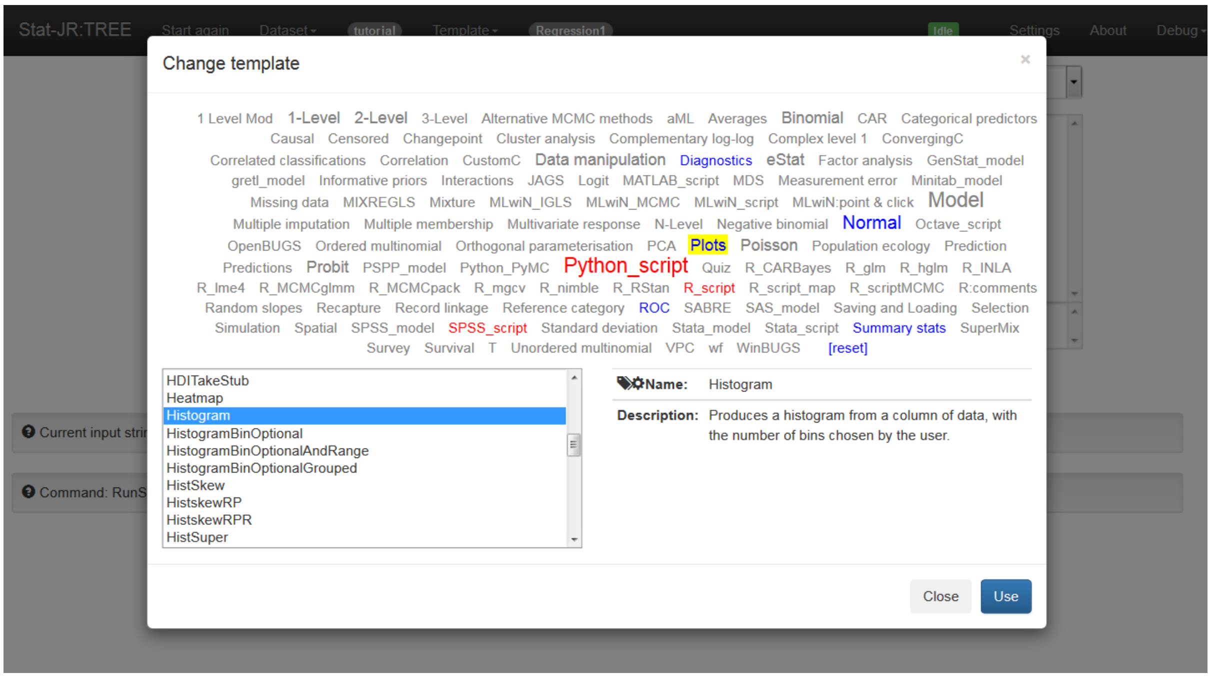
Here if we answer the question we will once again get output showing the means and standard deviations for the selected variables. Let's overwrite the workflow we saved earlier with this version, so save it as *section1_05.xml*.

[6] We're using the *Ask multiple variables* block here as it allows the user to select more than one variable in their answer; the *Ask single variable* block only allows the user to select one variable.

1.8 Plotting a histogram

We will now move on from working with the *AverageAndCorrelation* template and turn our attention to trying a second template and placing it in a workflow. This will be another operation one might do when beginning to look at a dataset, namely plotting a histogram of a variable to assess the shape of its distribution.

We can go back to TREE to identify the template we will need. If you don't still have TREE active you will need to restart it. Once you're back on the main TREE window, select the *Template* list and click on **Choose**. If you select *Plots* in the cloud of terms, you will see the list reduces to those templates which generate charts, including one called *Histogram*, which we can use.


Stat-JR:TREE
Start again
Dataset
tutorial
Template
Regression1
Idle
Settings
About
Debug
Change template
1 Level Mod 1-Level 2-Level 3-Level Alternative MCMC methods aML Averages Binomial CAR Categorical predictors Causal Censored Changepoint Cluster analysis Complementary log-log Complex level 1 ConvergingC Correlated classifications Correlation CustomC Data manipulation Diagnostics eStat Factor analysis GenStat_model gretl_model Informative priors Interactions JAGS Logit MATLAB_script MDS Measurement error Minitab_model Missing data MIXREGLS Mixture MLwiN_IGLS MLwiN_MCMC MLwiN_script MLwiN:point & click Model Multiple imputation Multiple membership Multivariate response N-Level Negative binomial Normal Octave_script OpenBUGS Ordered multinomial Orthogonal parameterisation PCA Plots Poisson Population ecology Prediction Predictions Probit PSPP_model Python_PyMC Python_script Quiz R_CARBayes R_glm R_hglm R_INLA R_lme4 R_MCMCglmm R_MCMCpack R_mgcv R_nimble R_RStan R_script R_script_map R_scriptMCMC R:comments Random slopes Recapture Record linkage Reference category ROC SABRE SAS_model Saving and Loading Selection Simulation Spatial SPSS_model SPSS_script Standard deviation Stata_model Stata_script Summary stats SuperMix Survey Survival T Unordered multinomial VPC wf WinBUGS [reset]
Current input stri
Command: RunS
Close
Use

HDITakeStub
Heatmap
Histogram
HistogramBinOptional
HistogramBinOptionalAndRange
HistogramBinOptionalGrouped
HistSkew
HistskewRP
HistskewRPR
HistSuper

Name: Histogram

Description: Produces a histogram from a column of data, with the number of bins chosen by the user.

Figure 23

In an earlier section we chose to run our template of interest in TREE and then export a workflow reflecting our choice of template, dataset and inputs via the **Make workflow** button. This time we'll try a different method.

Returning to the LEAF interface, make sure you have saved the previous workflow we were working on (*section1_05*). Then, delete the blocks specifically relating to our execution of the *AverageAndCorrelation* template (so that's the *Set Input*, *Template* and *Show* blocks) as shown below:

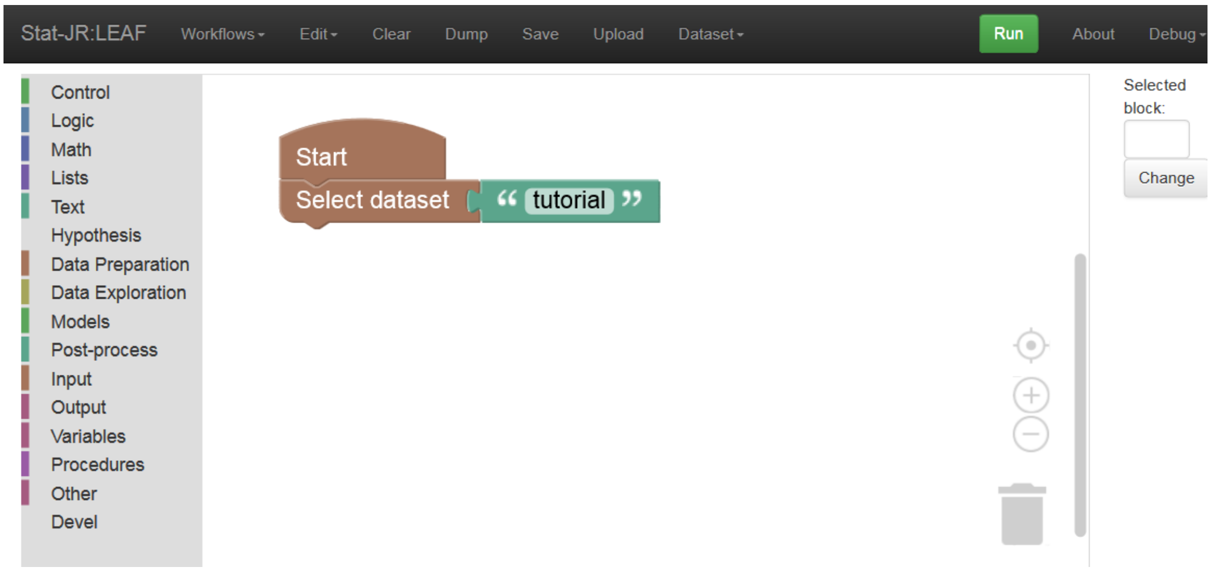


Figure 24

As an aside – whilst we've got this small set of blocks – it's worth noting that one of the features of using Blockly to realise Stat-JR's workflow system is that many of the syntactical rules are inherent in the shape of the blocks, and their readiness to fit together. As you can see, the *Select dataset* block has a slot on its right-hand side, like the side of a jigsaw piece. As you might imagine, this can only take another block which is appropriately shaped to fit into that slot. However, it can't take *any* such block: for example if you were to try to replace the current text block with a *not* block (from the **Logic** menu), you'll see it resists, like trying to join like poles of two magnets:

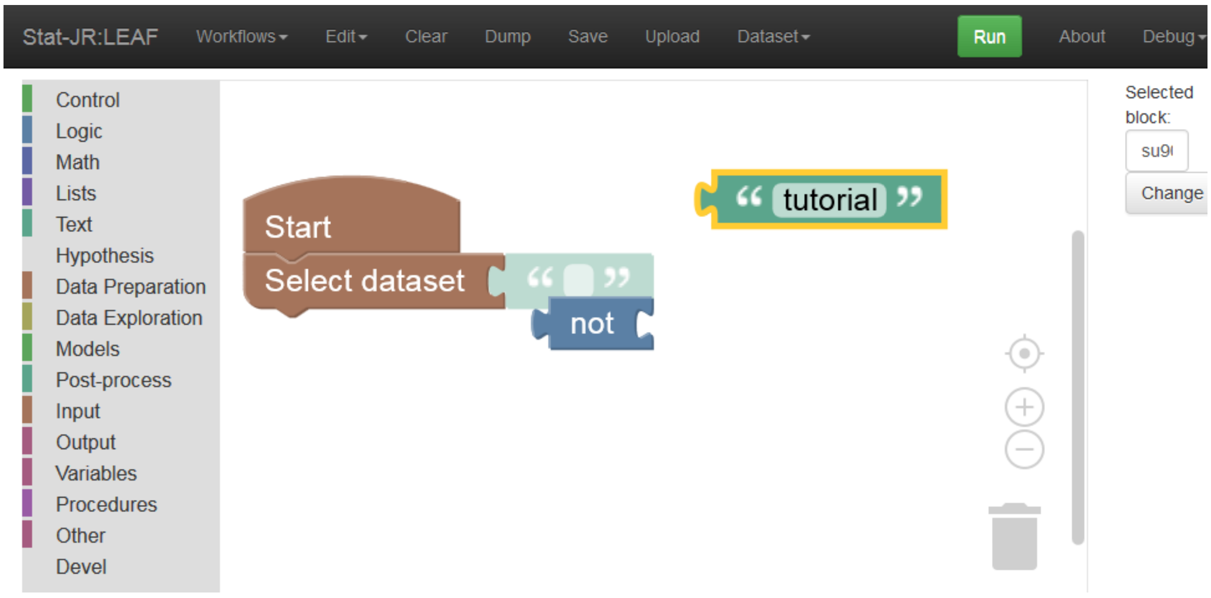


Figure 25

Clearly, this is the wrong sort of block (we can just delete it, as were only trying it out to prove a point!) In this instance, of course, the shadow block is suggesting an appropriate choice of block: a 'text' block, which is exactly what we had anyway, so we can just re-attach our "tutorial" text block, and we're back where we were.

Next we'll add a *Template* block indicating we wish to run the *Histogram* template we identified in TREE; to do this, pull in a *Template* block from the **Devel** menu on the left-hand side (alternatively we could, of course, have edited the Template block we just deleted) and write *Histogram* in the text block attached to the right of it, as shown below:

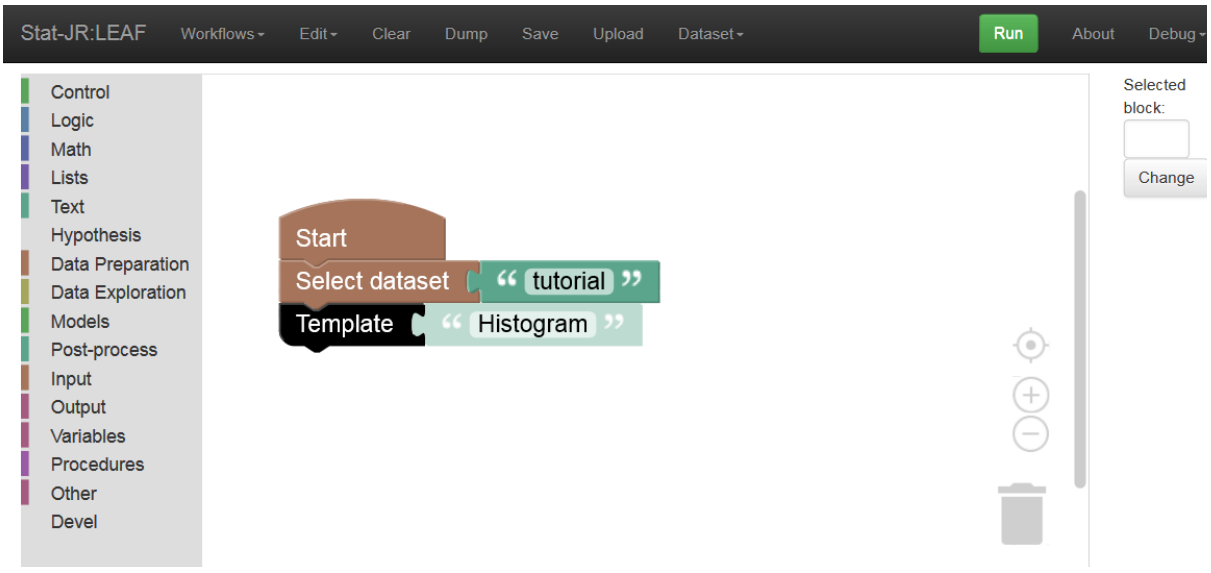
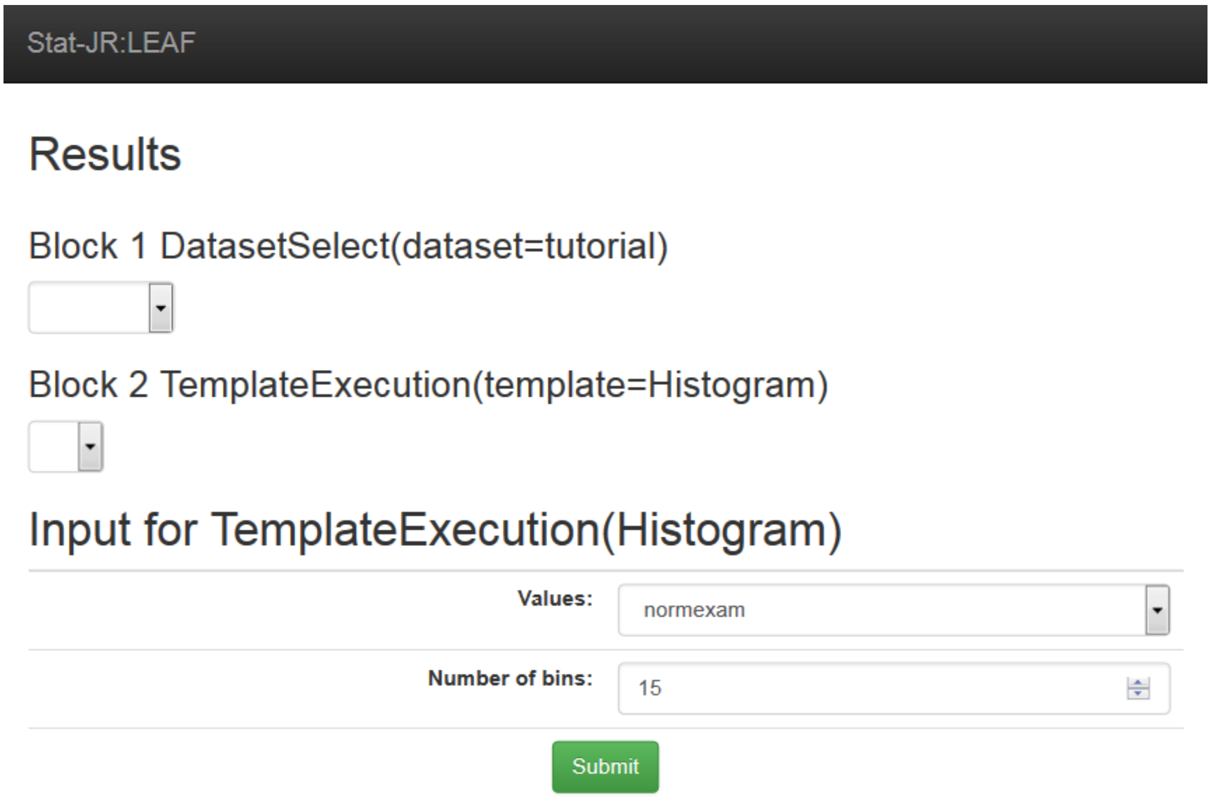


Figure 26

Now, if we press **Run**, it will prompt us for the inputs the template needs. In this example we've chosen to plot the variable *normexam* in a histogram with *15* bins:


Stat-JR:LEAF

Results

Block 1 DatasetSelect(dataset=tutorial)

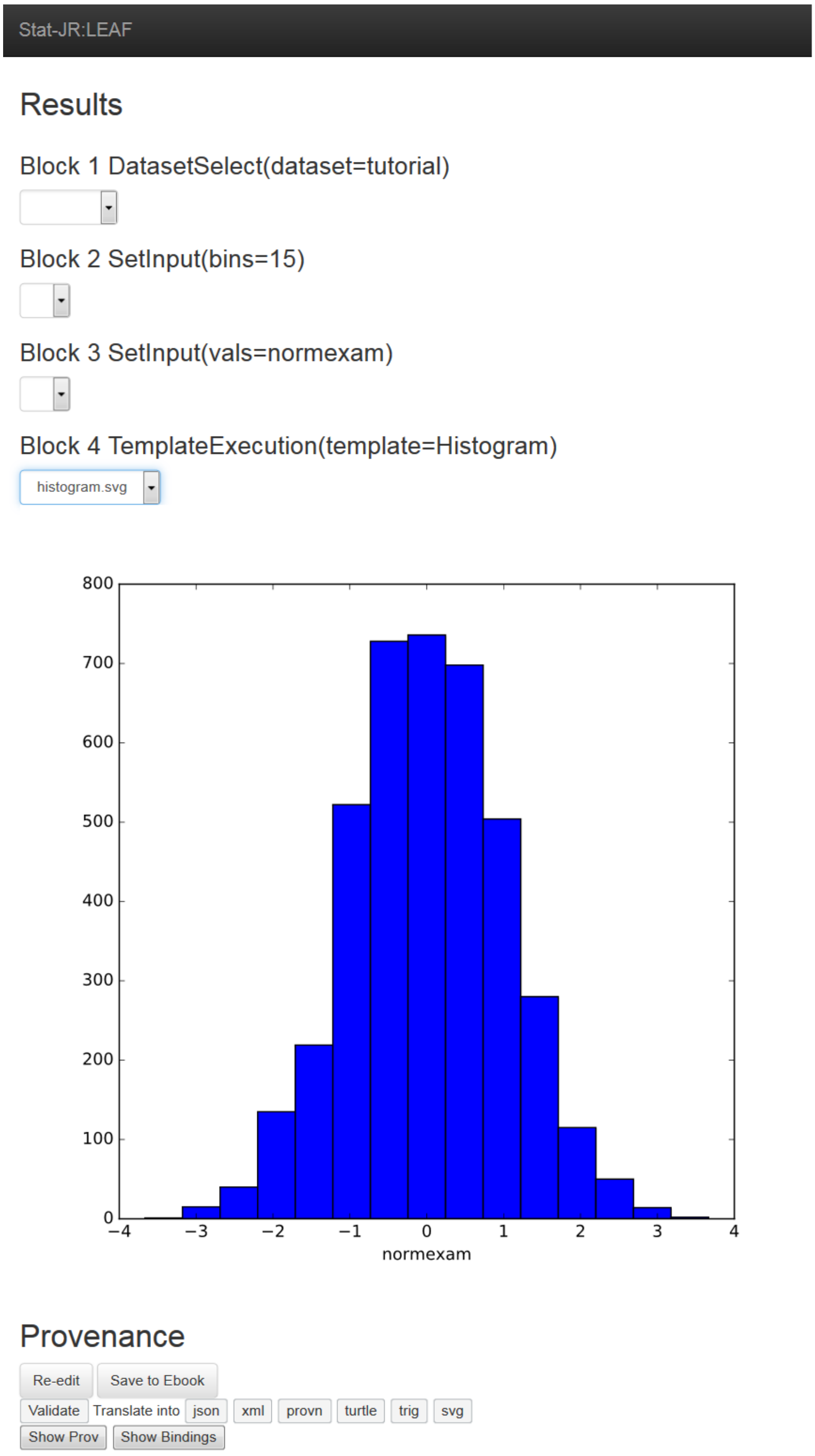
Block 2 TemplateExecution(template=Histogram)

Input for TemplateExecution(Histogram)

Values:	normexam
Number of bins:	15

Figure 27

Once you have pressed the **Submit** button, you will see that a final block appears (Block 4) pertaining to the template execution; from the pull-down underneath it select *histogram.svg*:


Stat-JR:LEAF

Results

Block 1 DatasetSelect(dataset=tutorial)

Block 2 SetInput(bins=15)

Block 3 SetInput(vals=normexam)

Block 4 TemplateExecution(template=Histogram)

histogram.svg

Provenance

Figure 28

So, we've been prompted for the input values the *Histogram* template needs, and have identified the relevant output object (*histogram.svg*). If we now press the **Re-edit** button, towards the bottom-left of the screen, we will see that a workflow appears containing the *Histogram* template's inputs (*bins* and *vals*) and the values we just assigned to them (*15* and *normexam*, respectively). What this workflow doesn't have is the output object (it's currently not possible to add that to a workflow via the **Re-edit** button), so let's add that ourselves by pulling in a *Show* block from the left-hand menu:

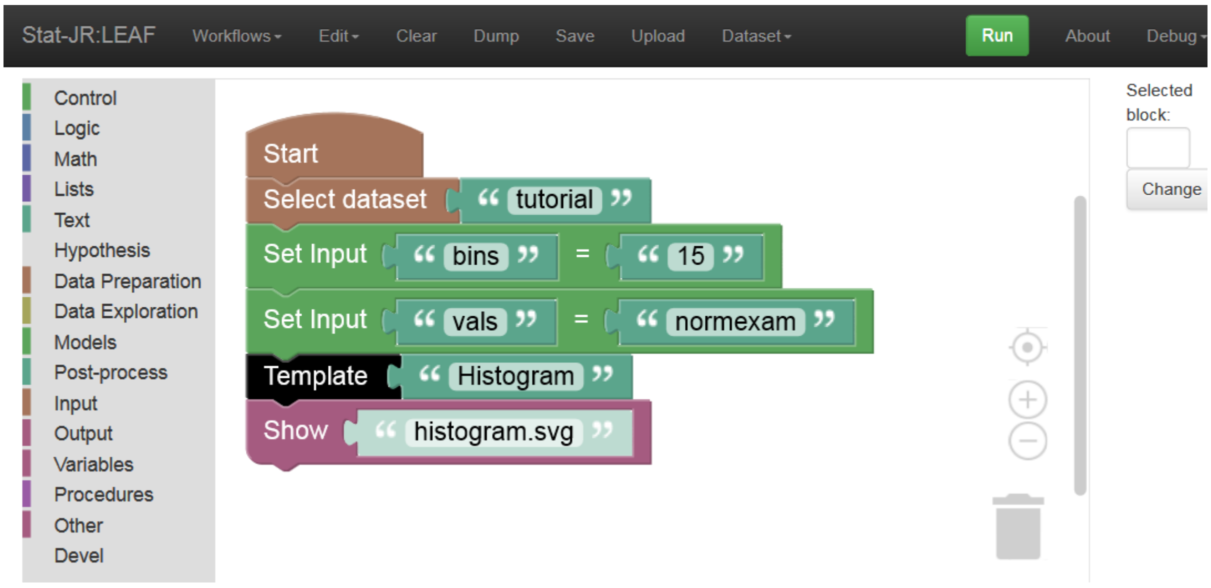


Figure 29

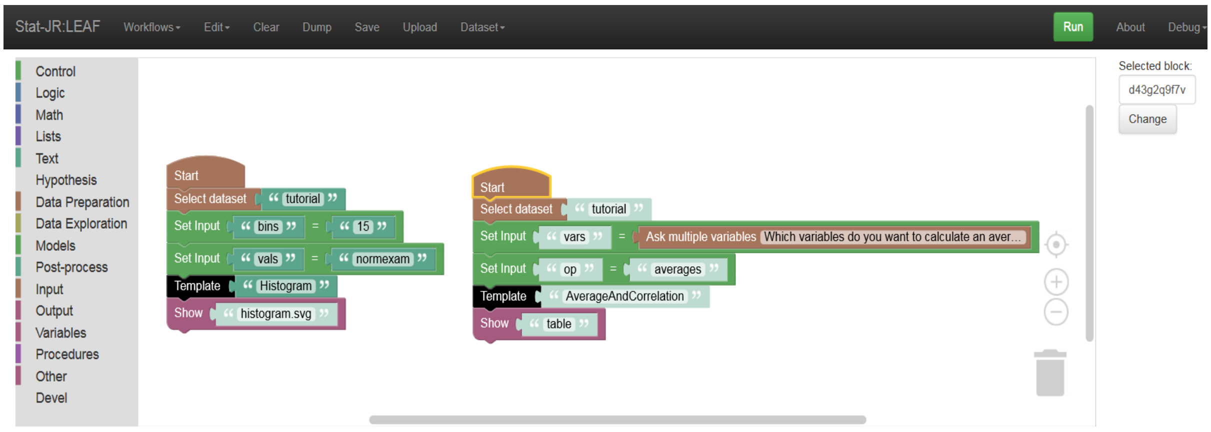
Now, if you **Run** this workflow, you see that the plot appears towards the bottom of the output.

Save this workflow as *section1_08.xml.*

1.9 Connecting up the operations

We have now created two workflows and an obvious next step is to join them together; to do this we can import the earlier workflow we produced and then append them. With the last workflow we constructed (*section1_08.xml*) still on the screen, select **Upload** from the black bar at the top, and navigate to the workflow we earlier saved as *section1_05.xml*. Having selected it and pressed **Open**, next choose **Import file** when prompted: this will bring that workflow into the same workspace as the current workflow (rather than first clearing the workspace).

The two workflows may appear alongside each other, or perhaps one may be overlaying the other; if the latter you can just move one aside so you can clearly see them both:

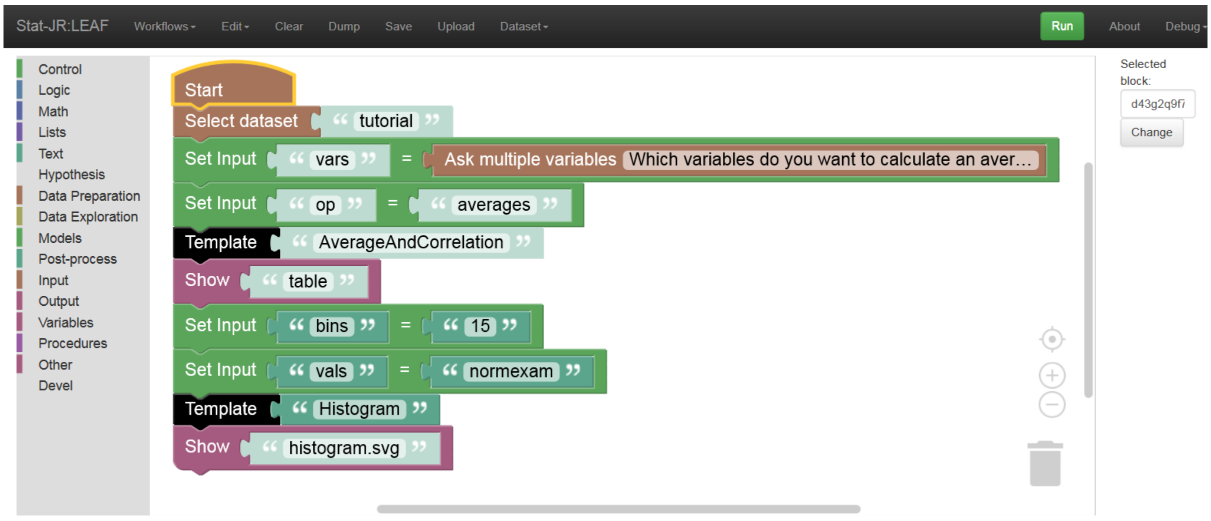


```
Start
Select dataset "tutorial"
Set Input "bins" = "15"
Set Input "vals" = "normexam"
Template "Histogram"
Show "histogram.svg"
```

```
Start
Select dataset "tutorial"
Set Input "vars" = Ask multiple variables Which variables do you want to calculate an aver…
Set Input "op" = "averages"
Template "AverageAndCorrelation"
Show "table"
```

Figure 30

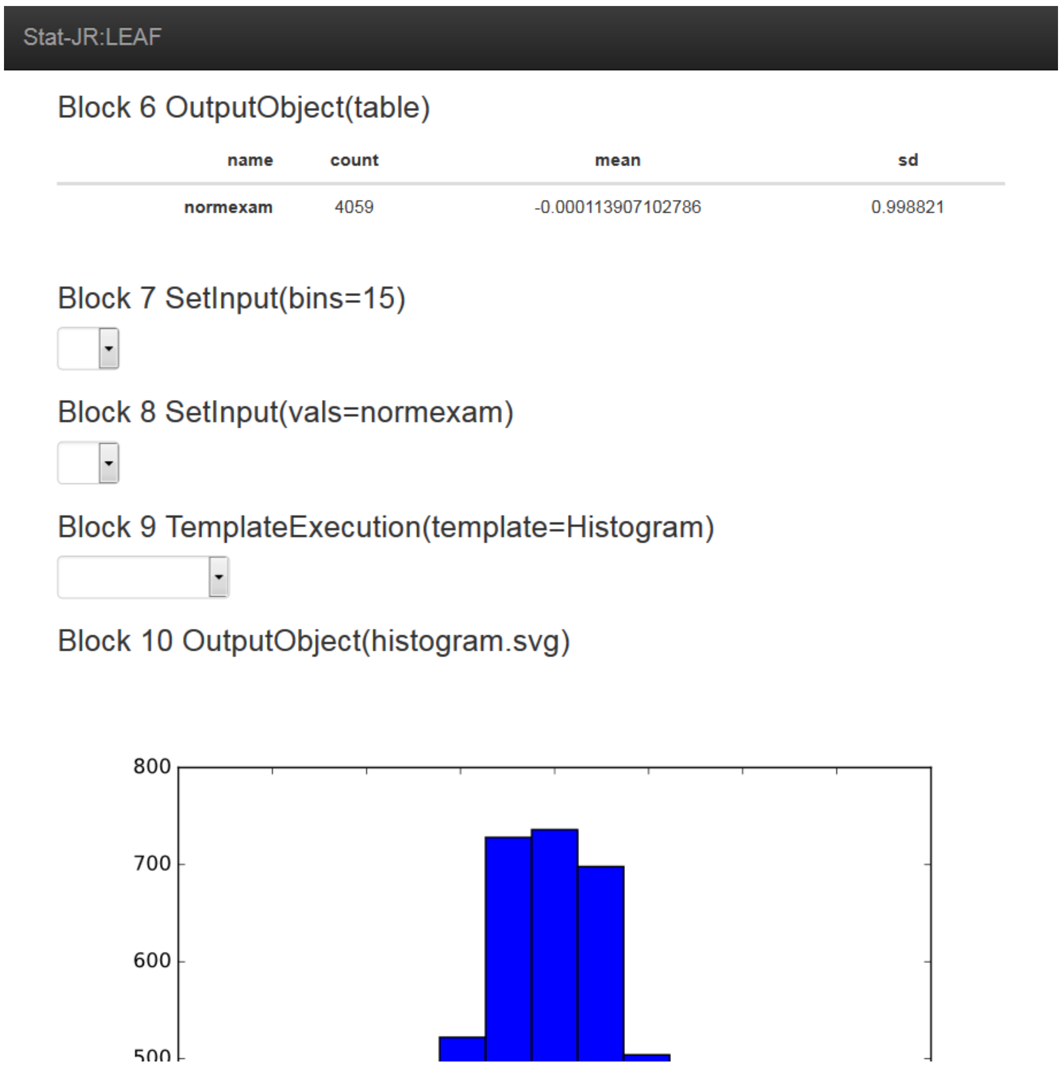
We now need to append the blocks pertaining to the histogram below those generating our summary statistics of interest; remember we only need one **Start** block and one **Select dataset** block (since both template executions use the same dataset):



```
Start
Select dataset "tutorial"
Set Input "vars" = Ask multiple variables Which variables do you want to calculate an aver…
Set Input "op" = "averages"
Template "AverageAndCorrelation"
Show "table"
Set Input "bins" = "15"
Set Input "vals" = "normexam"
Template "Histogram"
Show "histogram.svg"
```

Figure 31

If we press **Run**, and then answer the question when prompted (here we have chosen just *normexam*) we will see that both operations are done thus:


Stat-JR:LEAF

Block 6 OutputObject(table)

name	count	mean	sd
normexam	4059	-0.000113907102786	0.998821

Block 7 SetInput(bins=15)

Block 8 SetInput(vals=normexam)

Block 9 TemplateExecution(template=Histogram)

Block 10 OutputObject(histogram.svg)

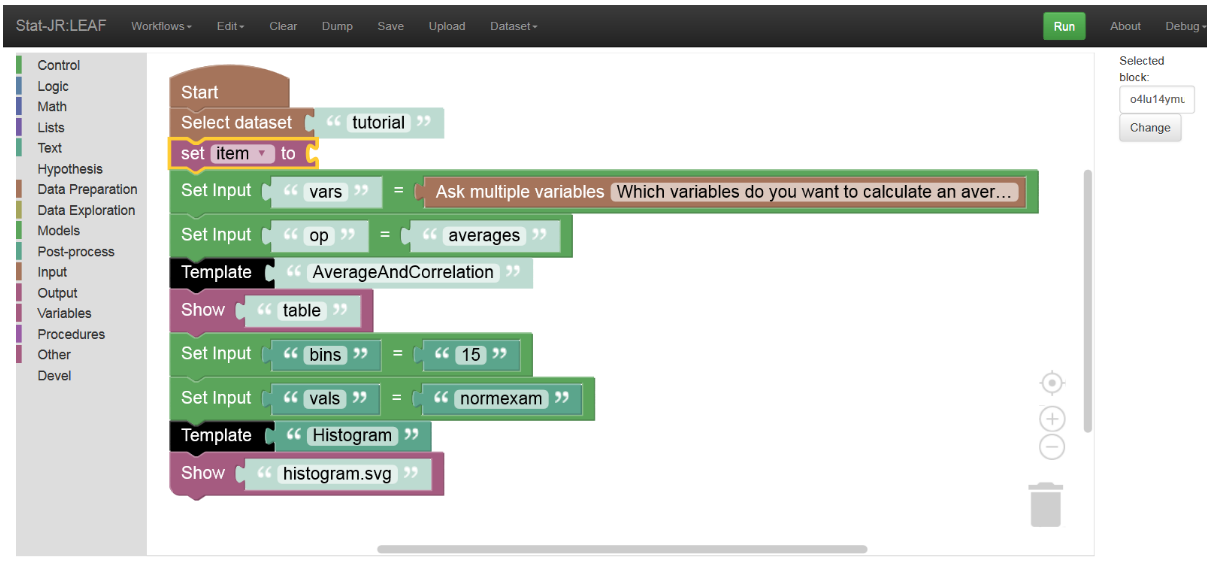
Figure 32

1.10 Using variables in a workflow

So we have now seen how we can join up two template executions in one workflow and it is easy to continue this with further operations to create a logfile-style workflow, either by appending blocks in LEAF or via the **Make workflow** button in TREE.

We have investigated how to ask questions to replace hard-wired inputs and add an element of interoperability. A natural extension of this is to ask a question where the answer is shared by several templates downstream. To do this we will introduce the concept of variables within a workflow and illustrate it by constructing a workflow that asks for a single input and then produces its average and its histogram.

You will see in the list to the left there is a menu entitled **Variables** and in this list is a red *set <item> to* block. Grab a copy of this block and place it in your workflow under the *Select dataset* block (if you place it in the approximate area and let go of the mouse button it should be added into the workflow thus):



```
Start
Select dataset "tutorial"
set item to
Set Input "vars" = Ask multiple variables Which variables do you want to calculate an aver…
Set Input "op" = "averages"
Template "AverageAndCorrelation"
Show "table"
Set Input "bins" = "15"
Set Input "vals" = "normexam"
Template "Histogram"
Show "histogram.svg"
```

Figure 33

By default the variable is called *item* but we can change this by clicking on the pull-down arrow to the side of it and selecting **New variable…** A window appears where we can enter a name; we will choose *response*:

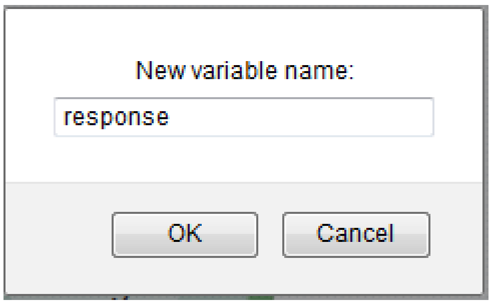
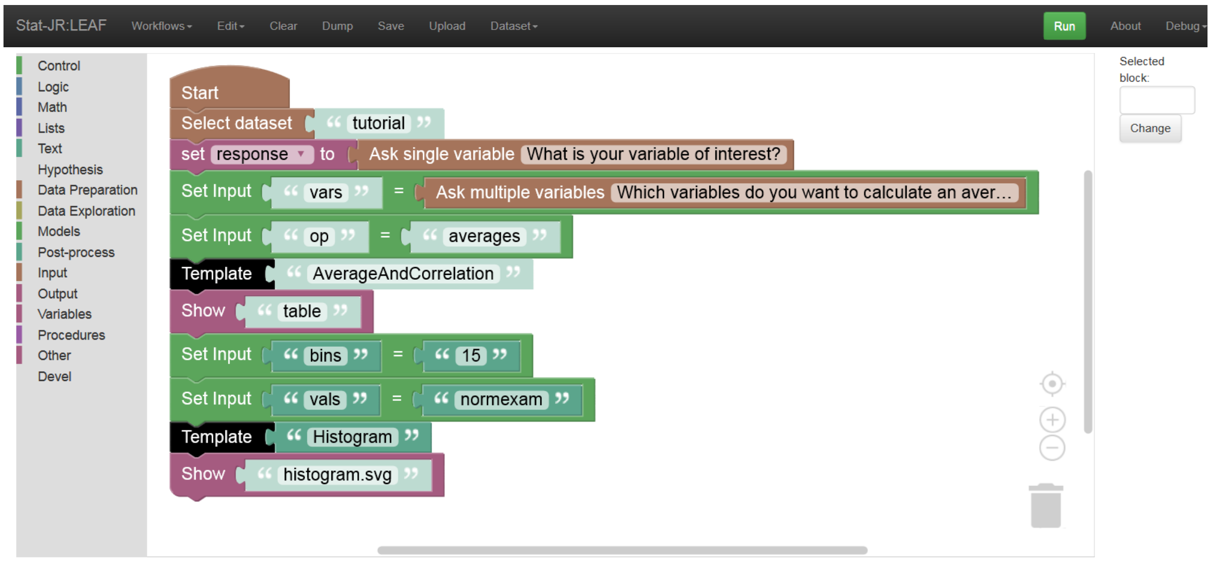


Figure 34

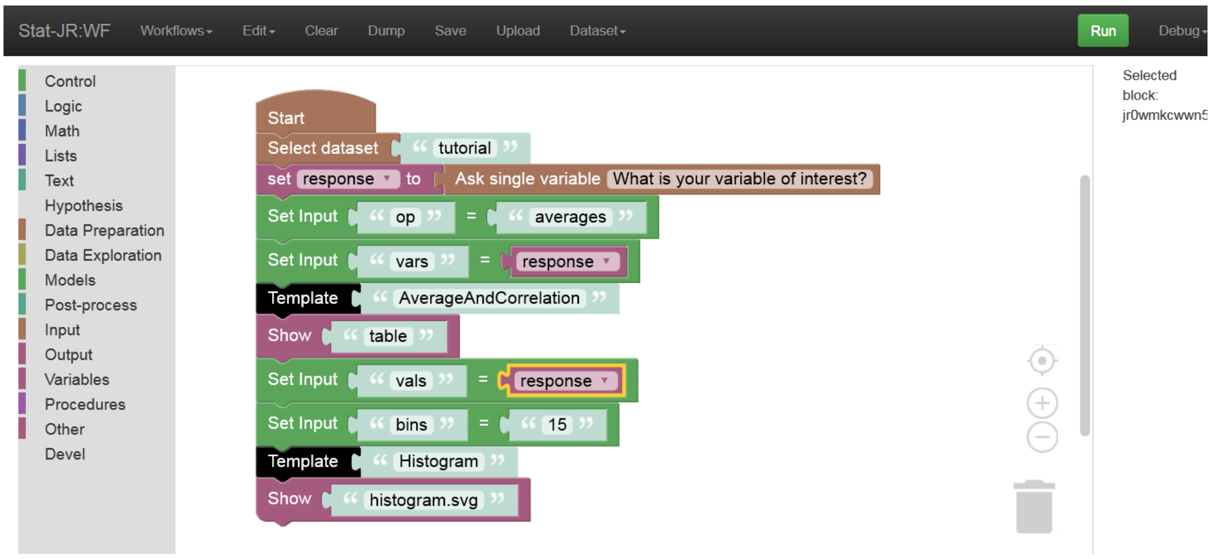
Clicking on **OK** will select *response* as our variable name. We now need to assign it a value (in this case the answer to a question), and so from the **Input** list select *Ask single variable* and move it to the right of response. We can then add the question text ("What is your variable of interest?") as shown below:



```
Start
Select dataset "tutorial"
set response to Ask single variable What is your variable of interest?
Set Input "vars" = Ask multiple variables Which variables do you want to calculate an aver…
Set Input "op" = "averages"
Template "AverageAndCorrelation"
Show "table"
Set Input "bins" = "15"
Set Input "vals" = "normexam"
Template "Histogram"
Show "histogram.svg"
```

Figure 35

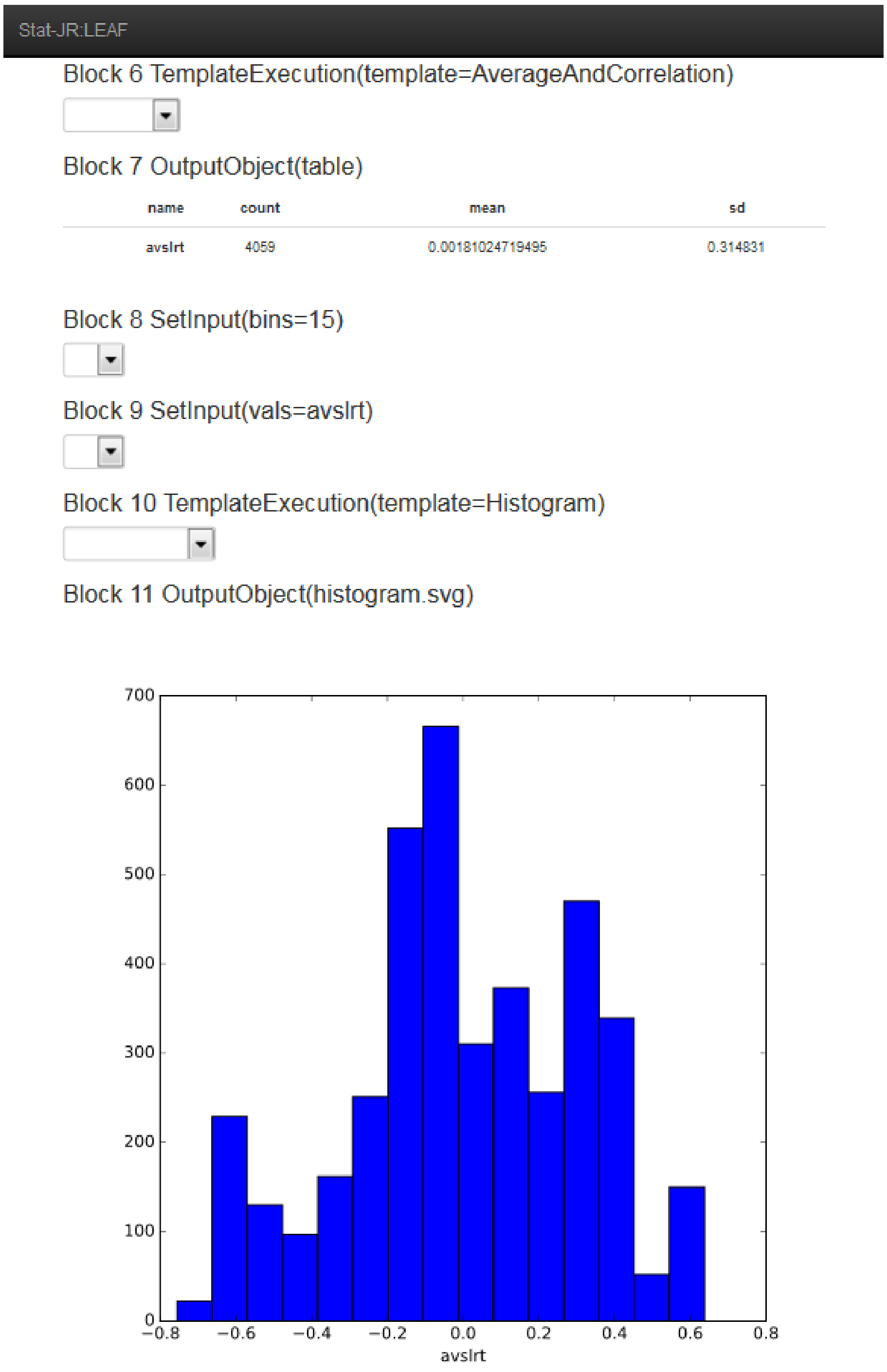
This has created a variable (called *response*), the value of which will be whatever the user chooses when prompted by the question "What is your variable of interest?" However, before running this workflow, we first need to slot this variable (*response*) into places in the workflow where it is to be used (as the values for inputs *vars* and *vals*, for example). Have a go at doing this yourself (you'll need a new type of block from this list on the left). Note that, as well as pulling multiple instances of the same block in from the left-hand menu, if you right-click on a block you can choose **Duplicate** from the resulting menu and a copy of the block appears (alternatively you can select the block(s) you wish to duplicate and press Ctrl-C then Ctrl-V to copy and paste). The completed workflow looks as follows:



```
Start
Select dataset "tutorial"
set response to Ask single variable What is your variable of interest?
Set Input "op" = "averages"
Set Input "vars" = response
Template "AverageAndCorrelation"
Show "table"
Set Input "vals" = response
Set Input "bins" = "15"
Template "Histogram"
Show "histogram.svg"
```

Figure 36

Hopefully you managed to find the block you needed.[7] We can now save this workflow as *section1_10.xml* before clicking on the **Run** button to run the workflow. In our example we've chosen *avslrt* in answer to the question:



Stat-JR:LEAF

Block 6 TemplateExecution(template=AverageAndCorrelation)

Block 7 OutputObject(table)

name	count	mean	sd
avslrt	4059	0.00181024719495	0.314831

Block 8 SetInput(bins=15)

Block 9 SetInput(vals=avslrt)

Block 10 TemplateExecution(template=Histogram)

Block 11 OutputObject(histogram.svg)

Figure 37

[7] Look under the **Variables** list. Once you've chosen the correct block, you can change the name away from *item* by selecting *response* from the drop-down list in the block.

