

Practical 1: Introduction to the Stat-JR workflow system

1.1 Overview

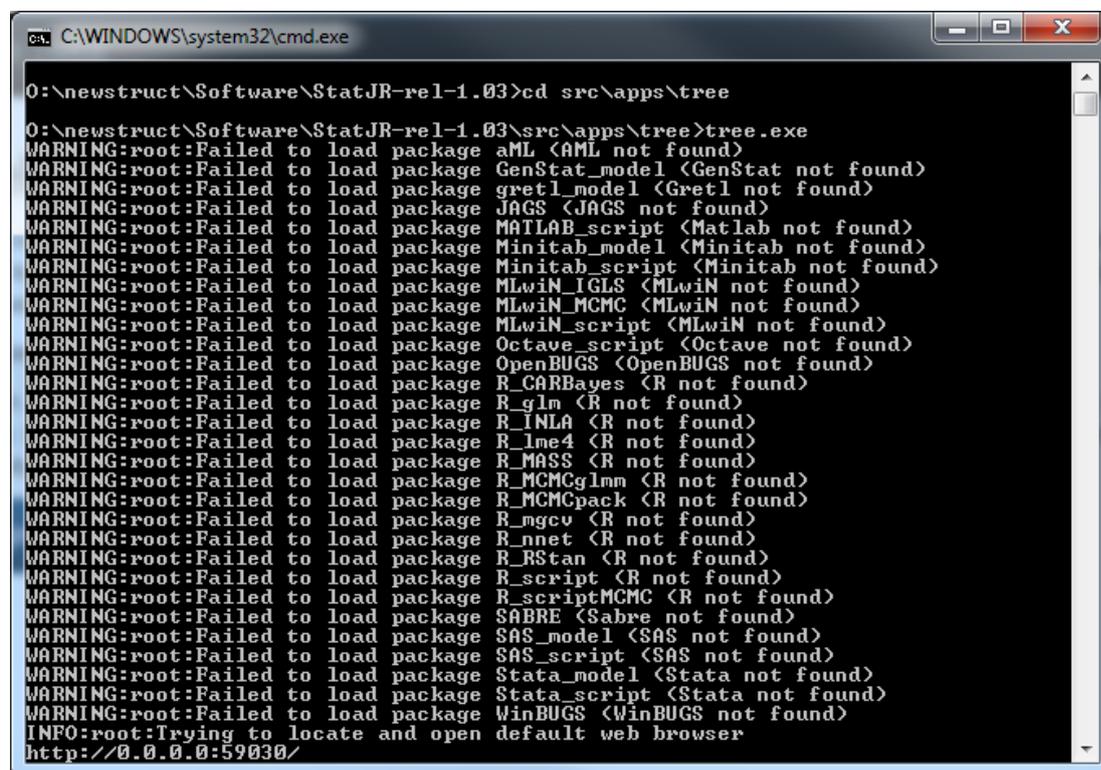
These practicals are designed to introduce new users to Stat-JR and in particular to its new workflow interface. Whilst the new workflow interface is still under development as part of the eBook grant, much of its structure and functionality is already in place, and today we will introduce the principles underlying how it works via a number of practical examples. As well as getting a handle on how the system works, any feedback you pass onto us will be very valuable in helping shape the development of the workflow system as it progresses over the remainder of the project.

In order to introduce the workflow interface, we will provide an overview of how to use the TREE (*Template Reading and Execution Environment*) 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 the workflow system, TREE and DEEP (*Documents with Embedded Execution and Provenance*) – 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 by clicking on the executable file *tree.cmd*. When we do this we will find a command window appears which looks something like the following:



```

cmd - C:\WINDOWS\system32\cmd.exe
O:\newstruct\Software\StatJR-rel-1.03>cd src\apps\tree
O:\newstruct\Software\StatJR-rel-1.03\src\apps\tree>tree.exe
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 MLwiN_IGLS (MLwiN not found)
WARNING:root:Failed to load package MLwiN_MCMC (MLwiN not found)
WARNING:root:Failed to load package MLwiN_script (MLwiN 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_INLA (R not found)
WARNING:root:Failed to load package R_lme4 (R not found)
WARNING:root:Failed to load package R_MASS (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_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 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
http://0.0.0.0:59030/

```

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 it will be running locally on the user's machine); this should open automatically after a few seconds, as follows¹:

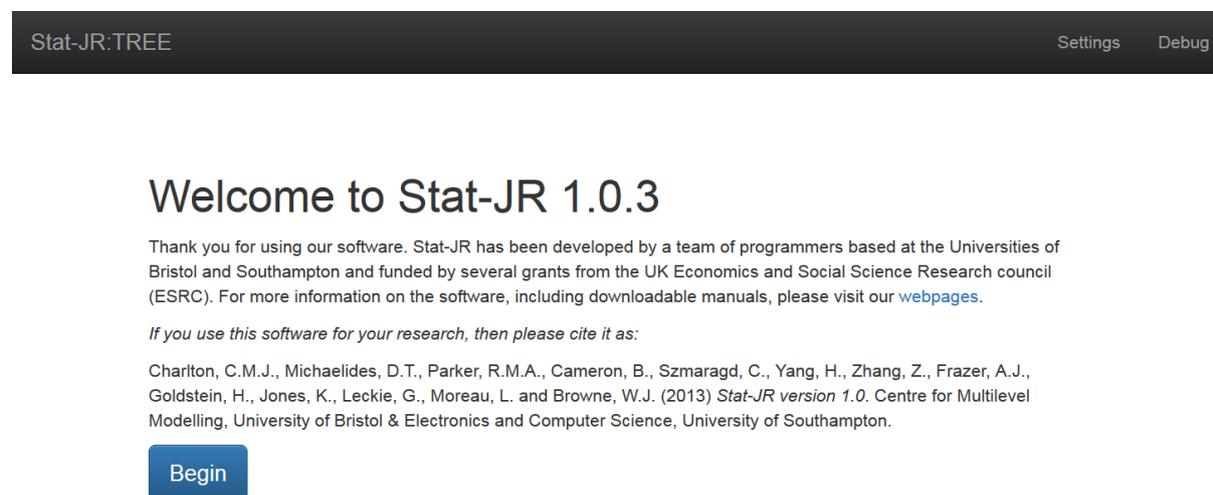


Figure 2

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

¹ 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. 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).

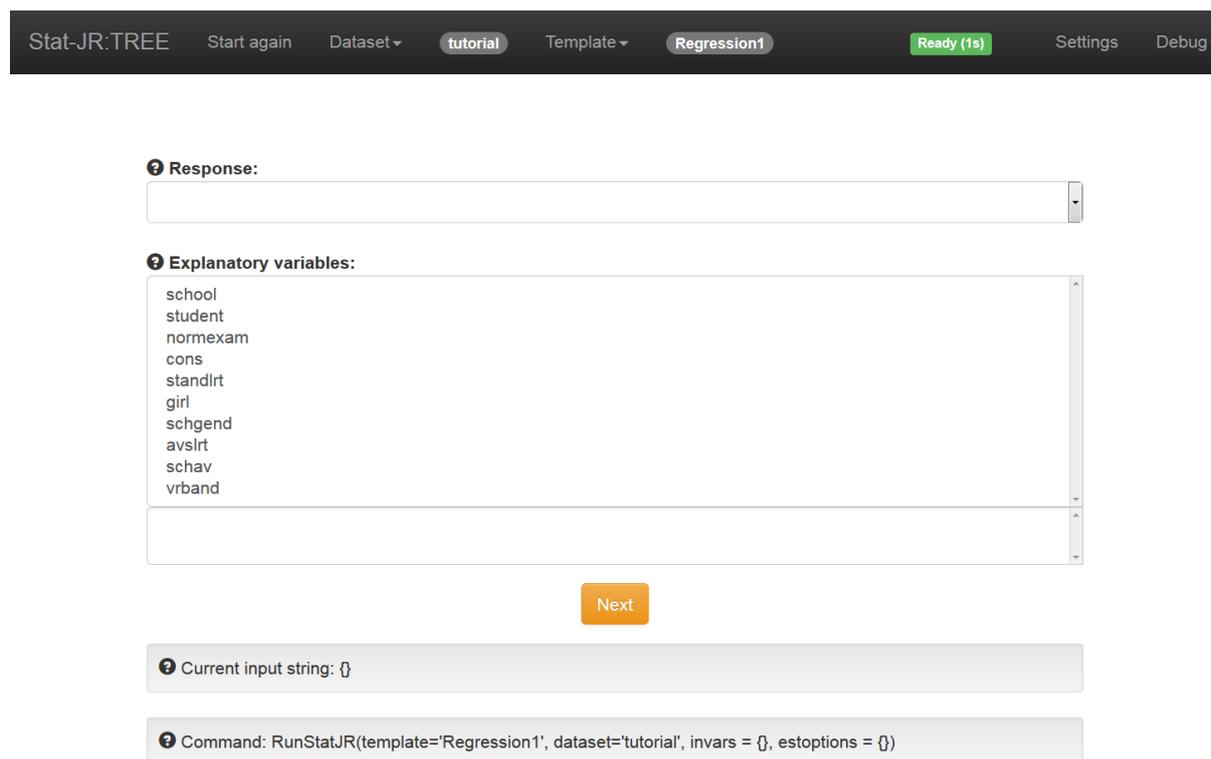


Figure 3

The TREE interface allows the user to try out one template at a time, using 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).

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 the `StatJR\datasets` folder, and then selecting **Debug > Reload datasets** (again, accessible via the black bar at the top of the browser window). If, instead, you have it (iii) 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 you to specify more particulars, and can also handle string variables).

In the case of options (i) and (iii) 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 `StatJR\datasets` folder) for use in the future sessions too.

So, via option (iii) (and downloading), Stat-JR will save your 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:

Stat-JR: TREE

Dataset name: tutorial Unload Duplicate Download

Data Summary Add variable Delete variable Edit data label Edit value labels

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

	school	student	normexam	cons	standirt	girl	schgend	avsirt	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:

Stat-JR: TREE

Dataset name: tutorial Unload Duplicate Download

Data Summary Add variable Delete variable Edit data label Edit value labels

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	-0.0001139127416	0.998821	Age 16 exam score (normalised)	
cons	4059	0	1	1	1.0	0.0	Constant	
standlrt	4059	0	-2.93495	3.01595	0.00181025476195	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	0.00181024719495	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

Page 1 of 1 View 1 - 10 of 10

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 (much like the *data viewer* in RStudio, which allows the user to explore aspects of their data independent of commands in the R console). 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:

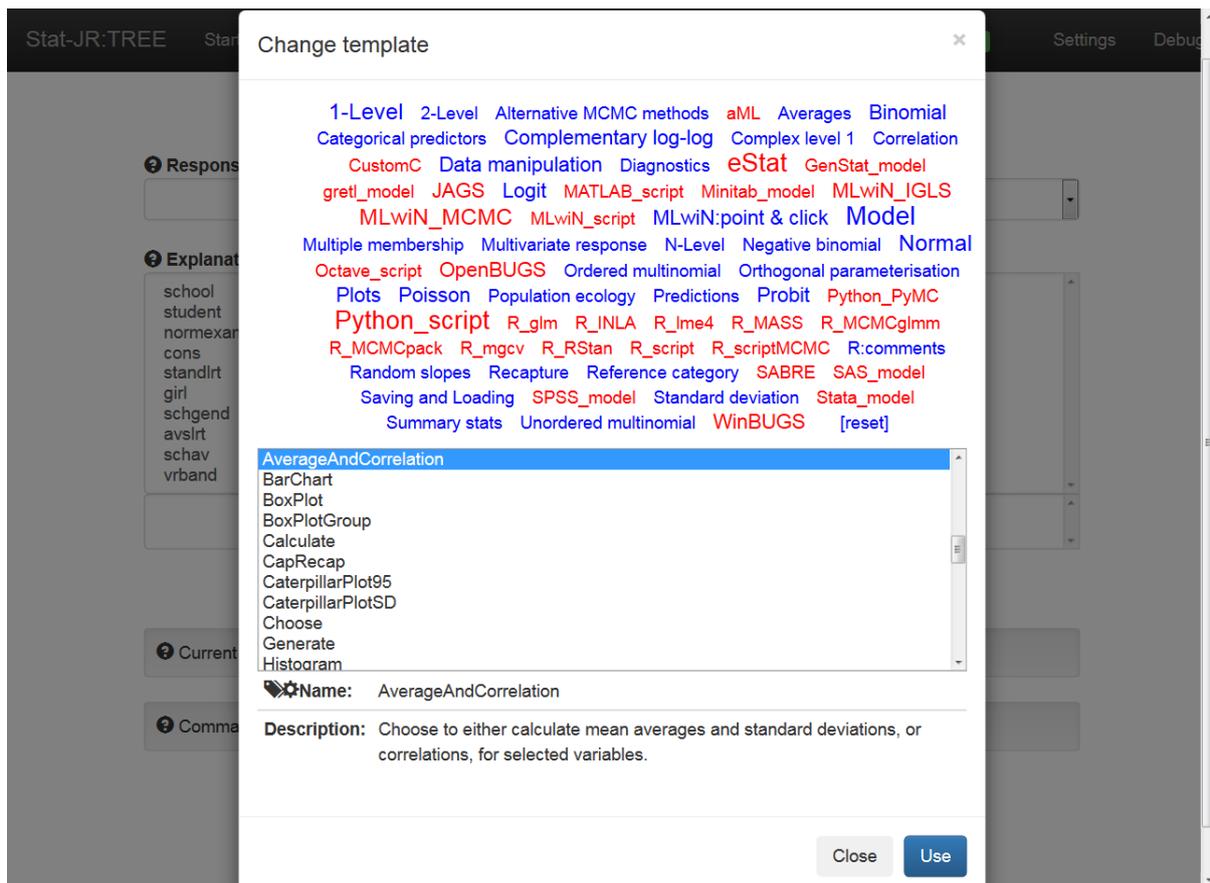


Figure 6

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*):

