

SOFA Statistics – Developing, packaging, & promoting a Python open source project

Dr Grant Paton-Simpson

Paton-Simpson & Associates Ltd
grant@sofastatistics.com

Abstract

SOFA Statistics (<http://www.sofastatistics.com>) is an open source desktop Python application with an emphasis on ease of use, learn as you go, and attractive output. This paper will cover my experiences with some of the technical aspects of the project as well as project management issues. Specific areas covered will include:

- Using the cross-platform GUI toolkit wxPython (including wxWebKit and the grid widgets)
- The different roles of Matplotlib and Raphael (Javascript and SVG) in the application.
- Experience using different database engines including SQLite, MySQL, MS Access, and MS SQL Server
- Issues with standard statistics modules e.g. SciPy
- Making python installer packages – currently only deb packages and Windows installers (using NSIS) – and managing releases.
- Using Sourceforge, Freshmeat, and Launchpad (including Bazaar)
- Promoting the project through announcements, blogging, and answering posts.

1. Introduction

SOFA stands for Statistics Open For All, and SOFA Statistics is an open source desktop Python application with an emphasis on ease of use, learn as you go, and attractive output. It is still early-release software (currently version 0.8.9) but progress is quite rapid and the program is already quite usable for making report tables and running key statistical tests.

2. Overview of Functionality

Projects are used in SOFA Statistics to store all related reports, data files, and configuration information in one place. The default project should be enough for most users. SOFA Statistics does not require users to import from SQL-type databases – instead they are able to directly link to them.

The application has the ability to connect to, view, and edit, data from multiple different database engines using the same interface. And data can be imported from csv files and from Excel spreadsheets.

Three types of report table can be produced – Column Measures (e.g. frequencies, row and column percentages etc); Row Summaries (e.g. Mean, median, standard

deviation); and Raw display of table data. Variable labels can be edited (e.g. 1 for Male and 2 for Female), and output is HTML. The Python scripts used to generate the output can also be exported for reuse, simple automation, and incorporation into other programs. It is possible to work seamlessly with data in different tables and even different database engines.

Output charting will be SVG and JavaScript using the RaphaelJS library[1], which is printable, beautiful, and completely open.

The statistical test selection dialog allows speed when the user knows what they want or helpful guidance when they don't (or need reassurance).