Here we see the mean and then a histogram for the *avslrt* variable; i.e. it's taken our answer and used it as input for two template executions.

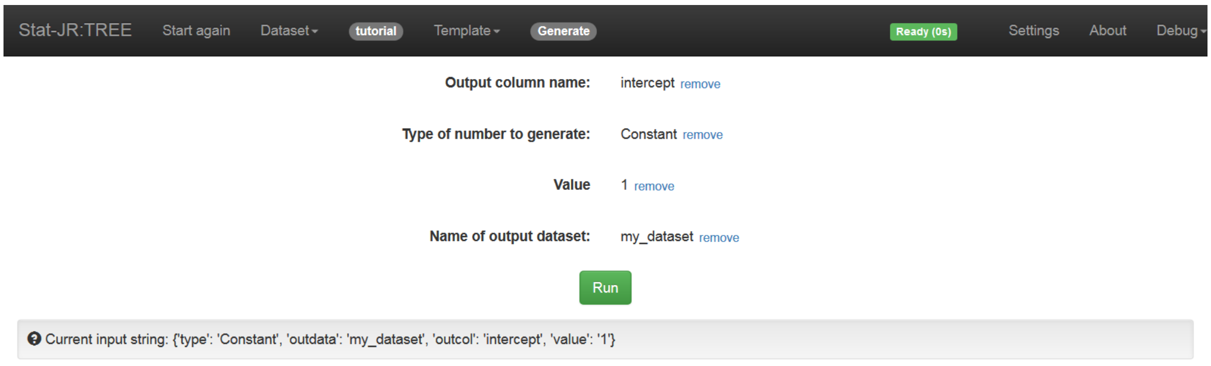
1.11 Running a statistical regression model and showing predictions

We will now move on to actually fitting a statistical model in Stat-JR. We will continue our approach of adding to our current workflow. We have so far seen how we can put together a sequence of operations in one workflow but up to now *outputs* from one template execution have not yet been used as inputs for the next template execution. We will remedy that by illustrating how to create predictions for our regression model based on the model fit.

We will begin by returning to TREE to fit a model using Stat-JR's built-in eStat MCMC engine. This time we're going to export the template executions we make in TREE as a workflow. We don't want to export earlier template executions, so it's probably easier to open a new TREE session and work with that.[8]

To do this we will use the *Regression1* template to fit a simple regression. The *Regression1* template requires the user to include a constant in their list of predictors if they want to fit an intercept. As it happens, the *tutorial* dataset we have been using has a constant of ones (the variable *cons*) which we could use, but since you may be using your own dataset which might not have a constant already in it, we'll show how to add a constant to the dataset using the template *Generate*.

Here, having selected the template *Generate* in TREE, we request our constant of ones as follows:



Output column name: intercept remove

Type of number to generate: Constant remove

Value 1 remove

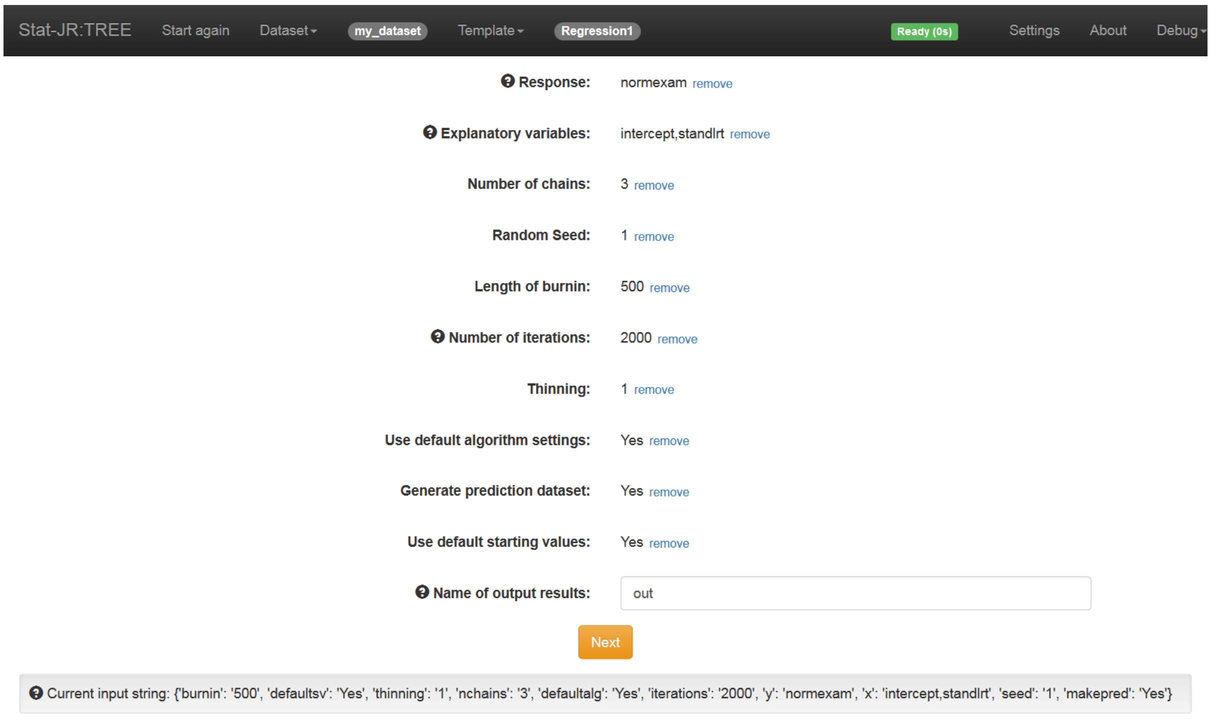
Name of output dataset: my_dataset remove

Current input string: {'type': 'Constant', 'outdata': 'my_dataset', 'outcol': 'intercept', 'value': '1'}

Figure 38

On pressing **Run** we create a variable consisting solely of ones called *intercept* in a new dataset called *my_dataset* (which is exactly the same as our original dataset, but with the new variable appended to the end). Selecting this modified dataset (*my_dataset*) from the list of datasets, and *Regression1* from the list of templates, we can now include this new variable as one of our predictors, setting up the inputs as follows:

[8] If you want to close the current TREE session to avoid possible confusion, then remember to close the browser tabs related your current TREE session *and* the associated command line window: this will be the one with TREE in the title bar. An alternative, of course, would be to work within the current TREE session, and then just delete any superfluous blocks we export as part of our workflow.

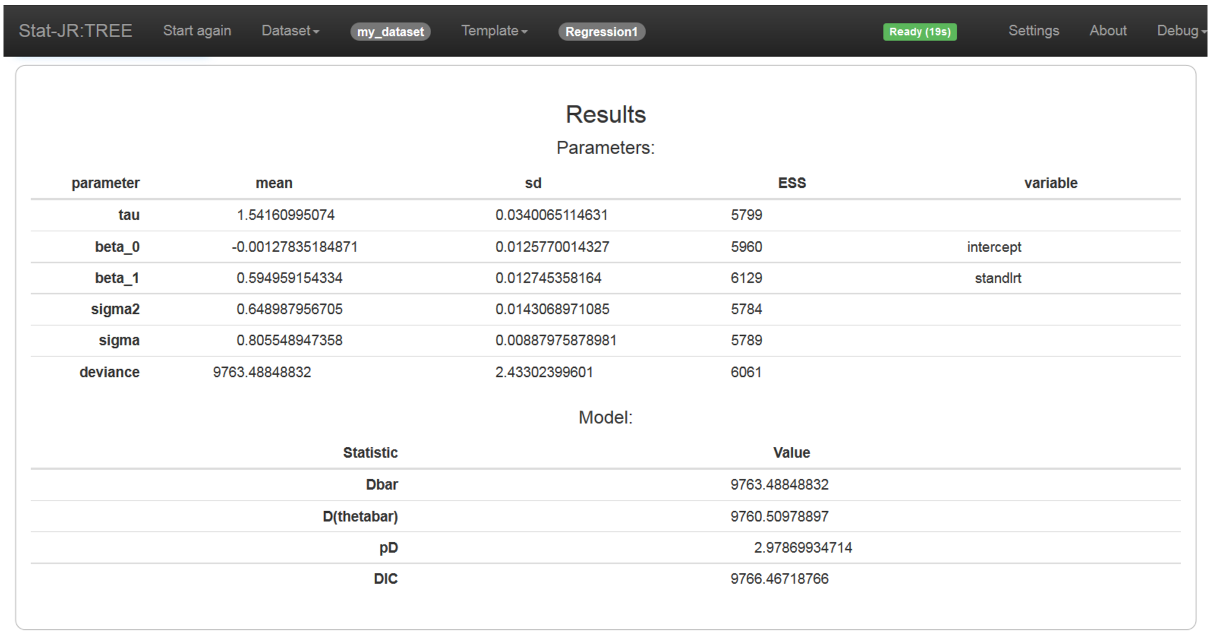


Response:	normexam remove
Explanatory variables:	intercept,standlrt remove
Number of chains:	3 remove
Random Seed:	1 remove
Length of burnin:	500 remove
Number of iterations:	2000 remove
Thinning:	1 remove
Use default algorithm settings:	Yes remove
Generate prediction dataset:	Yes remove
Use default starting values:	Yes remove
Name of output results:	out

Current input string: {'burnin': '500', 'defaultsv': 'Yes', 'thinning': '1', 'nchains': '3', 'defaultalg': 'Yes', 'iterations': '2000', 'y': 'normexam', 'x': 'intercept,standlrt', 'seed': '1', 'makepred': 'Yes'}

Figure 39

Here we are using the default settings for our MCMC estimation procedure[9], although we answer *Yes* to the prompt **Generate a prediction dataset**. Clicking on **Next** and **Run** will run the model and choosing *ModelResults* gives a summary of the model we have fitted thus:



Results

Parameters:

parameter	mean	sd	ESS	variable
tau	1.54160995074	0.0340065114631	5799	
beta_0	-0.00127835184871	0.0125770014327	5960	intercept
beta_1	0.594959154334	0.012745358164	6129	standlrt
sigma2	0.648987956705	0.0143068971085	5784	
sigma	0.805548947358	0.00887975878981	5789	
deviance	9763.48848832	2.43302399601	6061	

Model:

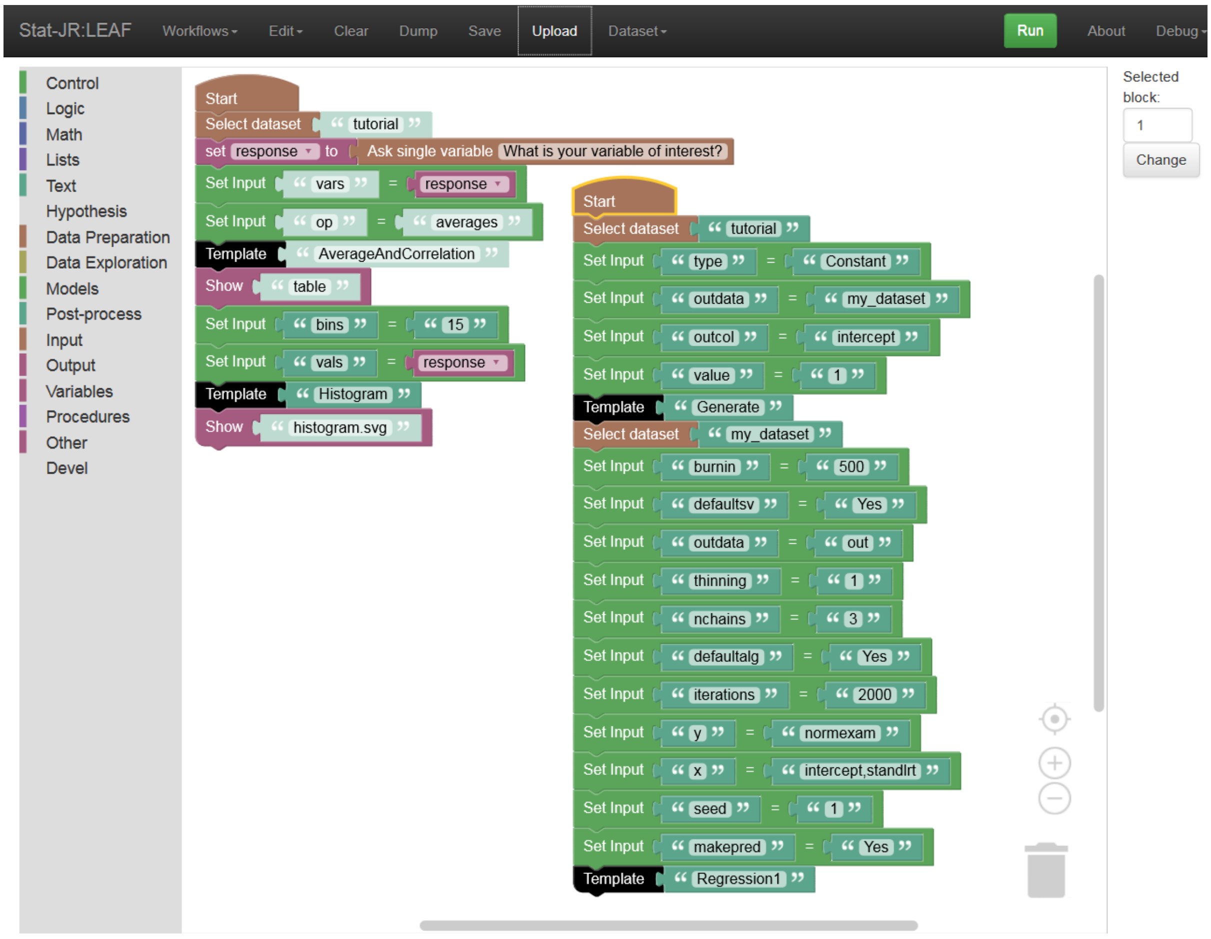
Statistic	Value
Dbar	9763.48848832
D(thetabar)	9760.50978897
pD	2.97869934714
DIC	9766.46718766

Figure 40

[9] This particular template can *only* use this estimation engine, although many others can use a wide variety of third-party software, including R, Stata, MLwiN, etc.

Now click on the **Make workflow** button and then, within the resulting box, the **Workflow** button, and save it somewhere you can access from within LEAF.

Return to the LEAF interface and then **Upload** the workflow you have just saved during your TREE session, again choosing to **Import file** so that your current workflow remains in the workspace. Again, once you have done so, one workflow might be partially overlaying the other, so if so just move one aside to clearly see them both:

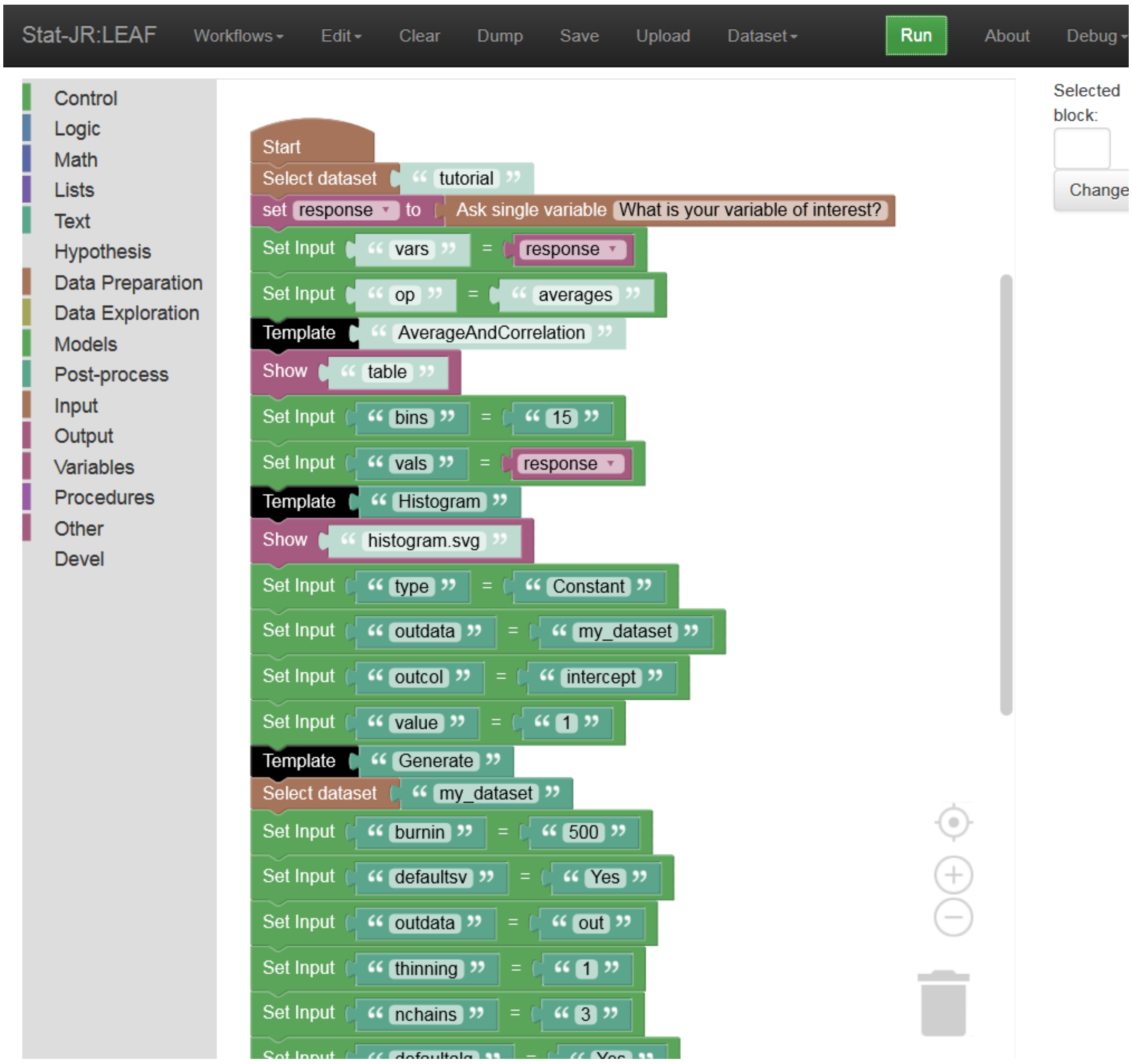


Start
Select dataset "tutorial"
set response to Ask single variable "What is your variable of interest?"
Set Input "vars" = response
Set Input "op" = "averages"
Template "AverageAndCorrelation"
Show "table"
Set Input "bins" = "15"
Set Input "vals" = response
Template "Histogram"
Show "histogram.svg"

Start
Select dataset "tutorial"
Set Input "type" = "Constant"
Set Input "outdata" = "my_dataset"
Set Input "outcol" = "intercept"
Set Input "value" = "1"
Template "Generate"
Select dataset "my_dataset"
Set Input "burnin" = "500"
Set Input "defaultsv" = "Yes"
Set Input "outdata" = "out"
Set Input "thinning" = "1"
Set Input "nchains" = "3"
Set Input "defaultalg" = "Yes"
Set Input "iterations" = "2000"
Set Input "y" = "normexam"
Set Input "x" = "intercept,standlrt"
Set Input "seed" = "1"
Set Input "makepred" = "Yes"
Template "Regression1"

Figure 41

We can attach this imported workflow (after first discarding the **Start** and **Select dataset** blocks at the top of it) to our existing one, as follows:



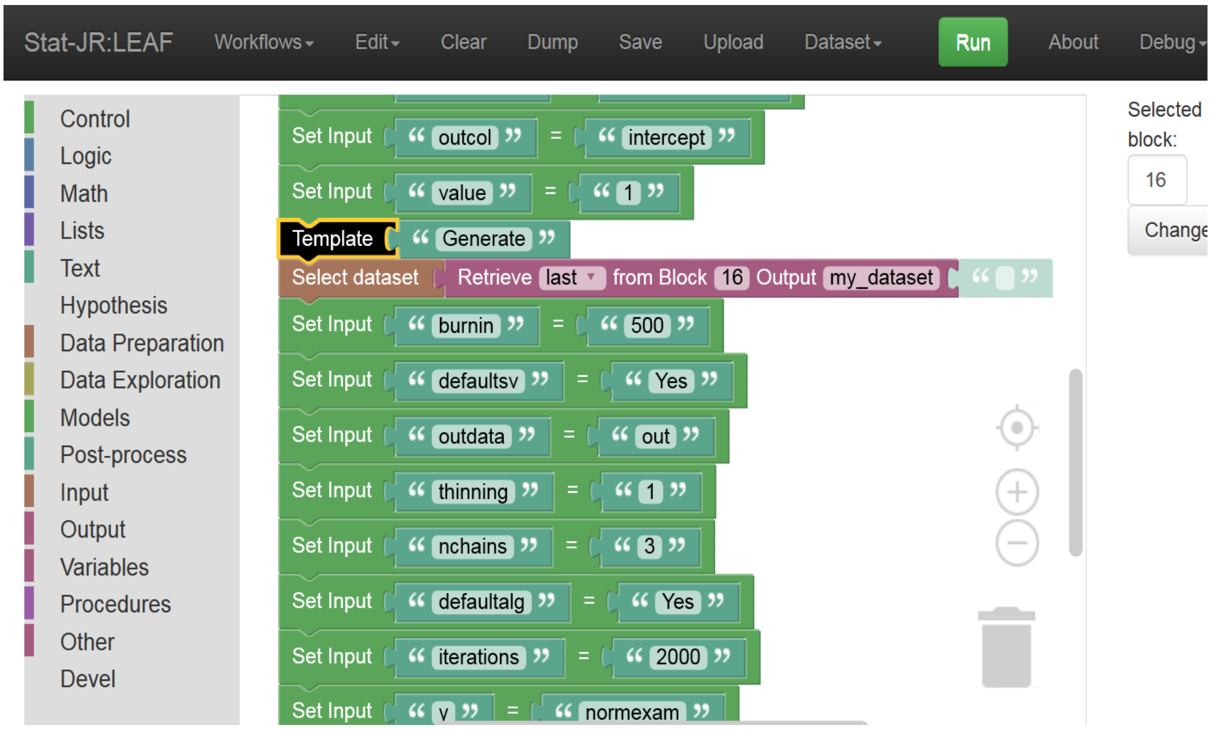
```
Start
Select dataset "tutorial"
set response to Ask single variable What is your variable of interest?
Set Input "vars" = response
Set Input "op" = "averages"
Template "AverageAndCorrelation"
Show "table"
Set Input "bins" = "15"
Set Input "vals" = response
Template "Histogram"
Show "histogram.svg"
Set Input "type" = "Constant"
Set Input "outdata" = "my_dataset"
Set Input "outcol" = "intercept"
Set Input "value" = "1"
Template "Generate"
Select dataset "my_dataset"
Set Input "burnin" = "500"
Set Input "defaultsv" = "Yes"
Set Input "outdata" = "out"
Set Input "thinning" = "1"
Set Input "nchains" = "3"
```

Figure 42

We're almost there, but there are a couple of changes we first need to make. Firstly, it is better practice within the LEAF system to explicitly extract the modified dataset we need from the relevant template execution, rather than rely on it being there in the global cache of datasets. So, remove the *Text* block to the right of the *Select dataset* block midway down the workflow (the one containing the text "*my_dataset"*), and replace it with a *Retrieve* block (found in the **Other** menu). The *Retrieve* block retrieves a named object from whatever stage of the workflow execution is cited in the block. Thus we have to give the object name we want (*my_dataset*)[10] and tell it which block to take this from. We perform the latter by referencing a unique ID code each block is assigned – it's the black *Template* block we need to reference (the one to which *"Generate"* is appended). If you select this block you will see the corresponding reference ID appears in the *"Selected block"* box towards the top right-hand side of the screen. In the example in the screenshot this is 16, but the unique block IDs don't always take this form: sometimes they are assigned long alphanumeric IDs (e.g. pofzefaivnqbos0n3x7h) – it just depends on the history of the workflow (e.g. whether it was created

[10] Note we could type this in the gap to the right of "Output", as we have done in the screenshot, or if you still have the text block we just removed you can attach it to the end of the *Retrieve* block, to the same effect.

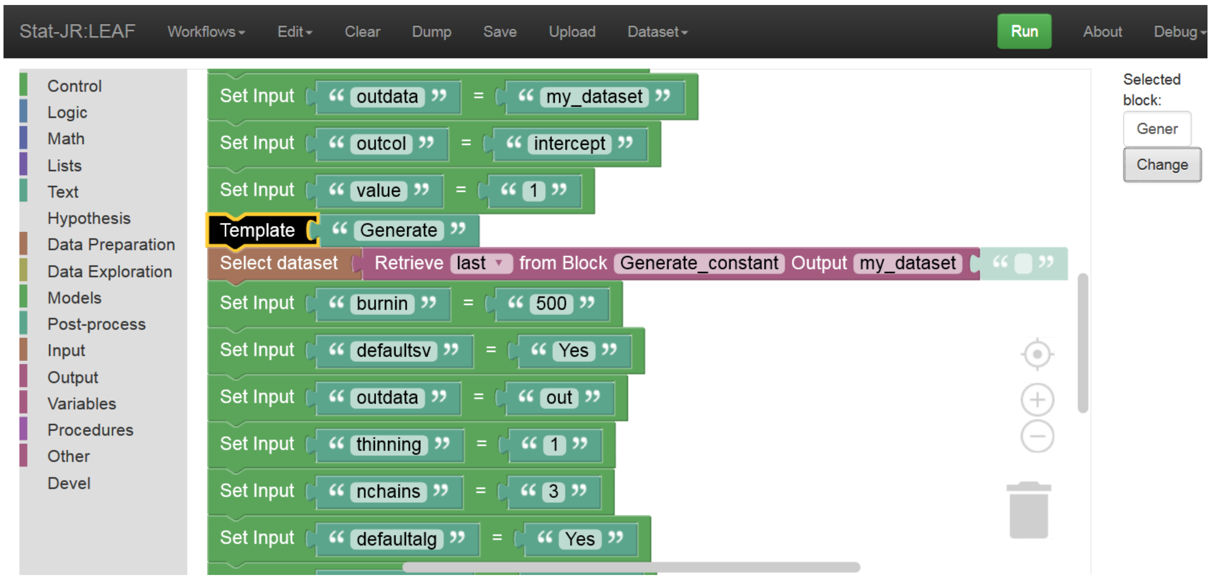
via the **Re-edit** button, etc.) You will also note that we've copied this block ID and pasted it in our *Retrieve* block (between "Block" and "Output"), as shown below:



Set Input "outcol" = "intercept"
Set Input "value" = "1"
Template "Generate"
Select dataset Retrieve last from Block 16 Output my_dataset ""
Set Input "burnin" = "500"
Set Input "defaultsv" = "Yes"
Set Input "outdata" = "out"
Set Input "thinning" = "1"
Set Input "nchains" = "3"
Set Input "defaultalg" = "Yes"
Set Input "iterations" = "2000"
Set Input "y" = "normexam"
Selected block: 16

Figure 43

Actually, let's change this block ID to something more meaningful: making sure the relevant *Template* block is still highlighted, type "Generate_constant" in the *"Selected block"* box and press the **Change** button. You'll see that the reference to it in the *Retrieve* block is also automatically modified to reflect this change:



Set Input "outdata" = "my_dataset"
Set Input "outcol" = "intercept"
Set Input "value" = "1"
Template "Generate"
Select dataset Retrieve last from Block Generate_constant Output my_dataset ""
Set Input "burnin" = "500"
Set Input "defaultsv" = "Yes"
Set Input "outdata" = "out"
Set Input "thinning" = "1"
Set Input "nchains" = "3"
Set Input "defaultalg" = "Yes"
Selected block: Gener

Figure 44

Note our choice of *last* in the *Retrieve* block simply tells the workflow to take the version of the object created the last time this block was executed (this becomes important within loops where the same block is called more than once).

Next, rather than hard-wiring our choice of response (*y*) variable in the subsequent model fit, we need to feed in the variable (*response*) defined above, so the value for the input "y" to the variable *response* as shown below:

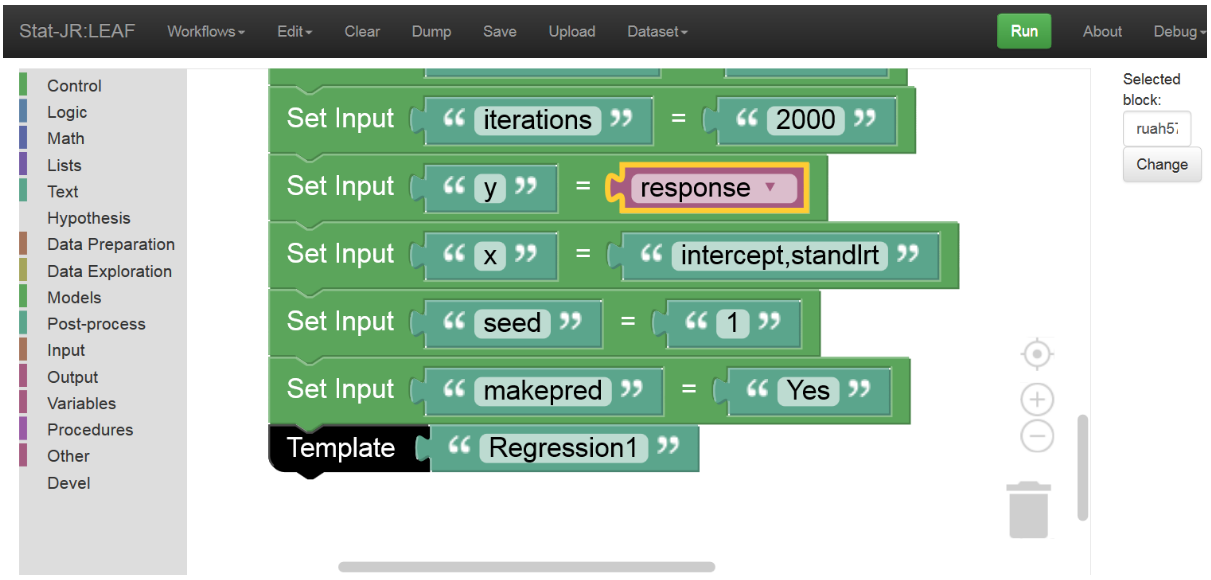


Figure 45

Finally, append a *Show* block to the end of the workflow requesting the output *ModelResults* be displayed:

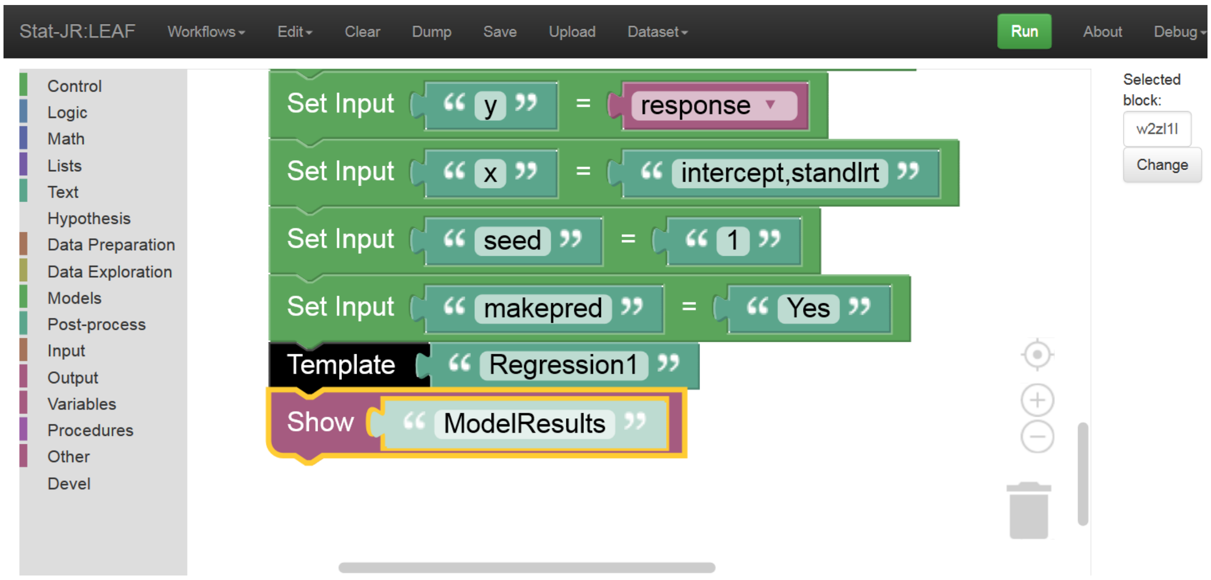
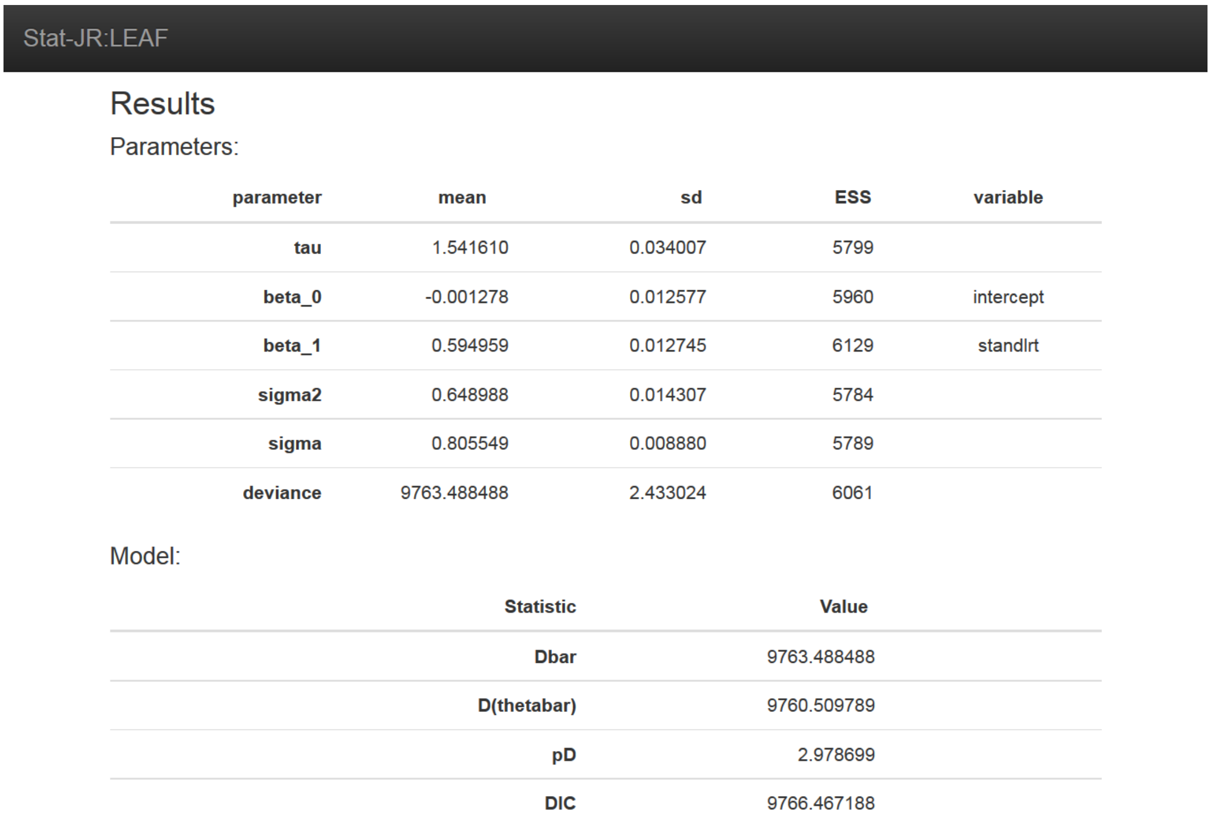


Figure 46

We will **Save** this workflow as *section1_11.xml* and then **Run** it (in this example choosing *normexam* as our variable of interest). Note that it will take a little longer for this workflow to finish its execution, and nothing will appear until the workflow has finished. If you scroll down to the bottom of the window after running it, it will look as follows:


Stat-JR:LEAF

Results

Parameters:

parameter	mean	sd	ESS	variable
tau	1.541610	0.034007	5799	
beta_0	-0.001278	0.012577	5960	intercept
beta_1	0.594959	0.012745	6129	standlrt
sigma2	0.648988	0.014307	5784	
sigma	0.805549	0.008880	5789	
deviance	9763.488488	2.433024	6061	

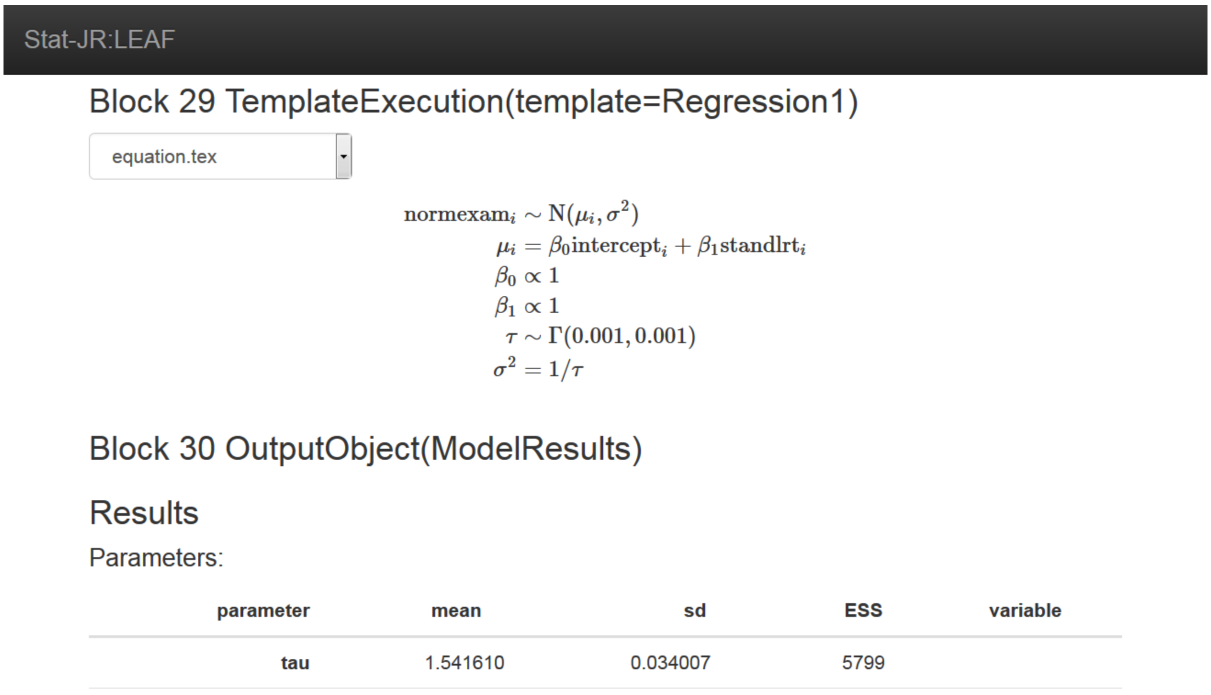
Model:

Statistic	Value
Dbar	9763.488488
D(thetabar)	9760.509789
pD	2.978699
DIC	9766.467188

Figure 47

So we see the results that we saw within TREE, from our model fit, appearing in the final block of the output.

