Documentation as code for SAS projects

Using SAS and SASpy with markdown

Frederico Muñoz

Contents

Using SAS and SASpy with markdown

Motivation

With SASpy we can use Jupyter notebooks to run SAS code. Since it uses HTML for the display it doesn't work for markdown documents without an HTML parser (which is iPython per default, but doesn't exist when running `knitr` or any other interpreter.

This is a significant limitation to those that might want to use it for producing documentation: while Jupyter is great for interactive notebooks it doesn't lend itself for producing other types of documents. This is especially true for anything that isn't HTML: it's currently not possible to have images in PDFs exported from JupyterLab, for example.

Additionally there are other reasons for choosing markdown over Jupyter when the goal is no interactive development but producing documents that will become final deliverables.

In order to overcome this we added a proof-of-concept `display` to SASpy that converts the HTML output into markdown, making it possible to directly use the results in Markdown documents. The focus was on the plots since for the tables the default Pandas format (or the `TEXT` one) can be directly used.

There are other approaches which could be cleaner: SAS ODS has different output formats (including LaTeX) that could be used to get something that could be directly used as well. Our approach completely bypasses this in order to achieve a usable literate programming solution that can be used to produce PDF output from a simple Markdown source file.

SAS, Python and R

SASpy is a Python library, and as such a Python engine will be used. This doesn't mean that a Python-only approach is required, however: knitr is a well-known R solution for producing this kind of documents and allows the use of Python code blocks.

We have tested two solutions:

- Knitr, with the Python engine.
- Codebraid, using the Jupyter kernel.

Assuming that there is a way to parse Markdown output from code chuncks any other solution (e.g. pweave) should also work.

The code

Using SAS through SASpy

```
import saspy
import pandas as pd
sas = saspy.SASsession(cfgname='mycfg', results='Pandas')
```

SAS server started using Context Data Mining compute context with SESSION_ID=110f424b-1782-4772-9fd7-03ac845d2d93-ses0000

```
hr_pd = pd.read_csv("./HR_comma_sep.csv")
hr = sas.df2sd(hr_pd)  # the short form of: hr = sas.dataframe2sasdata(hr_pd)
p = hr.columnInfo()
print(p.to_markdown(tablefmt="github"))
```

	Member	Num	Variable	Type	Len	Pos
0	WORK._DF	6	Work_accident	Num	8	40
1	WORK._DF	4	average_montly_hours	Num	8	24
2	WORK._DF	2	last_evaluation	Num	8	8
3	WORK._DF	7	left	Num	8	48
4	WORK._DF	3	number_project	Num	8	16
5	WORK._DF	8	promotion_last_5years	Num	8	56
6	WORK._DF	10	salary	Char	6	75
7	WORK._DF	9	sales	Char	11	64
8	WORK._DF	1	satisfaction_level	Num	8	0
9	WORK._DF	5	time_spend_company	Num	8	32

A heatmap:

```
hr.heatmap('last_evaluation', 'satisfaction_level')
```

Python code

Regular python code runs as expected.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2*np.pi, 1001)
x_tick_values = np.linspace(0, 2*np.pi, 5)
x_tick_labels = ['0', r'$\pi/2$', r'$\pi$', r'$3\pi/2$', r'$2\pi$']
plt.plot(x, np.cos(x), label=r'$\cos(x)$')
```

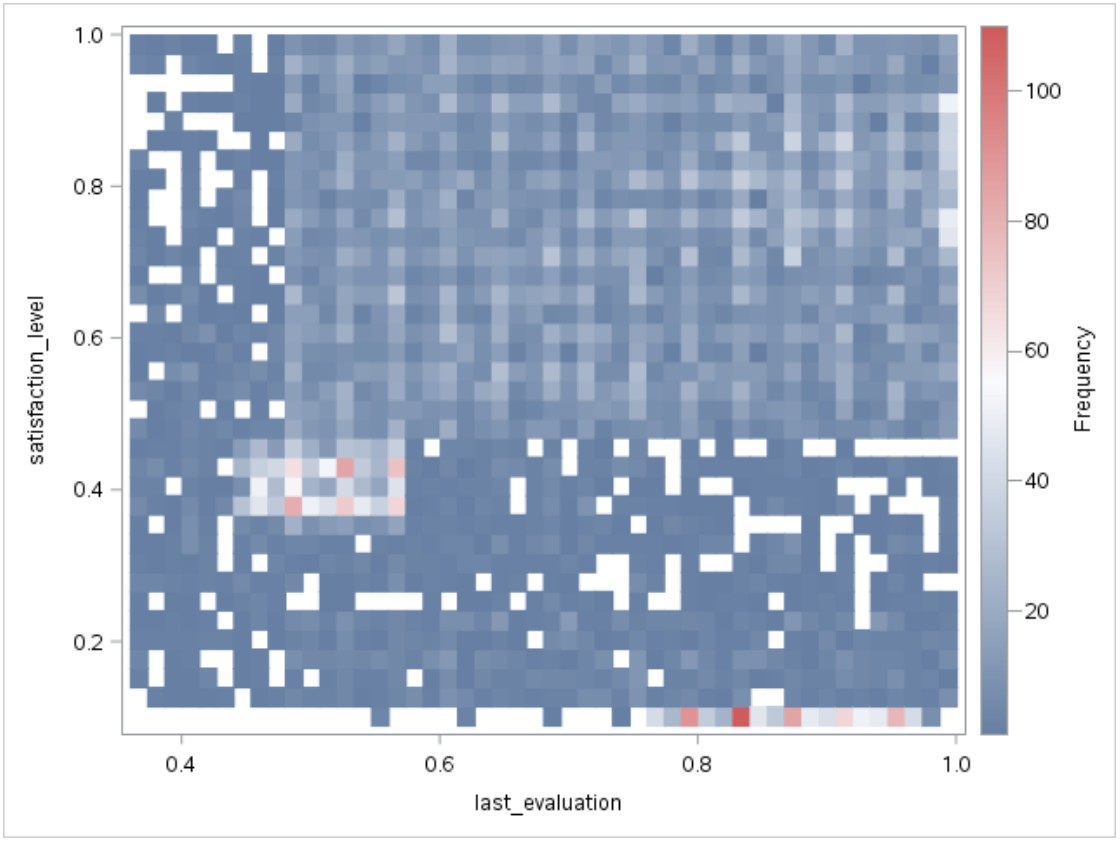

Figure 0.1: The SGPlot Procedure