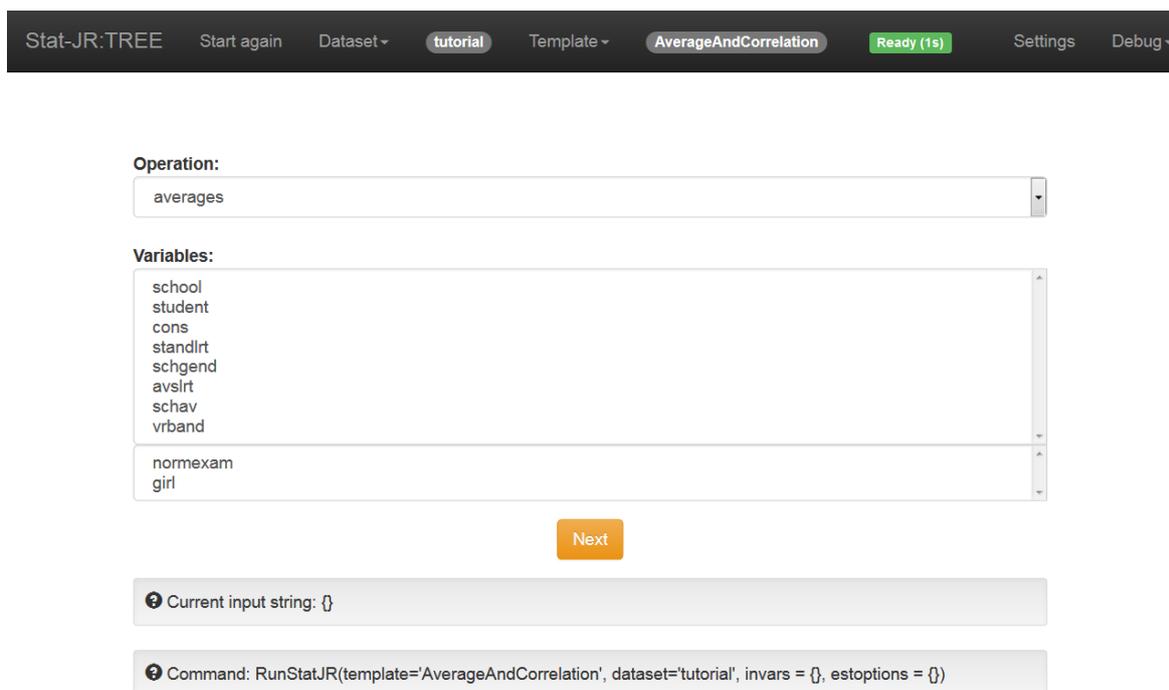


Figure 7

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](#)

[Download](#) [Add to ebook](#)