As we saw earlier, the *Show* block is not the only way to see outputs; we can view any of the output objects from the regression model fit via the pull-down list under the block above (Block 27 in this example) which represents the *Regression1* template run. For example if we choose *equation.tex* we get the following output:



Block 29 TemplateExecution(template=Regression1)

$$\begin{aligned} \mathrm{normexam}_i &\sim \mathrm{N}(\mu_i, \sigma^2) \\ \mu_i &= \beta_0 \mathrm{intercept}_i + \beta_1 \mathrm{standlrt}_i \\ \beta_0 &\propto 1 \\ \beta_1 &\propto 1 \\ \tau &\sim \Gamma(0.001, 0.001) \\ \sigma^2 &= 1/\tau \end{aligned}$$

Block 30 OutputObject(ModelResults)

Results

Parameters:

parameter	mean	sd	ESS	variable
tau	1.541610	0.034007	5799	

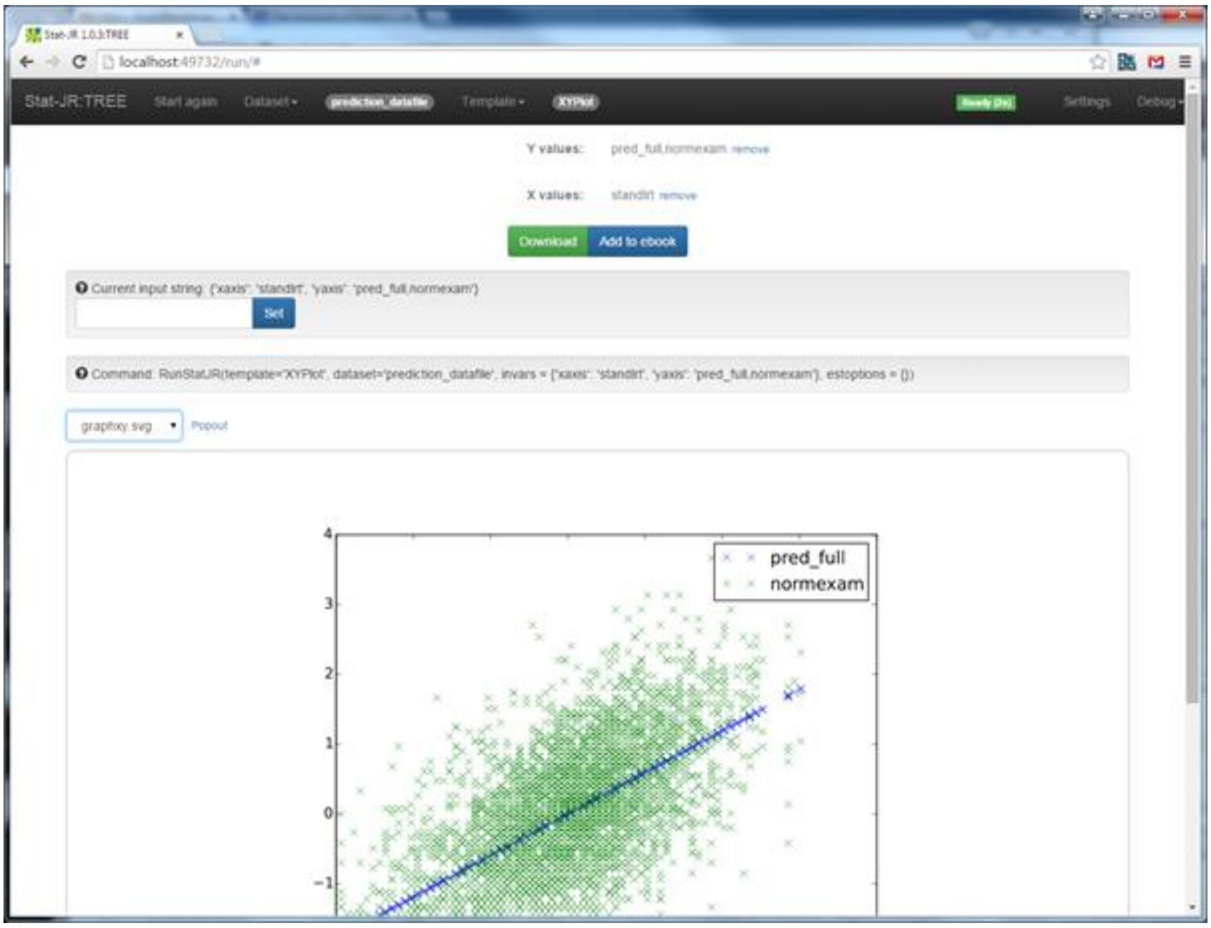
Figure 48

The only difference with this and the *Show* block is that the pull-down list is interactive, but it can only display one object at a time (whereas you could append several *Show* blocks on top of each other).

1.12 Adding predictions to the workflow

Going back to the TREE interface, since we selected the option to generate a prediction dataset we can look at the predictions graphically. The *Regression1* template has created a dataset object called *prediction_datafile* which we can select from the list of datasets in TREE (it will be in darker font to indicate it has been generated by the software and is loaded in the current session). Having chosen this as our dataset in TREE (it should appear in the black bar at the top once you have selected it) we can perform operations on it – e.g. plot predictions – by choosing an appropriate template (we will choose *XYPlot*) via the usual means.

Having chosen *XYPlot*, we can now set **Y values** to plot both the prediction and the original response variable (*pred_full* and *normexam*, in our example) and the **X values** to be our predictor variable of interest (*standlrt*, in this example). Clicking on **Next** and **Run** will give the following (if we select *graphxy.svg* from the list):



Y values: pred_full.normexam remove

X values: standlrt remove

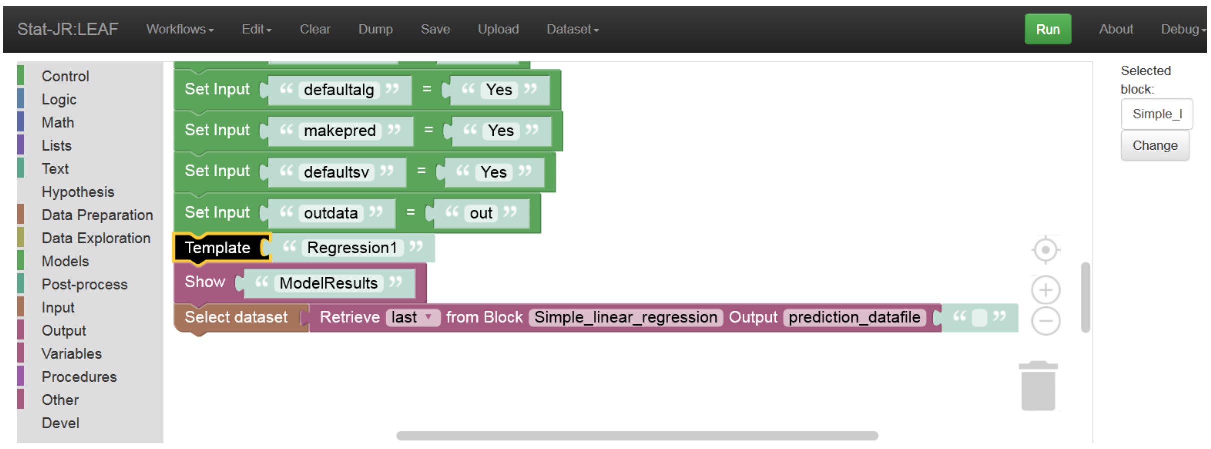
Current input string: {'xaxis': 'standlrt', 'yaxis': 'pred_full.normexam'}

Command: RunStatJR(template='XYPlot', dataset='prediction_datafile', invars = {'xaxis': 'standlrt', 'yaxis': 'pred_full.normexam'}, estoptions = {})

Figure 49

Here we see the data in green and the regression line in blue.

So, to add this to the workflow we will need to change dataset (to the *prediction_datafile* generated by the template). Let's return to the workflow interface and add the following to our existing workflow; note that we've changed the Block ID of the *Template* block for the *Regression1* template to "Simple_linear_regression" (by selecting the relevant *Template* block and changing its block ID via the "Selected block" box towards the top right-hand corner):



Set Input " defaultalg " = " Yes "
Set Input " makepred " = " Yes "
Set Input " defaultsv " = " Yes "
Set Input " outdata " = " out "
Template " Regression1 "
Show " ModelResults "
Select dataset Retrieve last from Block Simple_linear_regression Output prediction_datafile " "

Figure 50

As before, then, we change the dataset name via the *Select dataset* block, appending a *Retrieve* block to the end of it, and specifying in that block that we want to use the output object called *prediction_datafile* from the relevant template execution (the black *Template* block which runs the *Regression1* template).

For the graph, the input names and output objects are those we saw in TREE (although we can create these dynamically, based on the user's earlier choices: see Figure 125 in the Appendix for an example of how to do so) – we will leave these to you to add (e.g. see Section 1.4; remember to choose the corresponding template too; if in doubt, see Figure 125 in the Appendix). **Save** the resulting workflow as *section1_12.xml* and then click on **Run** to see what happens (in our example we again choose *normexam* when prompted). At the end of the run output you will see the prediction plot thus:

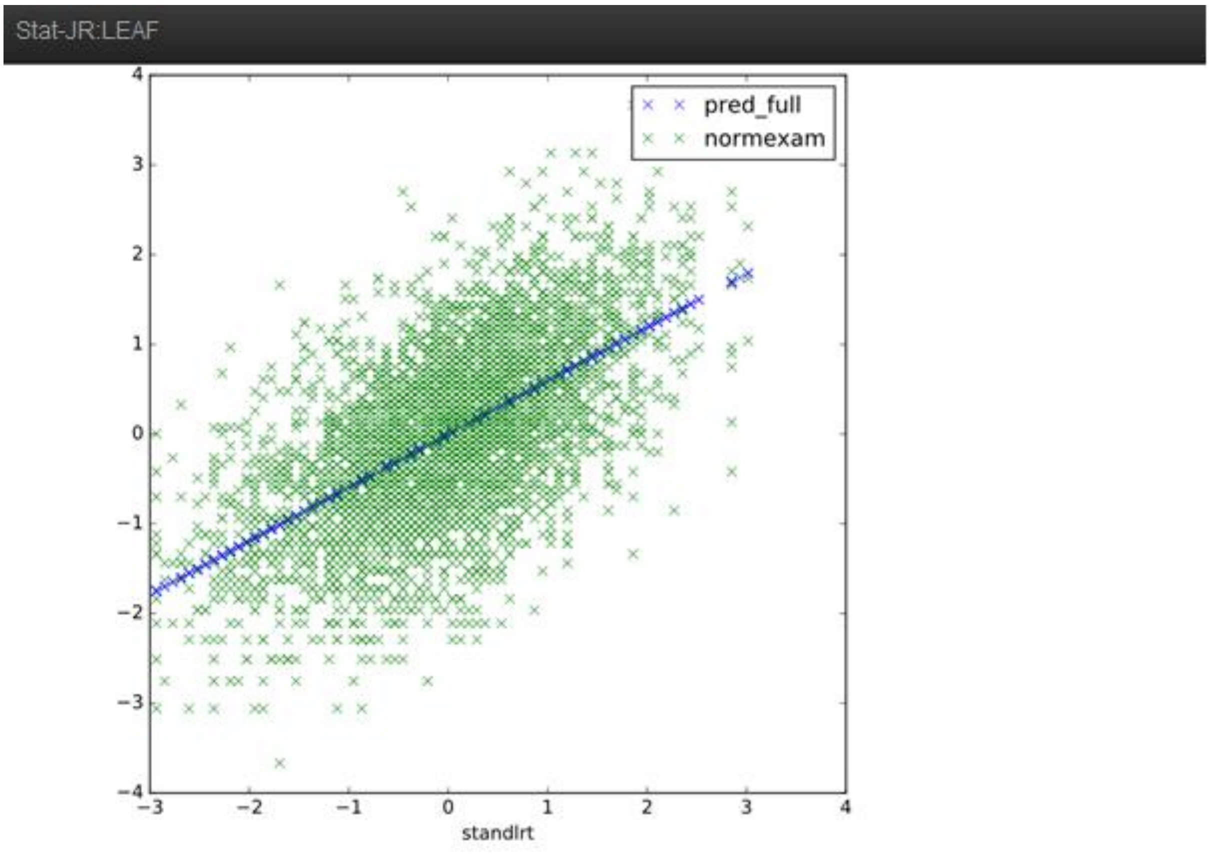


Figure 51

So here we have demonstrated how we can link together output (via an outputted dataset) from one template as input for another template.

1.13 What have we covered?

From this first session you should now be comfortable with using Stat-JR TREE: selecting a dataset and template, entering inputs, running it and inspecting the outputs. We've investigated how to use this information (the dataset, template, inputs and outputs of interest) to replicate the same operations in the Stat-JR workflow system, either doing so manually in LEAF, via TREE's **Make workflow** button, or via the **Re-edit** button in LEAF. In doing so we have covered:

- how to find and append blocks;

- duplicating and deleting blocks;
- saving workflows;
- including questions in workflows;
- using the same variable more than once in a workflow;
- retrieving output from one template execution for use in a later template execution;
- the functional relevance of the *Start* block.

1.14 What's next?

In the next section we will build on what we have covered and think about creating more interactive, generalised workflows for fitting regression models and also introduce the idea of a statistical analysis assistant. In doing so, we will also explore more of the workflow system's functionality.

Section 2 A statistical analysis assistant for conducting regression type models

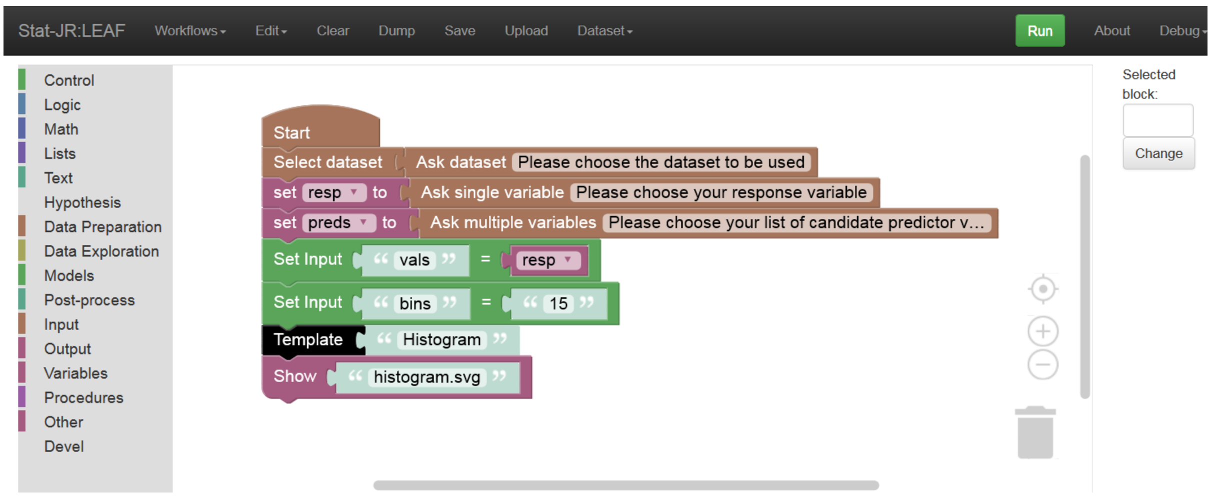
2.1 Overview

In the first section we introduced Stat-JR's TREE interface and its workflow system. By the end of that section 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 section 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 section 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 section 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:LEAF afresh or click on the **Clear** button to clear the current workflow, and set up the workflow using the palette of blocks accessible from the left-hand menu, as follows:



```
Start
Select dataset  Ask dataset  Please choose the dataset to be used
set resp to  Ask single variable  Please choose your response variable
set preds to  Ask multiple variables  Please choose your list of candidate predictor v...
Set Input  "vals"  =  resp
Set Input  "bins"  =  "15"
Template  "Histogram"
Show  "histogram.svg"
```

Figure 52

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 (truncated here) 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 *section2_02.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:

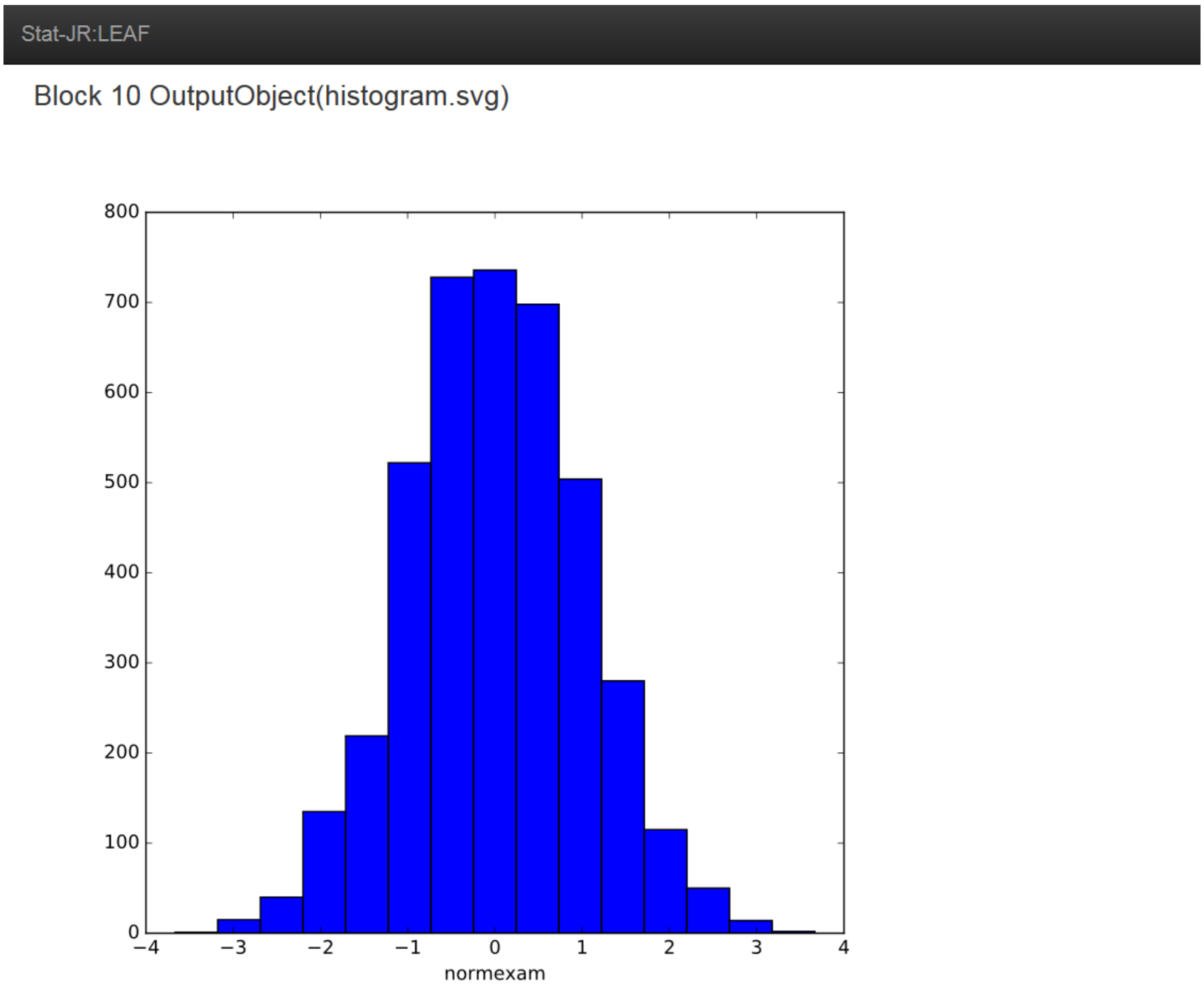
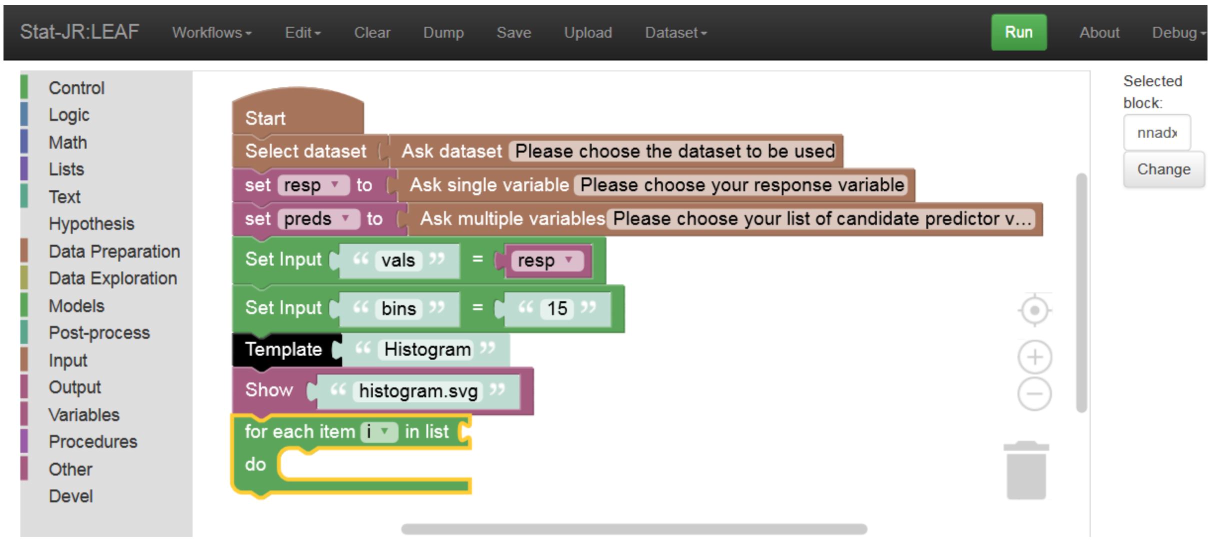


Figure 53

In this example we have chosen *normexam* as our response from the *tutorial* dataset 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:



```
Start
Select dataset  Ask dataset  Please choose the dataset to be used
set resp to  Ask single variable  Please choose your response variable
set preds to  Ask multiple variables  Please choose your list of candidate predictor v...
Set Input  " vals "  =  resp
Set Input  " bins "  =  " 15 "
Template  " Histogram "
Show  " histogram.svg "
for each item i in list
do
```

Figure 54

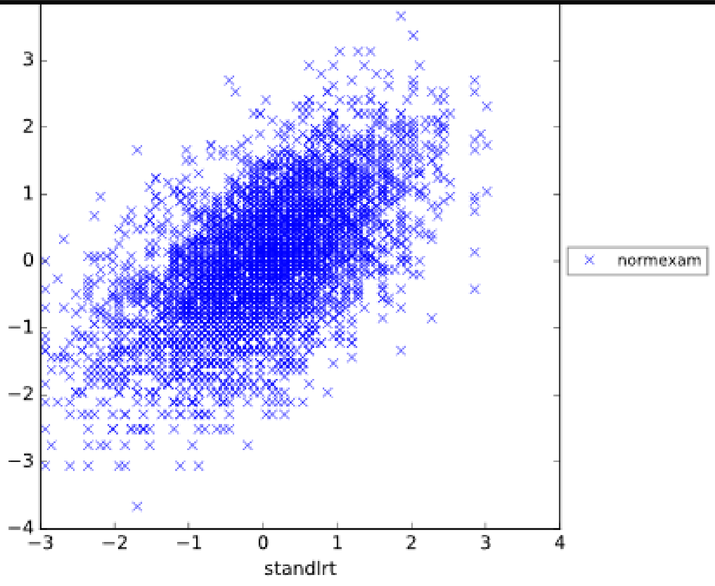
The *for-do* block has slots for two attached blocks (or sets of blocks) – the uppermost slot (to the right *of "…list")* requires a list containing elements to loop through, whilst the other slot, beneath, requires blocks defining what to do to each element of that list. The variable *i* will contain 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.[11]

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 section), showing the relevant output (graph) for each. Have a go at doing this yourself, and then compare it to our worked example in Figure 126 of the Appendix.

How did you get on? Save your workflow as *section2_03.xml*, and then **Run** it. In our example, below, we have chosen *normexam* as our response and the predictors *standlrt* and *avslrt*:

[11] 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).

Stat-JR:LEAF



Block 15 SetInput(yaxis=normexam)

Block 16 SetInput(xaxis=avslrt)

Block 17 TemplateExecution(template=XYPlot)

Block 18 OutputObject(graphxy.svg)

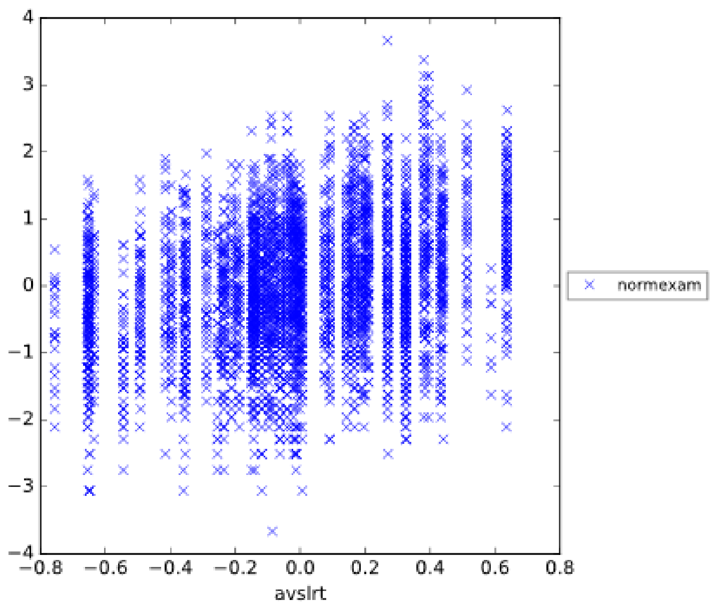


Figure 55

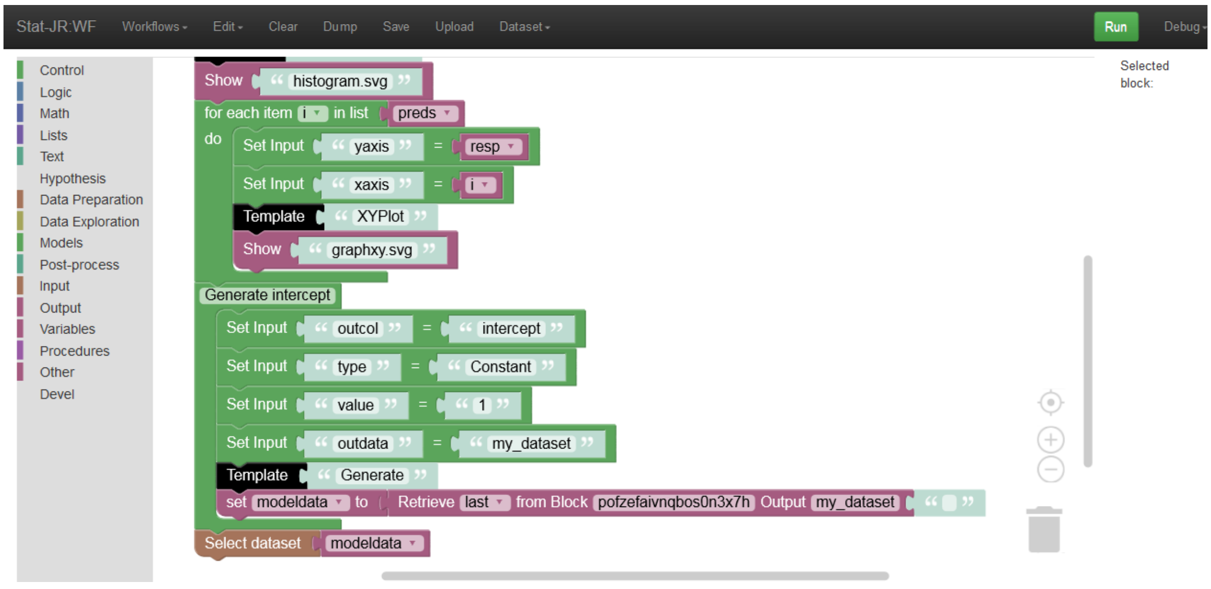
Here we see that both of the predictor variables we chose appear to have a positive relationship with the response (*normexam*), with *avslrt* plotted as discrete bands of points as this variable is constant for each school in our two level dataset.

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 section. 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 Section 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 Section 3).

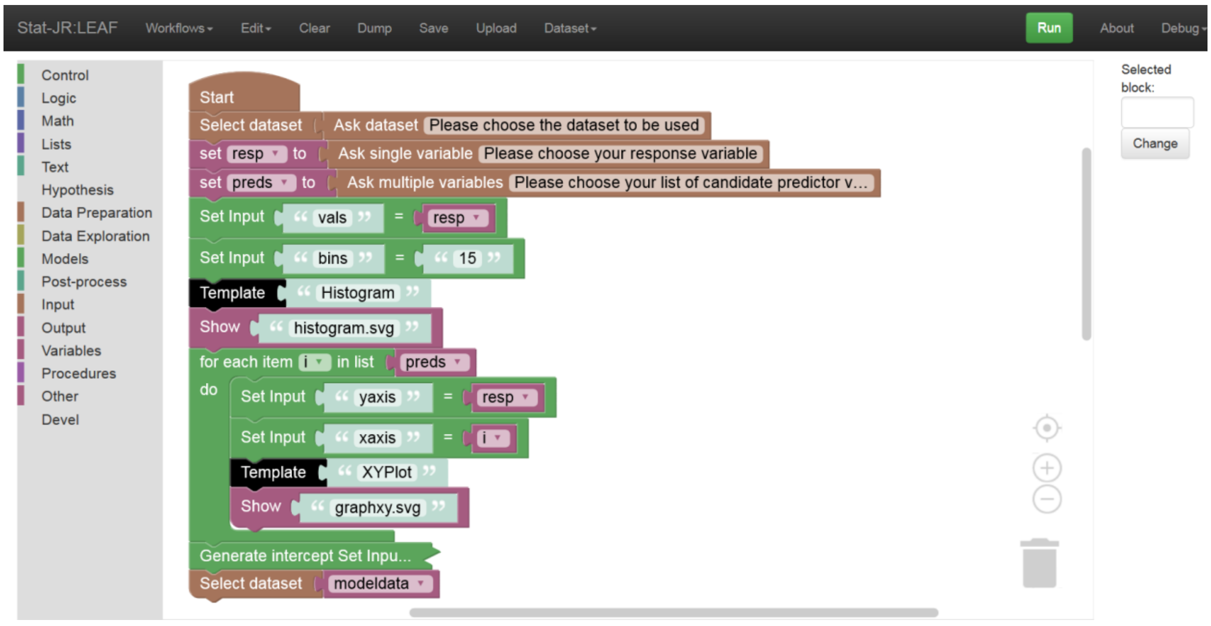
Here we've nested our completed *Set Input* blocks and a black (run) *Template* block all inside a *grouping* block, together with a *Retrieve* block. As you can see, we have given the *grouping* block an appropriate name ("*Generate intercept*") to describe the function of the blocks within. Instead of plugging the *Retrieve* block straight into a *Select Dataset* block, 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. We've also changed the default block ID for the black *Template* block associated with the *Generate* template to *"Generate_constant"* and used this to reference that block in the *Retrieve* block. Here we constructed this section of the workflow afresh, but as we've seen in earlier sections we could instead have (a) imported another workflow and taken our blocks of interest from that, deleting the rest (e.g. section1_11, which we made earlier: if you do this make sure you select **Import**, otherwise you will over-write the current workflow), (b) run the *Generate* template in TREE and imported the resulting workflow, or (c) used the *Re-edit* button to populate input values for the *Generate* template.



```
Show “ histogram.svg ”
for each item i in list preds
do  Set Input “ yaxis ” = resp
    Set Input “ xaxis ” = i
    Template “ XYPlot ”
    Show “ graphxy.svg ”
Generate intercept
    Set Input “ outcol ” = “ intercept ”
    Set Input “ type ” = “ Constant ”
    Set Input “ value ” = “ 1 ”
    Set Input “ outdata ” = “ my_dataset ”
    Template “ Generate ”
    set modeldata to Retrieve last from Block pofzefaivnqbos0n3x7h Output my_dataset “ ”
Select dataset modeldata
```

Figure 56

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



```
Start
Select dataset Ask dataset Please choose the dataset to be used
set resp to Ask single variable Please choose your response variable
set preds to Ask multiple variables Please choose your list of candidate predictor v...
Set Input “ vals ” = resp
Set Input “ bins ” = “ 15 ”
Template “ Histogram ”
Show “ histogram.svg ”
for each item i in list preds
do  Set Input “ yaxis ” = resp
    Set Input “ xaxis ” = i
    Template “ XYPlot ”
    Show “ graphxy.svg ”
Generate intercept Set Inpu...
Select dataset modeldata
```

Figure 57

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:

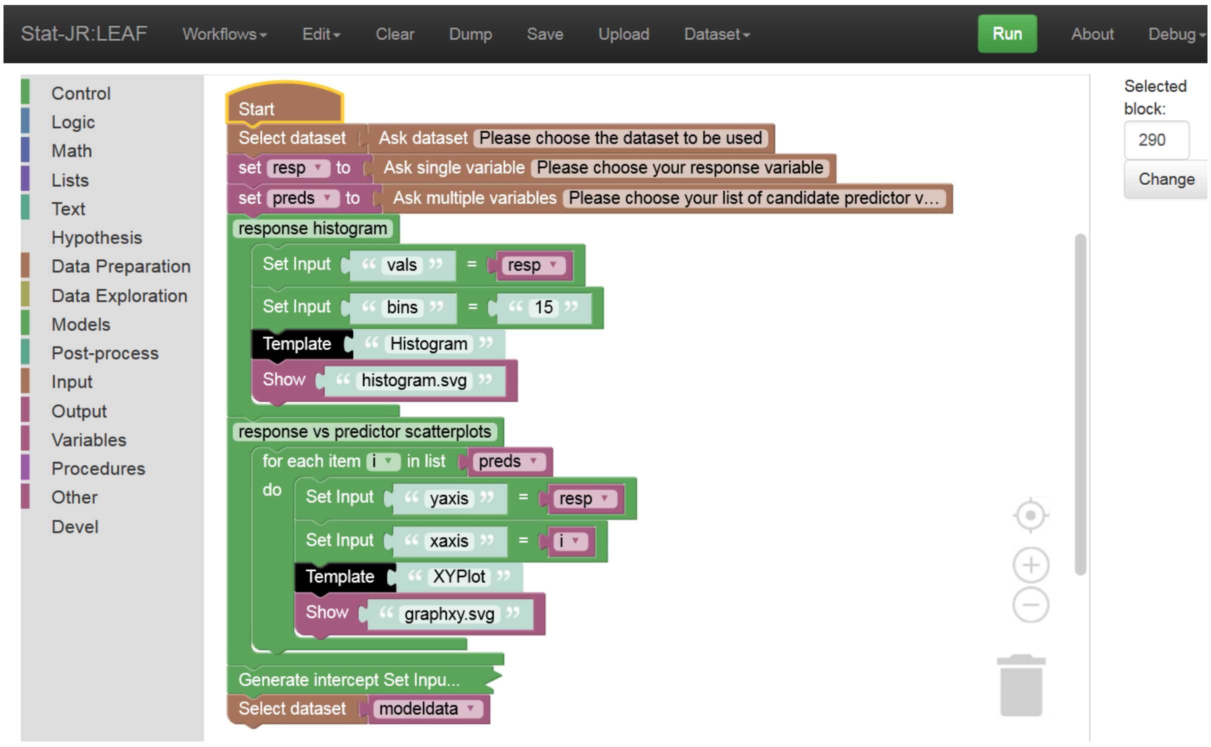


Figure 58

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

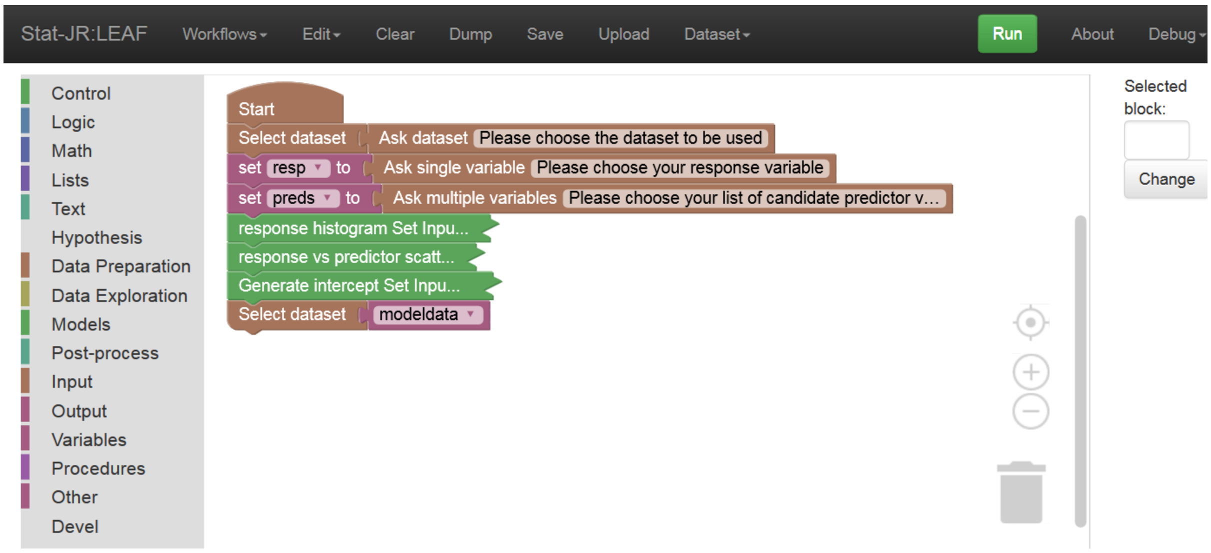
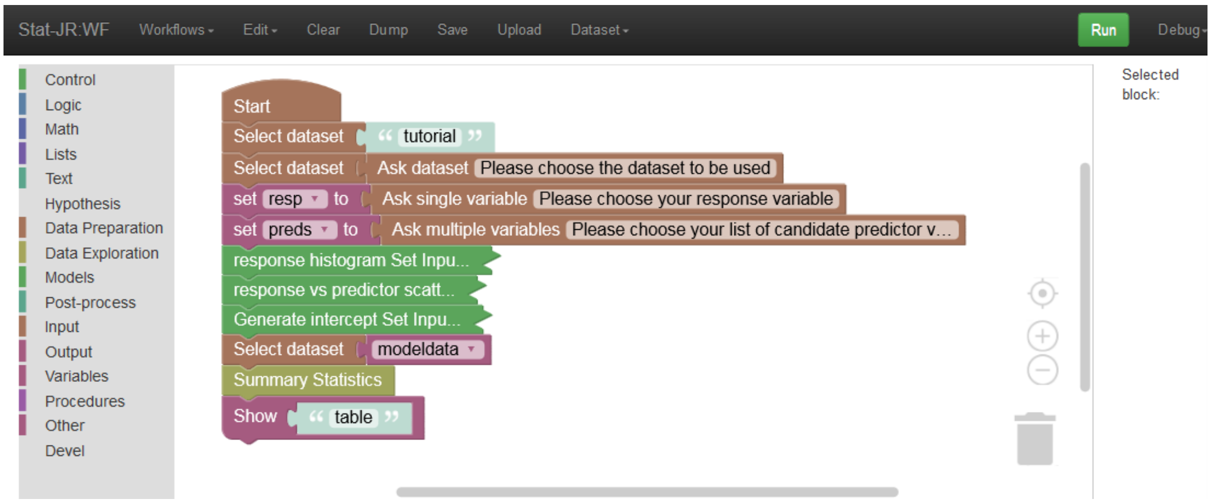


Figure 59

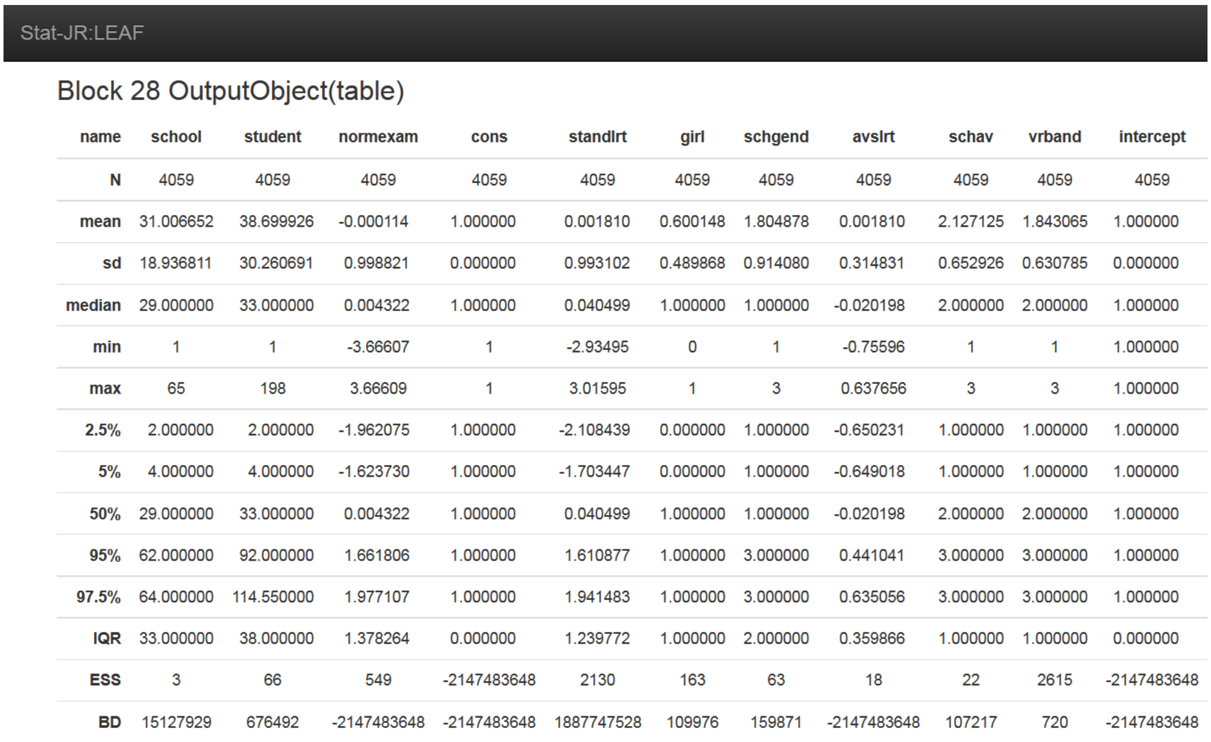
To complete the operation we will use a block we haven't yet investigated, namely the *Summary Statistics* block available in the **Data Exploration** block list. This block will produce summary statistics for the dataset and we can pull these out for display by adding a *Show* block for the *"table"* 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.



```
Start
Select dataset "tutorial"
Select dataset Ask dataset Please choose the dataset to be used
set resp to Ask single variable Please choose your response variable
set preds to Ask multiple variables Please choose your list of candidate predictor v...
response histogram Set Inpu...
response vs predictor scatt...
Generate intercept Set Inpu...
Select dataset modeldata
Summary Statistics
Show "table"
```

Figure 60

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



Block 28 OutputObject(table)

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

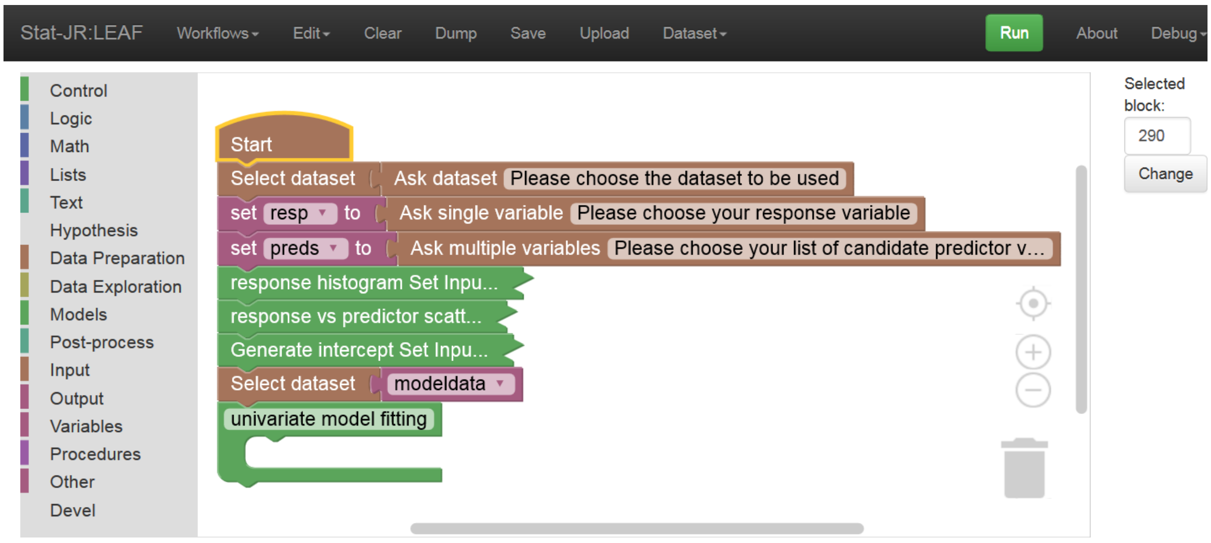
Figure 61

You will see that at the end we have the new column labelled *intercept*. (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).