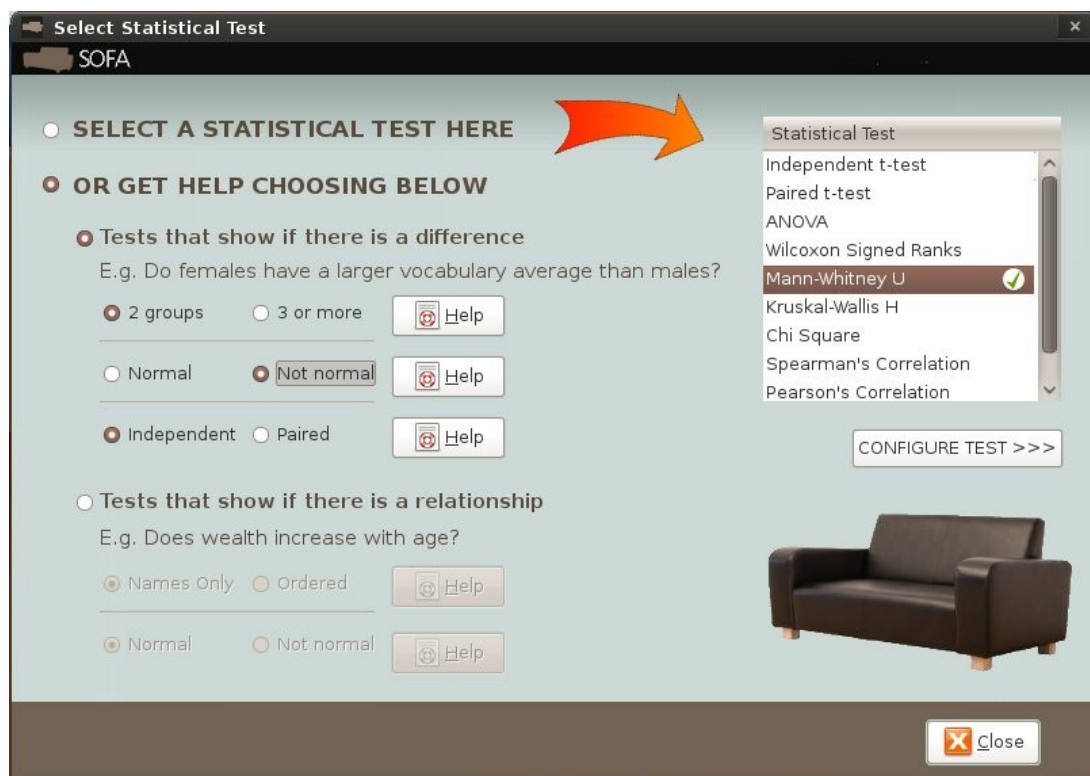


Figure 1.0. SOFA Statistics assistance in selecting appropriate tests.

3. wxPython GUI Toolkit

wxPython[2] is a mature cross-platform GUI toolkit for Python. The documentation is quite good, especially the “wxPython in Action” book, and it is often possible to get direct answers from the main developer Robin Dunn himself via the wxPython group[3]. Which is an enormous relief when you are pushing functionality to the limit e.g. using custom composite controls on a data grid and responding to custom events.

All the GUI code in SOFA Statistics is hand written and it is possible to build forms dynamically. For example, if you are on the Windows platform, the part of the project form collecting connection details adds in MS Access and MS SQL Server fields (assuming you are using those plug-ins). It also allows GUI code to be shared

between different forms – e.g. all the configuration settings or the database and table combo box functionality.

GUI coding is notorious for being challenging at times. Recently, I got some complex grid functionality working like clockwork in Linux, only to find it failed completely in Windows. Apparently Windows takes Return keystrokes out of the stream before you can access them. A Python paper wouldn't be complete without some code so here is the solution:

```
self.frame.Bind(wx.grid.EVT_GRID_EDITOR_CREATED,
                self.OnGridEditorCreated)

...

def OnGridEditorCreated(self, event):
    """
    Need to bind KeyDown to the control itself e.g. a choice
    control.
    wx.WANTS_CHARS makes it work.
    """
    control = event.GetControl()
    control.WindowStyle |= wx.WANTS_CHARS
    control.Bind(wx.EVT_KEY_DOWN, self.OnGridKeyDown)
    event.Skip()

...

def OnGridKeyDown(self, event):
    keycode = event.GetKeyCode()
    if keycode in (wx.WXK_TAB, wx.WXK_RETURN):
        etc
```

Two widgets merit extra comment - the grid and wxWebKit. The former is “very powerful, and very complex” according to Robin Dunn[4]. It was a relief to read Robin's comment because I certainly found the grid mentally demanding at times. It is extremely simple to get some simple things done with it but if you have particular demands, be prepared to do a lot of googling, learning, and experimenting.

wxWebkit[5] is very nearly ready to release properly (as at the time of writing) and it is the only way to do serious cross-platform HTML in wxPython. SOFA Statistics needed it for complex nested tables. Pushing all output through an HTML renderer has other advantages. It makes output easy to share, via intranets, the Internet, or even email. It also means the project can only benefit from developments in the web world. I intend to use Matplotlib[6] to make some of the auxiliary charting and RaphaelJS to produce the output charting. Both can be displayed in wxWebKit.

4. SQL databases and Python

SOFA Statistics can connect directly to SQLite, MySQL, PostgreSQL, MS Access, and MS SQL Server. Adding another database engine is as simple as taking an existing 500 line module e.g. `dbe_mysql.py` and changing the implementation of all the functions and methods. For example, `getDbTbls(cur, db)` must return a sorted list of table names. It soon becomes very clear that different SQL databases do

things very, very differently. A trivial example – in most SQL databases the following will count how many records meet a condition:

```
SELECT SUM(expression_here) AS freq FROM tbldata;
```

If True = 1, and False = 0 this works very well. If True is -1 and ABS() will take care of that. But PostgreSQL won't sum a boolean. And the SQLite approach to what data can be stored in fields is completely different from everything else I've come across. Sort of like dynamic typing. If you try to put something into a field it will go in somehow. Ideally, as what you would expect, but one way or another, it will go in. So you can't just get the type of a field and expect to have nothing but values of that type being served up to your statistics program.

5. Issues With Existing Modules

When doing something complicated like writing statistical algorithms, there are three very good reasons for trying to use existing standard libraries.