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

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

script.py [Popout](#)

```
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:

[Download](#) [Add to ebook](#)

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

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

table [Popout](#)

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* 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*

We will soon open up the workflow interface and build a simple workflow from scratch using this information, but we hope that in future you will be able to immediately save a workflow of your executions in TREE for direct translation into the workflow system.

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 = '''
```

```
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 at this stage you will see that there are two ways (via TREE, and via the template code itself) to find out the information we will need in the next section, when we write a workflow to execute the same operation.

1.5 STAT-JR Workflows

We will now open the workflow interface to Stat-JR. If you return to the main Stat-JR directory you will see that there is another executable described as *wf.cmd*. Clicking on this executable 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:

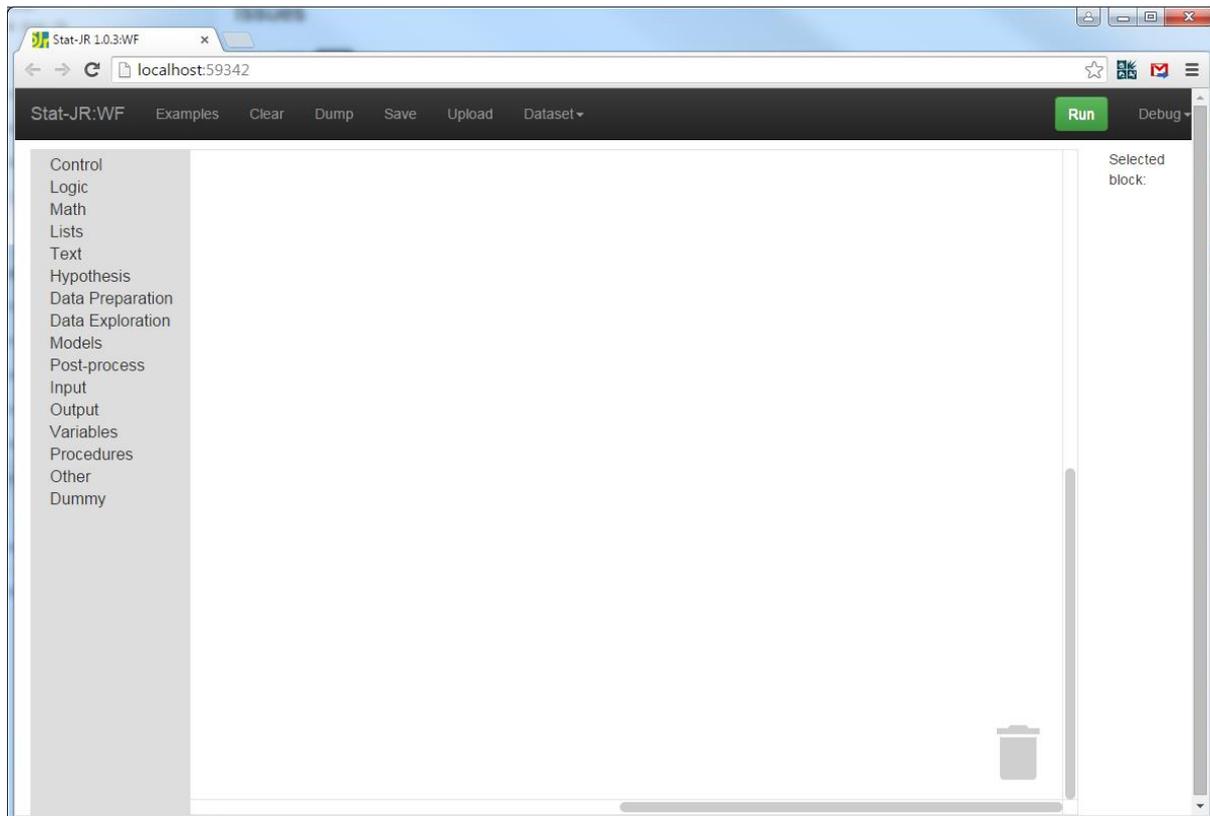


Figure 10

The workflow version of Stat-JR 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.

Here we will begin illustrating the workflow system by replicating the averaging we did in the TREE interface. The window shown above contains menus at the top and a panel to the left that contains a palette of blocks. If you click on the terms to the left you will see that blocks appear, e.g. clicking on **Control** results in the screen looking as follows:

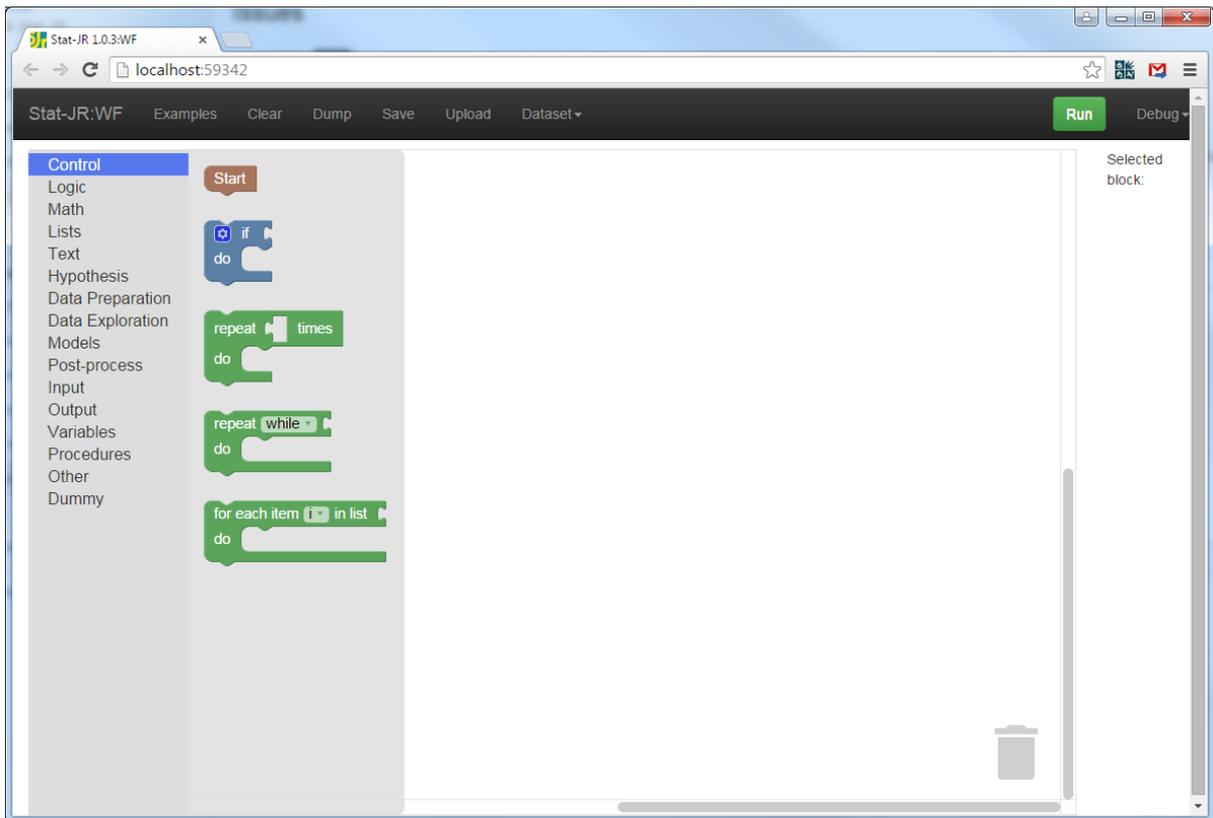


Figure 11

We will begin our workflow with the *Start* block whose simple purpose is to indicate the start of the workflow. To use it, click on it with the left mouse button and, holding down the button, drag it across to the white area; the window should now look thus:

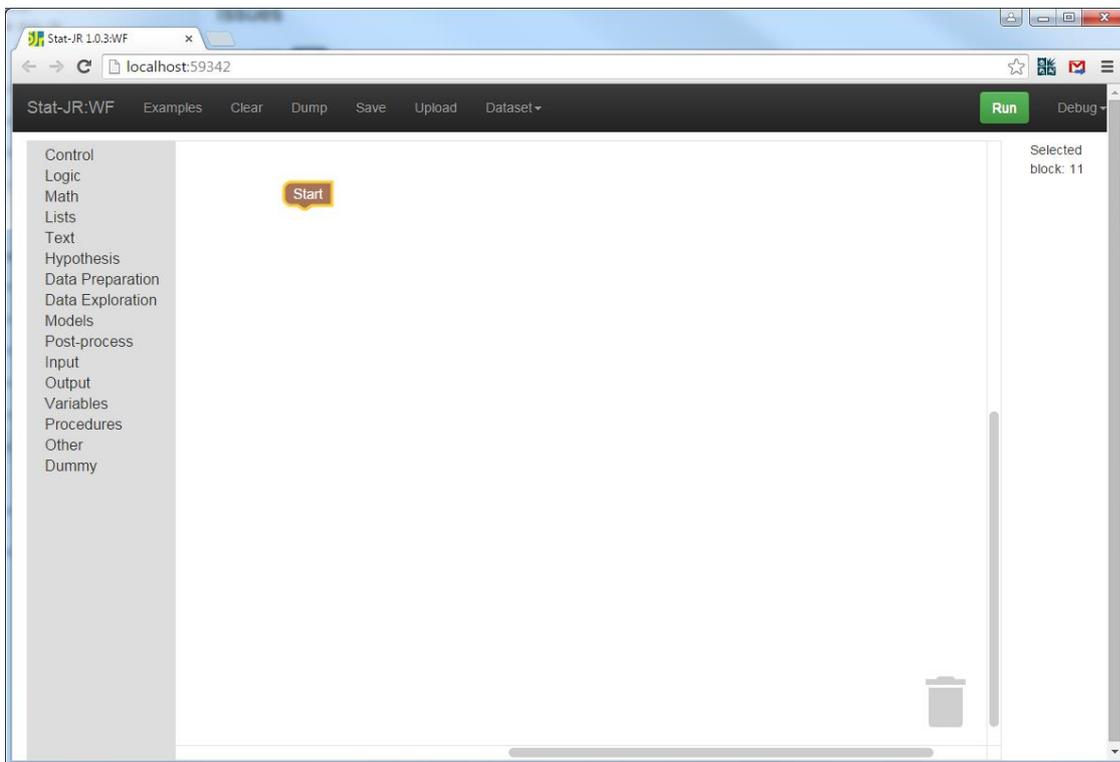


Figure 12

So we have a start (literally) but we need a dataset. In this example we will use the *tutorial* dataset we used earlier in TREE, but you can use a dataset of your own choice if you prefer. If you click on **Data Preparation** in the left-hand menu, you will see a block entitled *Select dataset*. Click on this block and drag it to below the *Start* block, so it will be the next block used in the workflow (workflows run sequentially downwards). It should join to it with a satisfying clicking noise (if your speakers are on), and visually ‘snap’ into place. The screen will look thus:

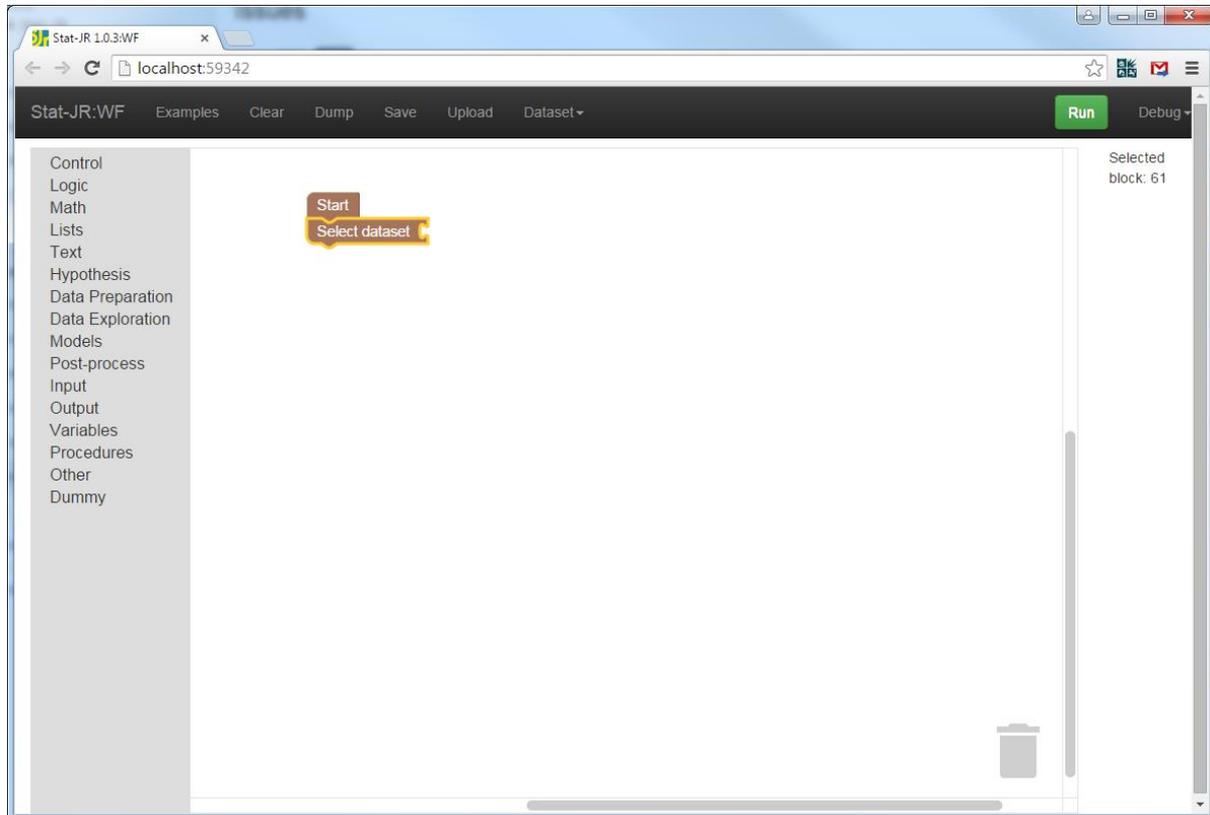


Figure 13