Save your workflow as *section2_04.xml*.

2.5 Univariable Models – running the models

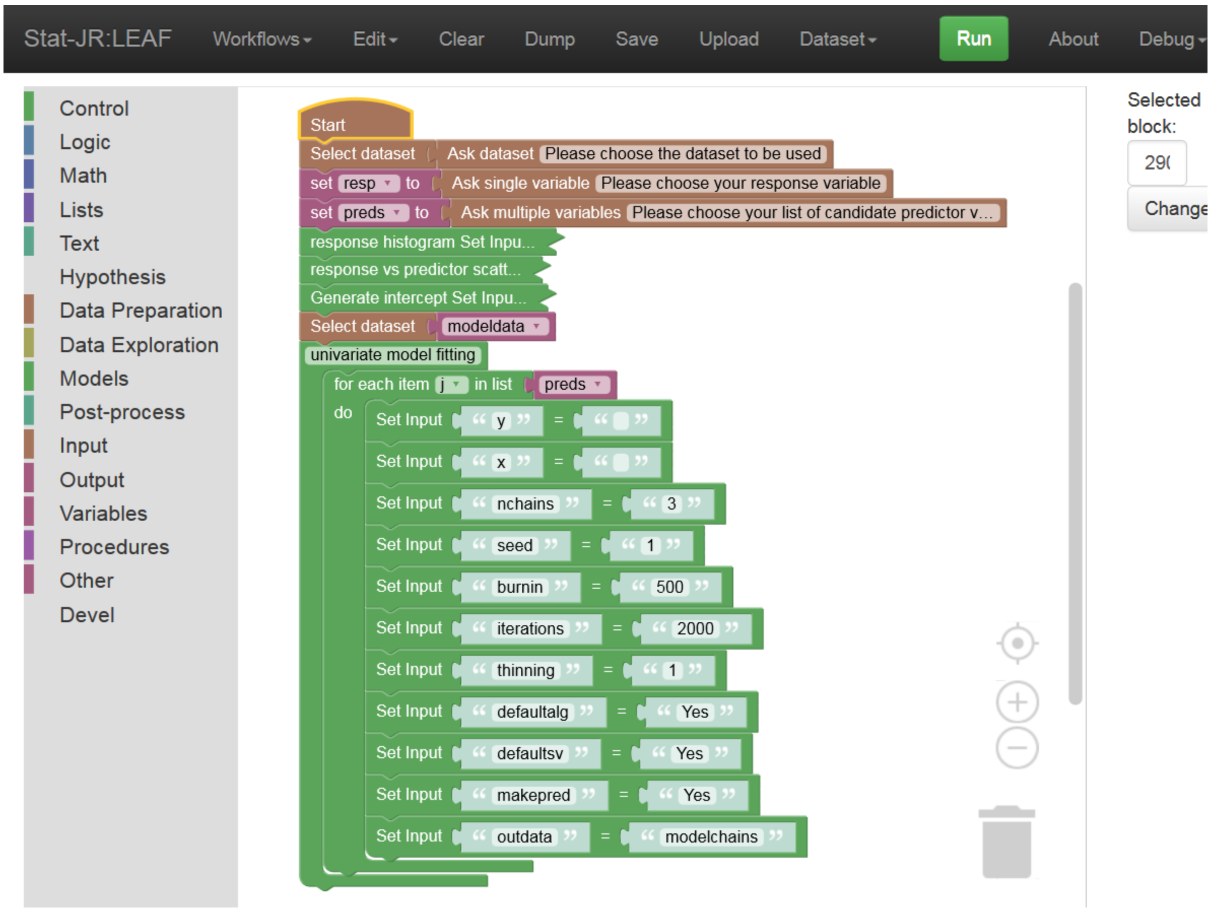
We will next perform the actual model fitting by looping through the list of predictors. Note, in this short example, we are assuming that all predictors will be treated as continuous rather than categorical variables and thus be included in the model in their current form rather than as a series of dummy variables. Recall that in the first section 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 bin the last two blocks (for *Summary Statistics*). We will add a grouping block which we will label as *“univariate model fitting”* thus:



Start
Select dataset Ask dataset Please choose the dataset to be used
set resp to Ask single variable Please choose your response variable
set preds to Ask multiple variables Please choose your list of candidate predictor v…
response histogram Set Inpu...
response vs predictor scatt...
Generate intercept Set Inpu...
Select dataset modeldata
univariate model fitting

Figure 62

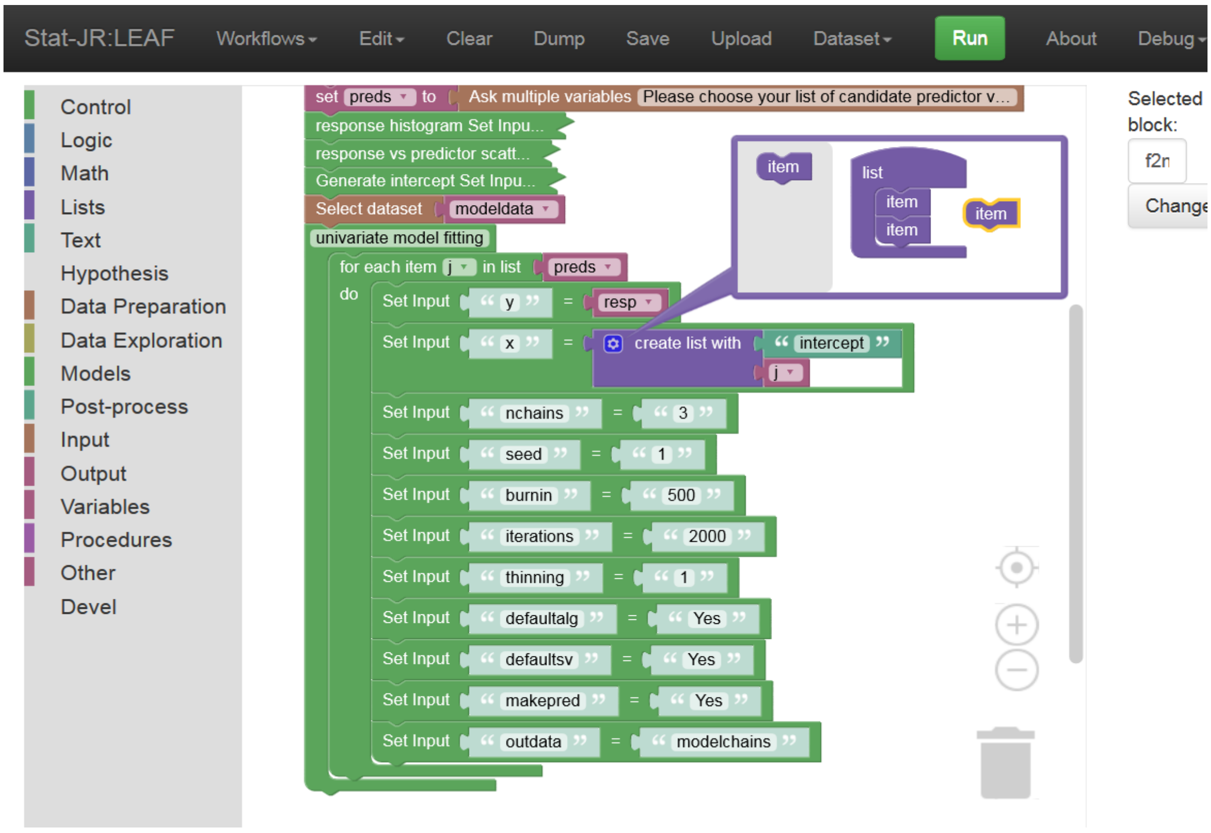
We will now need to loop over the predictors and so we will add a *for-do* loop block inside the *grouping* block and begin filling in the inputs required for a regression as shown below:



```
Start
Select dataset  Ask dataset  Please choose the dataset to be used
set resp to  Ask single variable  Please choose your response variable
set preds to  Ask multiple variables  Please choose your list of candidate predictor v...
response histogram Set Inpu...
response vs predictor scatt...
Generate intercept Set Inpu...
Select dataset  modeldata
univariate model fitting
  for each item j in list preds
  do
    Set Input " y " = "  "
    Set Input " x " = "  "
    Set Input " nchains " = " 3 "
    Set Input " seed " = " 1 "
    Set Input " burnin " = " 500 "
    Set Input " iterations " = " 2000 "
    Set Input " thinning " = " 1 "
    Set Input " defaultalg " = " Yes "
    Set Input " defaultsv " = " Yes "
    Set Input " makepred " = " Yes "
    Set Input " outdata " = " modelchains "
```

Figure 63

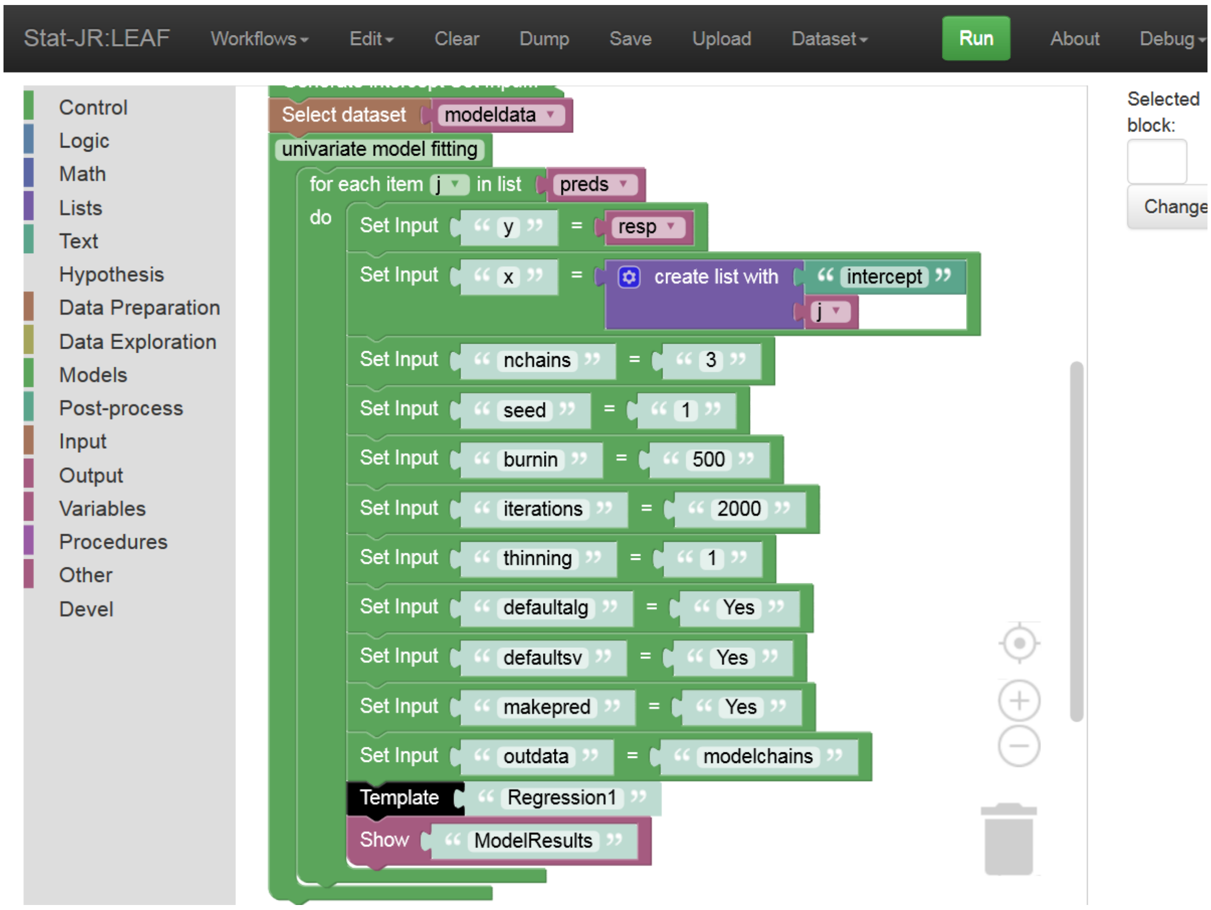
These inputs are just a reiteration of what we chose in the last section for this template, although we need to add the *y* and *x* variables. For *y* we simply choose the *resp* variable nominated (by the user) earlier in the workflow but for *x* we need to introduce a new block, namely the *create 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:



```
set preds to  Ask multiple variables  Please choose your list of candidate predictor v...
response histogram Set Inpu...
response vs predictor scatt...
Generate intercept Set Inpu...
Select dataset  modeldata
univariate model fitting
  for each item j in list preds
  do  Set Input "y" = resp
      Set Input "x" = create list with "intercept"
                                       j
      Set Input "nchains" = "3"
      Set Input "seed" = "1"
      Set Input "burnin" = "500"
      Set Input "iterations" = "2000"
      Set Input "thinning" = "1"
      Set Input "defaultalg" = "Yes"
      Set Input "defaultsv" = "Yes"
      Set Input "makepred" = "Yes"
      Set Input "outdata" = "modelchains"
```

Figure 64

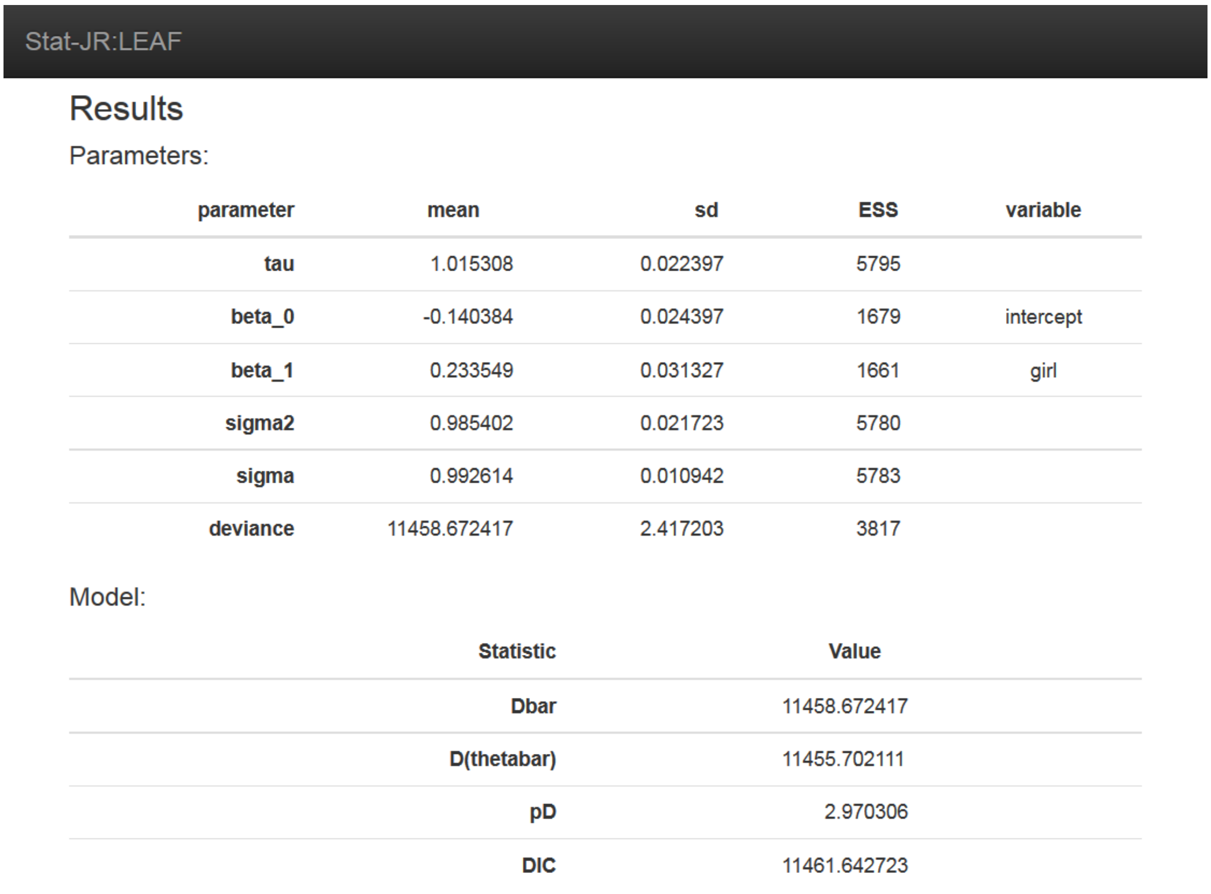
You’ll see we are creating a list that includes the variable *intercept* and whatever is the current predictor (as indexed by *j*) as we loop through the list. We next need to add the template block and we will also add a *Show* block for the *ModelResults* thus:



```
Select dataset modeldata
univariate model fitting
  for each item j in list preds
  do  Set Input "y" = resp
      Set Input "x" = create list with "intercept"
                                        j
      Set Input "nchains" = "3"
      Set Input "seed" = "1"
      Set Input "burnin" = "500"
      Set Input "iterations" = "2000"
      Set Input "thinning" = "1"
      Set Input "defaultalg" = "Yes"
      Set Input "defaultsv" = "Yes"
      Set Input "makepred" = "Yes"
      Set Input "outdata" = "modelchains"
      Template "Regression1"
      Show "ModelResults"
```

Figure 65

If we now save this workflow as *section2_05.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:LEAF

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

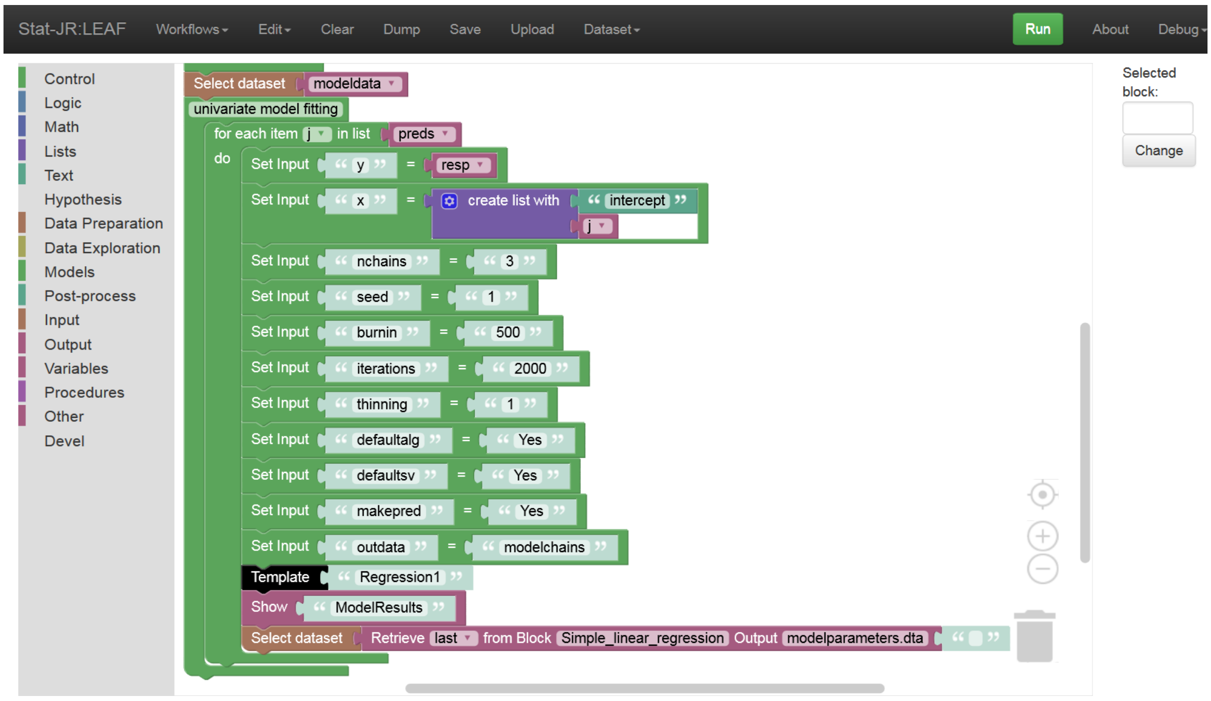
Figure 66

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:



```
Select dataset  modeldata
univariate model fitting
for each item j in list  preds
do  Set Input  "y"  =  resp
    Set Input  "x"  =  create list with  "intercept"
                                         j
    Set Input  "nchains"  =  "3"
    Set Input  "seed"  =  "1"
    Set Input  "burnin"  =  "500"
    Set Input  "iterations"  =  "2000"
    Set Input  "thinning"  =  "1"
    Set Input  "defaultalg"  =  "Yes"
    Set Input  "defaultsv"  =  "Yes"
    Set Input  "makepred"  =  "Yes"
    Set Input  "outdata"  =  "modelchains"
    Template  "Regression1"
    Show  "ModelResults"
    Select dataset  Retrieve  last  from Block  Simple_linear_regression  Output  modelparameters.dta  ""
```

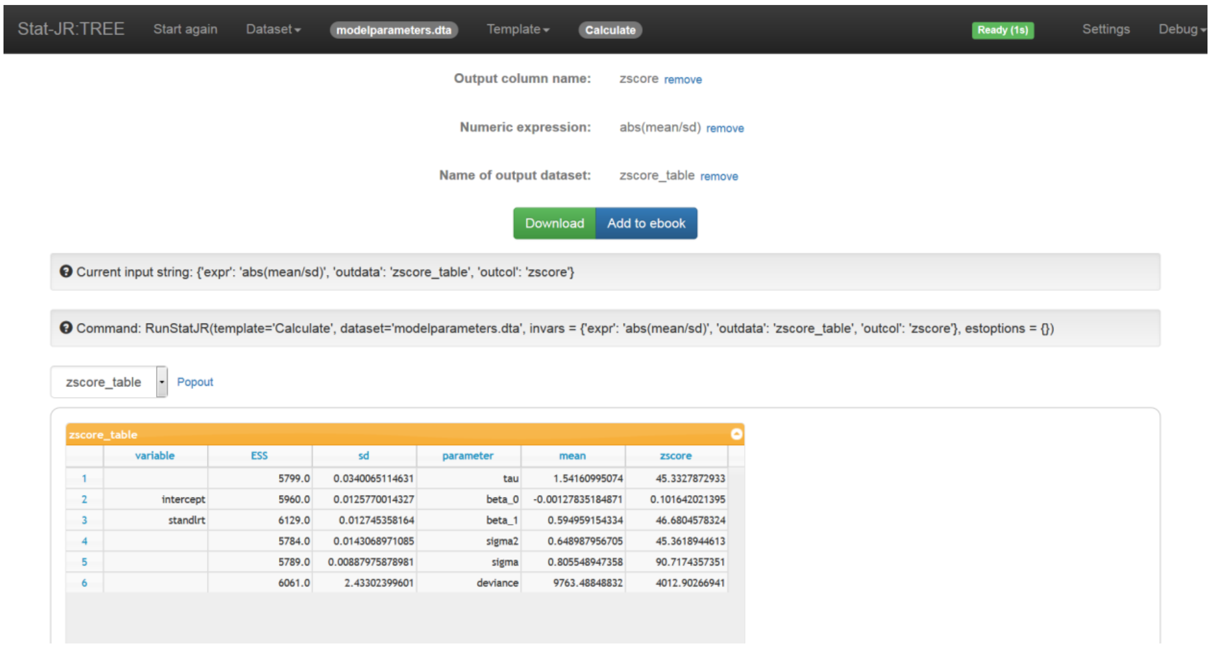
Figure 67

Here we've included another *Retrieve* statement, this time to pluck out the *modelparameters.dta* dataset, and have changed the block ID for the *Template* block associated with the *Regression1* template to *"Simple_linear_regression"*.

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.[12]

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:

[12] 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.



Output column name: zscore remove

Numeric expression: abs(mean/sd) remove

Name of output dataset: zscore_table remove

Current input string: {'expr': 'abs(mean/sd)', 'outdata': 'zscore_table', 'outcol': 'zscore'}

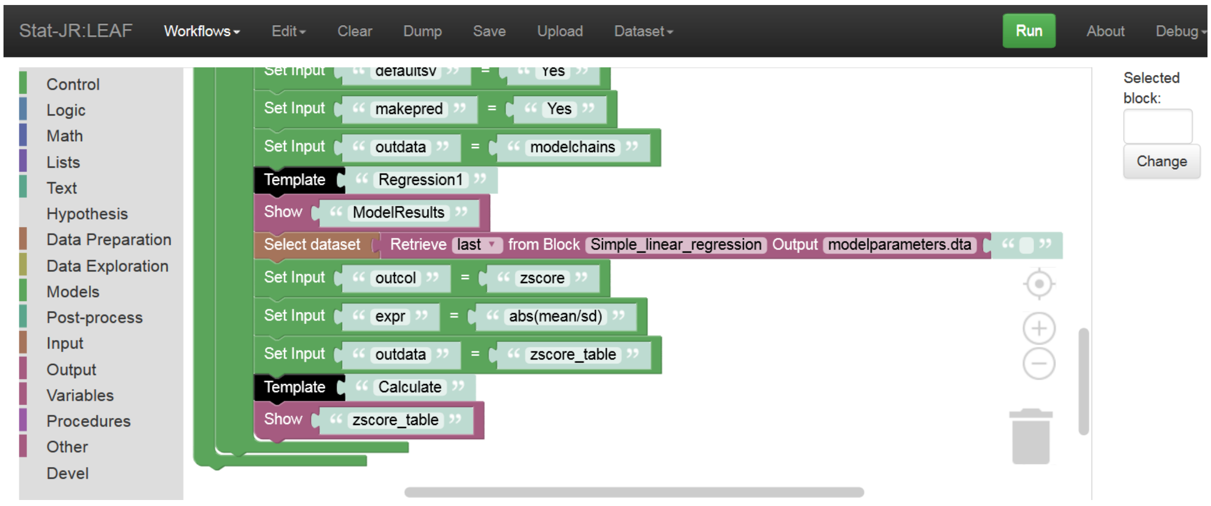
Command: RunStatJR(template='Calculate', dataset='modelparameters.dta', invars = {'expr': 'abs(mean/sd)', 'outdata': 'zscore_table', 'outcol': 'zscore'}, estoptions = {})

zscore_table

	variable	ESS	sd	parameter	mean	zscore
1		5799.0	0.0340065114631	tau	1.54160995074	45.3327872933
2	intercept	5960.0	0.0125770014327	beta_0	-0.00127835184871	0.101642021395
3	standlrt	6129.0	0.012745358164	beta_1	0.594959154334	46.6804578324
4		5784.0	0.0143068971085	sigma2	0.648987956705	45.3618944613
5		5789.0	0.00887975878981	sigma	0.805548947358	90.7174357351
6		6061.0	2.43302399601	deviance	9763.48848832	4012.90266941

Figure 68

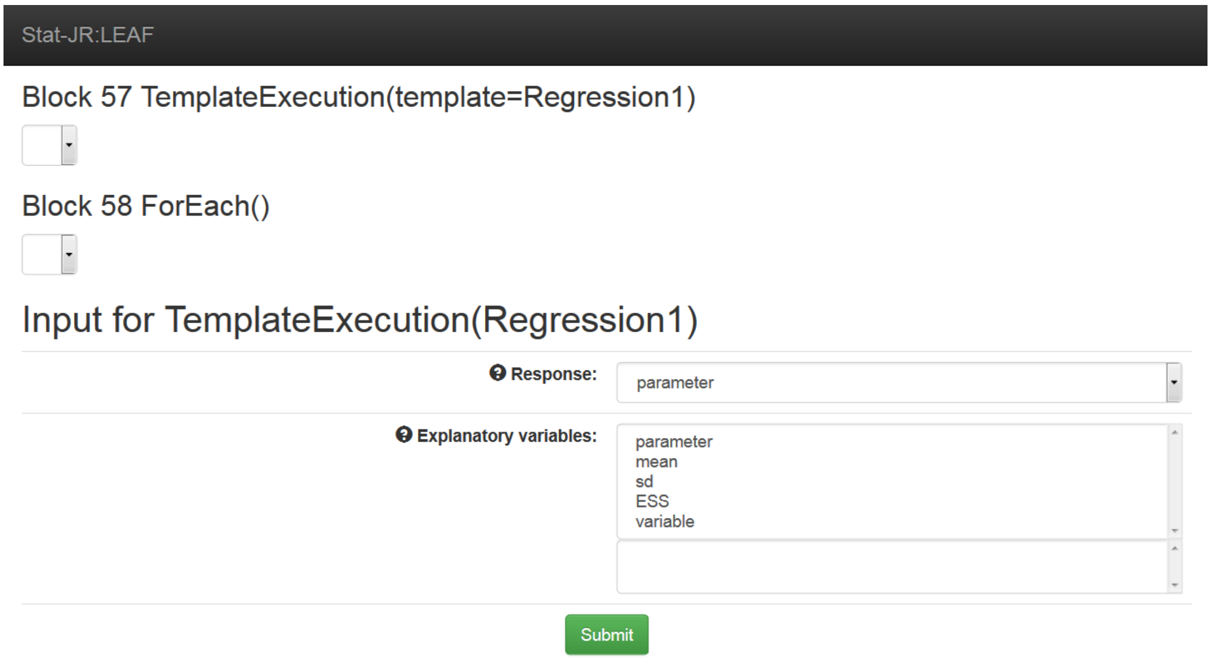
So let's now convert this into workflow blocks; of course we could save this as a workflow in TREE and export, but given it involves just a few blocks it's just as quick in this instance to assemble the blocks ourselves in LEAF:



```
Set Input " defaultsv " = " Yes "
Set Input " makepred " = " Yes "
Set Input " outdata " = " modelchains "
Template " Regression1 "
Show " ModelResults "
Select dataset Retrieve last from Block Simple_linear_regression Output modelparameters.dta " "
Set Input " outcol " = " zscore "
Set Input " expr " = " abs(mean/sd) "
Set Input " outdata " = " zscore_table "
Template " Calculate "
Show " zscore_table "
```

Figure 69

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:


Stat-JR:LEAF

Block 57 TemplateExecution(template=Regression1)

Block 58 ForEach()

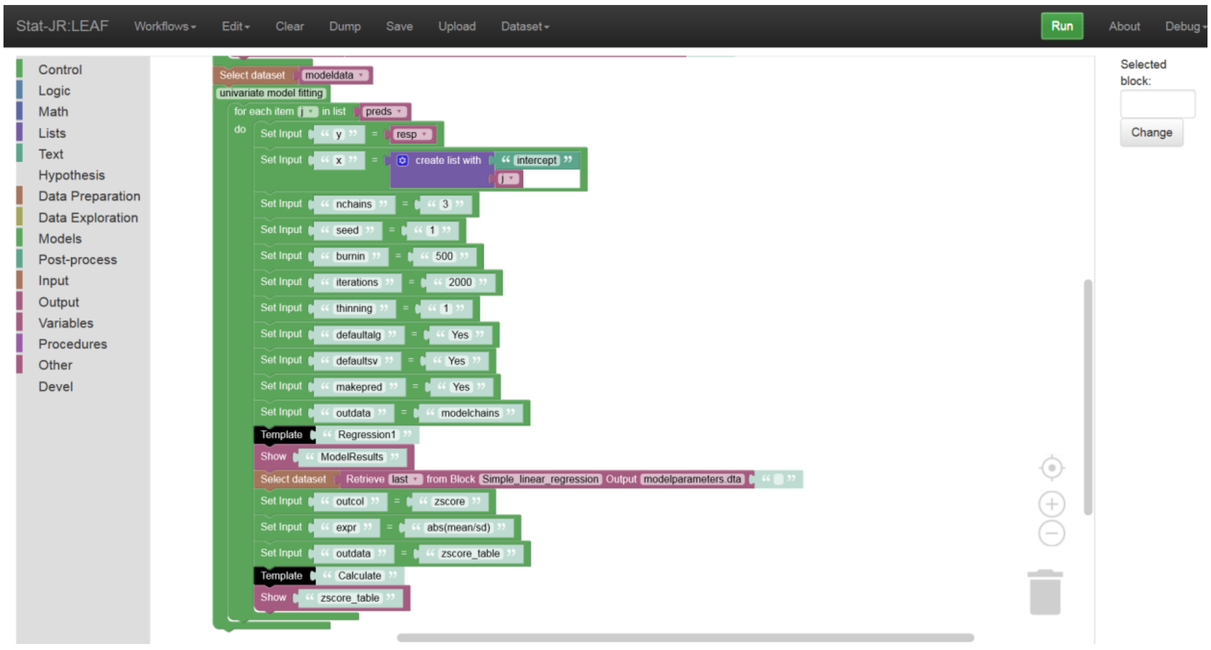
Input for TemplateExecution(Regression1)

Response:	parameter
Explanatory variables:	parameter mean sd ESS variable

Submit

Figure 70

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?

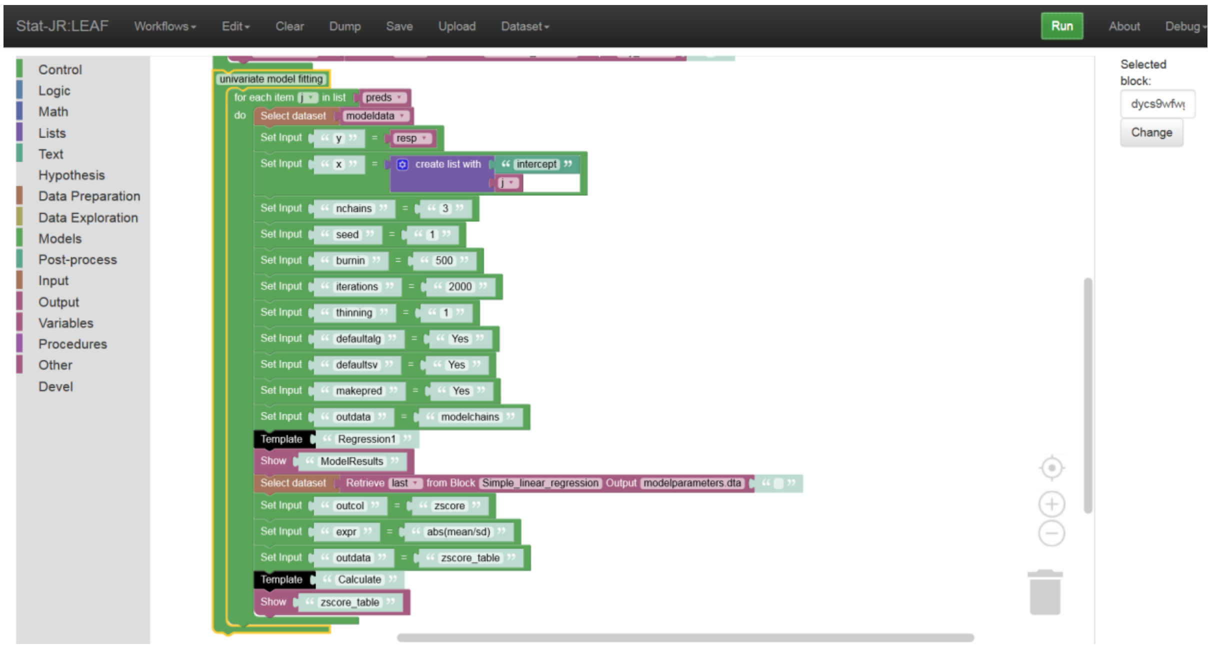


```
Select dataset  modeldata
univariate model fitting
  for each item j in list preds
  do  Set Input "y" = resp
      Set Input "x" = create list with "intercept"
                                       j
      Set Input "nchains" = "3"
      Set Input "seed" = "1"
      Set Input "burnin" = "500"
      Set Input "iterations" = "2000"
      Set Input "thinning" = "1"
      Set Input "defaultalg" = "Yes"
      Set Input "defaultsv" = "Yes"
      Set Input "makepred" = "Yes"
      Set Input "outdata" = "modelchains"
      Template "Regression1"
      Show "ModelResults"
      Select dataset  Retrieve last from Block Simple_linear_regression Output modelparameters.dta ""
      Set Input "outcol" = "zscore"
      Set Input "expr" = "abs(mean/sd)"
      Set Input "outdata" = "zscore_table"
      Template "Calculate"
      Show "zscore_table"
```

Figure 71

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.