- 1) You don't waste your time.
- 2) Rolling your own can be risky (very small differences can result in completely different answers e.g. using floating point rather than decimal maths).
- 3) You can improve things for everyone else.

But there can be problems with that approach.

- 1) Code which feels half-baked e.g. variables set but never used, or an algorithm which has x , calculates x^2 , and later on uses $\sqrt{x^2}$ rather than just x . Or there can be rounding errors and wasted processing, which can matter when working with millions of records.
- 2) Code which exposes inadequate outputs, or just produces a formatted string as the output so there is no ability to produce own format e.g. HTML.
- 3) Code which doesn't give the option of using “decimal” as opposed to floating point maths.
- 4) Including SciPy[7], for example, substantially increases the size of your Windows installer package.

Still, even if you roll your own, you can use the existing libraries to test your own algorithms against. Using nose[8] you can easily and routinely run hundreds of tests. NB it is not the volume of tests that matters as much as the edge cases that they expose. e.g. using data like 1000000.1, 1000000.3, 1000000.2 ... really puts ANOVA tests through their paces.

6. Making Installer Packages

6.1 Debian packages for Ubuntu

I was initially very daunted – could I even do this? Then I found a ShowMeDo video by Austrian Horst Jens[9]. His example was a Python project so he had very

similar requirements. I have created very detailed step-by-step guidelines for packaging SOFA Statistics and I have a small handful of files with content I reuse for each package e.g. rules. I faced one big issue. I needed to put the program files in one place when the installation was being conducted as sudo (namely /usr/share/pyshared/sofa/...) and then create some user folders and files when an given user open SOFA Statistics for the first time (namely /home/username/sofa/...).

6.2 Windows packages with NSIS

Nullsoft Scriptable Install System (NSIS)[10] was used to create the SOFA Statistics Windows packages. NSIS is free and is used by Firefox, OpenOffice etc. Its scripts must be written in a weird language – it seems like a cross between PHP and assembler! An example of the things you need to learn: macros can take parameters but functions cannot; you cannot have functions inside sections etc. There is an (over)abundance of documentation but the approach I found best was to start and then extend. I am still learning and I hope to give users the option of where to install soon (e.g. D drive).

One issue can be file size. You need to include installers for everything needed e.g. Python 2.6, numpy[11], and wxPython.

7. Release process

There are lots of little steps that you have to get each and every time you do a release. So it pays to document your process very carefully. I use dokuwiki[12] to help me with that.

The steps are:

- Do preparatory clean up.
- Make and test the new deb and Windows packages. I use VirtualBox[13] to give me identical install environments each time. NB along the way all code has been managed by Bazaar[14], including pushing it up to Launchpad[15].
- Add the new files to Sourceforge (I wanted to consolidate downloads to help me measure usage).
- Add a new release to Launchpad and Freshmeat complete with updated release notes and change log.
- Make announcements in both Launchpad and Freshmeat.
- Update the project homepage to account for the new download location, new features.
- Add a blog item to the project site.
- Update release version and release date on Wikipedia.
- Revisit any important threads commenting on open source statistics packages.

Soon I will also have to alert any people helping create translated releases e.g. Galician. These will probably lag by a few days from the main English release. Launchpad provides a lot of infrastructure and guidance for translation which is really good.

8. Conclusions

Creating an open source application is a long-term commitment which may require a broad range of technical skills. It also requires attention to the detail of processes. If, like myself, this is what you enjoy doing, then it is really stimulating. But you must make sure you go in with your eyes wide open.

- [1] <http://raphaeljs.com/>
- [2] <http://www.wxpython.org/>
- [3] <http://groups.google.com/group/wxpython-users>
- [4] <http://www.wxpython.org/OSCON2008/wxPython-Advanced-OSCON2008.pdf>
- [5] <http://wxwebkit.wxcommunity.com/>
- [6] <http://matplotlib.sourceforge.net/>
- [7] <http://www.scipy.org/>
- [8] <http://somethingaboutorange.com/mrl/projects/nose/0.11.1/>
- [9] <http://showmedo.com/videotutorials/video?name=linuxJensMakingDeb&fromSeriesID=37>
- [10] http://nsis.sourceforge.net/Main_Page
- [11] <http://numpy.scipy.org/>
- [12] <http://www.dokuwiki.org/dokuwiki>
- [13] <http://www.virtualbox.org/>
- [14] <http://bazaar-vcs.org/en/>
- [15] <https://launchpad.net/>