In fact, one of the strengths 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. The *Select dataset* block you have just introduced, for example, 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 try to fit the *not* block (from the **Logic** menu) into this slot, you’ll see it resists, like trying to join like poles of two magnets:

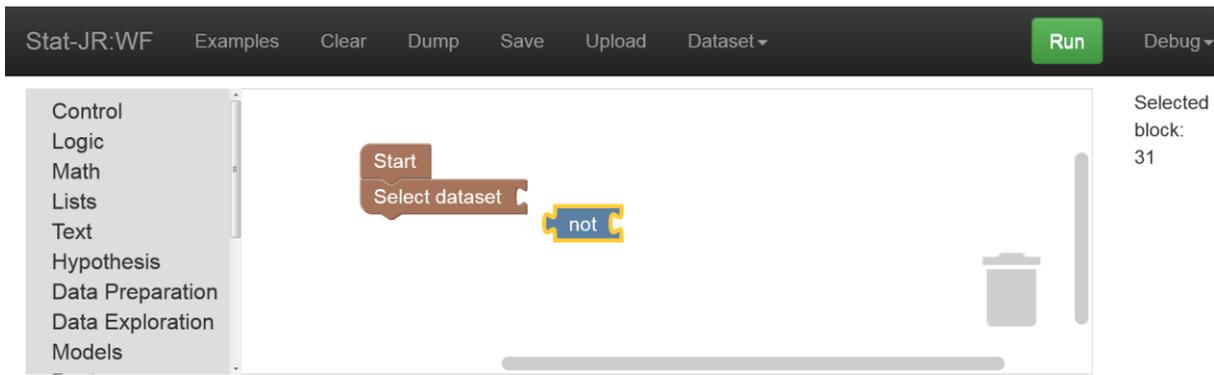


Figure 14

Clearly, this is the wrong sort of block (if you wish to tidy it away, you can select it and press the *Delete* button on your keyboard, or 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!) As it happens, what we need is a text block in which we can write the name of our dataset. If you click on **Text** in the left-hand menu, you will see a list of blocks; select the first one, which is a blank text string, and drag it so that it slots (successfully this time) into the *Select dataset* block thus:

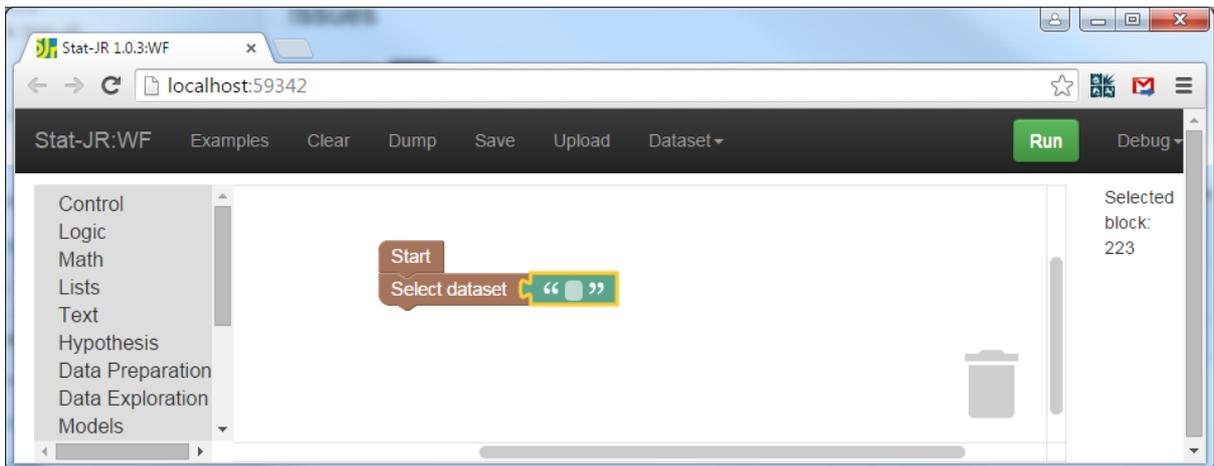


Figure 15

Next we can type in the name of our dataset of choice, in our example *tutorial*:

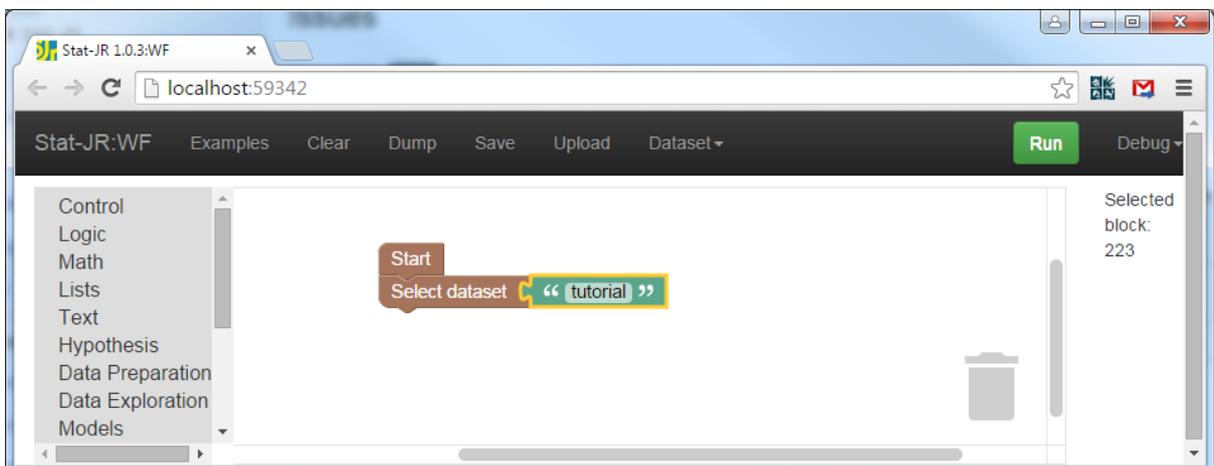


Figure 16

We will next include inputs for our template (we will nominate the particular template we wish to user later); we do this via the *Set Input* block which can be found in the **Models** menu, resulting in the following:

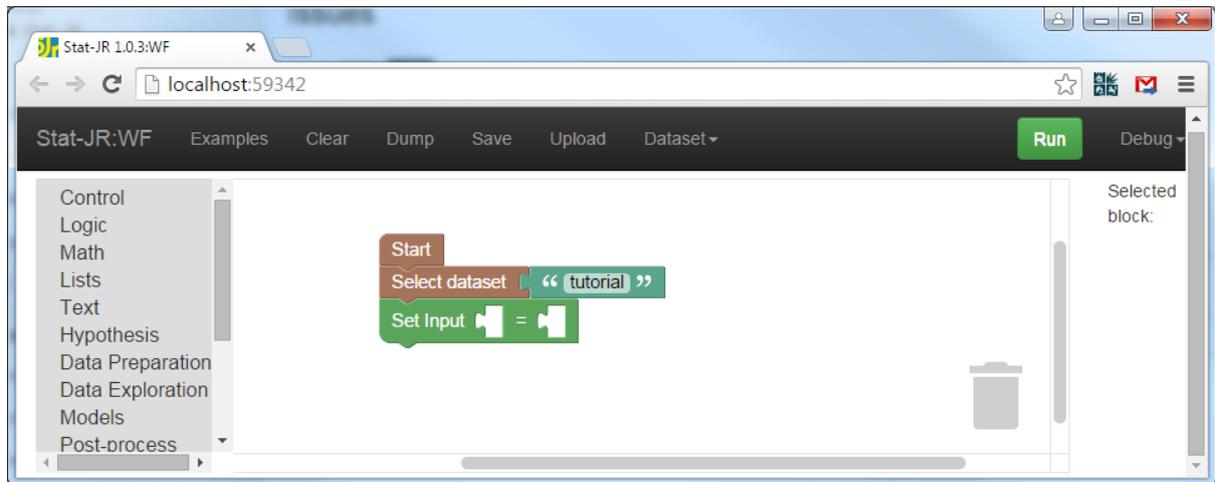


Figure 17

You will see that the *Set Input* block has two gaps: here we need to add blank text blocks from the **Text** list to the left thus:



Figure 18

In fact, we have two inputs to set up so this gives us the opportunity to show another feature of the workflow system. If you right-click on the *Set Input* block you can choose **Duplicate** from the menu that appears and a copy of the block (with embedded text blocks) appears (alternatively you can select the block(s) you wish to duplicate and press Ctrl-C then Ctrl-V to copy and paste). If you attach this to the workflow the screen will look as follows:

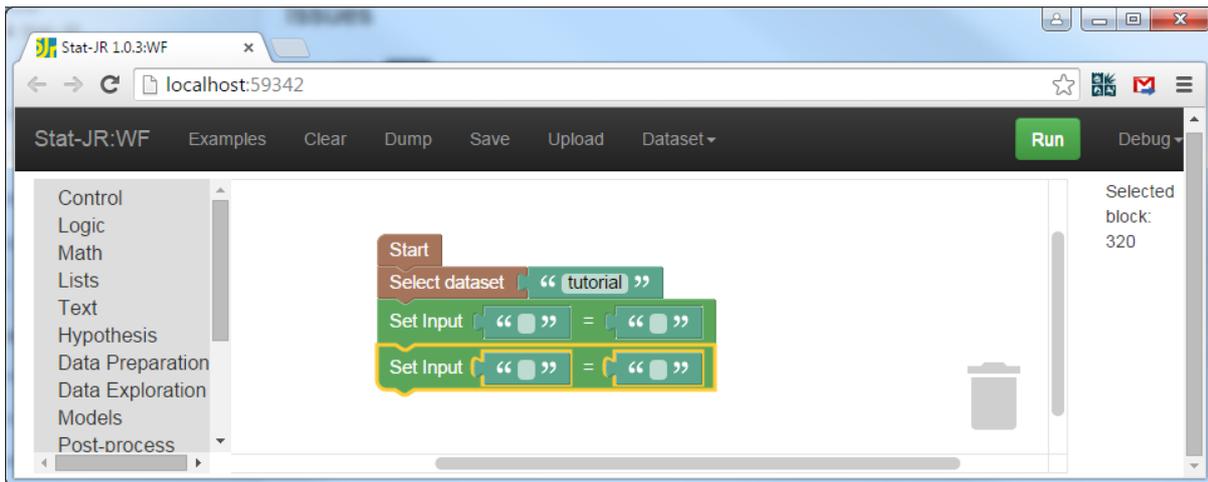


Figure 19

We can now fill in the inputs we noted earlier, namely “*op*” which takes value “*averages*”, and “*vars*” which takes value “*normexam, girl*”, resulting in the workflow now looking as follows:

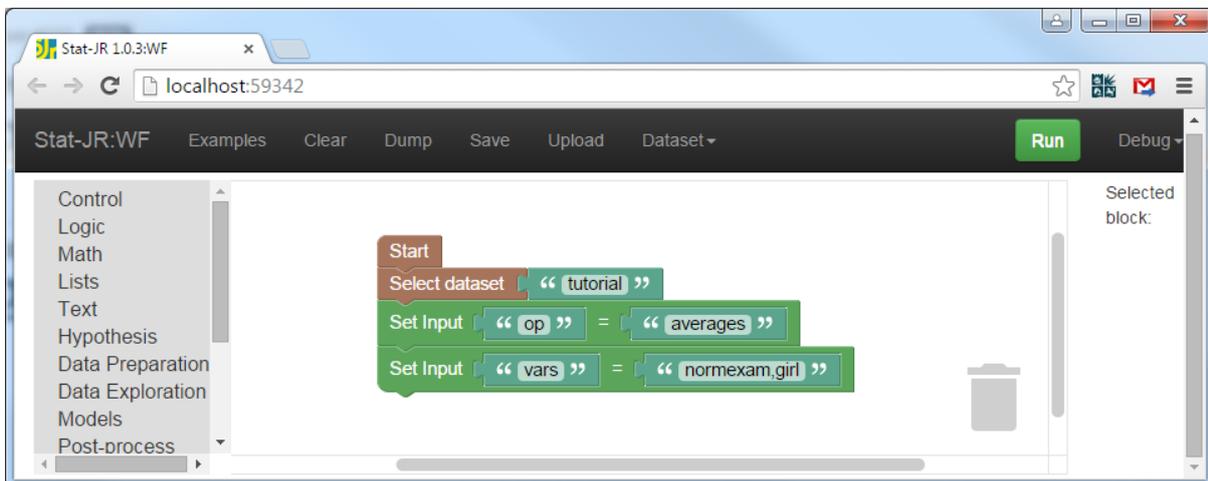


Figure 20

Next we add the template we wish to execute with these inputs; this currently requires the selection of the black *Template* block from the **Dummy** menu. This will run whatever template is named in a block appended to the right of it, and so we add a text block to this with the name *AverageAndCorrelation* thus:

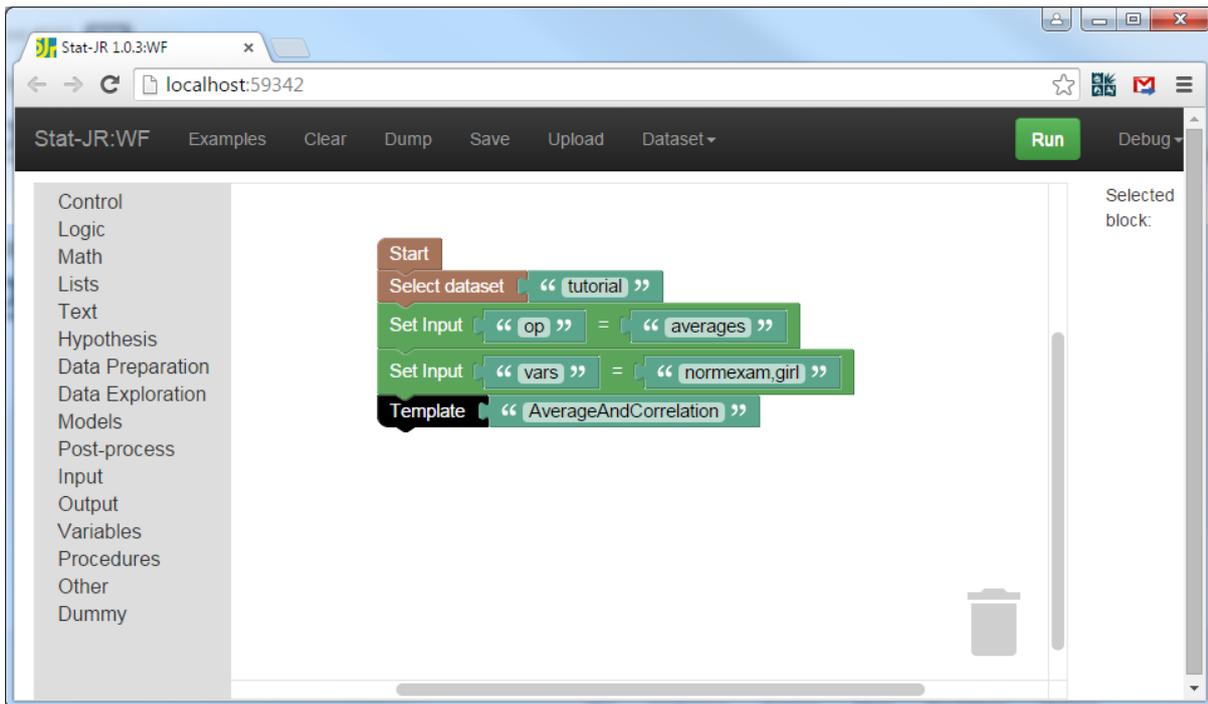


Figure 21

Finally we want to display an output resulting from the template’s execution. This is done via the **Show** block from the **Output** menu. Put this on the end of the workflow, together with an embedded text block in which we will type our output name (“table”) thus:

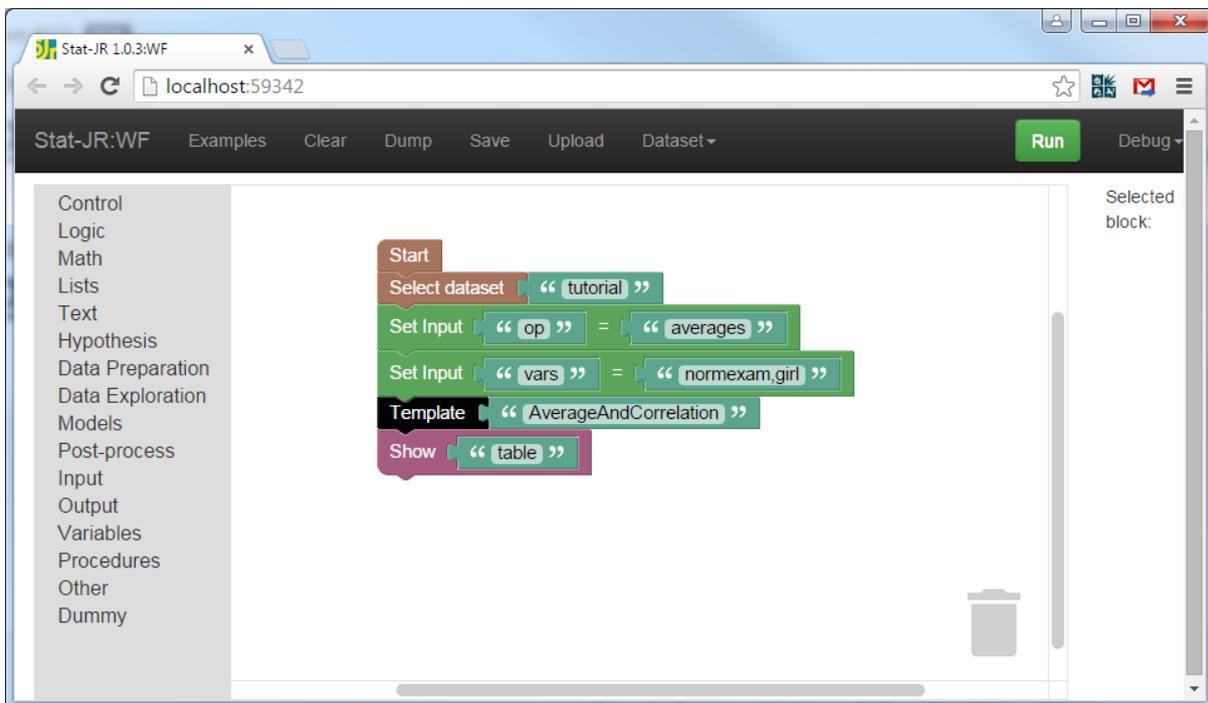


Figure 22