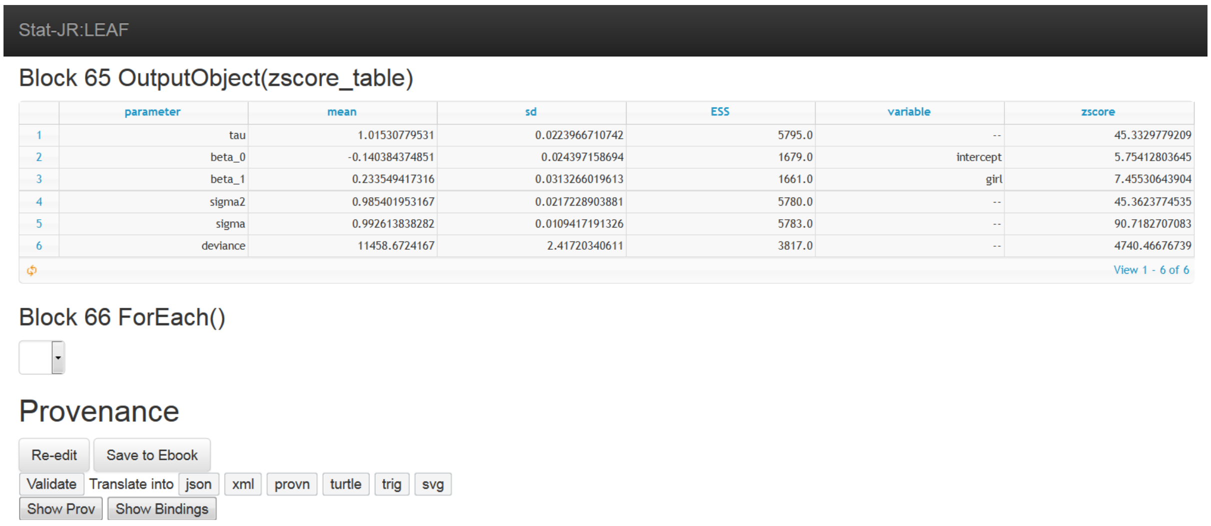
Let's remedy this by moving the *Select dataset: modeldata* blocks so that they appear *within* the loop. That way, the working dataset will be changed to the one the *Regression1* template inputs are expecting each time the loop begins again:



Stat-JR:LEAF
Workflows
Edit
Clear
Dump
Save
Upload
Dataset
Run
About
Debug
Control
Logic
Math
Lists
Text
Hypothesis
Data Preparation
Data Exploration
Models
Post-process
Input
Output
Variables
Procedures
Other
Devel
Selected block:
dycs9wfw
Change
univariate model fitting
for each item j in list preds
do Select dataset modeldata
Set Input "y" = resp
Set Input "x" = create list with "intercept" j
Set Input "nchains" = "3"
Set Input "seed" = "1"
Set Input "burnin" = "500"
Set Input "iterations" = "2000"
Set Input "thinning" = "1"
Set Input "defaultalg" = "Yes"
Set Input "defaultsv" = "Yes"
Set Input "makepred" = "Yes"
Set Input "outdata" = "modelchains"
Template "Regression1"
Show "ModelResults"
Select dataset Retrieve last from Block Simple_linear_regression Output modelparameters.dta " "
Set Input "outcol" = "zscore"
Set Input "expr" = "abs(mean/sd)"
Set Input "outdata" = "zscore_table"
Template "Calculate"
Show "zscore_table"

Figure 72

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


Stat-JR:LEAF

Block 65 OutputObject(zscore_table)

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

View 1 - 6 of 6

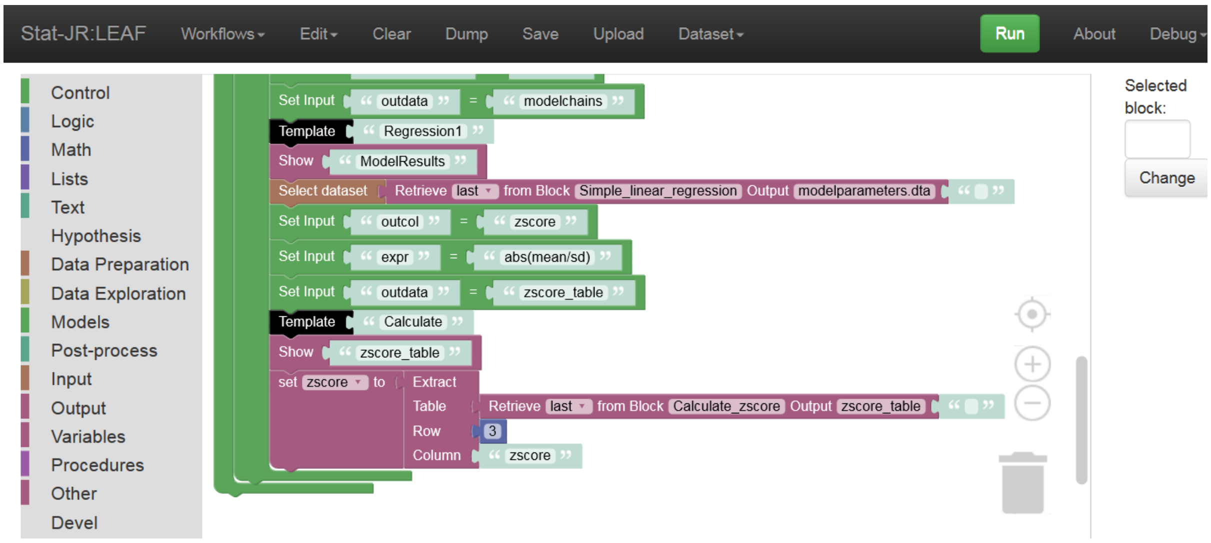
Block 66 ForEach()

Provenance

Re-edit Save to Ebook
Validate Translate into json xml provn turtle trig svg
Show Prov Show Bindings

Figure 73

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:



```
Set Input "outdata" = "modelchains"
Template "Regression1"
Show "ModelResults"
Select dataset Retrieve last from Block Simple_linear_regression Output modelparameters.dta ""
Set Input "outcol" = "zscore"
Set Input "expr" = "abs(mean/sd)"
Set Input "outdata" = "zscore_table"
Template "Calculate"
Show "zscore_table"
set zscore to Extract
                Table   Retrieve last from Block Calculate_zscore Output zscore_table ""
                Row     3
                Column  "zscore"
```

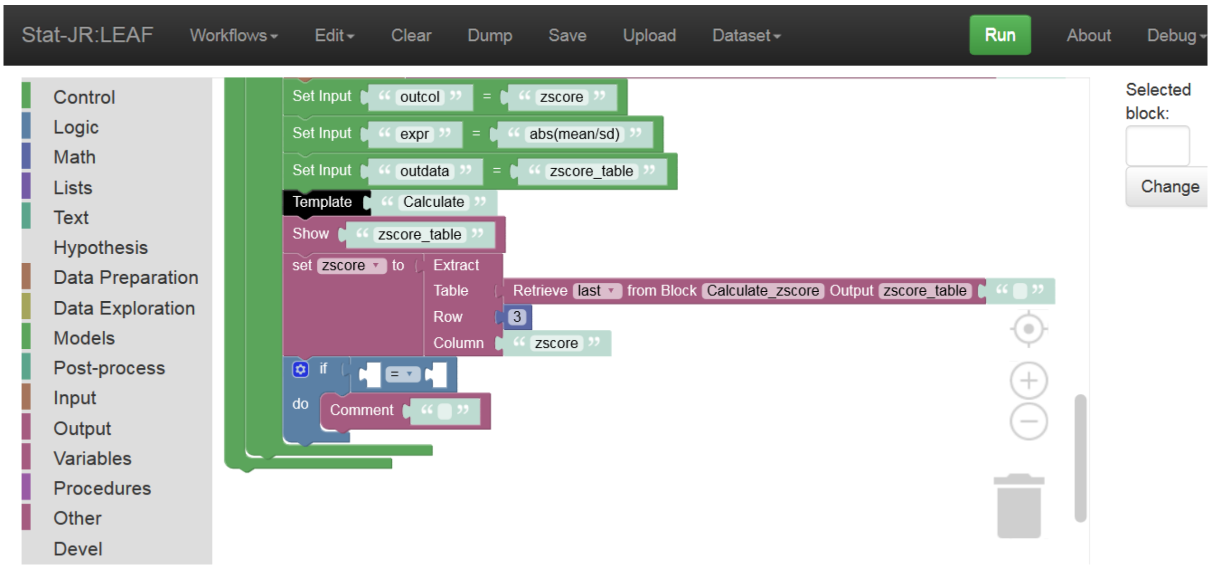
Figure 74

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*).[13]

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:

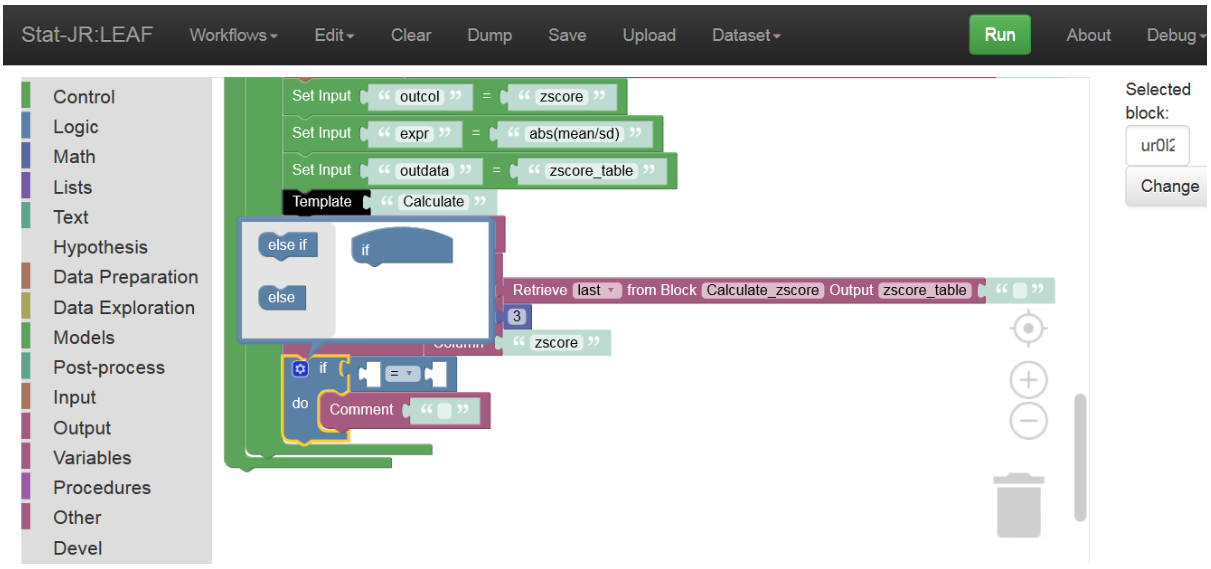
[13] NB: the blue block is retrieved from the **Math** block list, and the *Extract* block is from the **Other** block list.



Set Input "outcol" = "zscore"
Set Input "expr" = "abs(mean/sd)"
Set Input "outdata" = "zscore_table"
Template "Calculate"
Show "zscore_table"
set zscore to Extract Table Retrieve last from Block Calculate_zscore Output zscore_table ""
Row 3
Column "zscore"
if =
do Comment ""

Figure 75

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:



Set Input "outcol" = "zscore"
Set Input "expr" = "abs(mean/sd)"
Set Input "outdata" = "zscore_table"
Template "Calculate"
Retrieve last from Block Calculate_zscore Output zscore_table ""
3
"zscore"
if =
do Comment ""

Figure 76

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

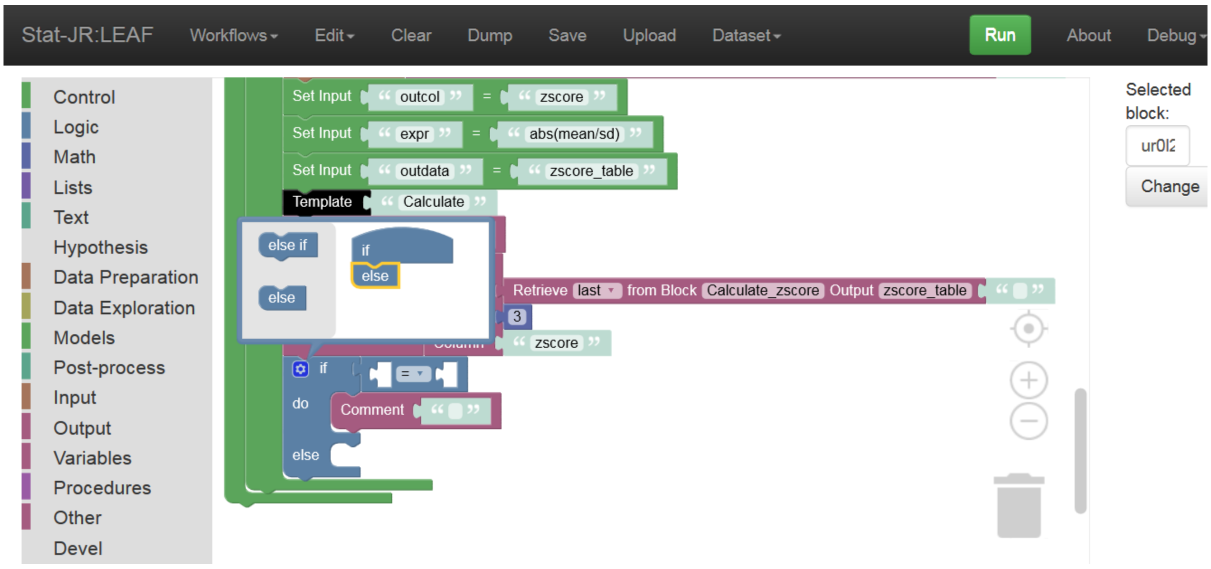


Figure 77

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…"):

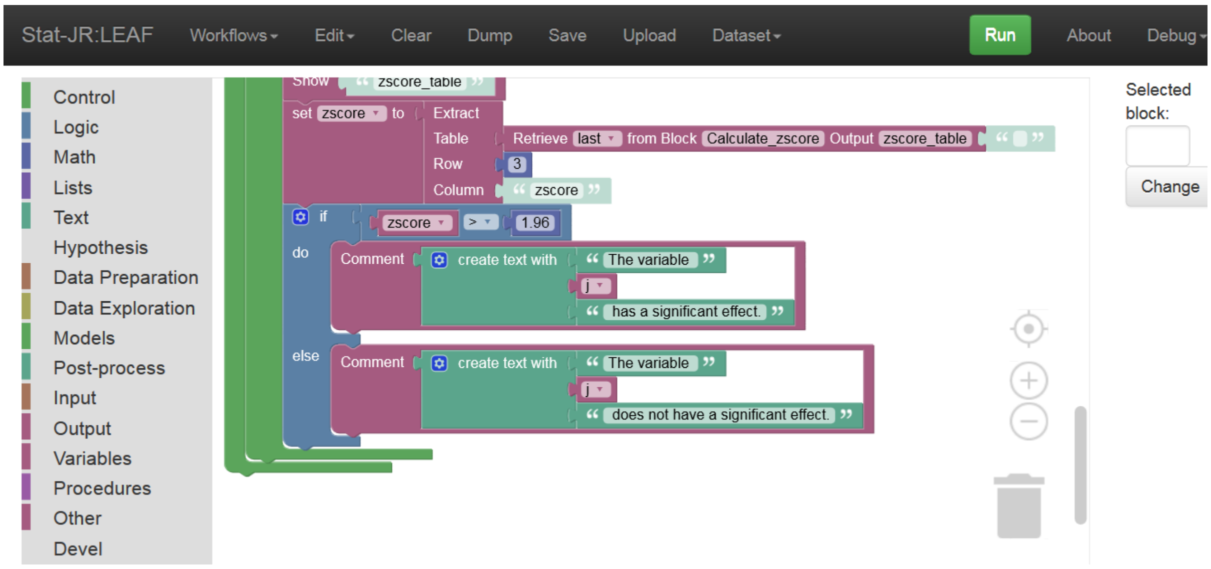
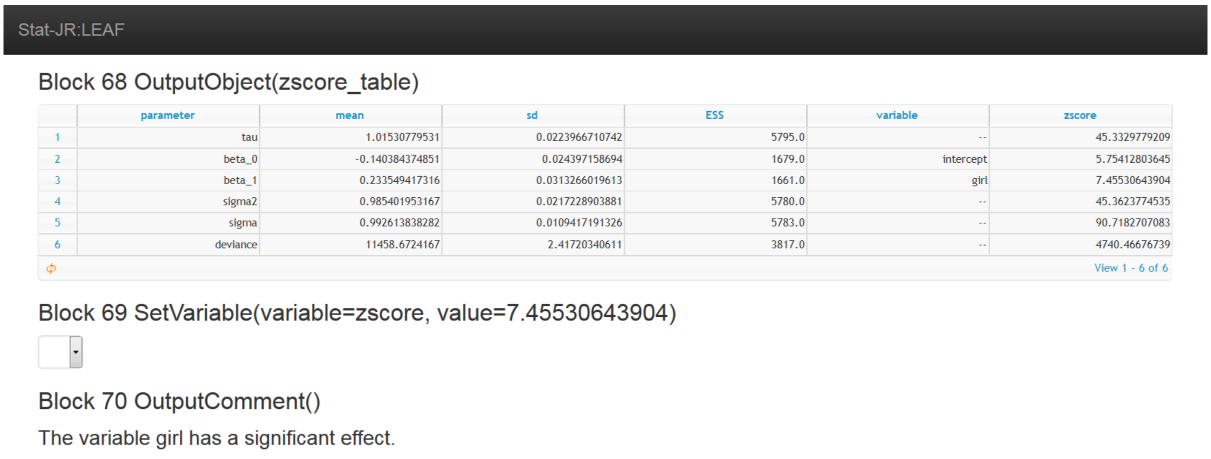


Figure 78

If we save this workflow as *section2_06.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*:



Block 68 OutputObject(zscore_table)

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

View 1 - 6 of 6

Block 69 SetVariable(variable=zscore, value=7.45530643904)

Block 70 OutputComment()

The variable girl has a significant effect.

Figure 79

So here we see the textual output indicating that *girl* has a significant effect on *normexam*[14]. 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

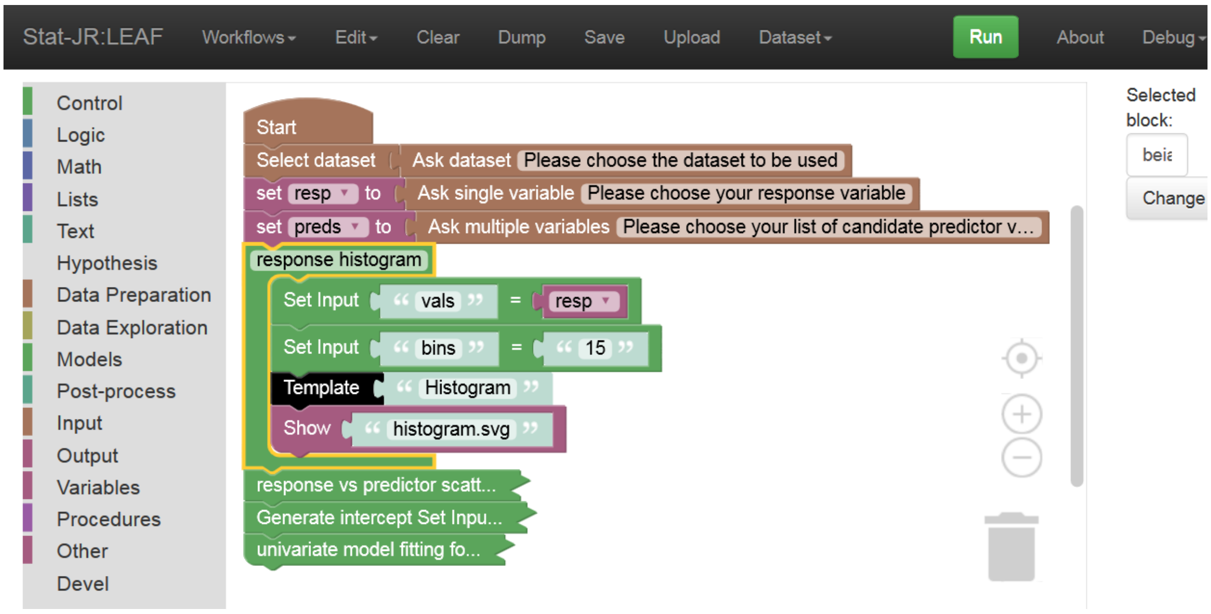
We will finish this section by introducing a couple of templates that have some built-in interrogation of their outputs. Firstly we will replace the *Histogram* template that we used towards the start of our workflow with a template by the name of *HistSkew* which gives textual feedback about the shape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

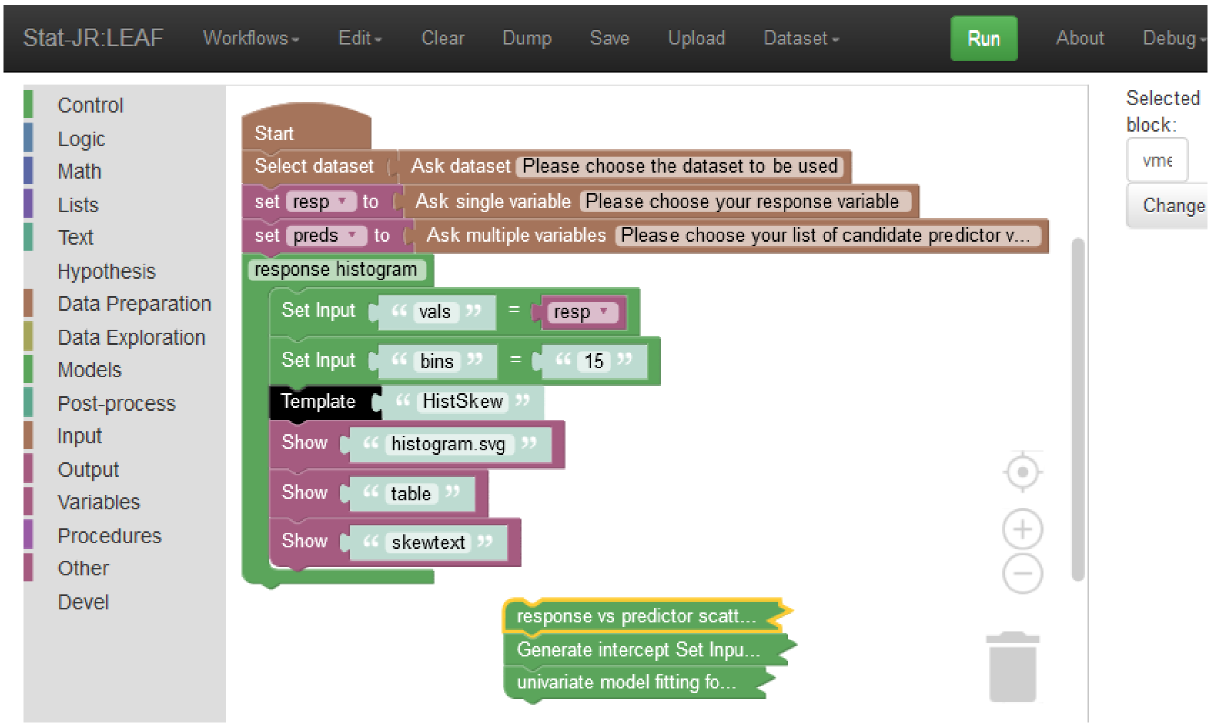
[14] Obviously this is quite a crude way of evaluating evidence for an effect (one would typically wish to look beyond simply considering whether a test statistic has an associated p-value below 0.05 or not), but this example nevertheless illustrates the functionality of the *if-do-else* block, using it to return conditional textual output.



Start
Select dataset Ask dataset Please choose the dataset to be used
set resp to Ask single variable Please choose your response variable
set preds to Ask multiple variables Please choose your list of candidate predictor v...
response histogram
Set Input " vals " = resp
Set Input " bins " = " 15 "
Template " Histogram "
Show " histogram.svg "
response vs predictor scatt...
Generate intercept Set Inpu...
univariate model fitting fo...

Figure 80

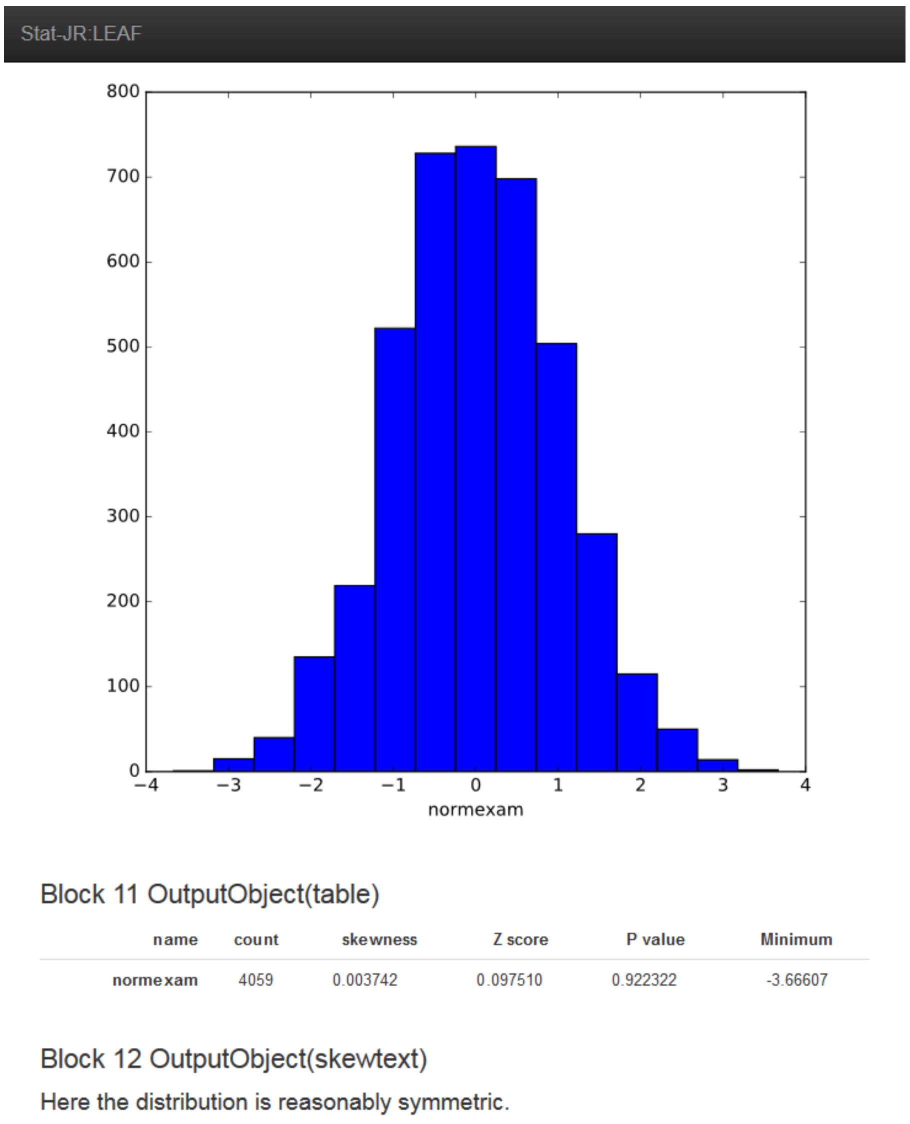
The *HistSkew* template actually has the same inputs as the *Histogram* template, so we only need to change the name in the *Template* block and then add the additional outputs as follows:



Start
Select dataset Ask dataset Please choose the dataset to be used
set resp to Ask single variable Please choose your response variable
set preds to Ask multiple variables Please choose your list of candidate predictor v...
response histogram
Set Input " vals " = resp
Set Input " bins " = " 15 "
Template " HistSkew "
Show " histogram.svg "
Show " table "
Show " skewtext "
response vs predictor scatt...
Generate intercept Set Inpu...
univariate model fitting fo...

Figure 81

Of course, we could have discovered the names of the outputs we required by running the *Histskew* template in TREE or LEAF, 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:



Block 11 OutputObject(table)

name	count	skewness	Z score	P value	Minimum
normexam	4059	0.003742	0.097510	0.922322	-3.66607

Block 12 OutputObject(skewtext)

Here the distribution is reasonably symmetric.

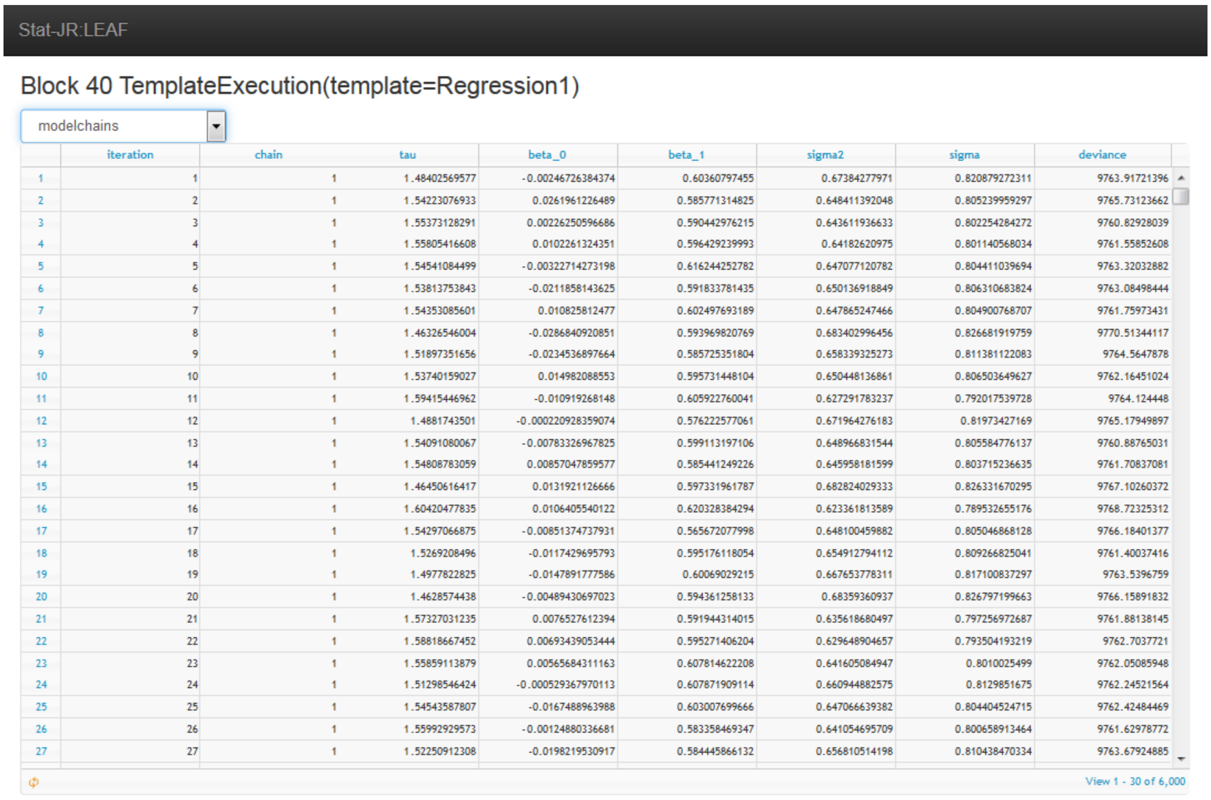
Figure 82

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).

Attach the latter part of the workflow back onto the former part of it, and save it as *section2_08.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:


Stat-JR:LEAF

Block 40 TemplateExecution(template=Regression1)

modelchains

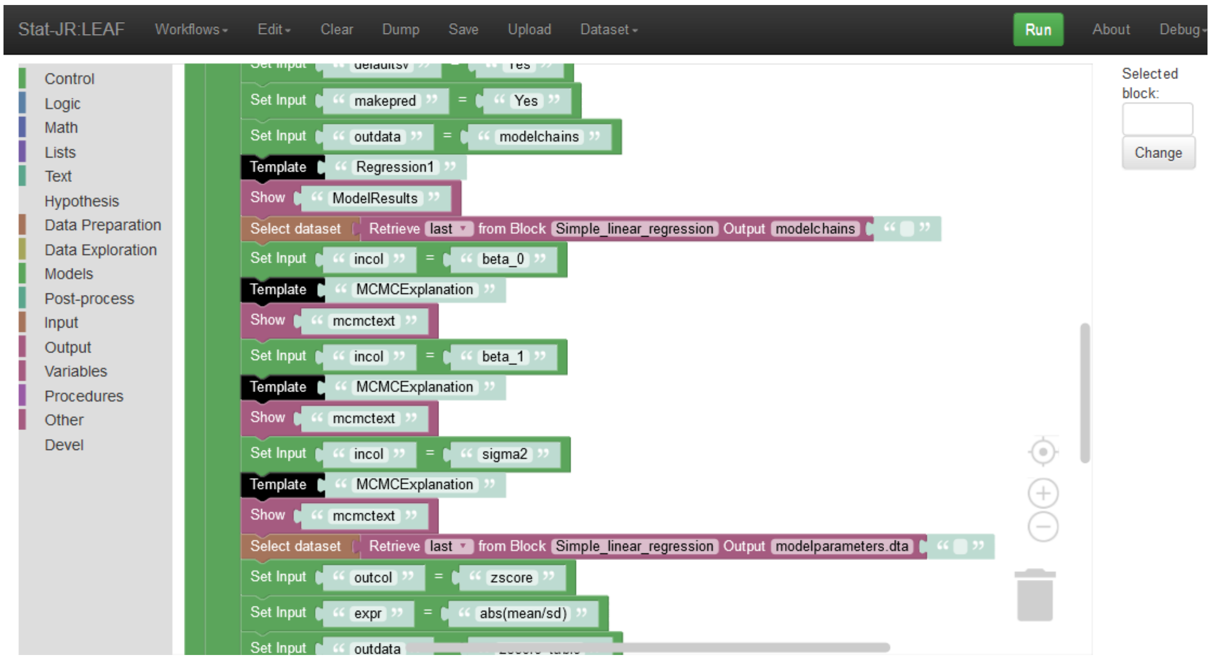
	iteration	chain	tau	beta_0	beta_1	sigma2	sigma	deviance
1	1	1	1.48402569577	-0.00246726384374	0.60360797455	0.67384277971	0.820879272311	9763.91721396
2	2	1	1.54223076933	0.0261961226489	0.585771314825	0.648411392048	0.805239959297	9765.73123662
3	3	1	1.55373128291	0.00226250596686	0.590442976215	0.643611936633	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
17	17	1	1.54297066875	-0.00851374737931	0.565672077998	0.648100459882	0.805046868128	9766.18401377
18	18	1	1.5269208496	-0.0117429695793	0.595176118054	0.654912794112	0.809266825041	9761.40037416
19	19	1	1.4977822825	-0.0147891777586	0.60069029215	0.667653778311	0.817100837297	9763.5396759
20	20	1	1.4628574438	-0.00489430697023	0.594361258133	0.68359360937	0.826797199663	9766.15891832
21	21	1	1.57327031235	0.0076527612394	0.591944314015	0.635618680497	0.797256972687	9761.88138145
22	22	1	1.58818667452	0.00693439053444	0.595271406204	0.629648904657	0.793504193219	9762.7037721
23	23	1	1.55859113879	0.00565684311163	0.607814622208	0.641605084947	0.8010025499	9762.05085948
24	24	1	1.51298546424	-0.000529367970113	0.607871909114	0.660944882575	0.8129851675	9762.24521564
25	25	1	1.54543587807	-0.0167488963988	0.603007699666	0.647066639382	0.804404524715	9762.42484469
26	26	1	1.55992929573	-0.00124880336681	0.583358469347	0.641054695709	0.800658913464	9761.62978772
27	27	1	1.52250912308	-0.0198219530917	0.584445866132	0.656810514198	0.810438470334	9763.67924885

View 1 - 30 of 6,000

Figure 83

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*:

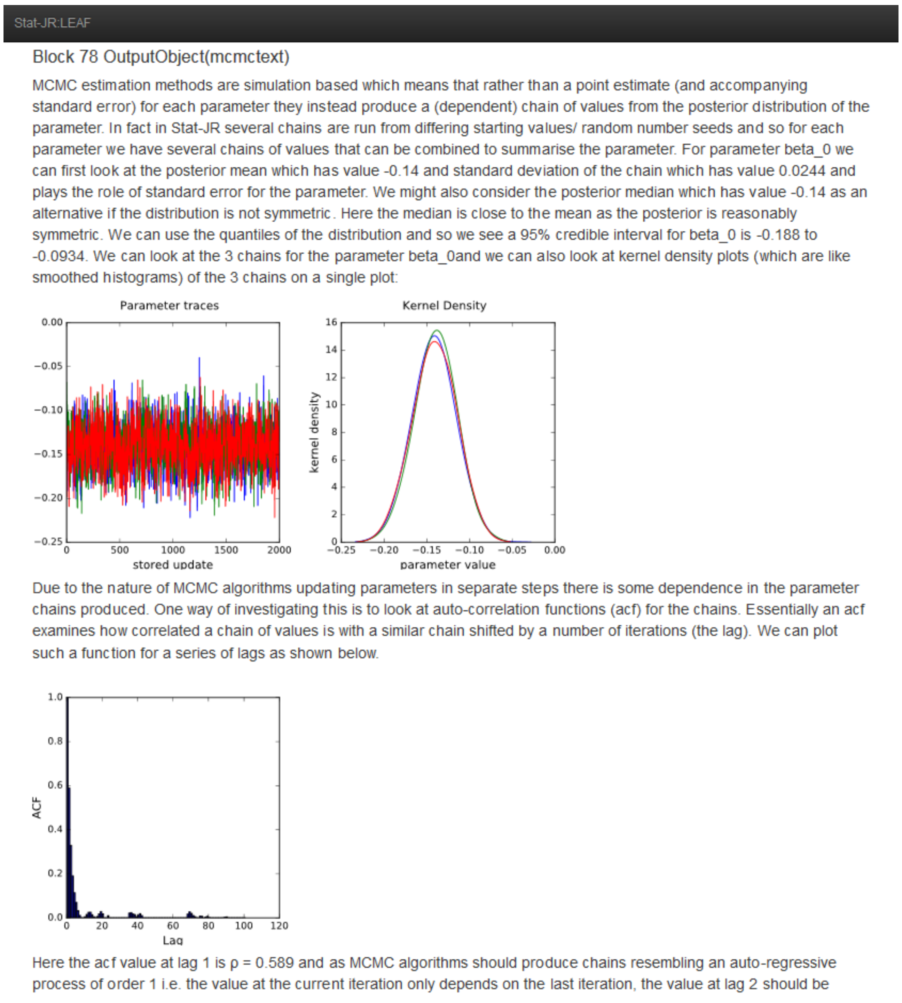


```
Set Input "defaultsv" = "Yes"
Set Input "makepred" = "Yes"
Set Input "outdata" = "modelchains"
Template "Regression1"
Show "ModelResults"
Select dataset Retrieve last from Block Simple_linear_regression Output modelchains " "
Set Input "incol" = "beta_0"
Template "MCMCExplanation"
Show "mcmctext"
Set Input "incol" = "beta_1"
Template "MCMCExplanation"
Show "mcmctext"
Set Input "incol" = "sigma2"
Template "MCMCExplanation"
Show "mcmctext"
Select dataset Retrieve last from Block Simple_linear_regression Output modelparameters.dta " "
Set Input "outcol" = "zscore"
Set Input "expr" = "abs(mean/sd)"
Set Input "outdata"
```

Figure 84

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 section, you may like to try converting this to a loop). Save this workflow as *section2_09.xml*.

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


Stat-JR:LEAF

Block 78 OutputObject(mcmctext)

MCMC estimation methods are simulation based which means that rather than a point estimate (and accompanying standard error) for each parameter they instead produce a (dependent) chain of values from the posterior distribution of the parameter. In fact in Stat-JR several chains are run from differing starting values/ random number seeds and so for each parameter we have several chains of values that can be combined to summarise the parameter. For parameter beta_0 we can first look at the posterior mean which has value -0.14 and standard deviation of the chain which has value 0.0244 and plays the role of standard error for the parameter. We might also consider the posterior median which has value -0.14 as an alternative if the distribution is not symmetric. Here the median is close to the mean as the posterior is reasonably symmetric. We can use the quantiles of the distribution and so we see a 95% credible interval for beta_0 is -0.188 to -0.0934. We can look at the 3 chains for the parameter beta_0and we can also look at kernel density plots (which are like smoothed histograms) of the 3 chains on a single plot:

Due to the nature of MCMC algorithms updating parameters in separate steps there is some dependence in the parameter chains produced. One way of investigating this is to look at auto-correlation functions (acf) for the chains. Essentially an acf examines how correlated a chain of values is with a similar chain shifted by a number of iterations (the lag). We can plot such a function for a series of lags as shown below.

Here the acf value at lag 1 is ρ = 0.589 and as MCMC algorithms should produce chains resembling an auto-regressive process of order 1 i.e. the value at the current iteration only depends on the last iteration, the value at lag 2 should be

Figure 85

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 since 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 What have we covered?

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

Section 3 Making workflows to support the LEMMA training materials

3.1 Overview

This section is somewhat different to the previous sections in that all the workflows have been written for you and the idea is simply that you follow them and learn some further features of the workflow system as you go. We will be using workflows that aim to produce the equivalent analyses presented in the MLwiN practical for Module 3 of the LEMMA training materials (written by Fiona Steele). It is therefore best to have a copy of the practical with you as you run through the workflow (the website supporting the online LEMMA training materials can be found here: http://www.bristol.ac.uk/cmm/learning/online-course/index.html).