This is our complete workflow and at this point it would be good to save it, so click on **Save** and choose a name (we will name it after this section of the practical, and choose *prac1_5.xml*) thus:

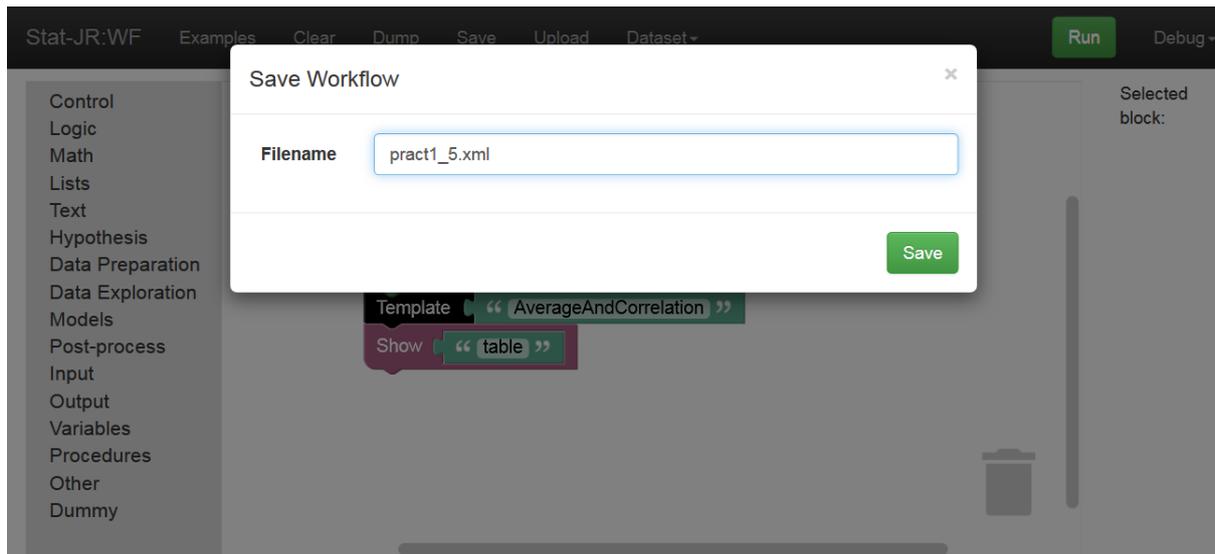


Figure 23

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

Now, clicking on the **Run** button will execute the workflow which will bring up another tab in the browser; in our example it looks as follows:

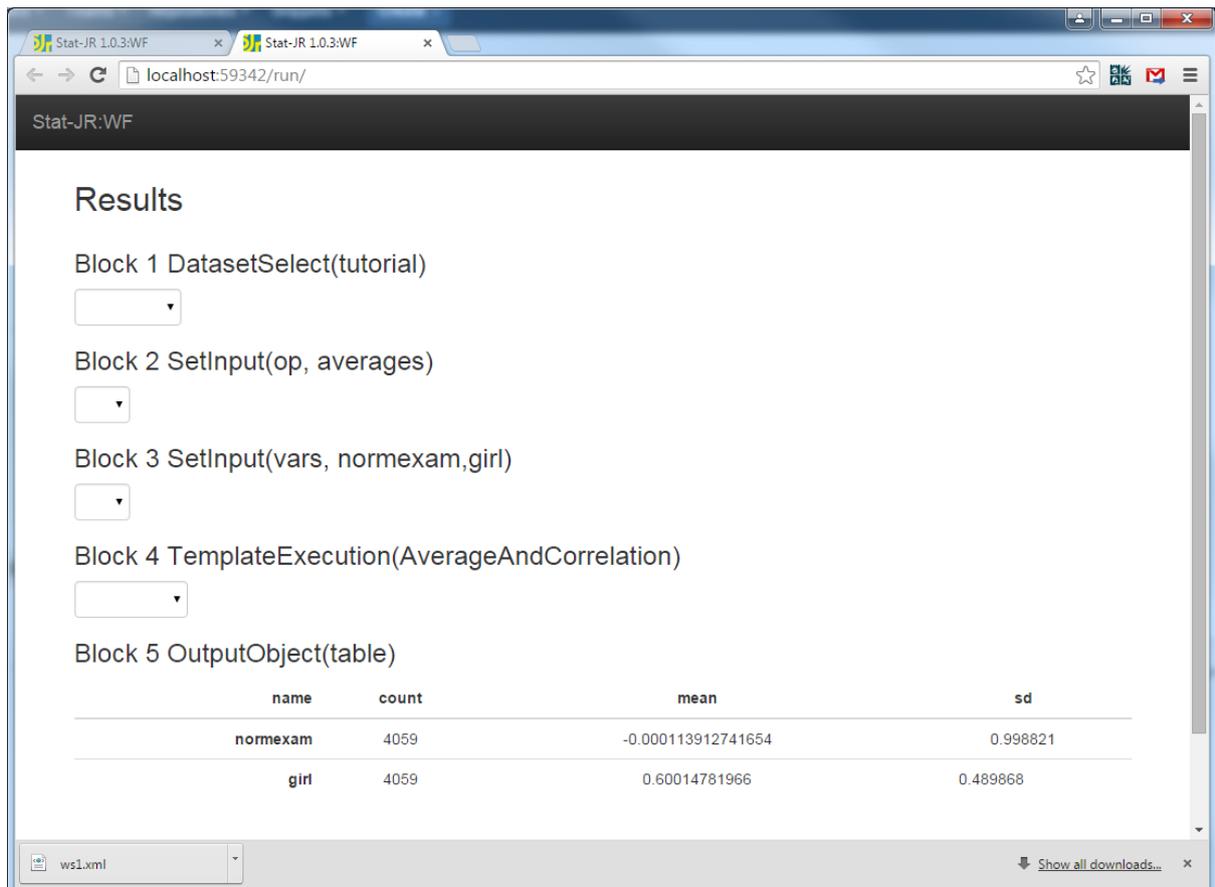


Figure 24

The current output from workflows is a little crude: essentially we get a list starting with “Block 1” and numbered through to “Block 5”, corresponding to the five blocks (counting vertically,

downstream from the *Start* block) in the workflow. The final *Show* block (Block 5) returns our requested table: this is the same as the output we saw in TREE.

1.6 Making our workflow interactive

As things stand we have made 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. Now click on the textual block that contains *normexam*, *girl* and holding the mouse button drag it to the waste bin in the bottom right of the pane. The screen will now look as follows:

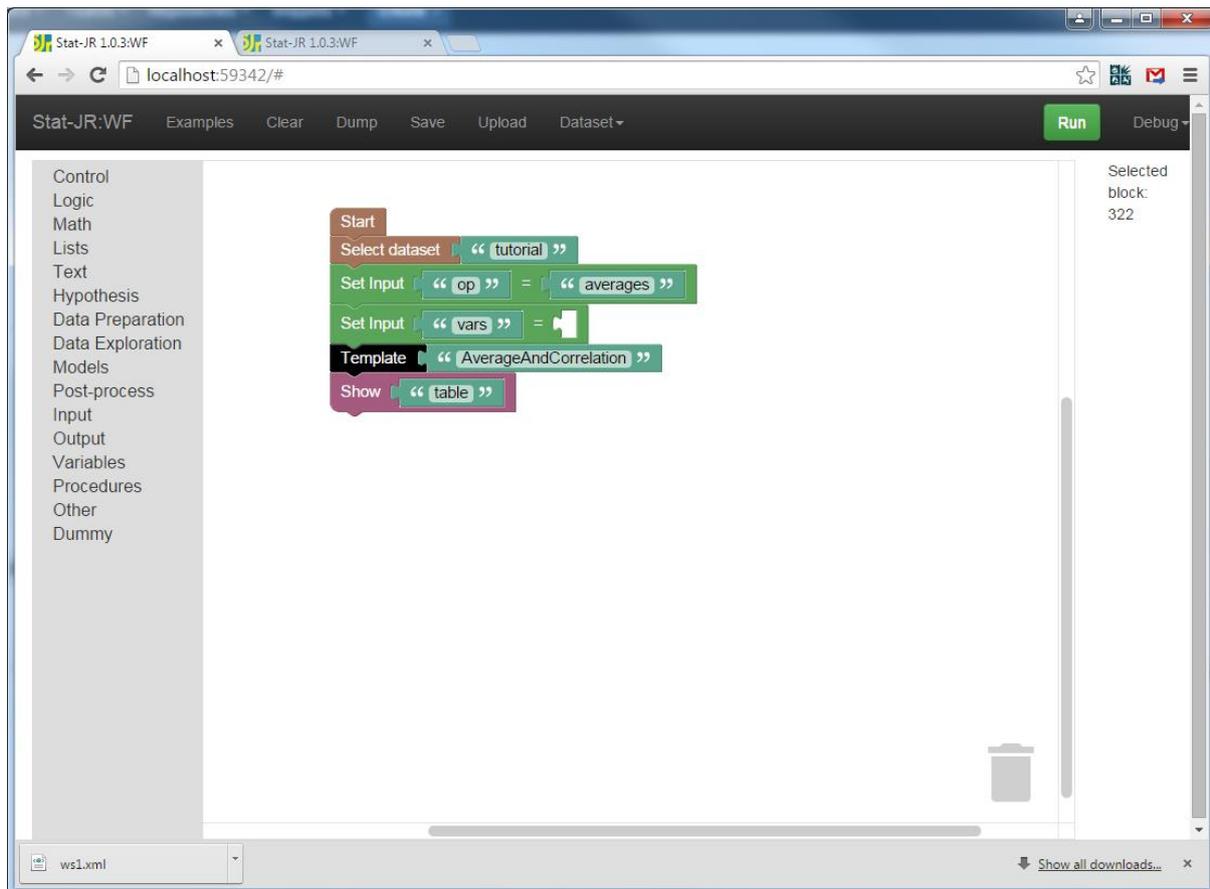


Figure 25

There is a gap in the inputs and this time clicking on **Run** the workflow stops at that point thus:

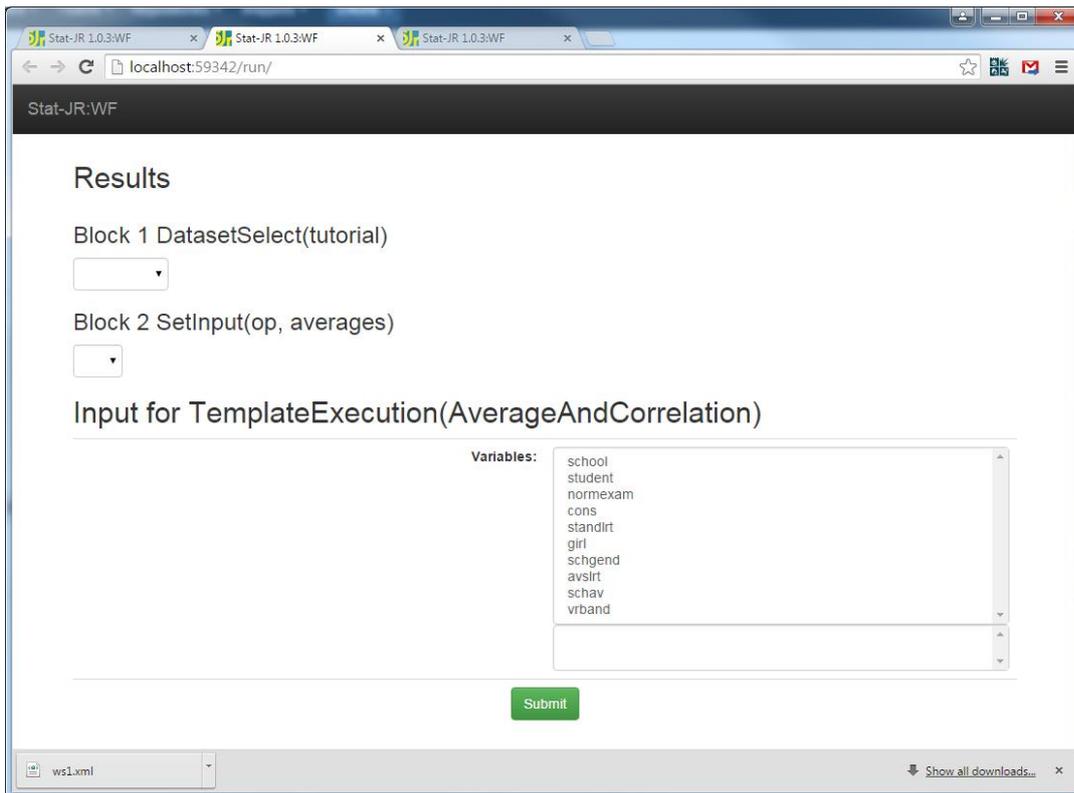


Figure 26

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

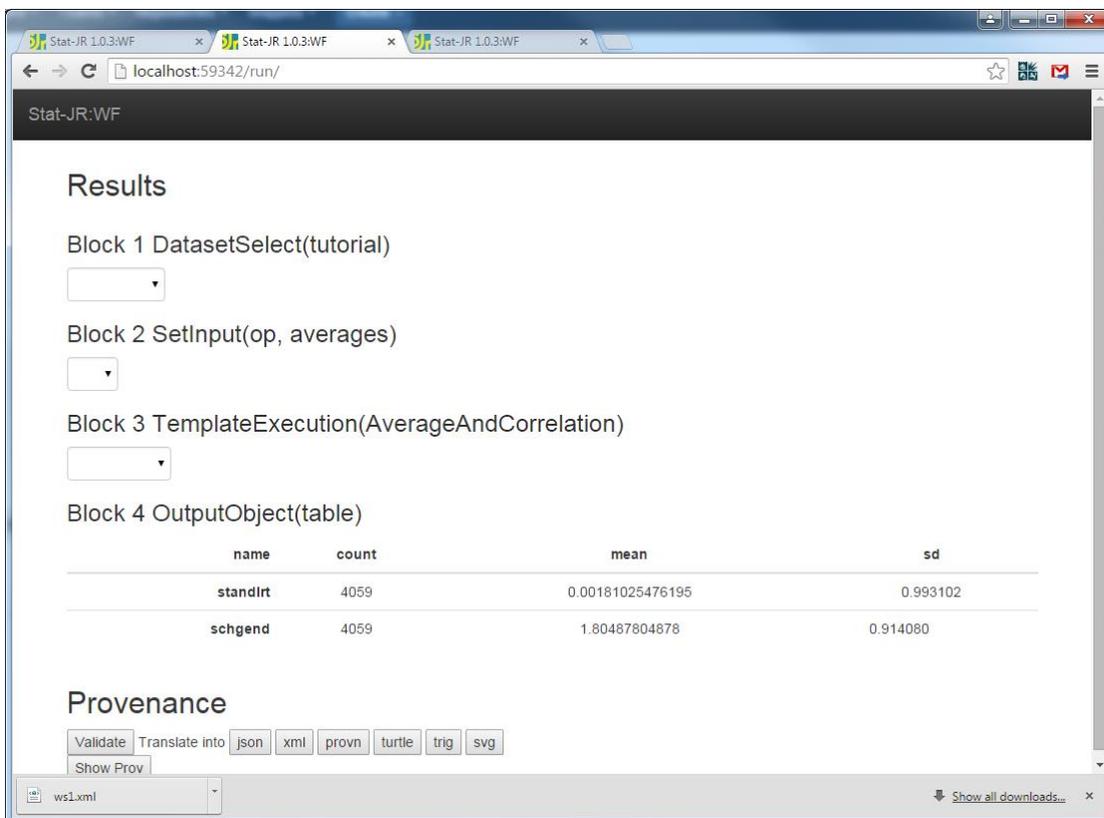


Figure 27

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 leaving the slot in the *Set Input* block empty, we can instead add a question block. So from the **Input** list of blocks select the *Ask multiple variables*² 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 thus:

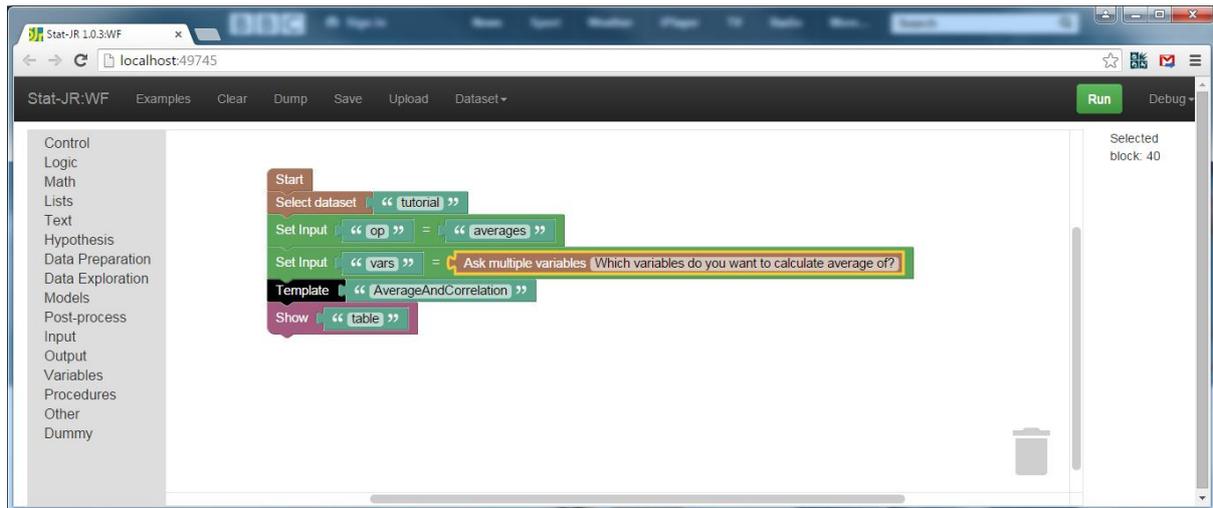


Figure 28

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

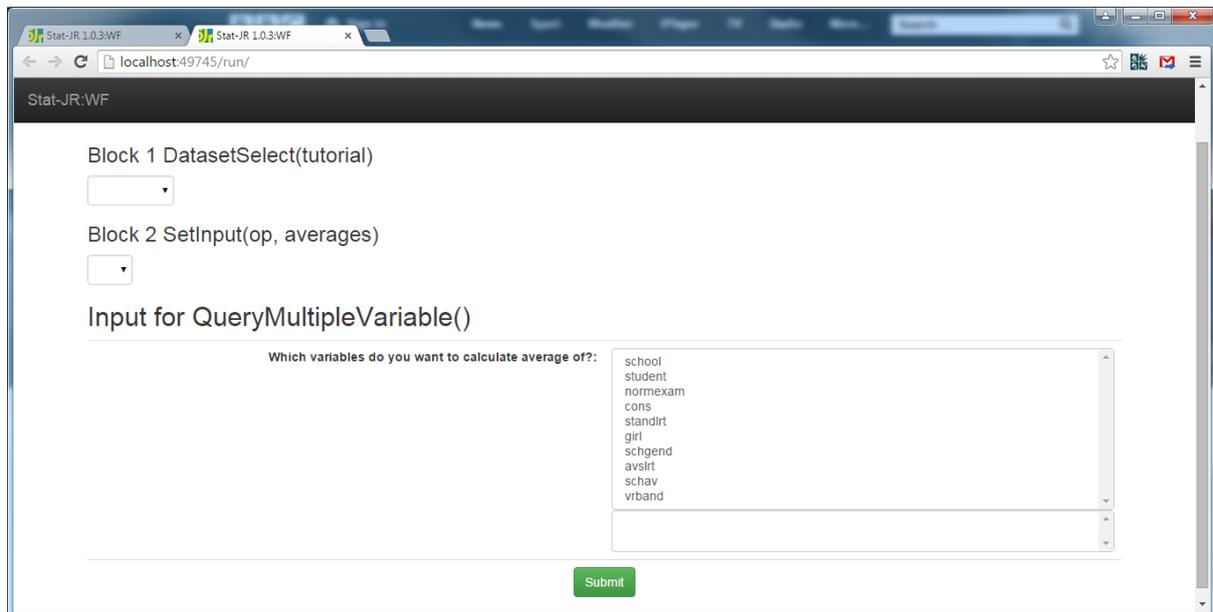


Figure 29

² 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.

Here if we answer the question we will once again get a fourth block showing the means and standard deviations for the selected variables.

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. Again we will first do this in TREE before moving across to the workflow system.

If you don't have TREE still active you will need to restart it. In the main TREE window we will need to choose the *Histogram* template from the list, so click on the *Template* list and click on **Choose** and highlight *Histogram* as shown:

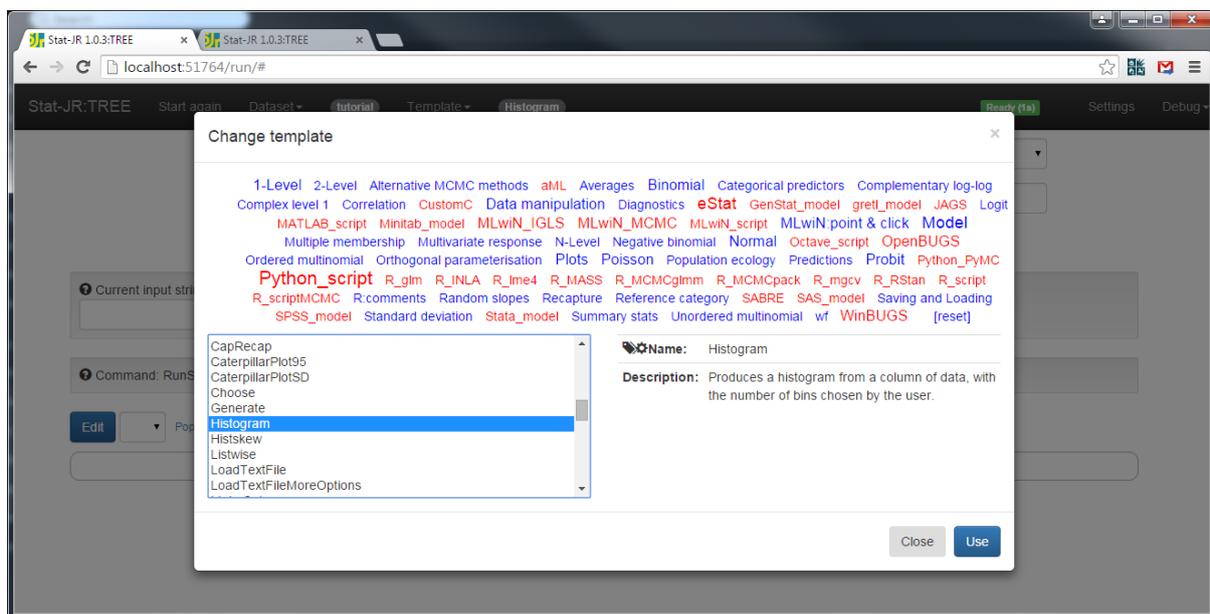


Figure 30

Now click on **Use** and the inputs for the *Histogram* template will appear. In our example we will choose *normexam* as the *Values* for which we wish to plot a histogram, and *15* for the *Number of bins*. Clicking on **Next** and **Run**, and selecting *histogram.svg* from the object list, gives the following screenshot:

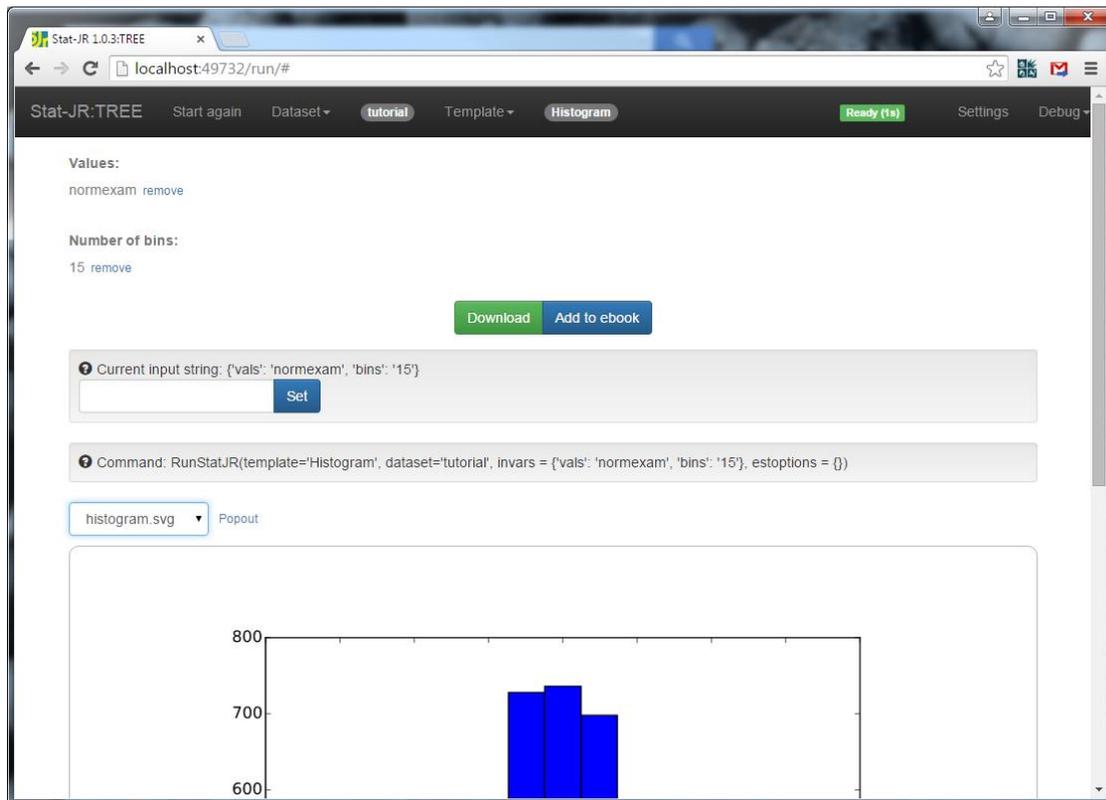


Figure 31

So here we have the inputs that we require (*vals* and *bins*) along with the output that we want to show (*histogram.svg*; again you could glean this information by looking at the template code itself in the *templates* directory if you so wished).

We will now return to the workflow system with our workflow for the averages still visible. Rather than start from scratch we will break up the current workflow by clicking on the *Set Input* block for “*op*” and moving it and the following blocks to the right so that the screen looks as follows:

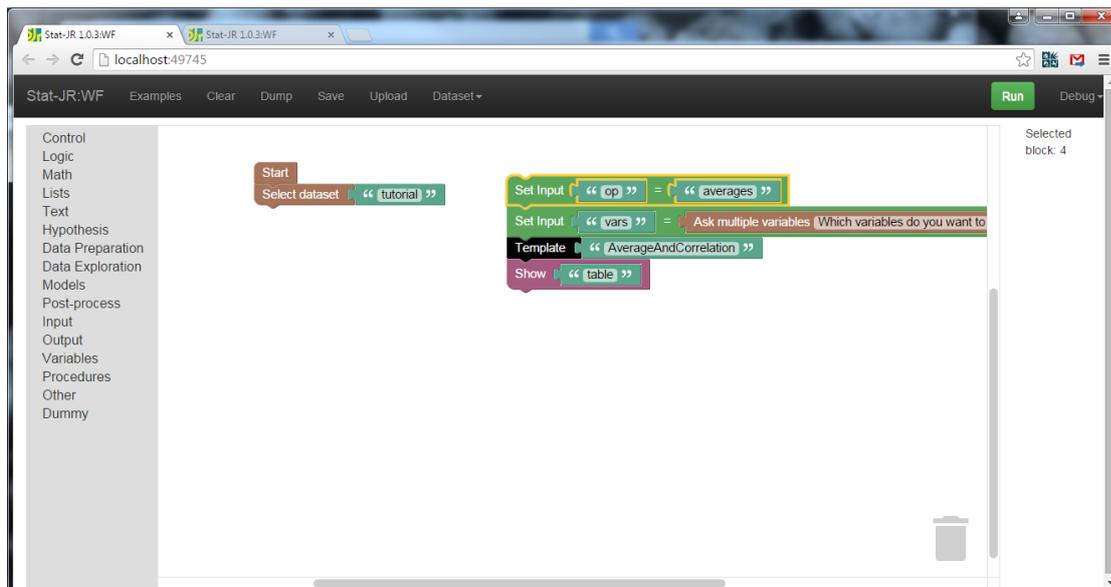


Figure 32

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. We will next add the blocks to produce the histogram. These will look similar to those for the first workflow as we will need two *Set Input* blocks (from the **Models** list), one *Template* block (from the **Dummy** list) and one *Show* block (from the **Output** list) along with several text blocks (from the **Text** list; alternatively we can duplicate blocks we already have elsewhere in the central workflow pane, and modify as appropriate).

We show below the workflow and so see if you can replicate it for yourself:

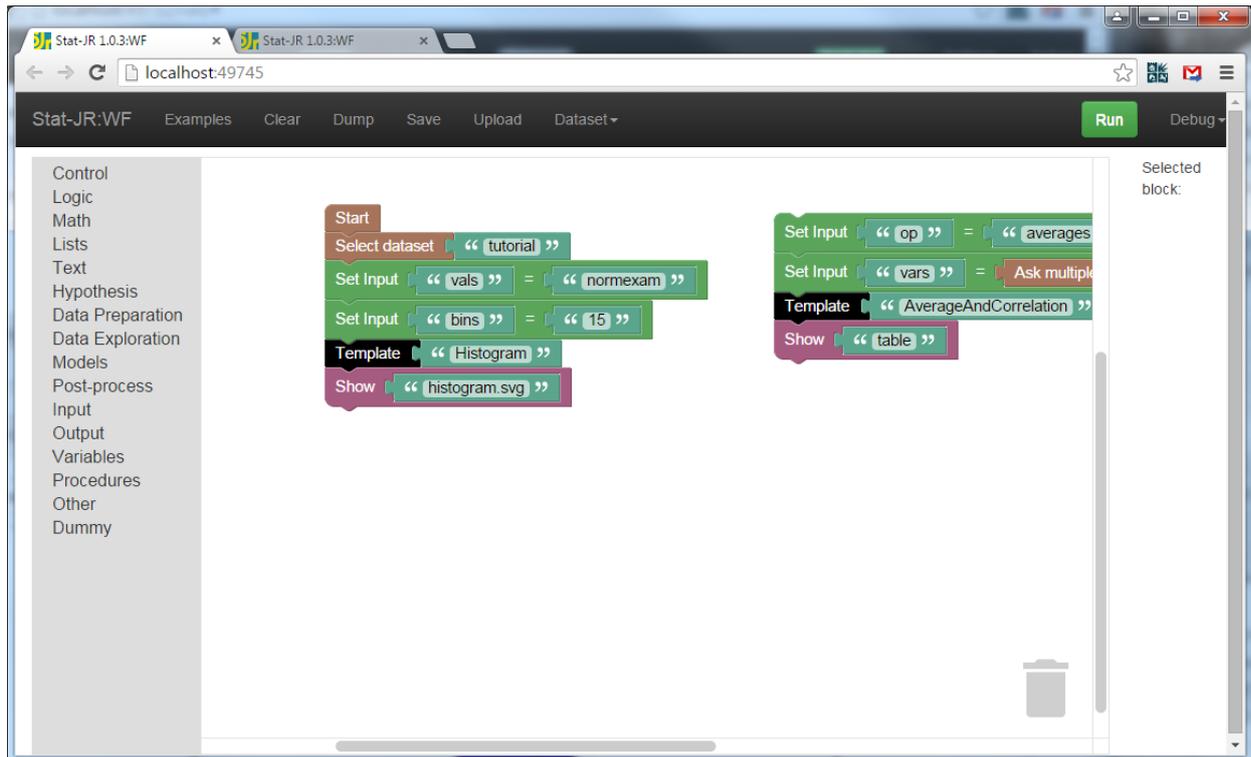


Figure 33

To test it out we click on the **Run** button and get the following:

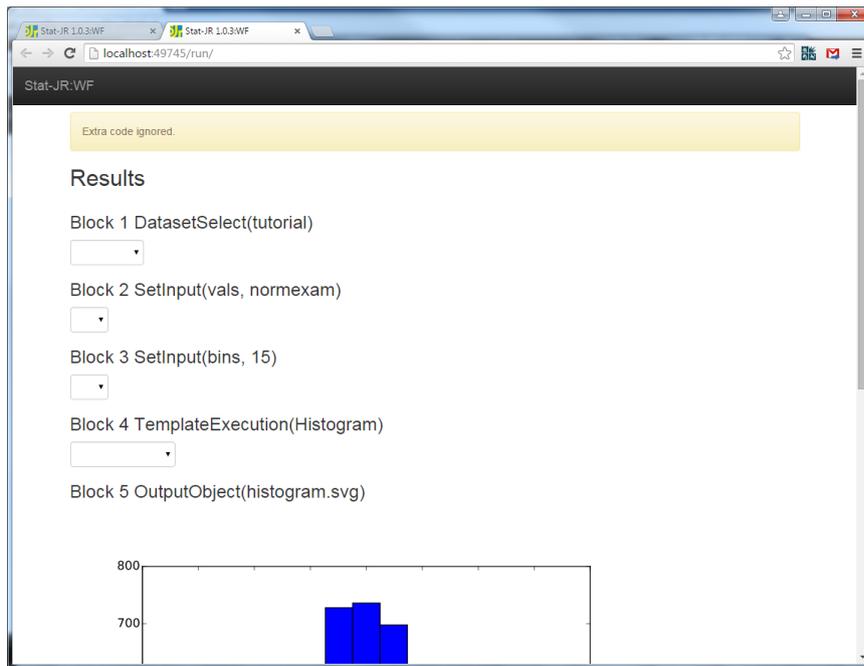


Figure 34

Here the initial comment that “Extra code is ignored” is simply the system telling us that there are blocks on the workflow screen that aren’t currently part of the workflow (i.e. it’s detected that we detached some from the active workflow headed by the *Start* block). We then see, in block five, the histogram we requested. It would be good to save your workflow at this point, so return to the main workflow window and click on **Save**. This time save it as *prac1_8.xml*.

1.9 Connecting up the operations

We have now created two workflows and an obvious next step is to join them together. Fortunately we have both workflows on the screen and so we can quite easily join them. Have a go at doing this yourself to produce the following:

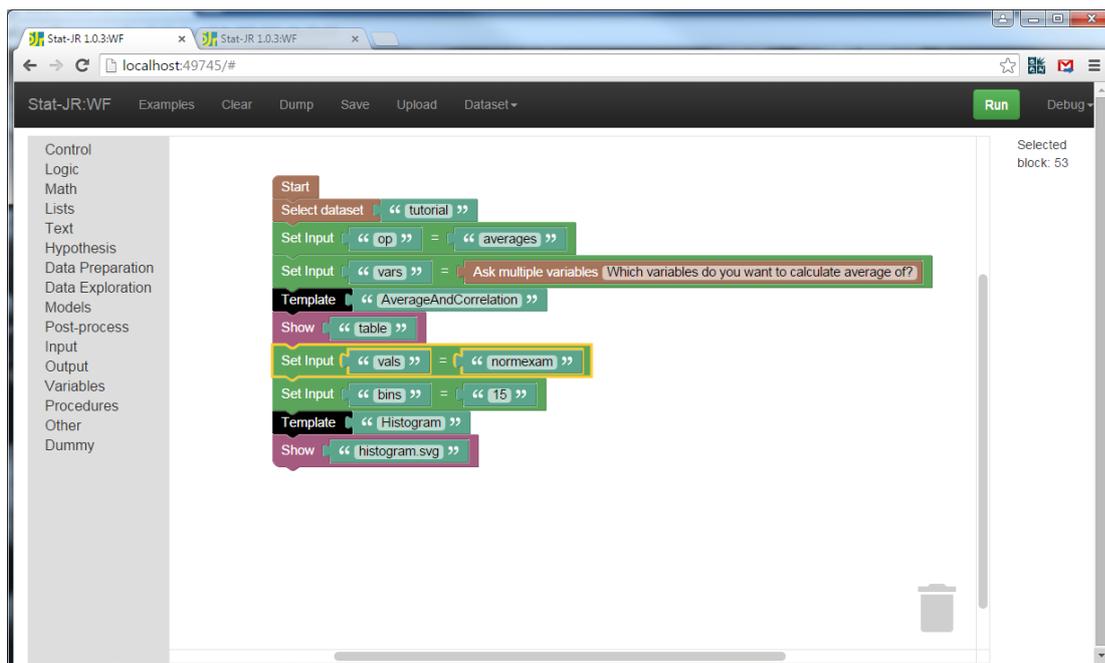


Figure 35

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

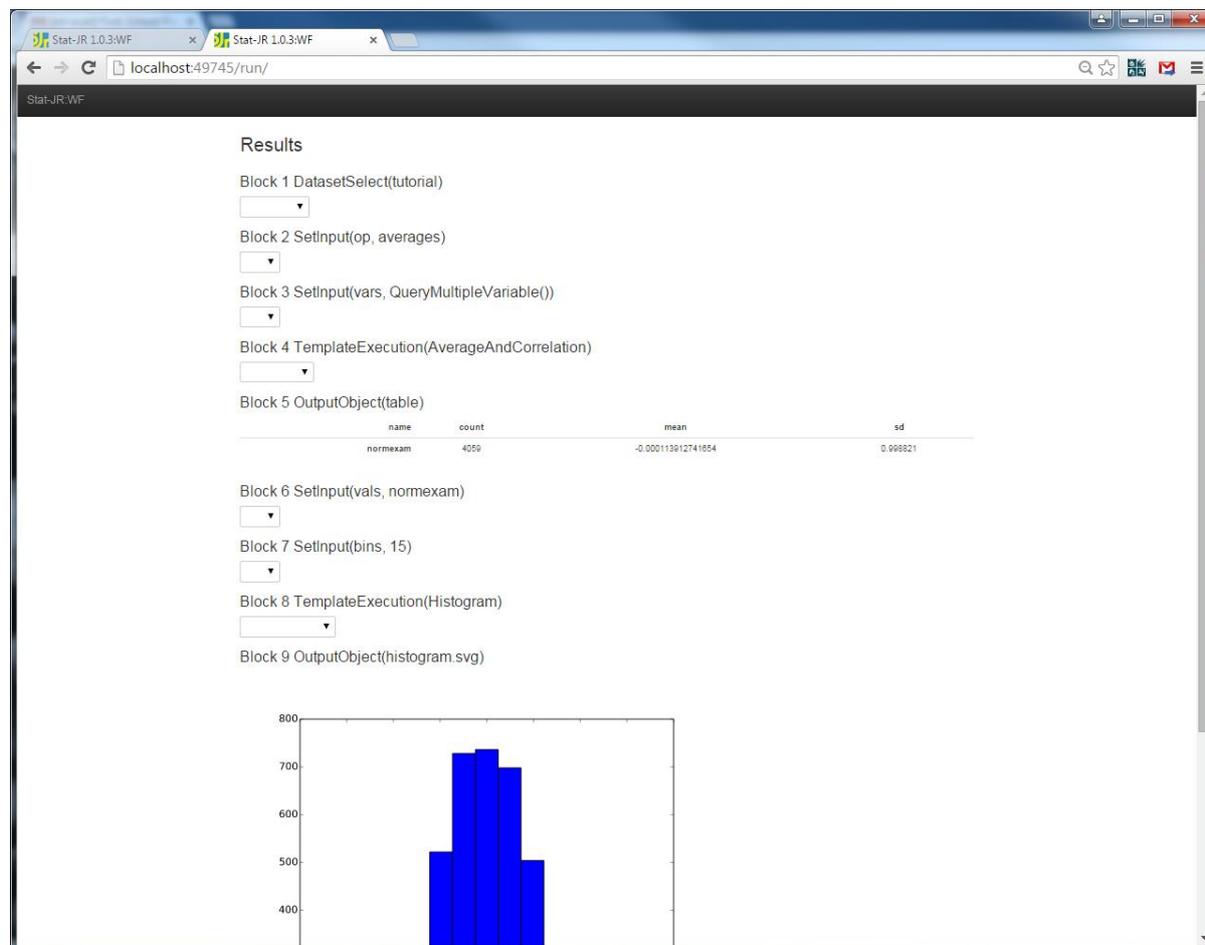


Figure 36

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 to replicate your analysis. As mentioned earlier, in the future we hope to add a feature into TREE so that this is done automatically.

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 lists 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):

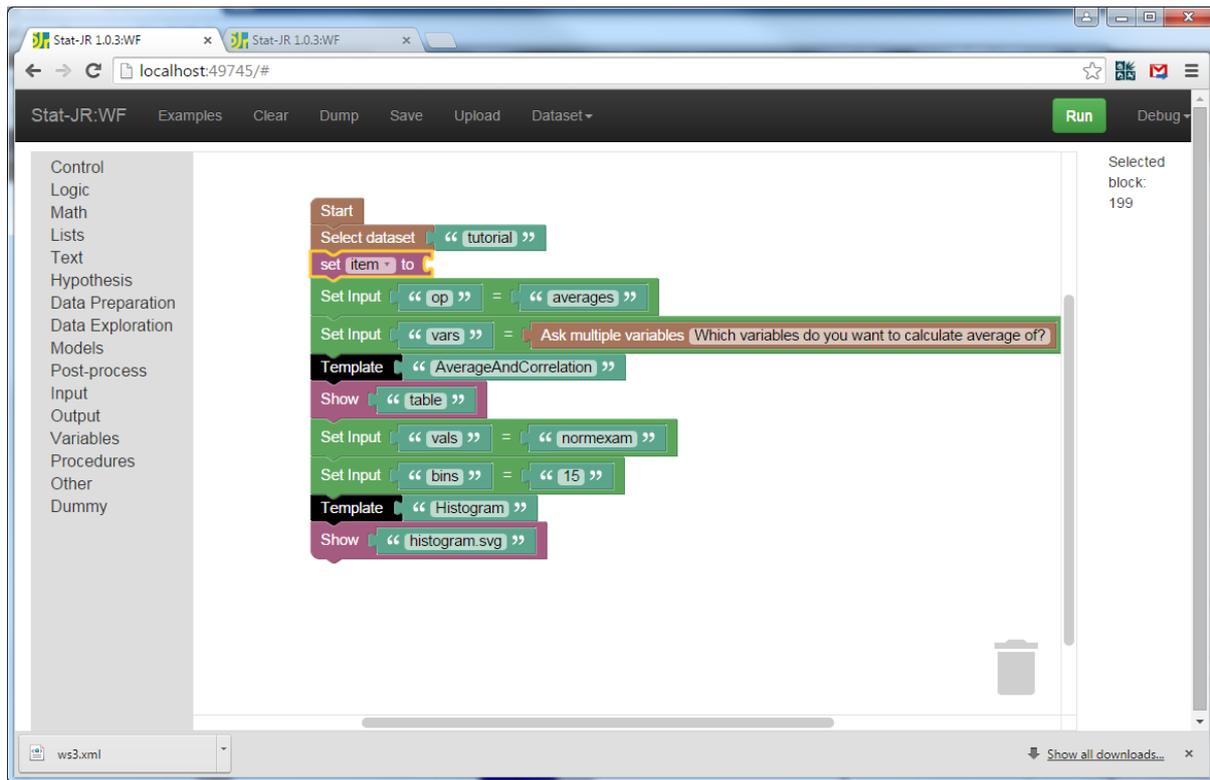


Figure 37

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*:

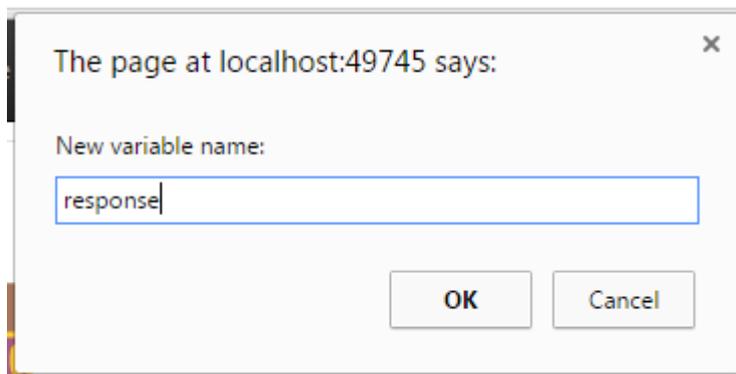


Figure 38

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 as shown below:

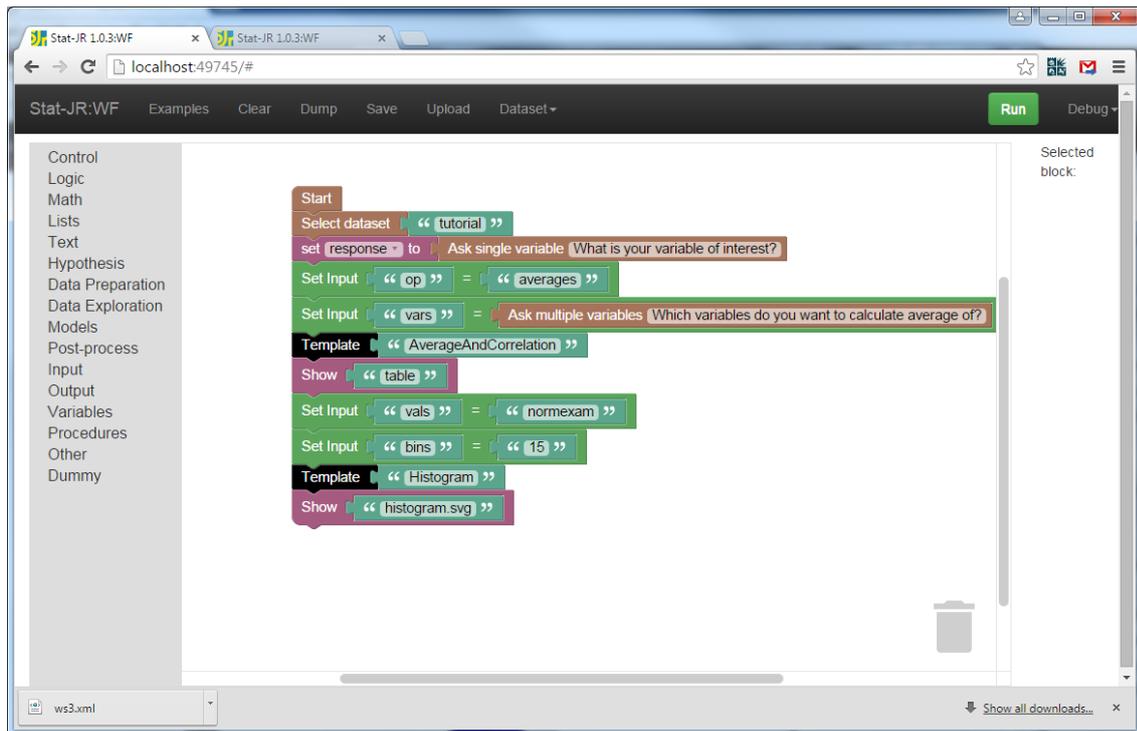


Figure 39