3.2 Introducing procedures

To begin you will need to start up Stat-JR:LEAF, or return to the workflow screen if it's already open, and press **Clear**. The screen will look as follows:

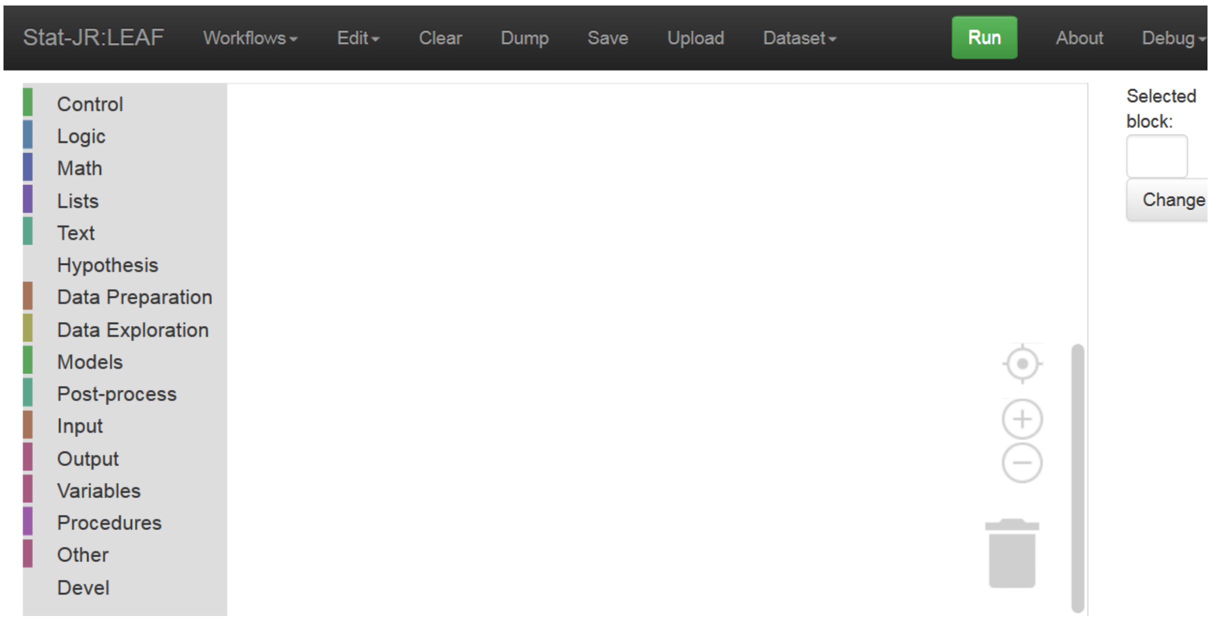


Figure 86

The workflows we will use can be opened from **Workflows > LEAF_Guide** (via the black bar at the top of the LEAF interface); there's one for each of the five sections of the LEMMA Module 3 practical, and you will find them saved as *lemma3_1* … *lemma3_5.* Let's open the first of these, *lemma3_1*:

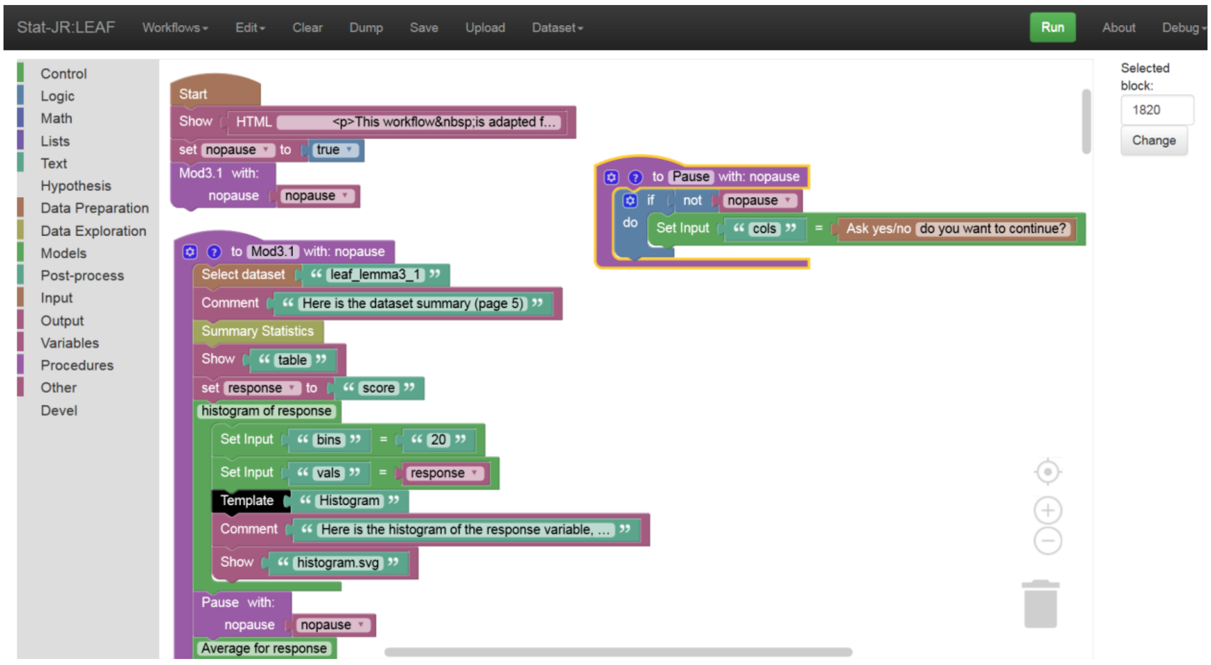


Figure 87

This looks somewhat different to the workflows you have seen previously. There appears to be a rather short workflow contiguous to the *Start* block consisting of a *Show* block, one *set <variable>* block and a purple block that we haven't yet come across. There are then some other blocks elsewhere in the workspace, which do not appear to be connected to the main workflow.

The purple blocks relate to *procedure*s; these are available from the **Procedures** menu which, if you click on it, looks as follows:

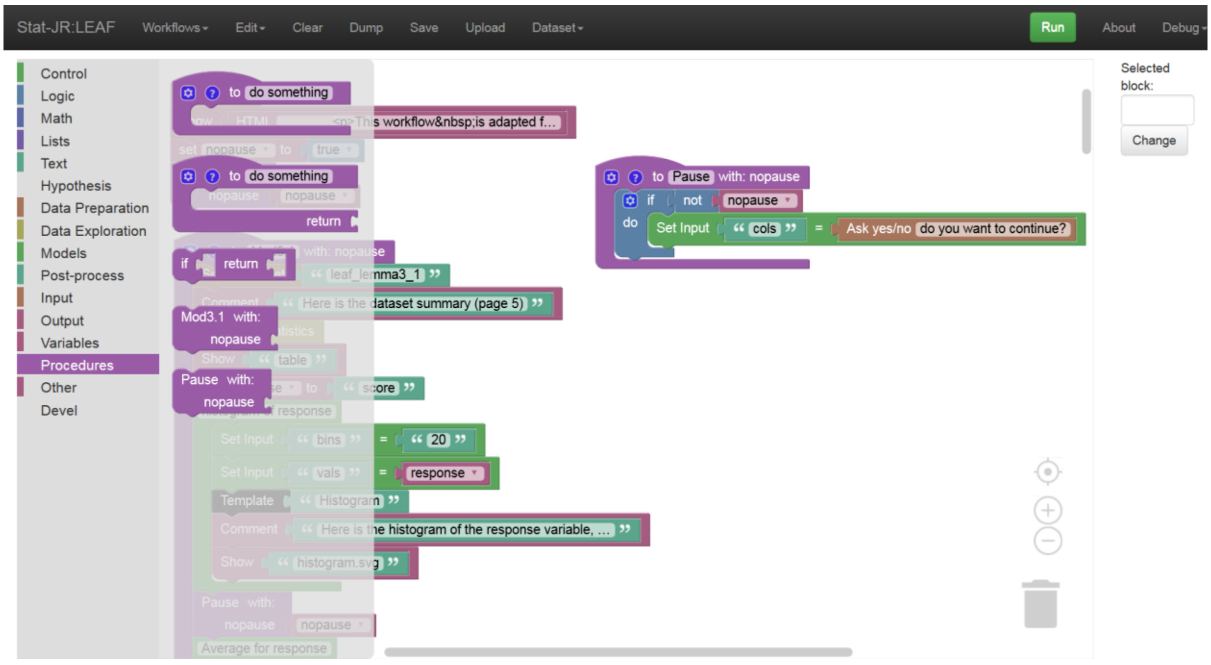


Figure 88

The top three blocks are always found in this list, whilst the others appeared in the list as we added *procedure* blocks when building our workflow – we describe this further below. We used the top block to create the procedures in this workflow. It looks similar to the *grouping* block we used in

Section 2 – we can change the name away from "do something" and place some blocks within the 'mouth' of the *procedure* block which will be executed when the workflow reaches it – but one important difference is that it's call-able: we've been able to place it away from the main workflow because we can call it from there.

To demonstrate, if we bring another of these onto the central workflow pane, and give it a name in place of the default *"do something"* (we've chosen *"run a model"* in this example), another purple block (with *"run a model"* written on it) has now appeared at the bottom of the **Procedures** list:

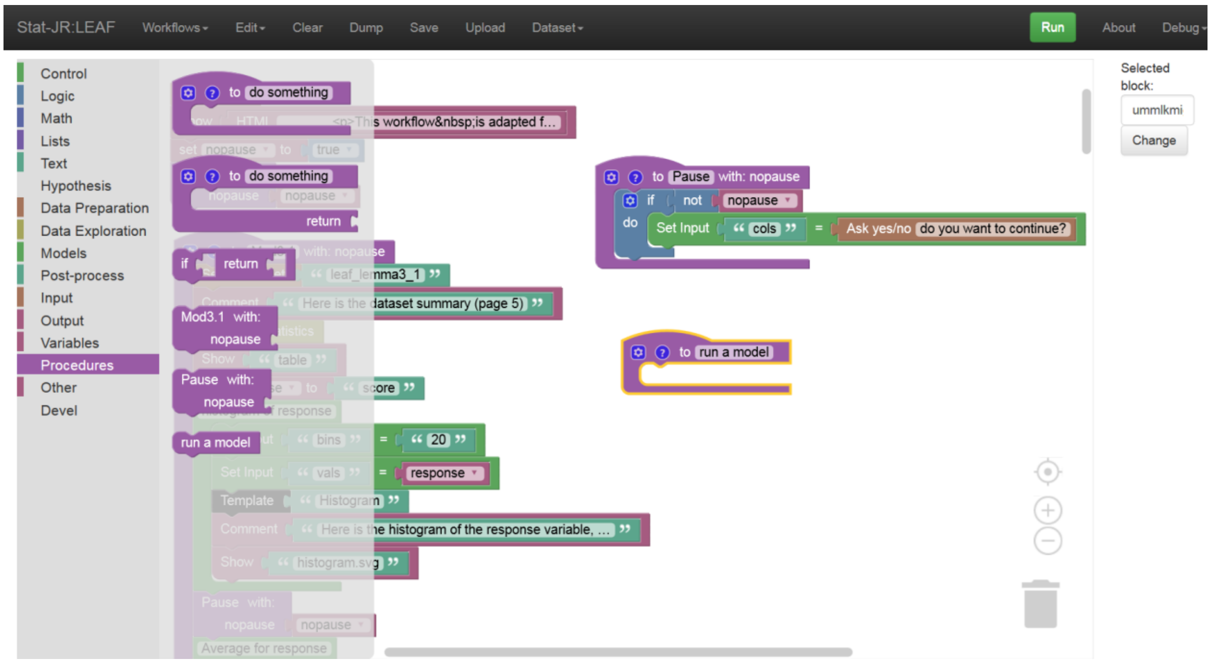


Figure 89

If we next click on the blue button on the procedure block we have just introduced, we can modify the procedure block, requesting that it accept a named input when called – here, in keeping with a number of the other procedures we produced when originally writing this workflow, we've called the named input *"nopause":*

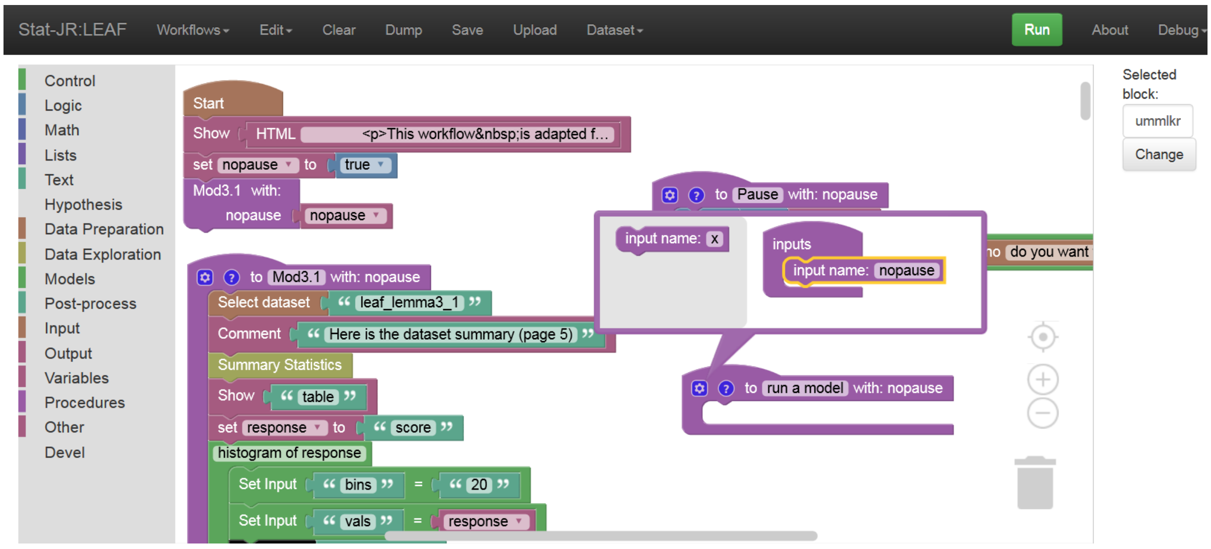


Figure 90

Now if we look again at the list of blocks under the **Procedures** list we see the block at the bottom has been modified, and now looks just like the ones immediately above it:

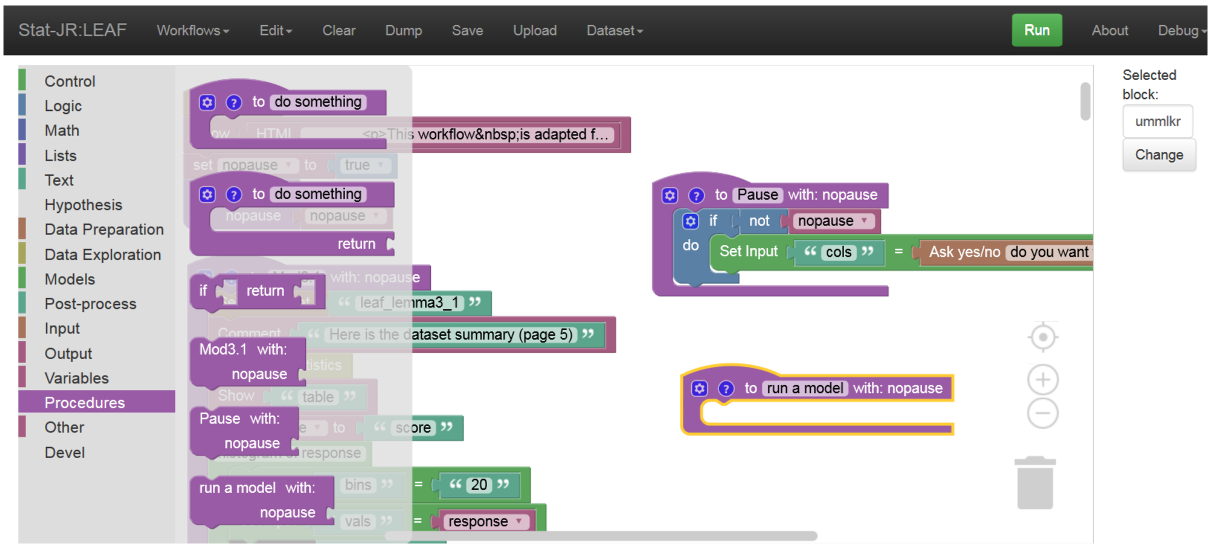


Figure 91

This new block (the one which has appeared at the bottom of the left-hand list) is the block we would use if we wished to call the procedure we've just defined (although we've not defined it fully: we haven't added any blocks in the procedure itself in this simple example). Since we've modified it such that it takes an input (called *"nopause"*), we can use this input to control internal aspects of the procedure.

Let's bin the procedure we've just made (you'll notice the **Procedures** list is modified appropriately) and turn our attention back to the original workflow. It has two procedures defined: one named *Pause* and another named *Mod3.1*; this latter procedure carries out all the instructions in section 1 of the LEMMA Module 3 practical. The procedure called *Pause* is a lot shorter; like the dummy procedure we produced for illustration a moment ago, it takes an input called *nopause*. Looking inside the *Pause* procedure we can see that it evaluates this input via an *if-do* block (as used in Section 2). The use of *not* when evaluating the *nopause* item means that if *nopause* is 'false' then it will set an input called *"cols"* (the name we've given it here is incidental) to be a Boolean yes/no question asking the user whether they wish to continue or not.

The calls to the *Pause* procedure are within the large *Mod3.1* procedure, whilst the *nopause* input it uses is defined in the main workflow, just three blocks below the *Start* block. It's currently set to 'true', which means that the whole workflow would run without pausing when you click **Run**; if it were instead set to 'false' the user would be prompted with the question "do you want to continue?" whenever the *pause* procedure was called.

So here a procedure is being used not for data analytical purposes, but to simply modify the interface for the user; *cf.* the other procedure, *Mod3.1*, which *is* used for data analytical purposes.

Below we run through each of the five workflows (*lemma3_1* … *lemma3_5*) in turn.

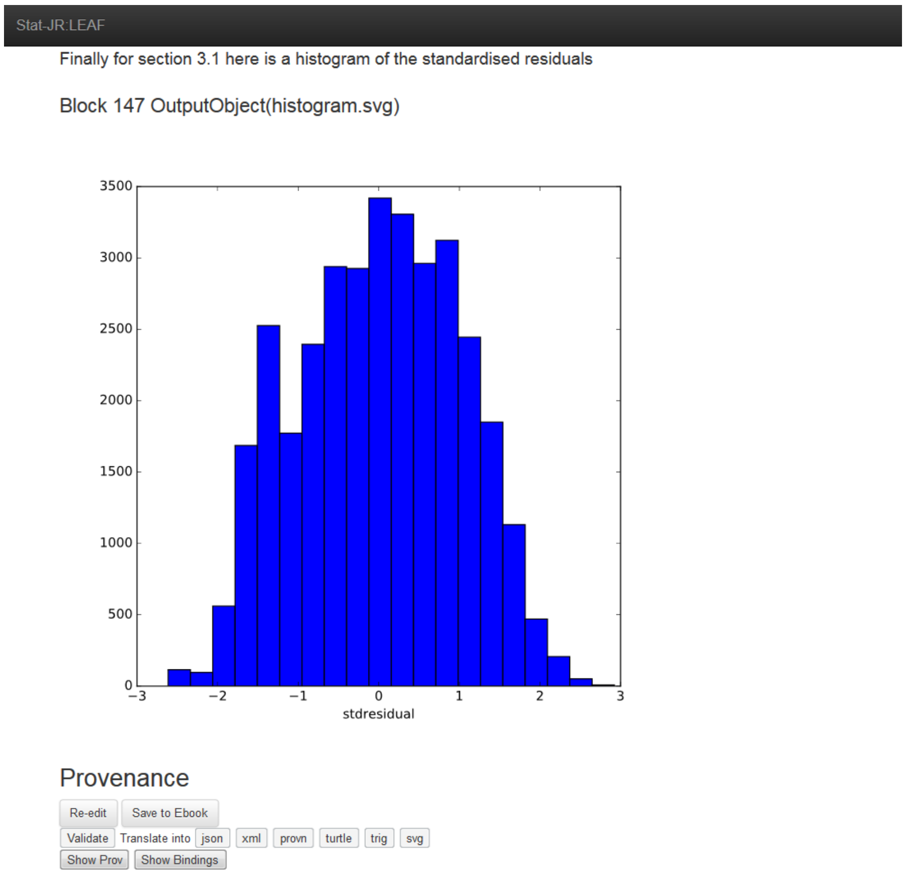
3.3 LEMMA P3.1: Regression with a single continuous explanatory variable

So let's now look at the first section of LEMMA Module 3.

If you press **Run**, the workflow will execute; this will take some time (you may see a flurry of activity in the command line window running in the background), so whilst its running we can look at the workflow blocks themselves.

You will see that the procedure consists of green *grouping* blocks identifying what each section of code does, with occasional *Comment* blocks. These *Comment* blocks link the workflow to the page numbers in the LEMMA documentation, and the text in the *Comment* blocks will be printed out in the output. The workflow is also punctuated by calls to the *Pause* procedure which, as described above, if set to 'false' would pause the workflow and present the user with a prompt asking whether they would like it to continue or not. Otherwise, the functional structure of the workflow is much like that encountered in the last two sections: *Set Input* blocks specifying the inputs for subsequent *Template* executions, and *Show* blocks displaying certain outputs from those executions.

Once it's run to completion you will see the following output if you scroll down:


Stat-JR:LEAF

Finally for section 3.1 here is a histogram of the standardised residuals

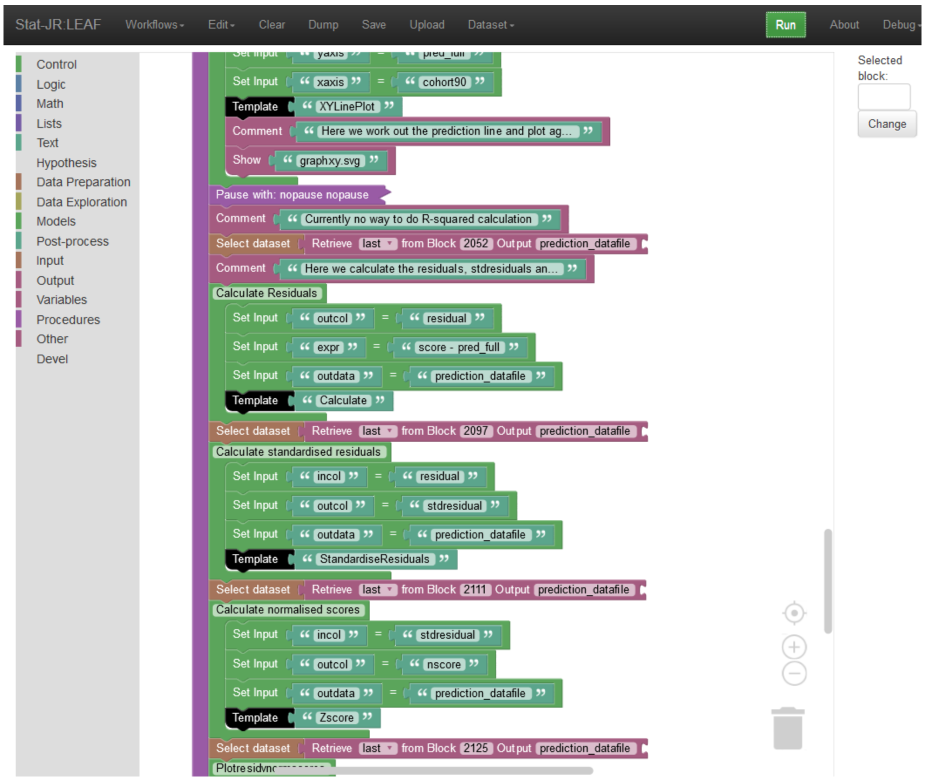
Block 147 OutputObject(histogram.svg)

Figure 92

Here is the histogram from page 24 of the MLwiN practical. If you have two screens (or two windows) you can have the workflow-code window and the workflow-results window up together to see their correspondence.

In this first section, whilst a number of the blocks are familiar to us from the first two sections, some of templates aren't. We make use of the *Tabulate* template that can produce quite a wide range of summary statistics in tabular form and is designed to mimic the MLwiN tabulate window. We also use a *RecodeValues* template for recoding the values of a categorical variable, again mimicking the MLwiN window for recoding values.

Scrolling down the workflow reveals some other new templates we have used:



```
    Set Input "yaxis" = "pred_full"
    Set Input "xaxis" = "cohort90"
    Template "XYLinePlot"
    Comment "Here we work out the prediction line and plot ag..."
    Show "graphxy.svg"
Pause with: nopause nopause
Comment "Currently no way to do R-squared calculation"
Select dataset Retrieve last from Block 2052 Output prediction_datafile
Comment "Here we calculate the residuals, stdresiduals an..."
Calculate Residuals
    Set Input "outcol" = "residual"
    Set Input "expr" = "score - pred_full"
    Set Input "outdata" = "prediction_datafile"
    Template "Calculate"
Select dataset Retrieve last from Block 2097 Output prediction_datafile
Calculate standardised residuals
    Set Input "incol" = "residual"
    Set Input "outcol" = "stdresidual"
    Set Input "outdata" = "prediction_datafile"
    Template "StandardiseResiduals"
Select dataset Retrieve last from Block 2111 Output prediction_datafile
Calculate normalised scores
    Set Input "incol" = "stdresidual"
    Set Input "outcol" = "nscore"
    Set Input "outdata" = "prediction_datafile"
    Template "Zscore"
Select dataset Retrieve last from Block 2125 Output prediction_datafile
Plotresidvn
```

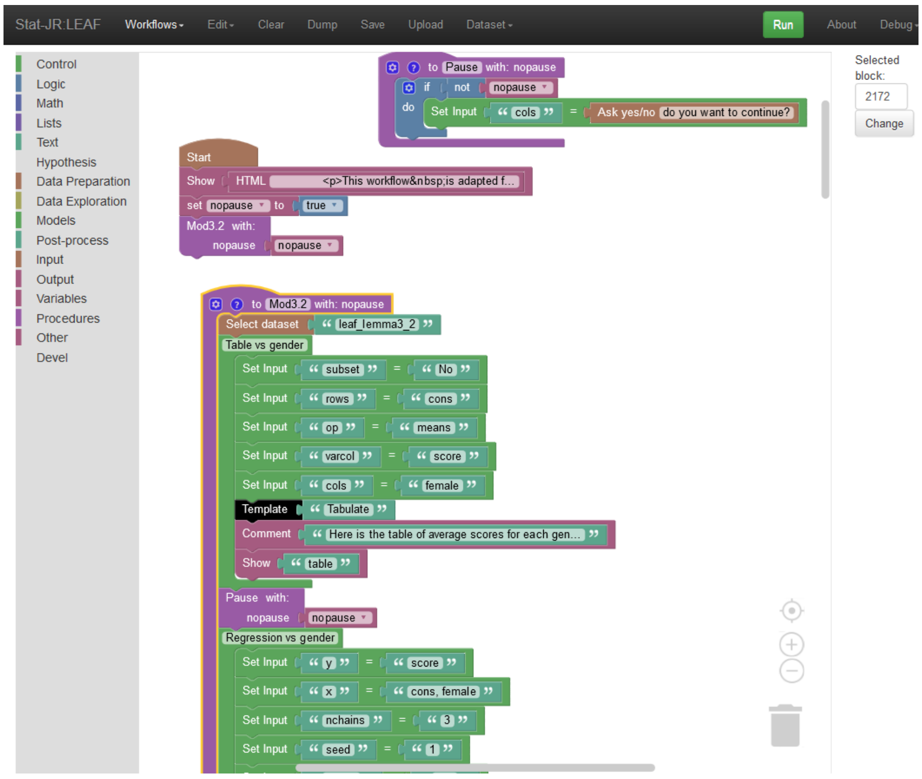
Figure 93

Here we use the *XYLinePlot* template which is simply a variant of the *XYPlot* that plots lines rather than points. Then, having switched dataset to the *prediction_datafile* generated from the model fit, we use in quick succession: *Calculate* (which you have encountered before) to create residuals from responses and fitted values; *Standardise*, to create standardised residuals from raw residuals and their standard errors; and *Zscore*, to create normalised scores from the (standardised) residuals. Each of these templates adds a column to a dataset, each of which we save using the same name, ensuring we are using the correct (latest) version by appending the *Retrieve* block onto a *Select dataset* block.

As an exercise you might take your own dataset and see if you can, by choosing one response and one predictor, replicate this exploratory analysis on your dataset: which aspects of the workflow would you need to modify to accommodate your own dataset, and how?

3.4 LEMMA P3.2: Comparing groups: regression with a single categorical explanatory variable

We can now look at the next workflow, *lemma3_2* (accessible via **Workflows > LEAF_Guide**):



```
to Pause with: nopause
    if  not nopause
    do  Set Input "cols" = Ask yes/no do you want to continue?

Start
Show HTML <p>This workflow&nbsp;is adapted f...
set nopause to true
Mod3.2 with:
    nopause nopause

to Mod3.2 with: nopause
    Select dataset "leaf_lemma3_2"
    Table vs gender
        Set Input "subset" = "No"
        Set Input "rows" = "cons"
        Set Input "op" = "means"
        Set Input "varcol" = "score"
        Set Input "cols" = "female"
        Template "Tabulate"
        Comment "Here is the table of average scores for each gen..."
        Show "table"
    Pause with:
        nopause nopause
    Regression vs gender
        Set Input "y" = "score"
        Set Input "x" = "cons, female"
        Set Input "nchains" = "3"
        Set Input "seed" = "1"
```

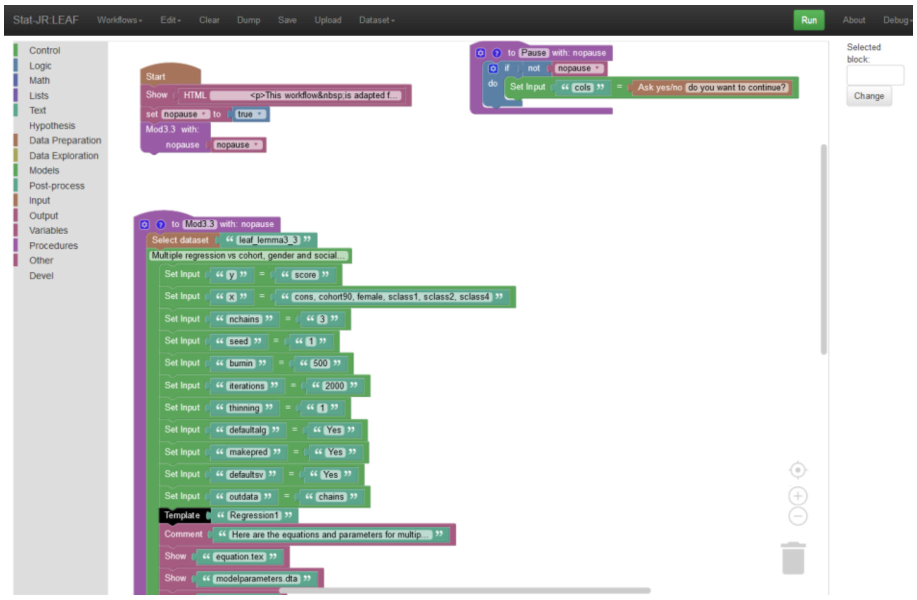
Figure 94

Section 3.2 of the LEMMA MLwiN practical covers some basic modelling of categorical predictor variables. The workflow here consists of some tabulations of the response variable (*score*) for different categorical variables using the *Tabulate* template, several calls to the *Regression1* template for model fitting and much use of the *Calculate* template to create dummy variables for the different categories of social class and the different cohorts, primarily because the models fitted have differing base categories.

There are no really new templates or blocks here so we suggest you simply **Run** the section and cross-reference it with the relevant LEMMA materials. If you have your own data and it contains categorical predictors you might like to adapt the code to your dataset.

3.5 LEMMA P3.3: Regression with more than one explanatory variable (multiple regression)

We can now look at the next workflow, *lemma3_3*:



```
Start
Show  HTML  <p>This workflow&nbsp;is adapted f...
set nopause to true
Mod3.3 with:
    nopause  nopause

to Pause with: nopause
  if  not  nopause
  do  Set Input  " cols "  =  Ask yes/no  do you want to continue?

to Mod3.3 with: nopause
  Select dataset  " leaf_lemma3_3 "
  Multiple regression vs cohort, gender and social...
    Set Input  " y "  =  " score "
    Set Input  " x "  =  " cons, cohort90, female, sclass1, sclass2, sclass4 "
    Set Input  " nchains "  =  " 3 "
    Set Input  " seed "  =  " 1 "
    Set Input  " burnin "  =  " 500 "
    Set Input  " iterations "  =  " 2000 "
    Set Input  " thinning "  =  " 1 "
    Set Input  " defaultalg "  =  " Yes "
    Set Input  " makepred "  =  " Yes "
    Set Input  " defaultsv "  =  " Yes "
    Set Input  " outdata "  =  " chains "
    Template  " Regression1 "
    Comment  " Here are the equations and parameters for multip... "
    Show  " equation.tex "
    Show  " modelparameters.dta "
```

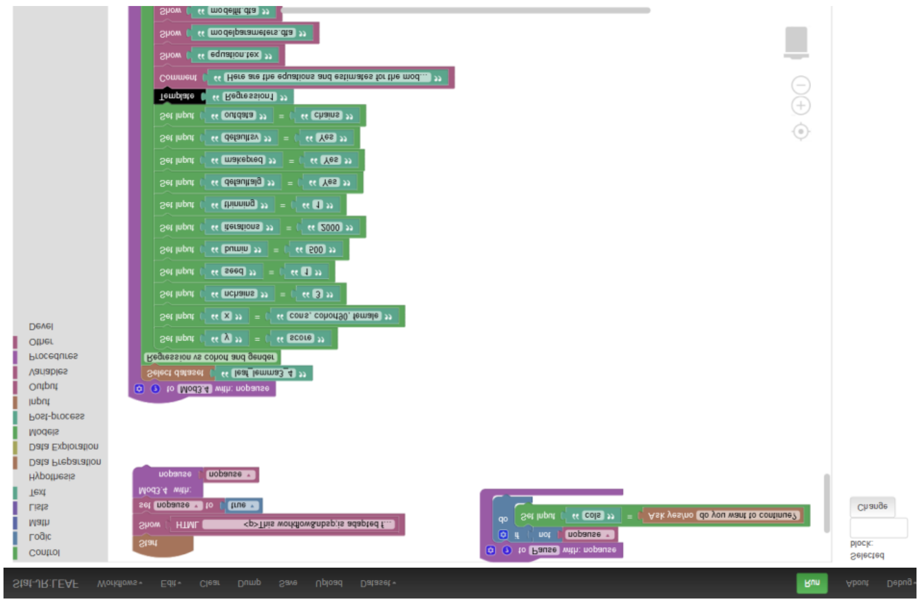
Figure 95

This workflow is actually quite short as this section of the LEMMA 3 MLwiN practical simply introduces the concept of multiple regression by placing the three predictor variables, investigated separately up to that point, into the same model. It does this using the *Regression1* template, with which we are familiar. It then also displays a tabulation of two categorical variables to show how social class has changed over time.

Again we suggest you **Run** the code and investigate whether it replicates the LEMMA materials, and if it relates to your own dataset try modifying it accordingly before moving onto the next section.

3.6 LEMMA P3.4: Interaction effects

This section is quite a long one in the training materials. Here is workflow *lemma3_4*:

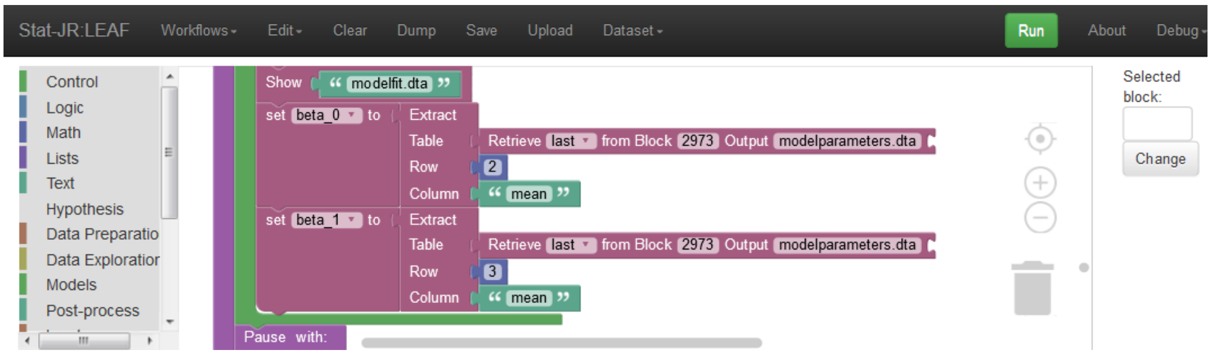


```
Start
Show  HTML  <p>This workflow&nbsp;is adapted f...
set nopause to true
Mod3.4 with:
    nopause  nopause

to Pause with: nopause
  if  not  nopause
  do  Set Input  " cols "  =  Ask yes/no  do you want to continue?

to Mod3.4 with: nopause
  Select dataset  " leaf_lemma3_4 "
  Regression vs cohort and gender
    Set Input  " y "  =  " score "
    Set Input  " x "  =  " cons, cohort90, female "
    Set Input  " nchains "  =  " 3 "
    Set Input  " seed "  =  " 1 "
    Set Input  " burnin "  =  " 500 "
    Set Input  " iterations "  =  " 2000 "
    Set Input  " thinning "  =  " 1 "
    Set Input  " defaultalg "  =  " Yes "
    Set Input  " makepred "  =  " Yes "
    Set Input  " defaultsv "  =  " Yes "
    Set Input  " outdata "  =  " chains "
    Template  " Regression1 "
    Comment  " Here are the equations and estimates for the mod... "
    Show  " equation.tex "
    Show  " modelparameters.dta "
    Show  " modelfit.dta "
```

Figure 96

In this section of the LEMMA materials the concept of interactions is introduced, and several regressions are fitted. To start with a multiple regression of cohort and gender on the hedonism score is fitted. The resulting model fit is plotted using a template we haven't come across previously, *XYGroupPlotLine,* which plots separate lines for each group. Another new template, *Choose*, is then used to select subsets of the data, firstly all boys and secondly all girls, and separate regressions of the hedonism score on cohort are performed for each subset. To illustrate interactions, the *Calculate* template is used to create the interaction term and a model including it is fitted. The *XYGroupPlotLine* template is used again, plotting separate lines that are not parallel for the two genders.

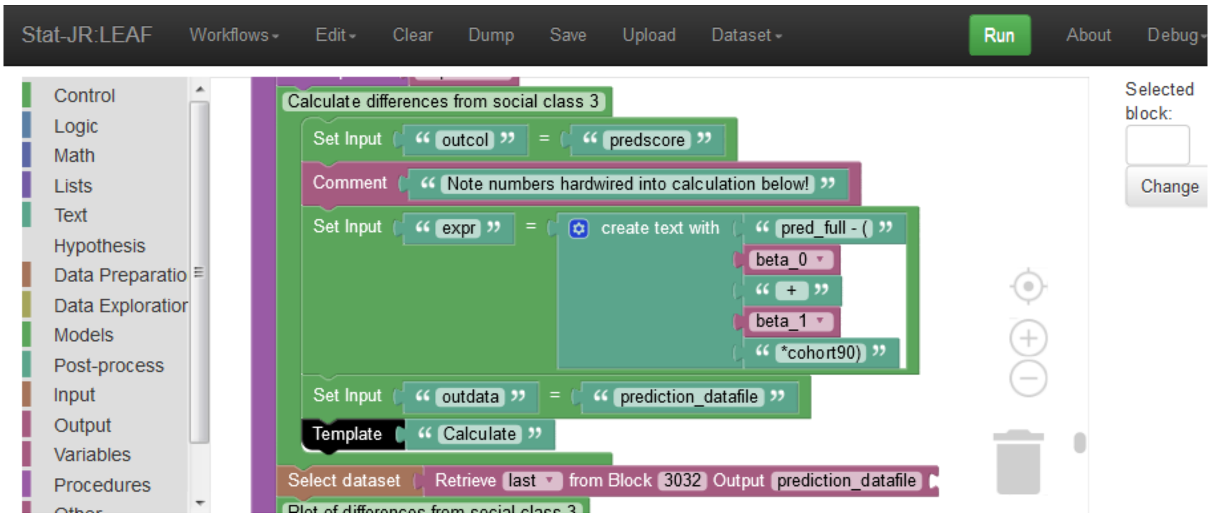
Attention then moves from gender to social class, which has more categories. The *Calculate* template is used to create interactions before the *Regression1* template is used to fit a model including these interactions. The fit of the model is illustrated in two ways using the *XYGroupPlotLine* template. Firstly a straightforward predicted line plot with a line for each social class and then a plot of the differences for each social class from a base category. Here the workflow illustrates how to extract values from a table of results thus:



```
Show "modelfit.dta"
set beta_0 to Extract
    Table   Retrieve last from Block 2973 Output modelparameters.dta
    Row     2
    Column  "mean"
set beta_1 to Extract
    Table   Retrieve last from Block 2973 Output modelparameters.dta
    Row     3
    Column  "mean"
Pause with:
```

Figure 97

This takes the values of the means of *beta_0* and *beta_1* from the model fit, which are then used to create the differences (stored as *predscore*) from the base category (social class 3) as shown below:



```
Calculate differences from social class 3
    Set Input "outcol" = "predscore"
    Comment "Note numbers hardwired into calculation below!"
    Set Input "expr" = create text with "pred_full - ("
                                        beta_0
                                        "+"
                                        beta_1
                                        "*cohort90)"
    Set Input "outdata" = "prediction_datafile"
    Template "Calculate"
Select dataset Retrieve last from Block 3032 Output prediction_datafile
Plot of differences from social class 3
```

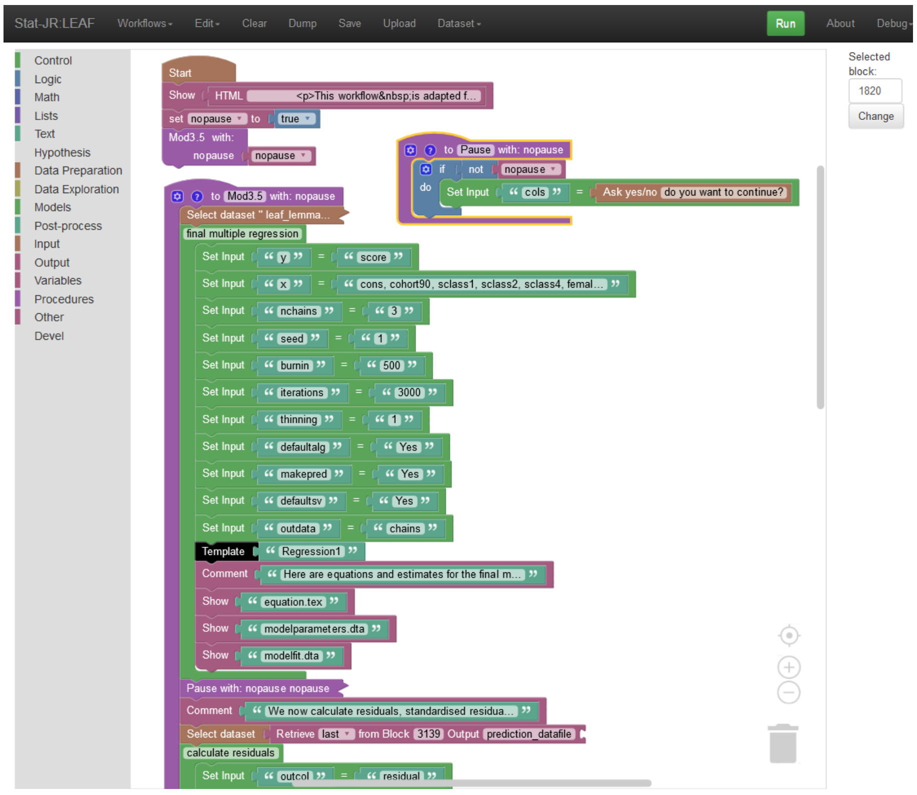
Figure 98

Finally the plot is constructed before a final model without interactions is constructed for comparison purposes.