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). The completed workflow looks as follows:

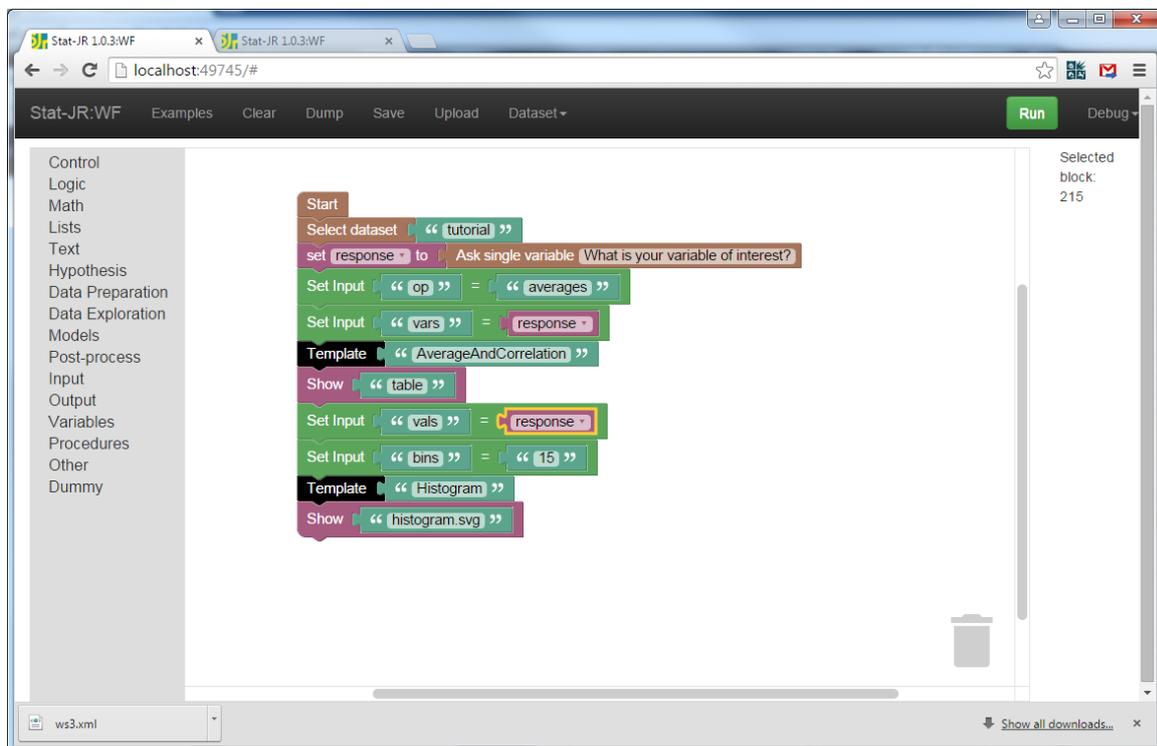


Figure 40

Hopefully you managed to find the block you needed.³ We can now save this workflow as *prac1_10.xml* before clicking on the **Run** button to run the workflow. In our example we've chosen *avslrt* in answer to the question:

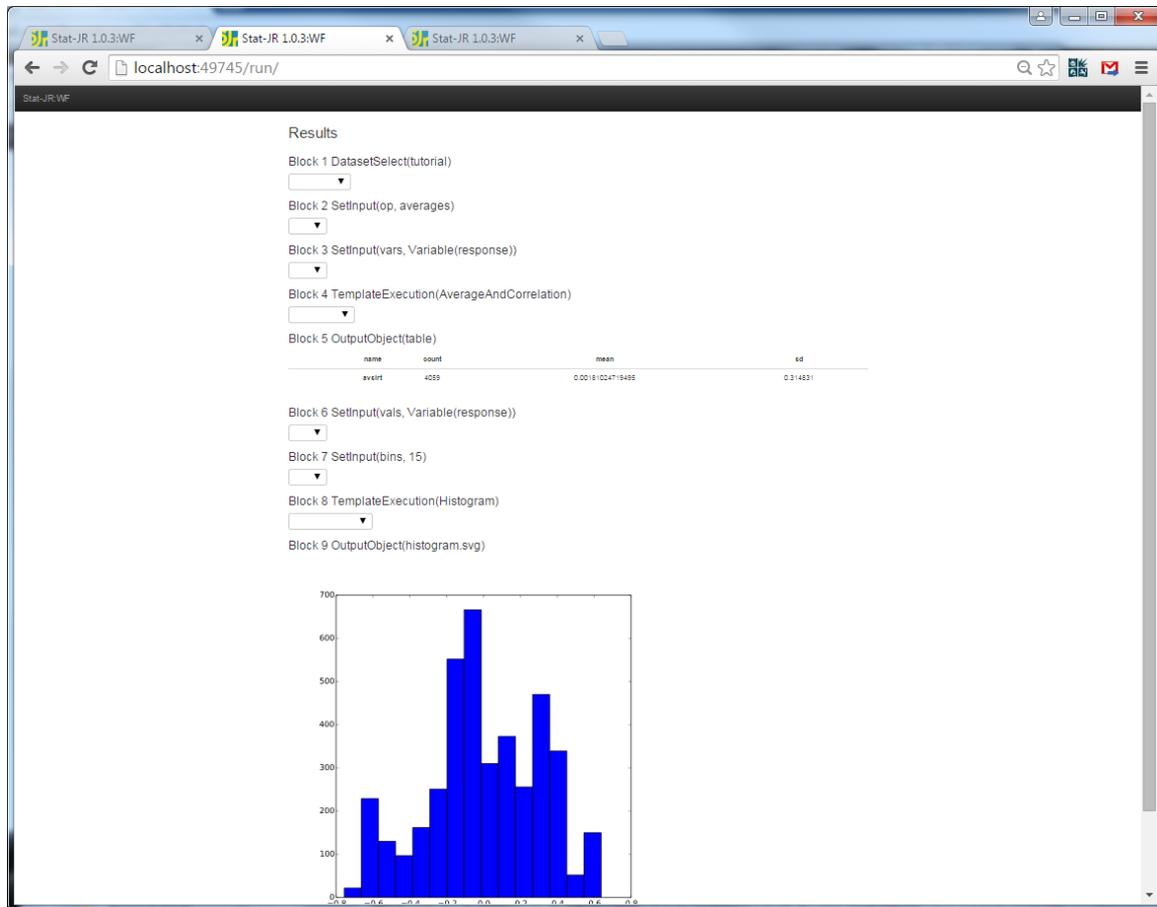


Figure 41

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. 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*.

³ 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, having selected the template *Generate* in TREE, we request our constant of ones as follows:

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

Output column name: intercept remove

Type of number to generate: Constant remove

Value: 1 remove

Name of output dataset: my_dataset remove

Download Add to ebook

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

Command: RunStatJR(template='Generate', dataset='tutorial', invars = {'outcol': 'intercept', 'outdata': 'my_dataset', 'type': 'Constant', 'value': '1'}, estoptions = {})

my_dataset Popout

my_dataset	school	student	normexam	cons	standlrt	girl	schgend	avslrt	schav	vrband	intercept
1	1	1	0.261324	1	0.619059	1	1	0.166175	2	1	1.0
2	1	2	0.134067	1	0.205802	1	1	0.166175	2	2	1.0
3	1	3	-1.72288	1	-1.36458	0	1	0.166175	2	3	1.0
4	1	4	0.967586	1	0.205802	1	1	0.166175	2	2	1.0
5	1	5	0.544341	1	0.371105	1	1	0.166175	2	2	1.0

Figure 42

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; you can inspect the dataset either by selecting *my_dataset* from the pull-down list of outputs to view it in the results pane, as shown above, or by choosing **Dataset > View** once you have selected it as the current dataset via **Dataset > Choose**).

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:

Stat-JR: TREE Start again Dataset ▾ my_dataset Template ▾ Regression1 Ready (8s) Settings Debug ▾

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 remove

Run

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

Figure 43

Here we are using the default settings for our MCMC estimation procedure⁴, 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:

The screenshot displays the following data:

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 44

As before, the list of inputs can be viewed in the *Current input string* box and we will need to include all these in our workflow to replicate the regression fit⁵. There are also a lot more outputs in the pull-down list that we might like to include in the workflow via *Show* blocks too.

So first we will return to the Stat-JR workflow system and continue with our existing workflow and add blocks to the end of it as follows (we've blown up the latter part of the workflow so that you can see the details):

⁴ 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.