Again we suggest you try running the section to satisfy yourself you understand the code and see how it replicates the section in the LEMMA materials. If you have suitable data you might consider modifying the code to use your own dataset.

3.7 LEMMA P3.5: Checking model assumptions in multiple regression

In this final short workflow (*lemma3_5*), the various predictor variables are brought together in one final model. This model isn't so easy to show graphically so instead this section focusses on checking the fit of the model.



```
Start
Show HTML <p>This workflow&nbsp;is adapted f...
set nopause to true
Mod3.5 with:
    nopause nopause

to Pause with: nopause
    if not nopause
    do Set Input “cols” = Ask yes/no do you want to continue?

to Mod3.5 with: nopause
    Select dataset " leaf_lemma...
    final multiple regression
        Set Input “y” = “score”
        Set Input “x” = “cons, cohort90, sclass1, sclass2, sclass4, femal...”
        Set Input “nchains” = “3”
        Set Input “seed” = “1”
        Set Input “burnin” = “500”
        Set Input “iterations” = “3000”
        Set Input “thinning” = “1”
        Set Input “defaultalg” = “Yes”
        Set Input “makepred” = “Yes”
        Set Input “defaultsv” = “Yes”
        Set Input “outdata” = “chains”
        Template “Regression1”
        Comment “Here are equations and estimates for the final m...”
        Show “equation.tex”
        Show “modelparameters.dta”
        Show “modelfit.dta”
    Pause with: nopause nopause
    Comment “We now calculate residuals, standardised residua...”
    Select dataset Retrieve last from Block 3139 Output prediction_datafile
    calculate residuals
        Set Input “outcol” = “residual”
```

Figure 99

In practice this section very closely resembles the first section only with a more complex model, i.e. all the calculations and plots are ones we have seen before. We suggest you **Run** the section to replicate the LEMMA materials and then consider what you would do with your own dataset.

3.8 What have we covered?

In this section we have demonstrated the use of *procedure* blocks to group sections of workflow together which can then be called from elsewhere in the workflow. We have also encountered a number of new templates, and have more generally demonstrated how we might use the tools of Stat-JR's workflow system to replicate the outputs found in the LEMMA training materials. Given Stat-JR's ability to interoperate with a wide variety of third-party statistical software packages (R, MLwiN, Stata, etc.) this workflow could eventually be modified to allow the user to toggle between packages. In doing so it could expose the scripts (R scripts, *.do* files, etc.) used to run each execution, so that the user can cross-reference the script and outputs from those packages, and gain insight into how the same operation might be achieved by a number of different packages.

At this point you should have the tools to try out other things. For example, you may like to consider fitting other models to your own dataset, perhaps even using other model-fitting templates (e.g. *1LevelMod*, *2LevelMod*) which you can always test first using TREE.

Section 4 Translating a workflow into an eBook

In this section we will create a workflow within LEAF and explore exporting it to Stat-JR's eBook-reading interface, DEEP.

To start things off we will load up the workflow system which will give us the usual window as shown:

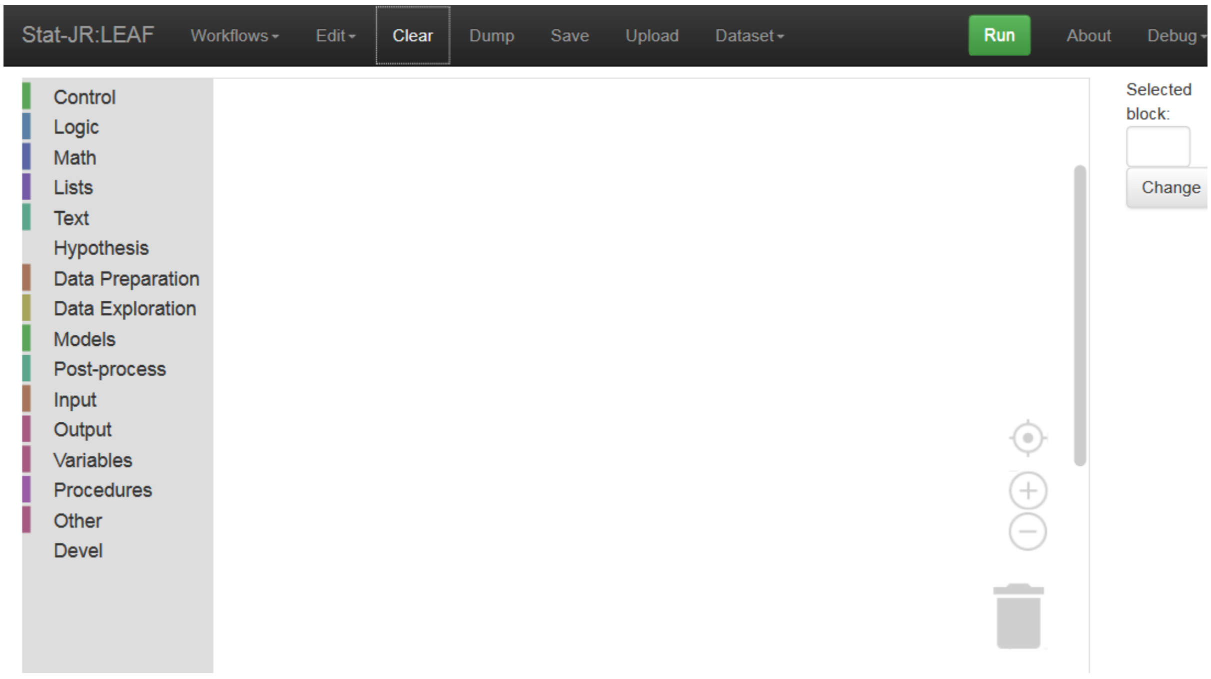
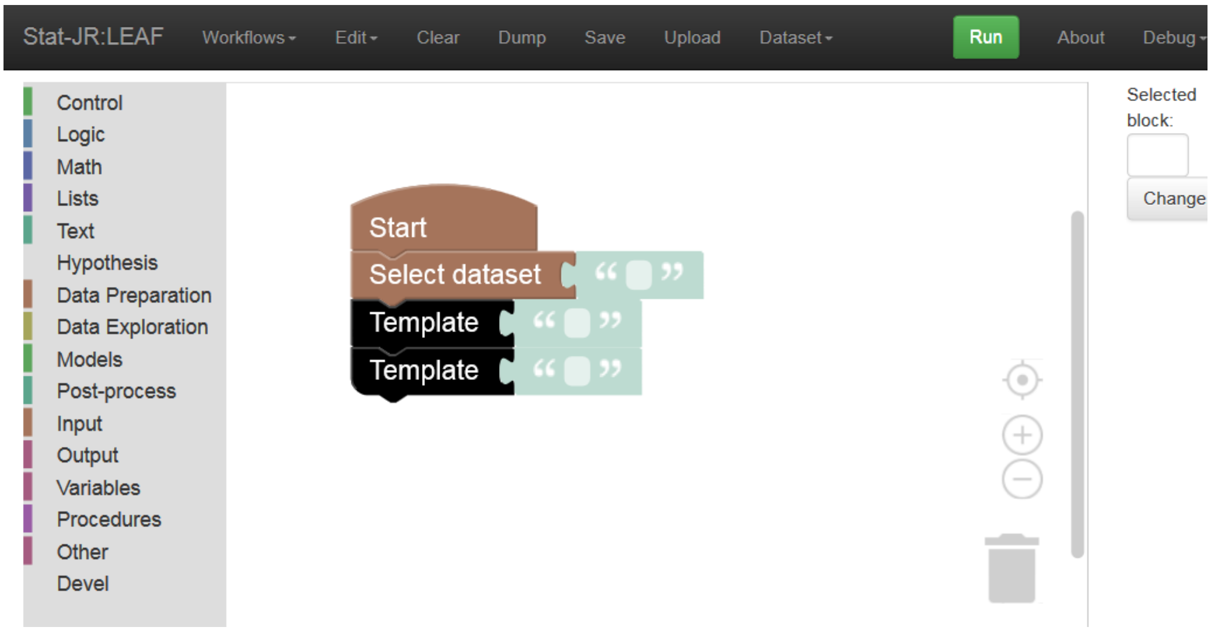


Figure 100

We'll quickly construct a workflow using the **Re-edit** button: a tool we briefly touched on in Section 1.8. We will start off by producing what we might call a 'skeleton workflow': a stripped down workflow that only contains the *Start* block, the *Select Dataset* block and the black *Template* blocks.

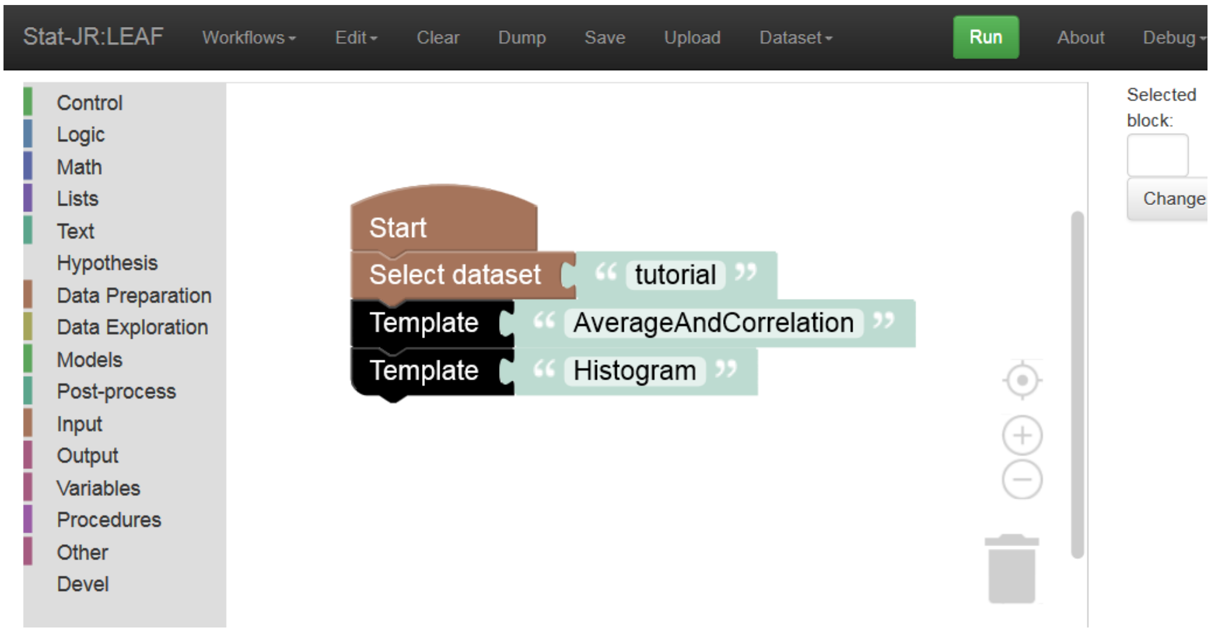
To create this skeleton workflow pick the *Start* block from the **Control** menu, the *Select dataset* block from the **Data Preparation** menu and two *Template* blocks from the **Devel** menu so that the workflow looks as follows:



Start
Select dataset " "
Template " "
Template " "

Figure 101

Next we will fill in the shadow blocks so that we have the names of the dataset (we're using the *tutorial* dataset we used earlier) and some templates which will allow us to explore aspects of the dataset (you may recall we encountered these two templates in Section 1):

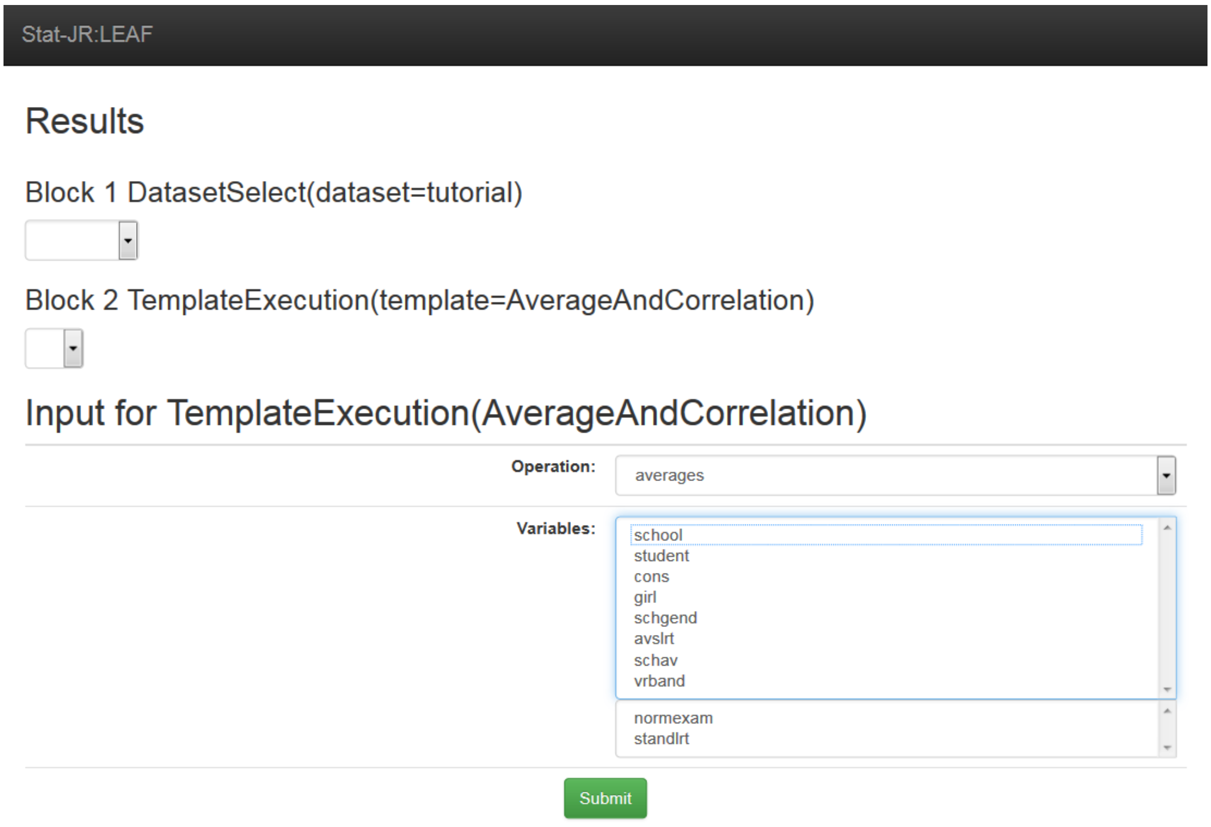


Start
Select dataset " tutorial "
Template " AverageAndCorrelation "
Template " Histogram "

Figure 102

This is a valid workflow but doesn't contain any inputs and so, as we saw in Section 1, when input values are not specified the user will instead be prompted for them when the workflow runs. Note that in this example, since we haven't specified how the input values are to be requested (e.g. via a prompt of our own choosing), the questions will simply be those that we observe in TREE.

If we click on **Run** we will see the following:


Stat-JR:LEAF

Results

Block 1 DatasetSelect(dataset=tutorial)

Block 2 TemplateExecution(template=AverageAndCorrelation)

Input for TemplateExecution(AverageAndCorrelation)

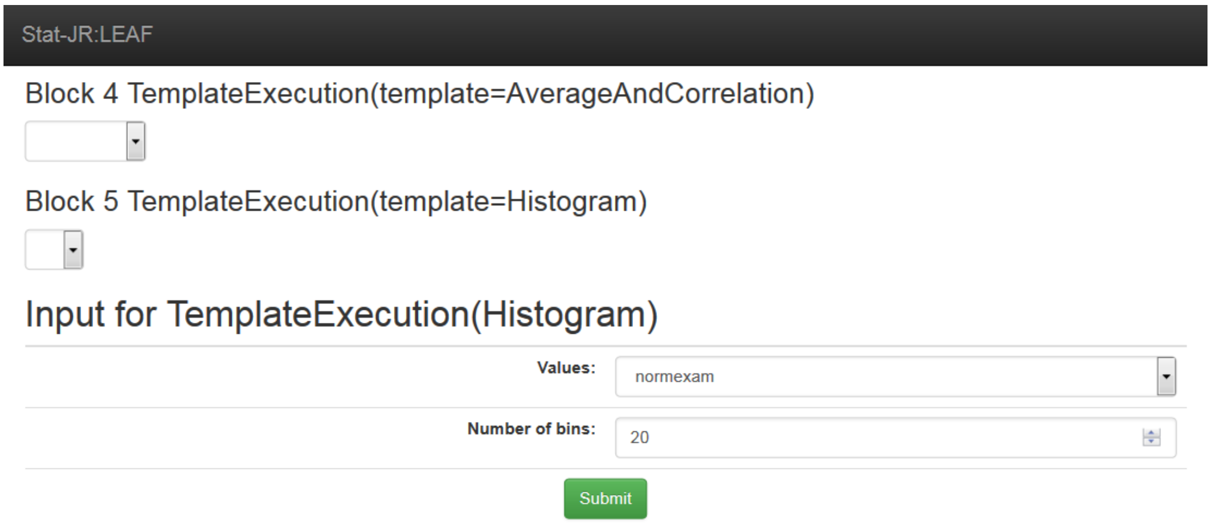
Operation: averages

Variables: school, student, cons, girl, schgend, avslrt, schav, vrband

normexam, standlrt

Figure 103

Here we see prompts for the input values for the *AverageAndCorrelation* template, which we can fill in. Here we have chosen *averages* as the **Operation**, and *normexam* and *standlrt* as the **Variables**; after pressing **Submit** we see the following:


Stat-JR:LEAF

Block 4 TemplateExecution(template=AverageAndCorrelation)

Block 5 TemplateExecution(template=Histogram)

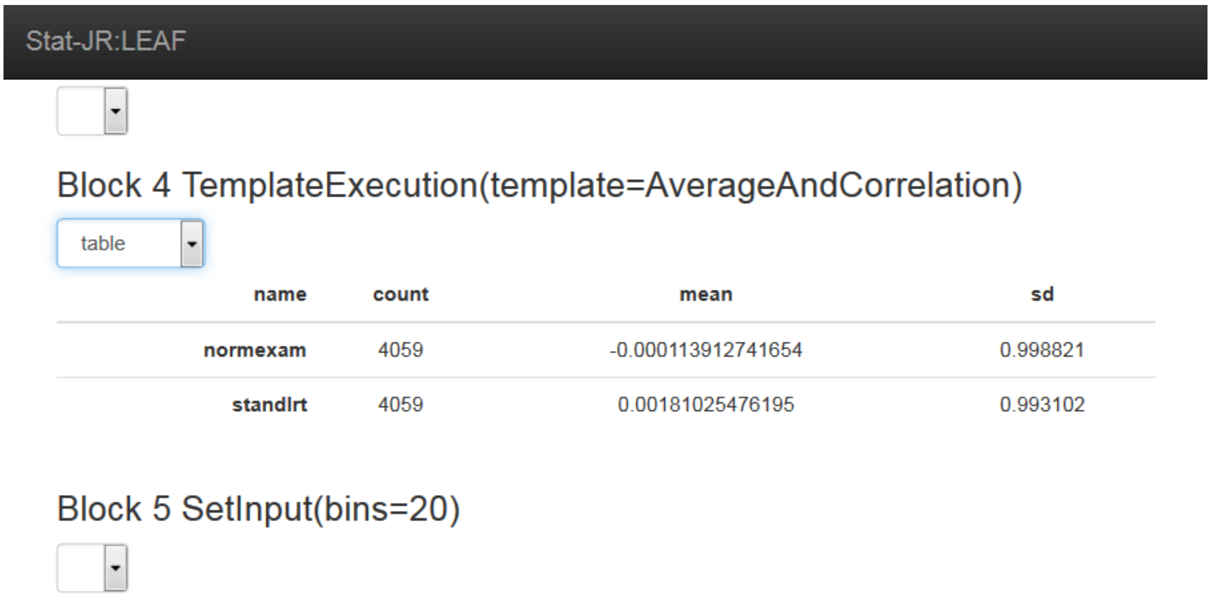
Input for TemplateExecution(Histogram)

Values: normexam

Number of bins: 20

Figure 104

So we're now being prompted for values for the *Histogram* template inputs; we've selected *normexam* as the **Values** to plot, and have typed *20* into the **Number of bins**. On pressing **Submit**, the relevant output from the AverageAndCorrelation template, *table*, and that from the *Histogram* template, *histogram.svg*, will be created but not displayed until we select them from the pull-down lists associated with each template execution; here first is the output *table* from the block titled TemplateExecution(template=AverageAndCorrelation):


Stat-JR:LEAF

Block 4 TemplateExecution(template=AverageAndCorrelation)

table

name	count	mean	sd
normexam	4059	-0.000113912741654	0.998821
standlrt	4059	0.00181025476195	0.993102

Block 5 SetInput(bins=20)

Figure 105

...and next the output *histogram.svg* from the pull-down list associated with the *Histogram* template execution:

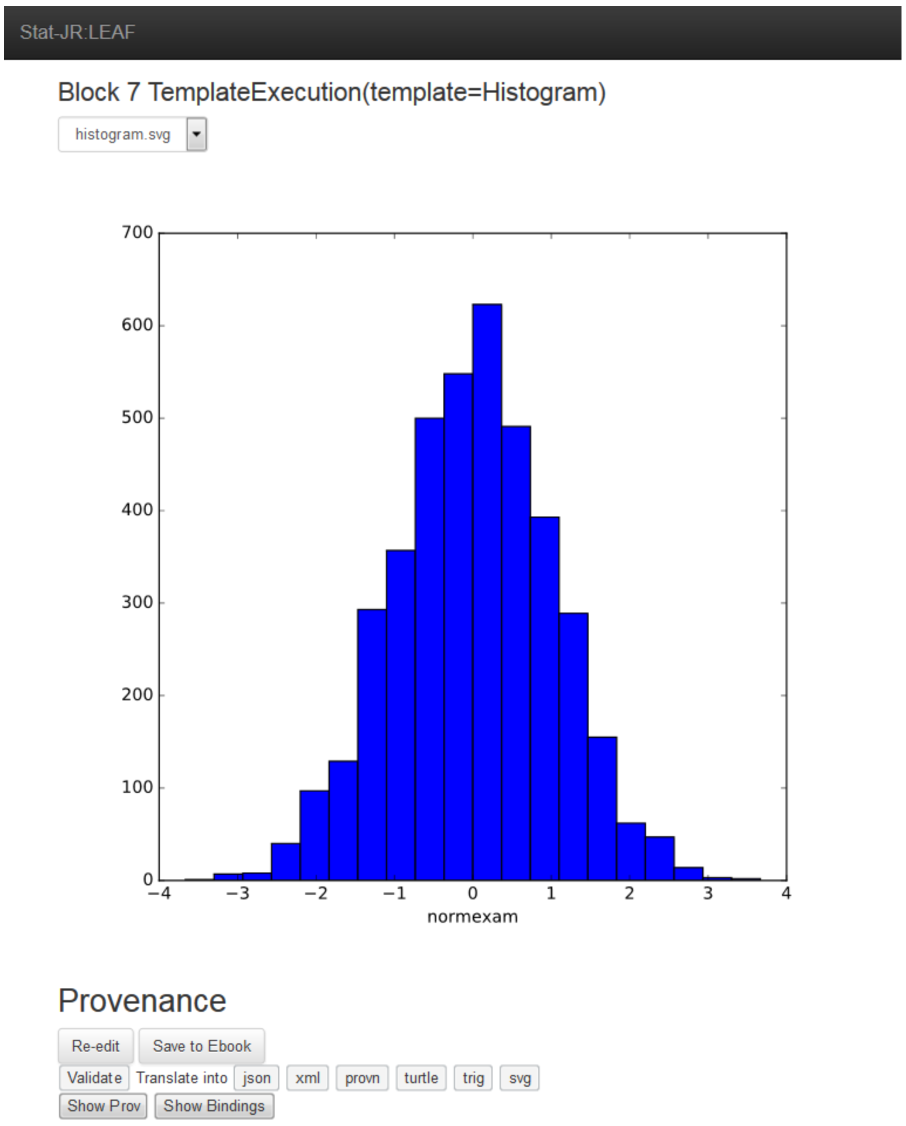
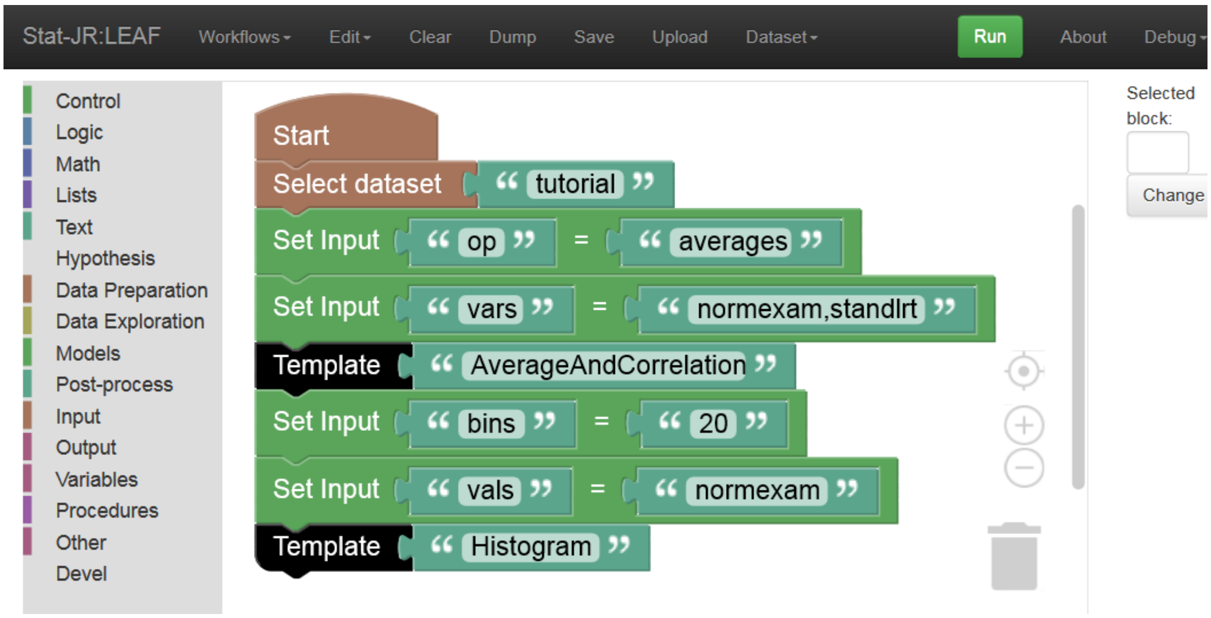


Figure 106

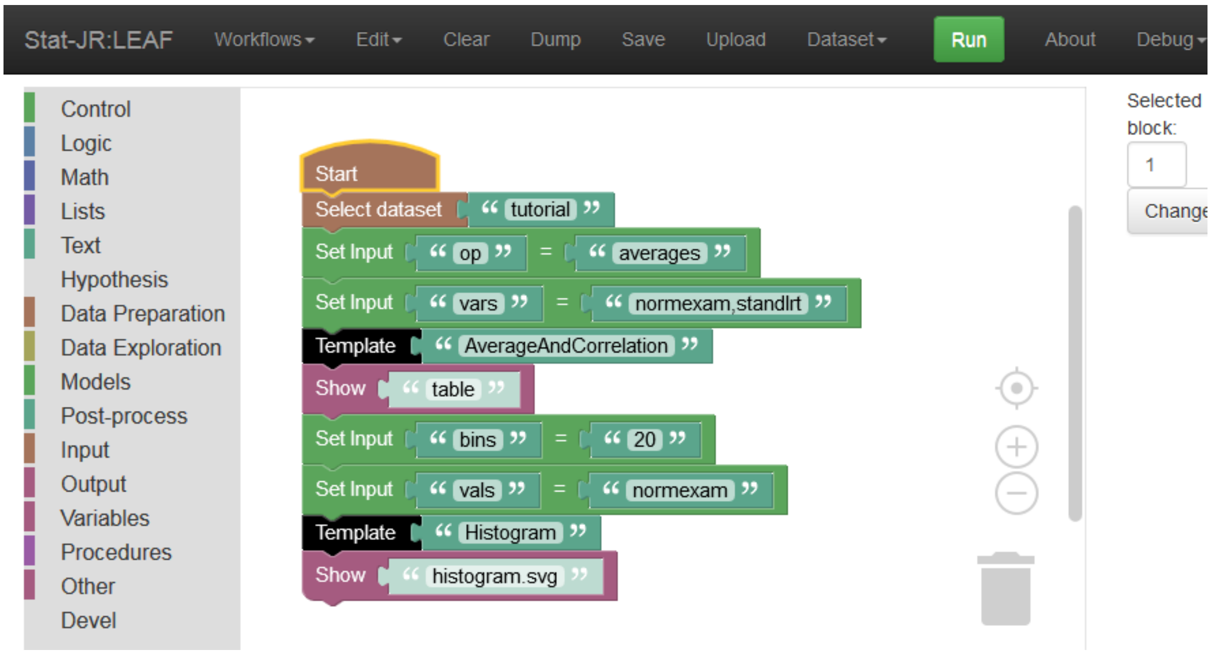
At the bottom of the screen you can see the **Re-edit** button: as we saw in Section 1.8, this will return the full workflow including the values for the inputs we specified. So, click on this now and you should see the following:



Start
Select dataset "tutorial"
Set Input "op" = "averages"
Set Input "vars" = "normexam,standlrt"
Template "AverageAndCorrelation"
Set Input "bins" = "20"
Set Input "vals" = "normexam"
Template "Histogram"

Figure 107

So this workflow is a log of the workflow we have just run. If we wish to include the output objects then we need to add two *Show* blocks from the **Output** menu after the respective *Template* blocks, as follows:



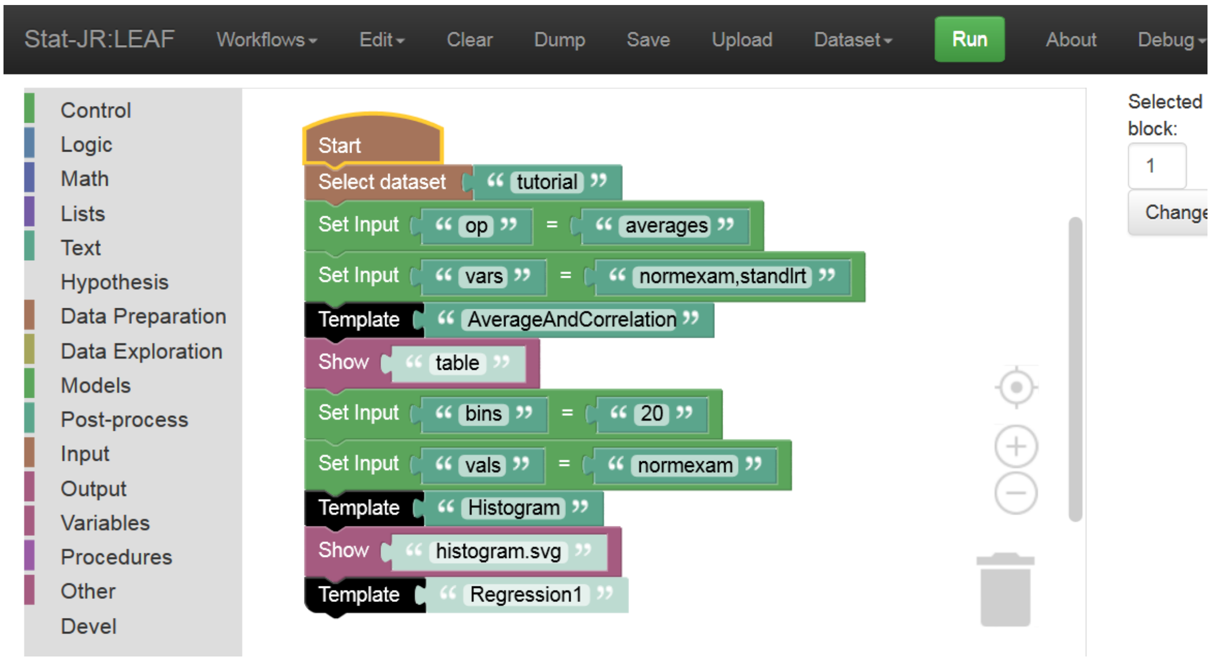
Start
Select dataset "tutorial"
Set Input "op" = "averages"
Set Input "vars" = "normexam,standlrt"
Template "AverageAndCorrelation"
Show "table"
Set Input "bins" = "20"
Set Input "vals" = "normexam"
Template "Histogram"
Show "histogram.svg"

Figure 108

If we were to run this workflow now it would automatically execute to completions, using the input values we chose earlier, and displaying the two outputs we've place in the *Show* blocks.

Save this workflow as *section4_01.xml*.

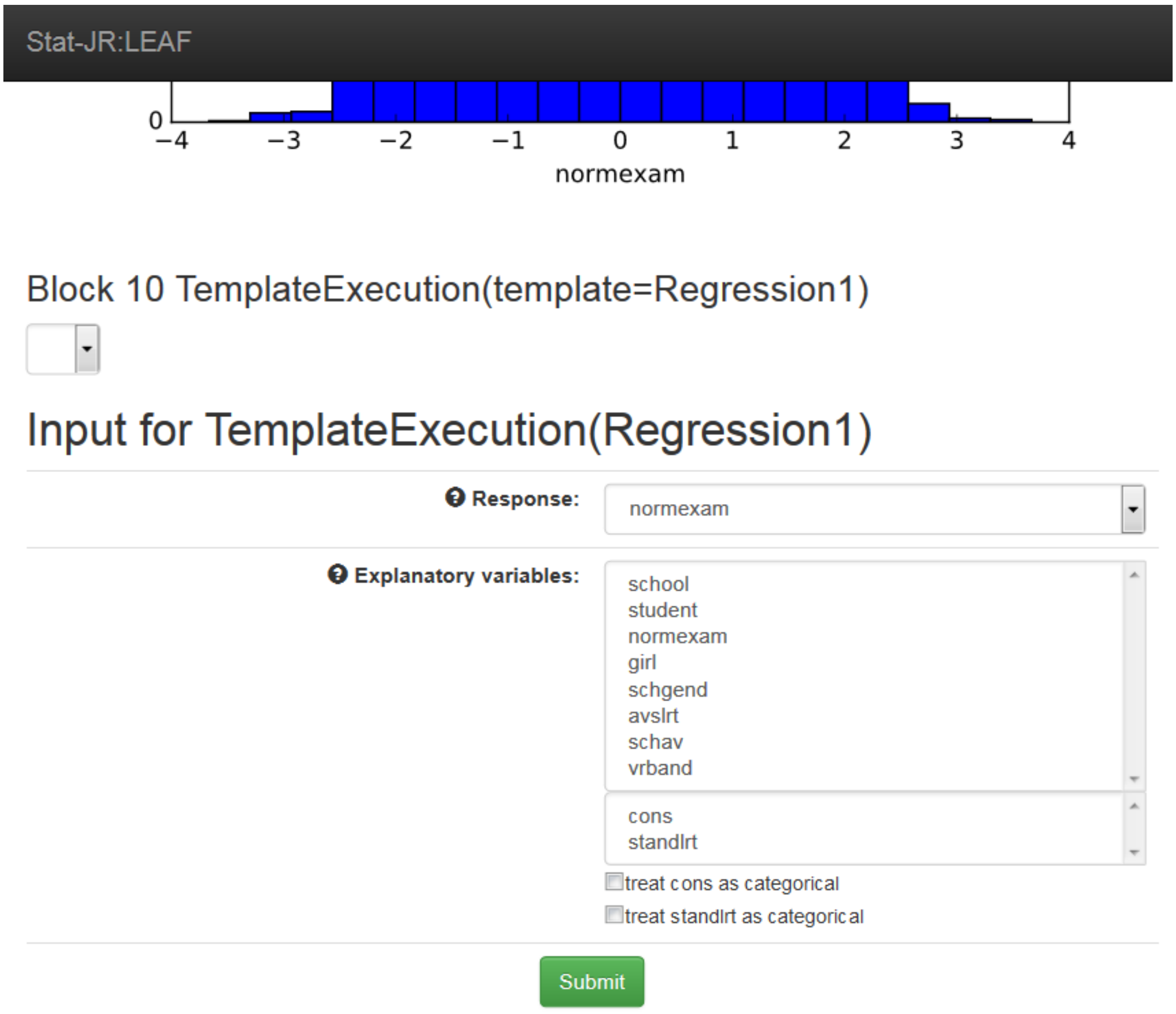
We will next extend our workflow by considering a regression, as we did in Section 1. Here we will simply add an additional *Template* block (from the **Devel** menu) to the end of the workflow and type *Regression1* in the associated shadow block:



```
Start
Select dataset "tutorial"
Set Input "op" = "averages"
Set Input "vars" = "normexam,standlrt"
Template "AverageAndCorrelation"
Show "table"
Set Input "bins" = "20"
Set Input "vals" = "normexam"
Template "Histogram"
Show "histogram.svg"
Template "Regression1"
```

Figure 109

If we next press **Run** then, after the templates has calculated the averages and generated the histogram, we will need to fill in the many *Regression1* inputs that appear in sequential stages thus (here we're regressing *normexam* on *standlrt*, and are including the constant of ones (*cons*) already in the dataset as a predictor in order to fit an intercept to the model, *cf.* earlier examples in which we demonstrated creating a constant anew):



Block 10 TemplateExecution(template=Regression1)

Input for TemplateExecution(Regression1)

Response: normexam

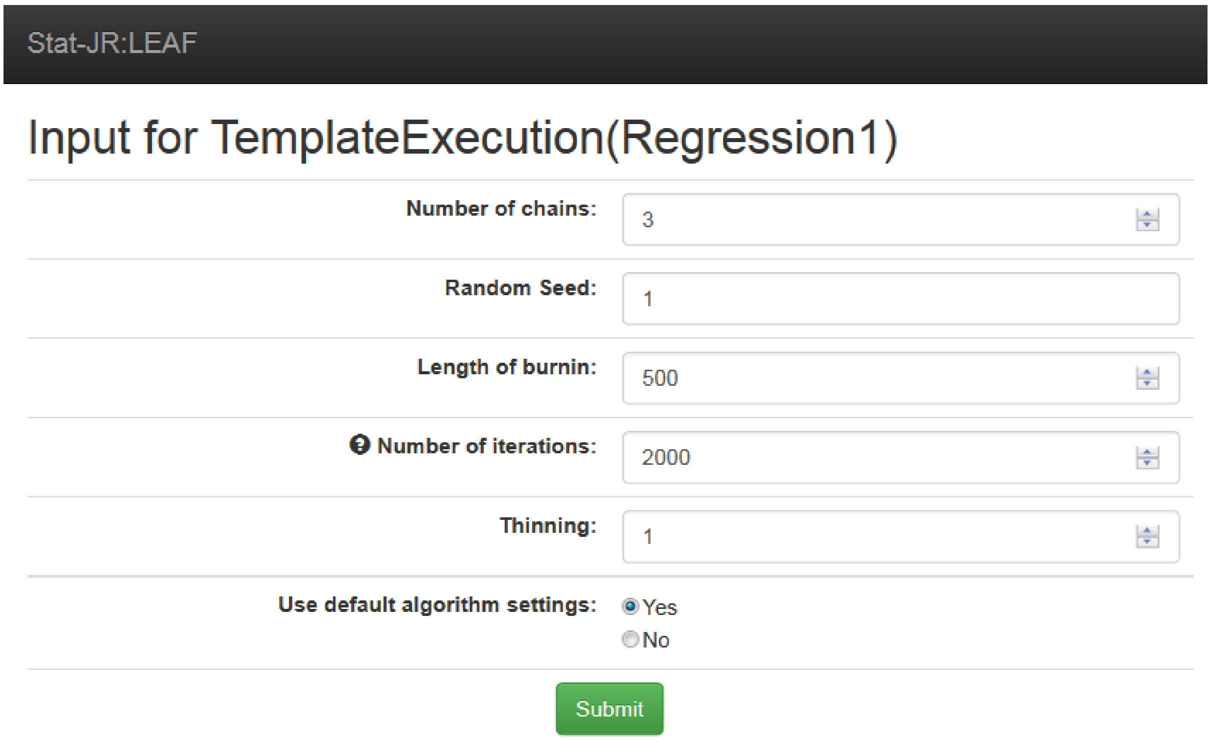
Explanatory variables:
school
student
normexam
girl
schgend
avslrt
schav
vrband

cons
standlrt

☐treat cons as categorical
☐treat standlrt as categorical

Figure 110

…and (we're just accepting the defaults here)…

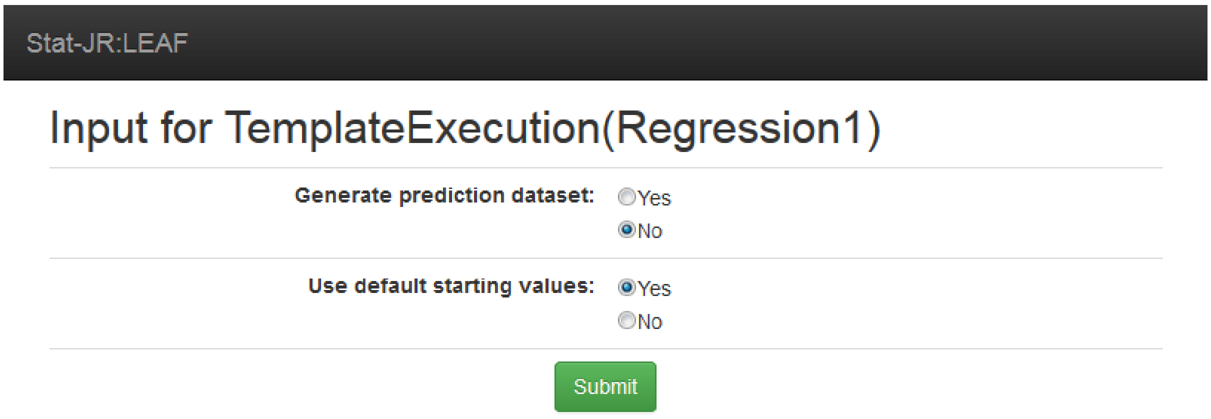

Stat-JR:LEAF

Input for TemplateExecution(Regression1)

Number of chains:	3
Random Seed:	1
Length of burnin:	500
Number of iterations:	2000
Thinning:	1
Use default algorithm settings:	[✓ Yes] No

Figure 111

...and...