⁵ Note that the workflow system has a **Model Fit** block which can be used instead of the *Template* block which fills in defaults for some of the (estimation) inputs, but we will not use this here (as our general aim is to make explicit the connection between operations in TREE and the workflow system).



Figure 45

Here, then, we are again assuming we *don't* already have a constant in our dataset, and so we first add one using the *Generate* template as we did before. We then need to change the working dataset name to that of our new dataset with our constant in it. This is done by appending the *Retrieve* block (found in the **Other** menu) to the end of the *Select dataset* block. 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*) and tell it which block to take this from. We perform the latter by referencing a unique number each block is assigned – it's the black *Template* block we need to reference (the one to which "Generate" is appended) and in the example in the screenshot this is number 38. This number will likely be different for you: you can find out by selecting the block you

need to reference and seeing which number pops up in the right-hand pane (see Figure 46). These referencing numbers may change as you move blocks around and add new sections, but it will always reference correctly (i.e. if the number of a block changes, then this will be automatically updated in the *Retrieve* block itself). 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).



Figure 46

Here we have ordered the inputs in the same order as the questions in TREE (this isn't necessary for the template to execute correctly, it just helps us ensure we included them all!) We can use the *response* variable we defined earlier as the input for "y". We will **Save** this workflow as *prac1_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:WF

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.

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:

Stat-JR:WF

Block 27 TemplateExecution(Regression1)

equation.tex

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

Block 28 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_dataset* 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):

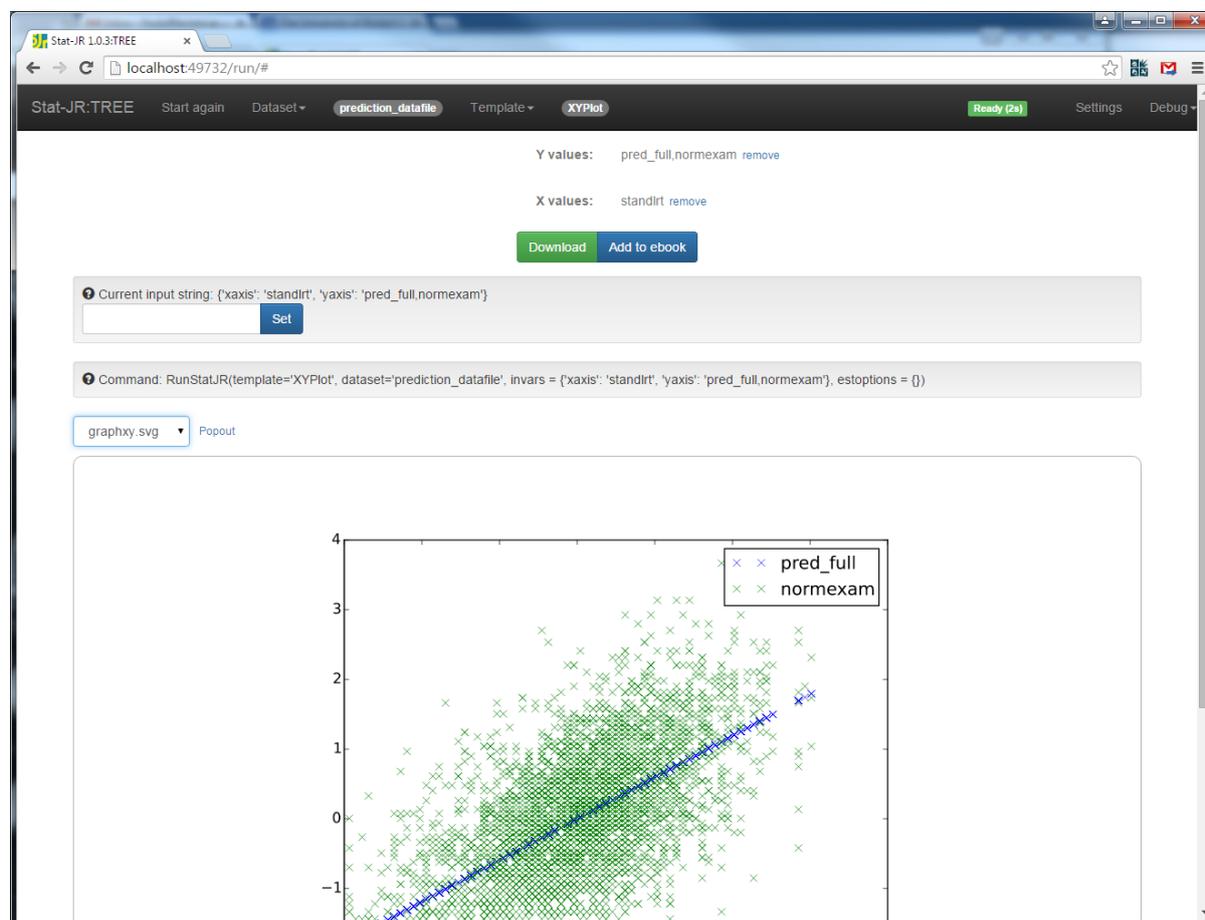


Figure 49

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:



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 – we will leave these to you to add (remember to choose the corresponding template too; if in doubt, see Figure 52 in the Appendix). **Save** the resulting workflow as *prac1_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:

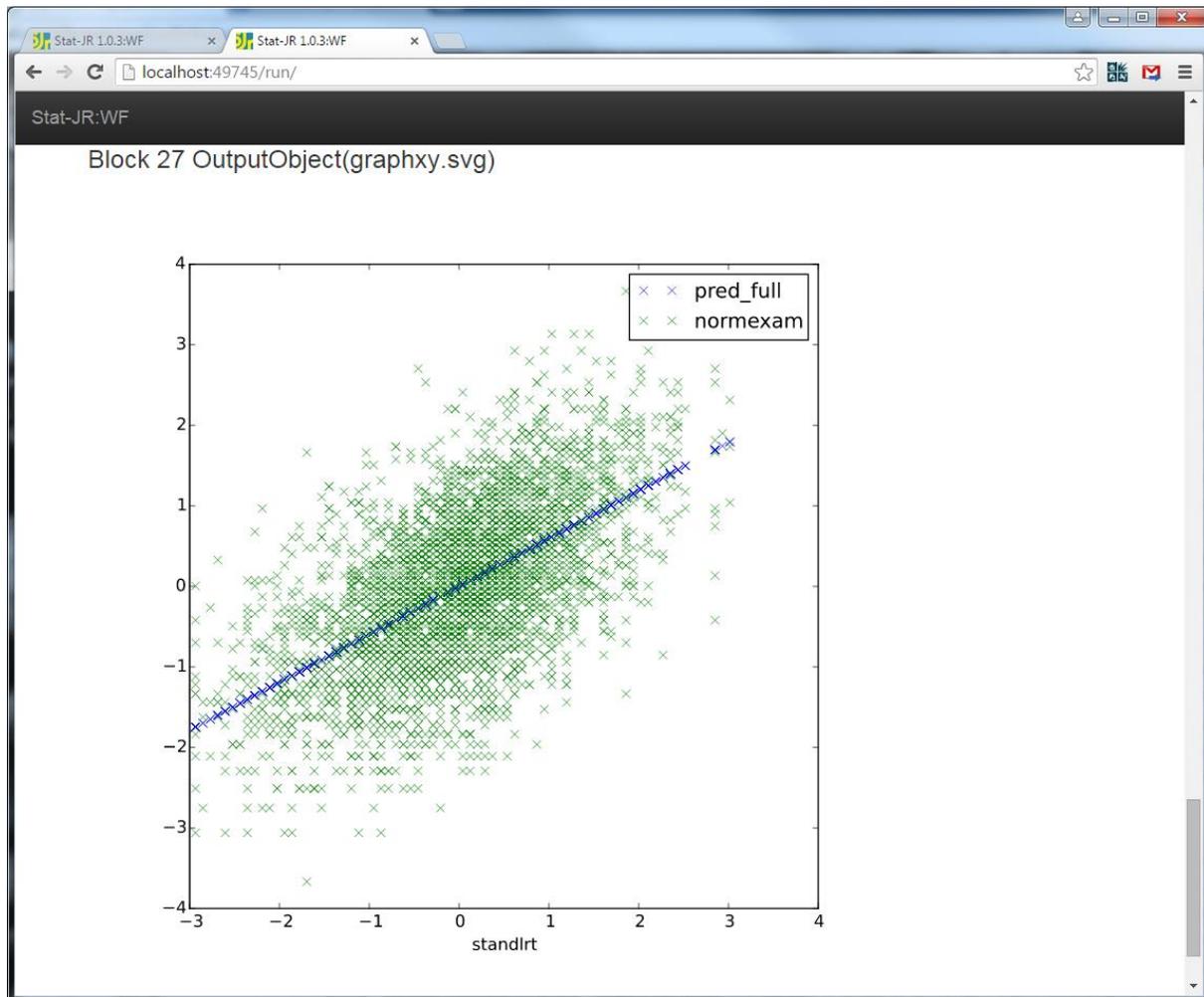


Figure 51

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. 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 practical 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.

1.15 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 *prac1_12.xml*.

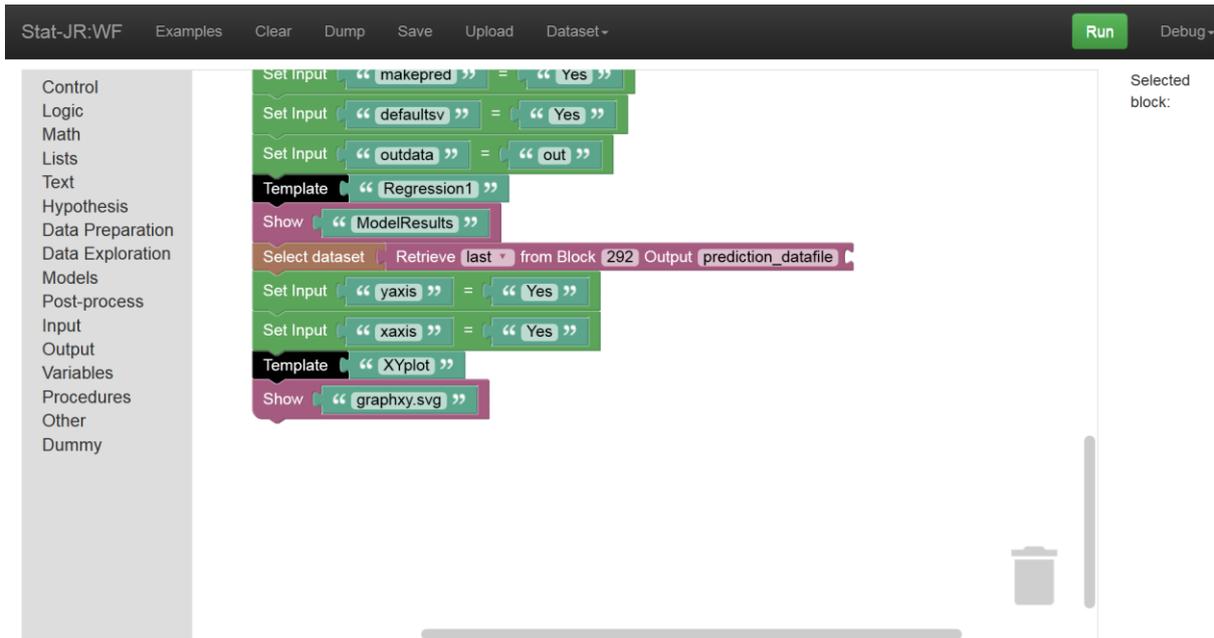


Figure 52