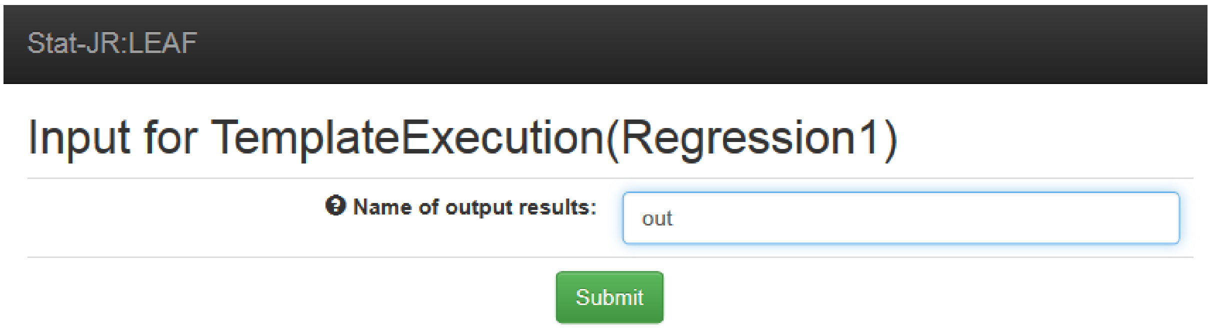

Stat-JR:LEAF

Input for TemplateExecution(Regression1)

Generate prediction dataset:	Yes [✓ No]
Use default starting values:	[✓ Yes] No

Figure 112

...and finally...

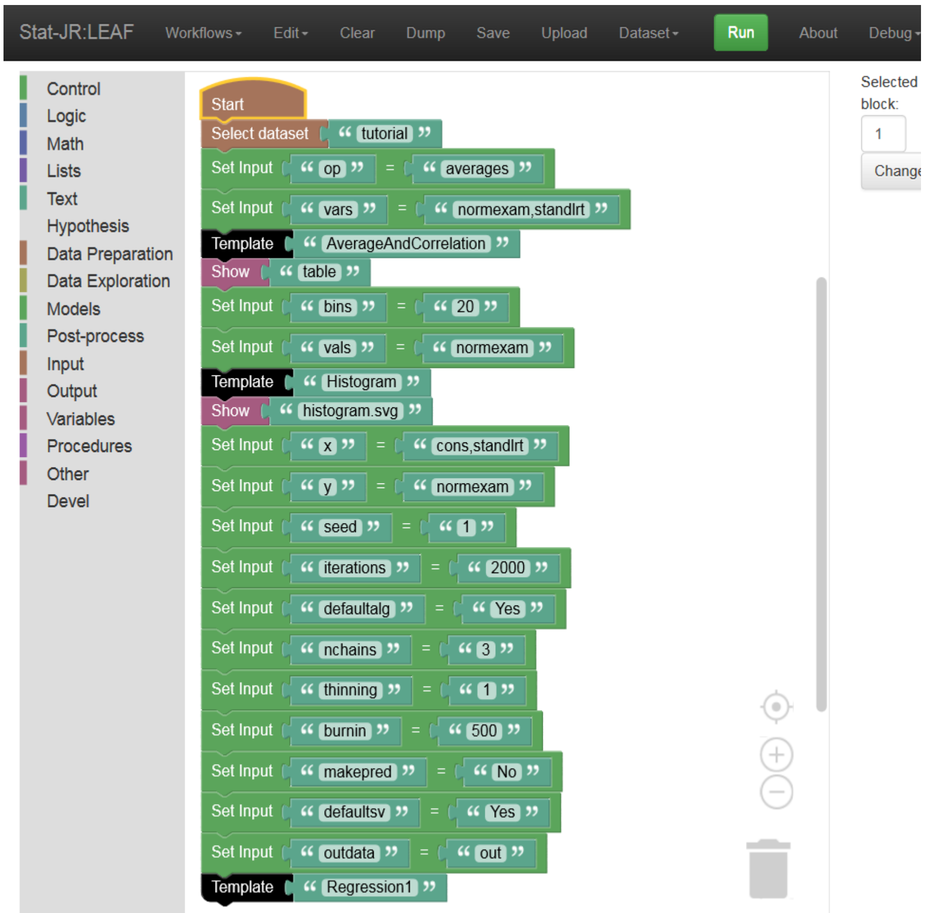

Stat-JR:LEAF

Input for TemplateExecution(Regression1)

Name of output results: out

Figure 113

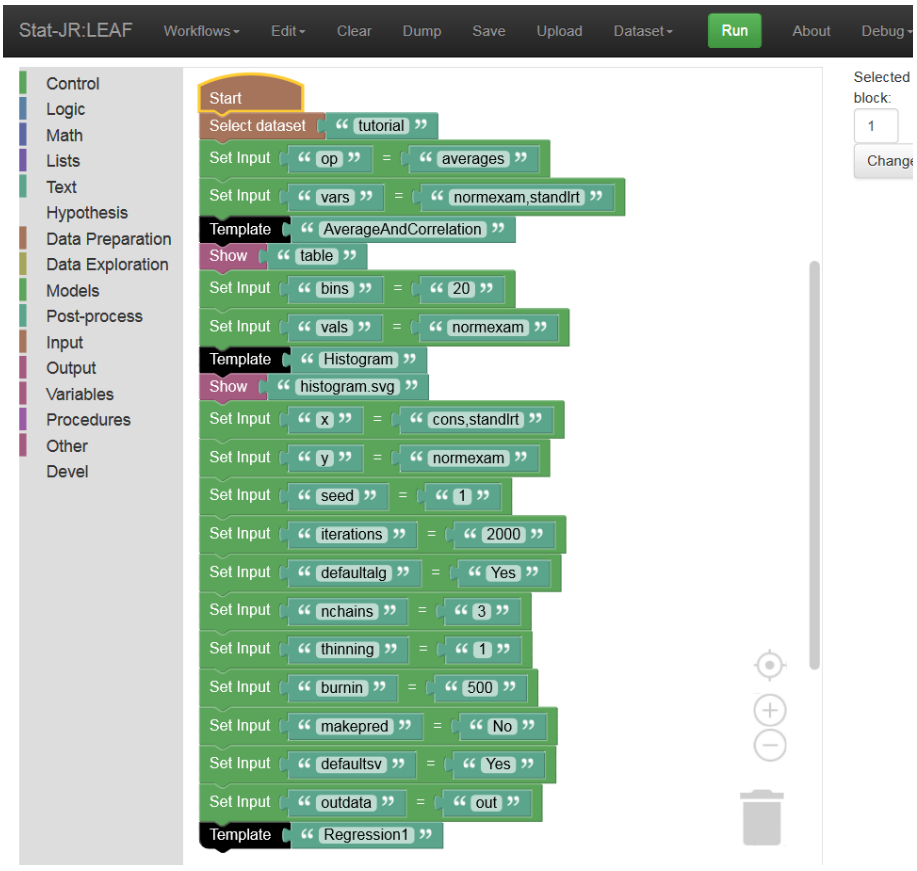
Once we have specified this last value, and clicked **Submit**, then the template will run. Once it has finished we can display outputs from the template execution, for example here we have chosen *ModelResults*:



```
Start
Select dataset "tutorial"
Set Input "op" = "averages"
Set Input "vars" = "normexam,standlrt"
Template "AverageAndCorrelation"
Show "table"
Set Input "bins" = "20"
Set Input "vals" = "normexam"
Template "Histogram"
Show "histogram.svg"
Set Input "x" = "cons,standlrt"
Set Input "y" = "normexam"
Set Input "seed" = "1"
Set Input "iterations" = "2000"
Set Input "defaultalg" = "Yes"
Set Input "nchains" = "3"
Set Input "thinning" = "1"
Set Input "burnin" = "500"
Set Input "makepred" = "No"
Set Input "defaultsv" = "Yes"
Set Input "outdata" = "out"
Template "Regression1"
```

Figure 114

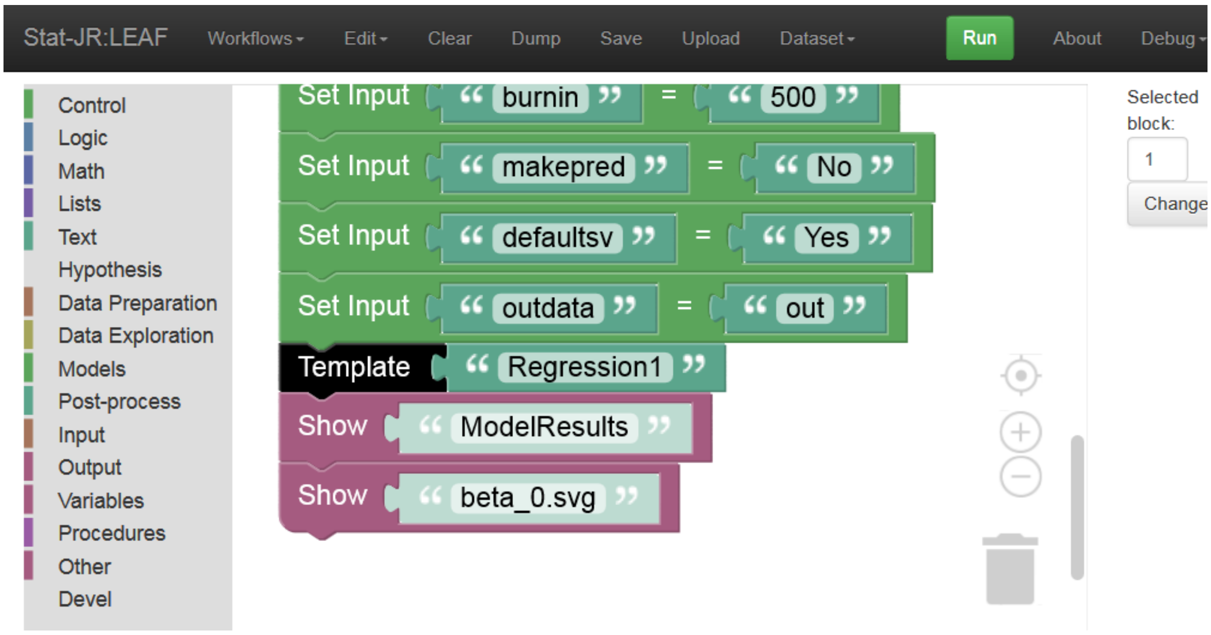
If we click on **Re-edit** we will now get a longer workflow that, if executed, would run all three templates in order:



Start
Select dataset "tutorial"
Set Input "op" = "averages"
Set Input "vars" = "normexam,standlrt"
Template "AverageAndCorrelation"
Show "table"
Set Input "bins" = "20"
Set Input "vals" = "normexam"
Template "Histogram"
Show "histogram.svg"
Set Input "x" = "cons,standlrt"
Set Input "y" = "normexam"
Set Input "seed" = "1"
Set Input "iterations" = "2000"
Set Input "defaultalg" = "Yes"
Set Input "nchains" = "3"
Set Input "thinning" = "1"
Set Input "burnin" = "500"
Set Input "makepred" = "No"
Set Input "defaultsv" = "Yes"
Set Input "outdata" = "out"
Template "Regression1"

Figure 115

We will add a couple of *Show* blocks to show the *ModelResults* and *beta_0.svg* output objects (the latter consists of six plots of various MCMC diagnostics for the beta_0 (intercept) parameter):

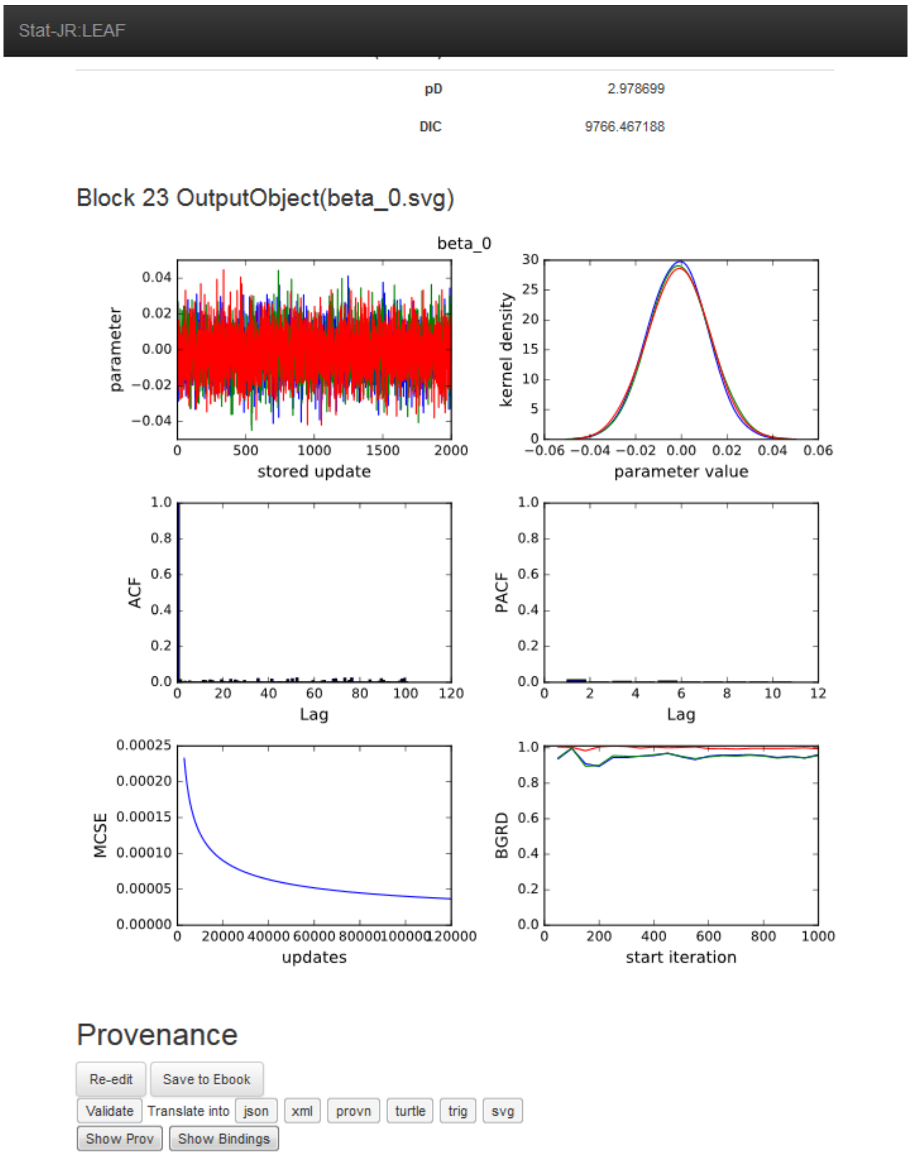


Set Input "burnin" = "500"
Set Input "makepred" = "No"
Set Input "defaultsv" = "Yes"
Set Input "outdata" = "out"
Template "Regression1"
Show "ModelResults"
Show "beta_0.svg"

Figure 116

Save this workflow as *section4_02.xml*.

Now if we run the workflow the three templates will be executed in order, and the four output objects will be shown. The sixway plot (*beta_0.svg*) can be seen in the window below:



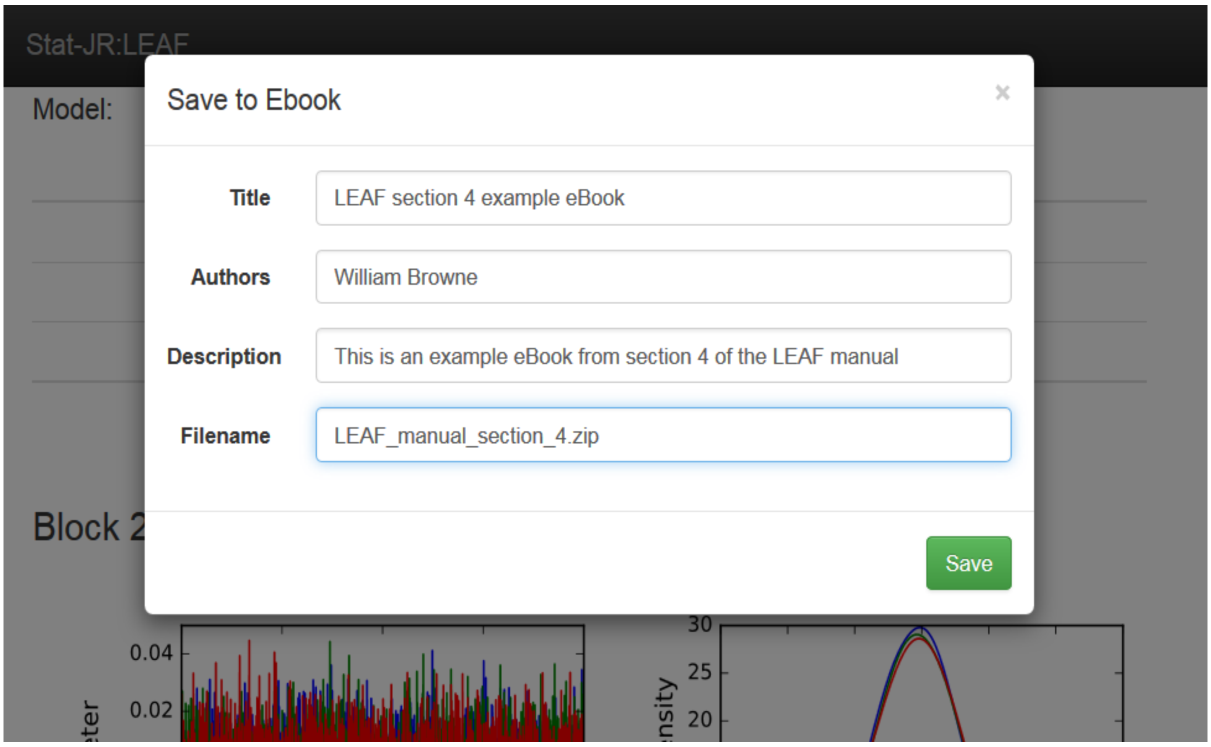
pD	2.978699
DIC	9766.467188

Block 23 OutputObject(beta_0.svg)

Figure 117

As you might imagine, if we were to press **Re-edit** again we should get exactly the workflow we just ran, however we will instead click on the **Save to Ebook** button.

When we do this a popup appears for which we will specify the requested information as follows:



Title: LEAF section 4 example eBook

Authors: William Browne

Description: This is an example eBook from section 4 of the LEAF manual

Filename: LEAF_manual_section_4.zip

Figure 118

Clicking **Save** then gives a standard **Save As** window in which we then need to save the *zip* file to a directory from which we can retrieve it from within Stat-JR's DEEP interface.

The DEEP interface is Stat-JR's eBook interface and can be accessed by selecting **All programs > Centre for Multilevel Modelling > StatJR – DEEP.** As with the LEAF interface, a console window will pop-up and after a few moments the DEEP front-end will be displayed in a web browser, as shown below:

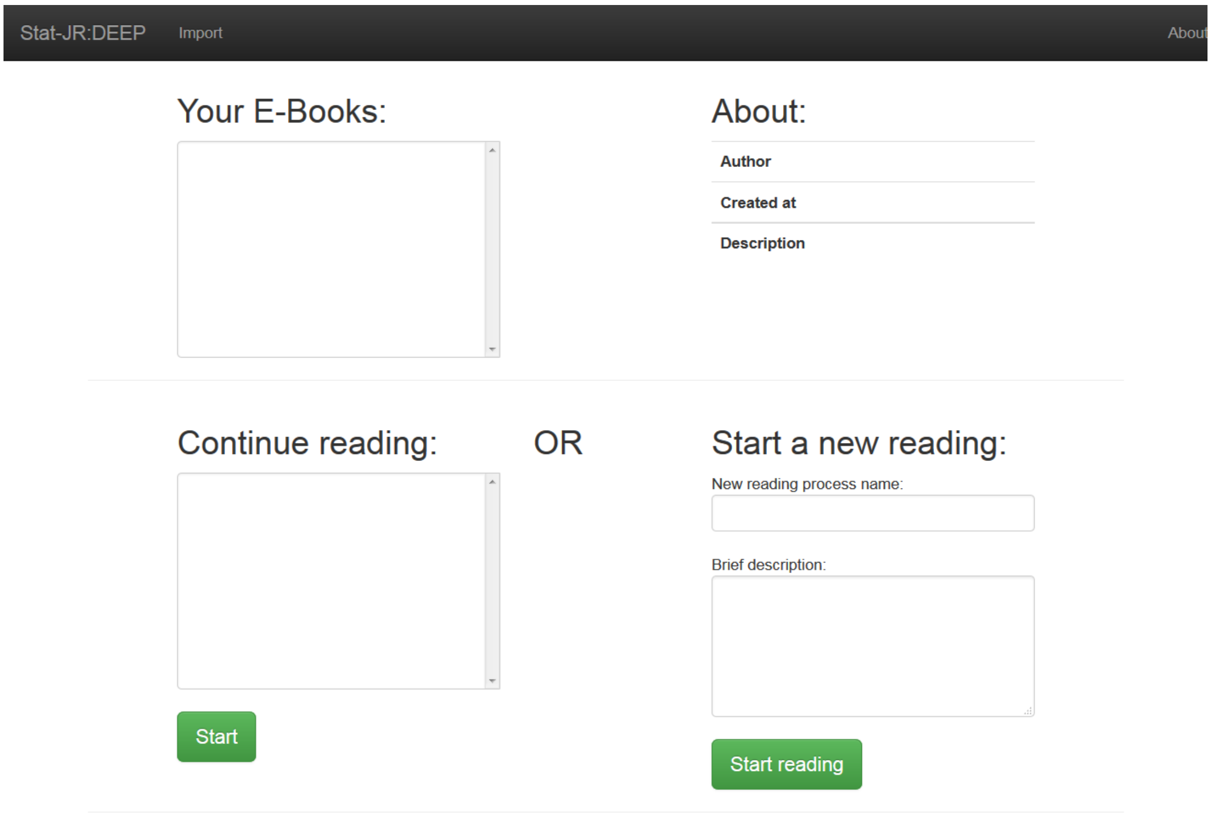


Figure 119

To select the eBook you have created you will need to click on the **Import** button (in the black bar at the top):

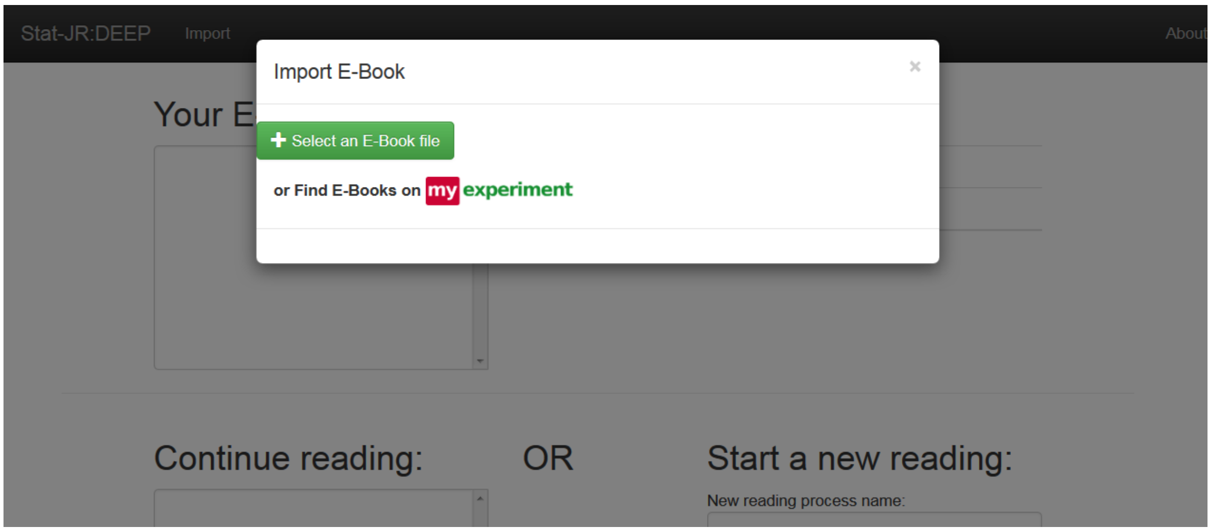
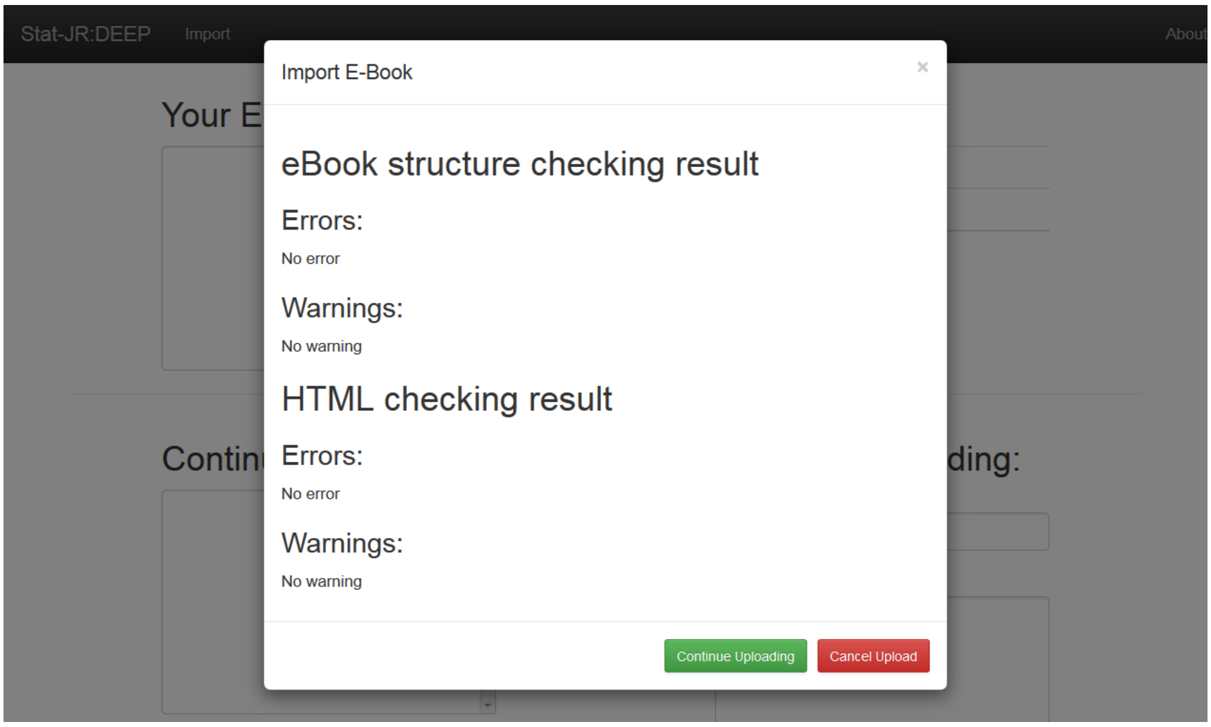


Figure 120

Next choose to **Select an E-book file** and select the file you saved from the workflow system, and when prompted click **Continue Uploading**:


Import E-Book

eBook structure checking result

Errors:

No error

Warnings:

No warning

HTML checking result

Errors:

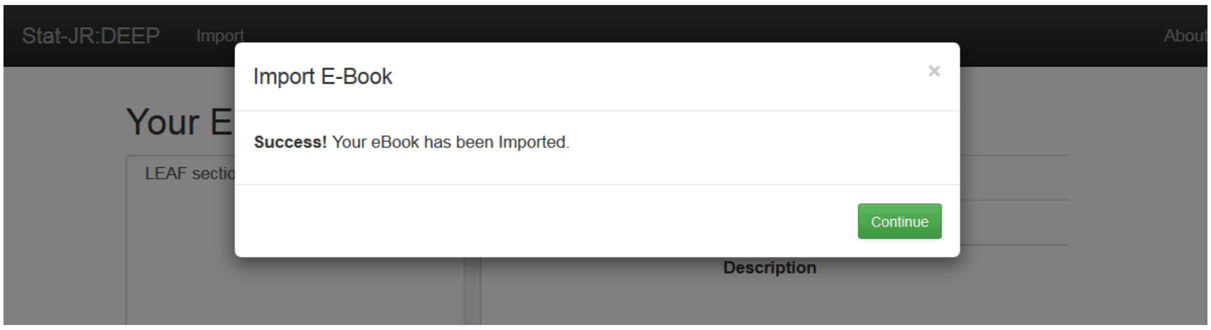
No error

Warnings:

No warning

Figure 121

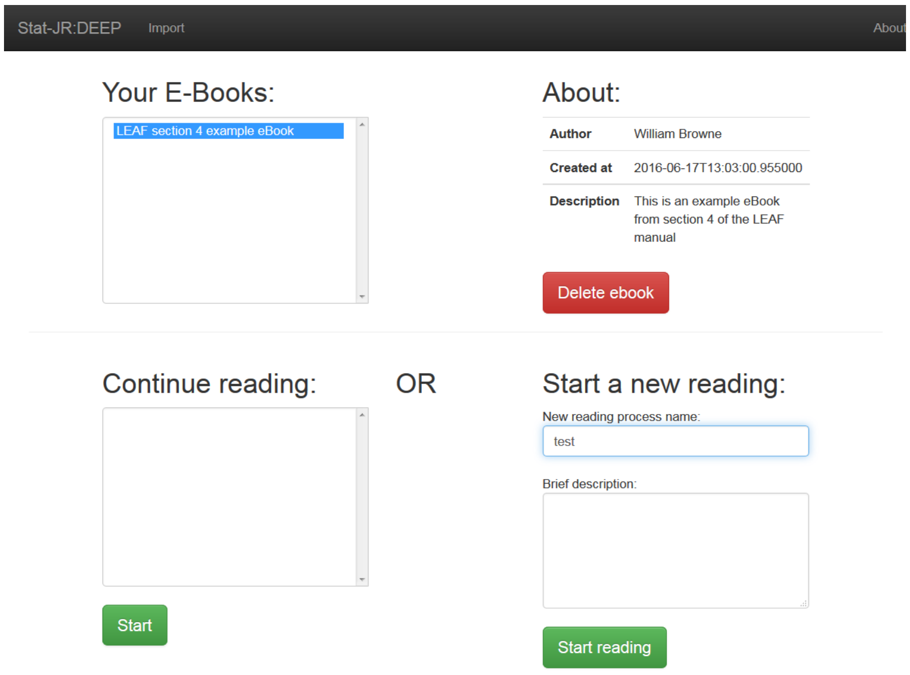
...after which we (hopefully) receive confirmation our eBook has been successfully imported:


Import E-Book

Success! Your eBook has been Imported.

Figure 122

It now appears in the list of **Your E-Books** in the top left pane; select this eBook in the list so that it is highlighted (associated meta-information, such as the **Author** and **Description**, will then appear under **About**). Then, under **Start a new reading,** type a **New reading process name** (we have chosen *test*, althought it doesn't really matter what name you choose):



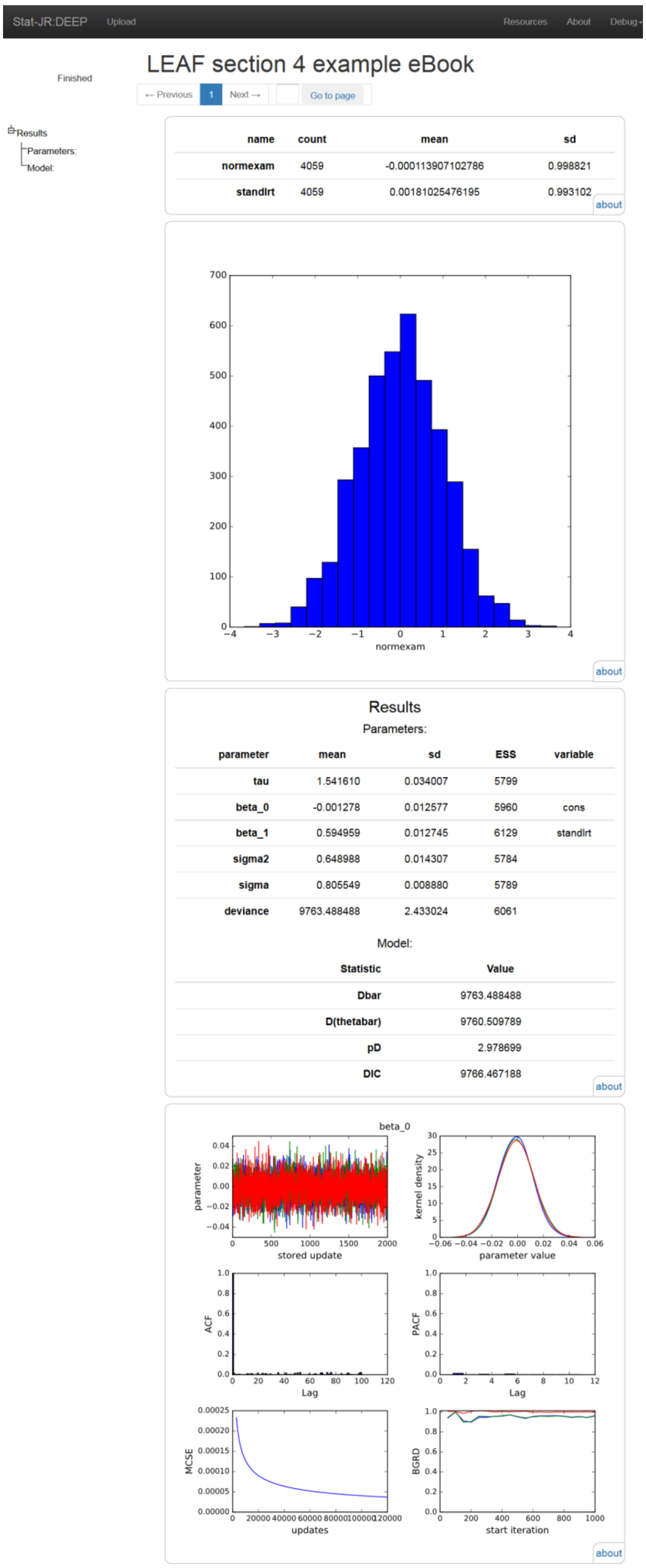
Author	William Browne
Created at	2016-06-17T13:03:00.955000
Description	This is an example eBook from section 4 of the LEAF manual

New reading process name: test

Brief description:

Figure 123

Clicking on **Start reading** will fire up the eBook and we will get a largely blank page with the progress gauge in the top-left corner stating “Running Workflow”. Soon this will indicated it has “Finished” and we will be left with the following:



LEAF section 4 example eBook

name	count	mean	sd
normexam	4059	-0.000113907102786	0.998821
standlrt	4059	0.00181025476195	0.993102

Results

Parameters:

parameter	mean	sd	ESS	variable
tau	1.541610	0.034007	5799	
beta_0	-0.001278	0.012577	5960	cons
beta_1	0.594959	0.012745	6129	standlrt
sigma2	0.648988	0.014307	5784	
sigma	0.805549	0.008880	5789	
deviance	9763.488488	2.433024	6061	

Model:

Statistic	Value
Dbar	9763.488488
D(thetabar)	9760.509789
pD	2.978699
DIC	9766.467188

Figure 124

Essentially we have a rather skeleton-like eBook where the outputs from the *Show* blocks in the workflow appear as objects in boxes in a one-page eBook.

Currently the LEAF system will simply create this skeleton eBook, but we can then consider adding to the eBook structure etc.; see the *Stat-JR DEEP eBook Reader & Authoring Guide* for more information.

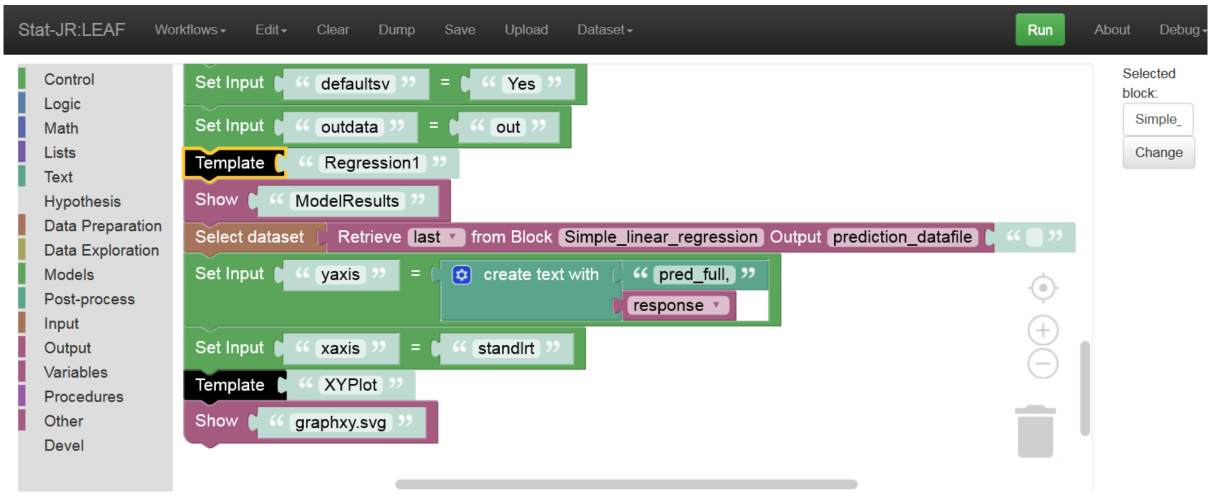
4.1 What have we covered?

In this section we have demonstrated how to create a workflow by starting with a ‘skeleton’ and filling in the template inputs ourselves when prompted, before selecting **Re-edit** to ‘complete the loop’ and construct the workflow corresponding to our choices.

Then we have investigated exporting this as a Stat-JR eBook, to be opened and read in the Stat-JR:DEEP interface.

Section 5 Appendix

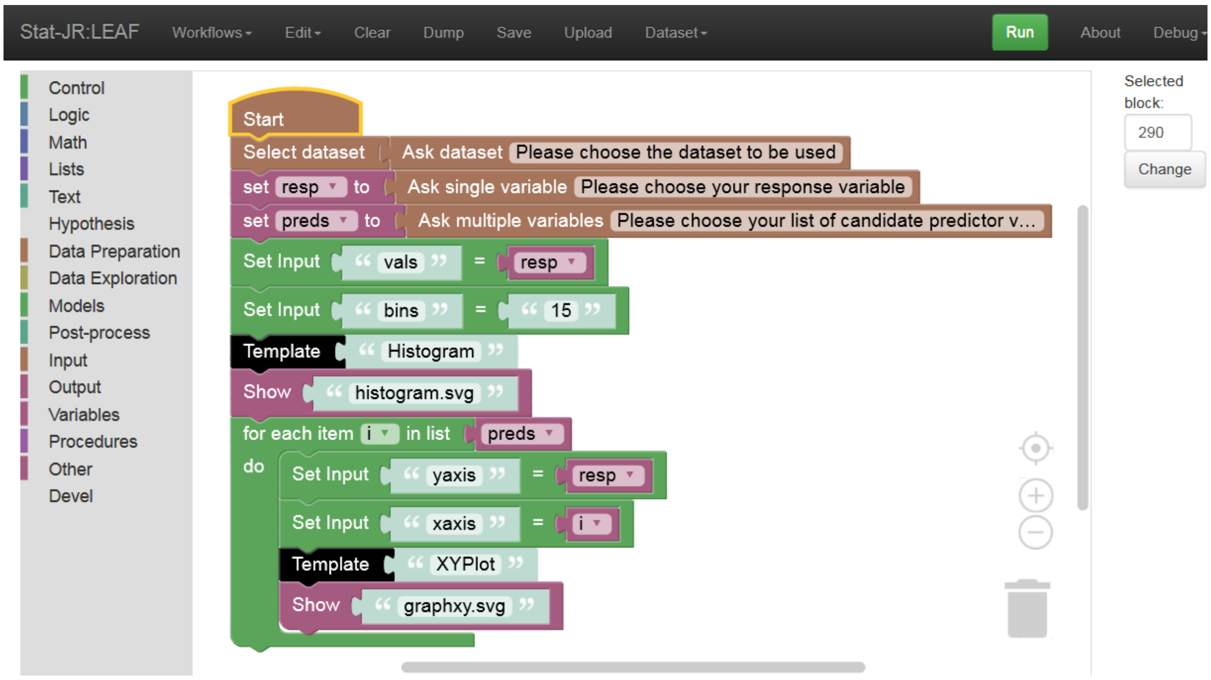
From Section 1.12, here's the end of the workflow with our prediction-plotting blocks added to it; remember to save the workflow as *section1_12.xml*.



```
Set Input "defaultsv" = "Yes"
Set Input "outdata" = "out"
Template "Regression1"
Show "ModelResults"
Select dataset  Retrieve last from Block Simple_linear_regression Output prediction_datafile ""
Set Input "yaxis" = create text with "pred_full,"
                                     response
Set Input "xaxis" = "standlrt"
Template "XYPlot"
Show "graphxy.svg"
```

Figure 125

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



```
Start
Select dataset  Ask dataset Please choose the dataset to be used
set resp to  Ask single variable Please choose your response variable
set preds to  Ask multiple variables Please choose your list of candidate predictor v...
Set Input "vals" = resp
Set Input "bins" = "15"
Template "Histogram"
Show "histogram.svg"
for each item i in list preds
do  Set Input "yaxis" = resp
    Set Input "xaxis" = i
    Template "XYPlot"
    Show "graphxy.svg"
```

Figure 126

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 *section2_03.xml*.