```
plt.plot(x, np.sin(x), label=r'$\sin(x)$')
plt.legend(prop={'size': 12})
plt.grid()
plt.show()
```

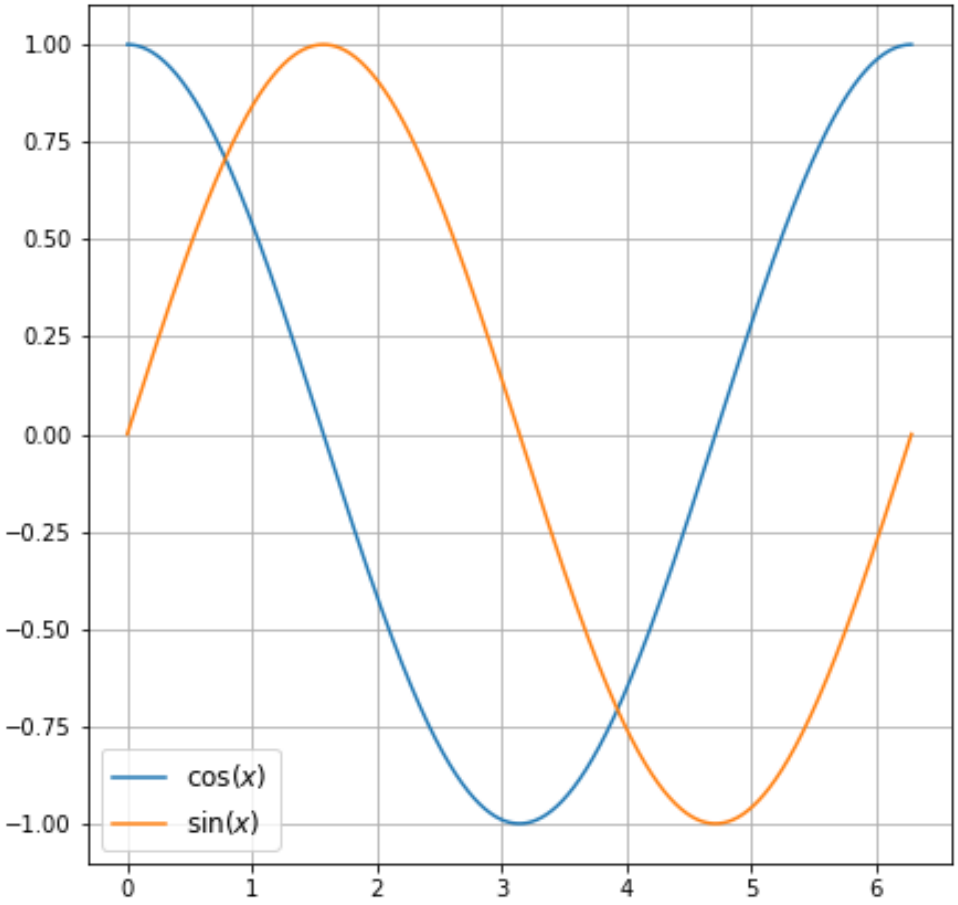

Figure 0.2: Matplotlib doing its thing

R code

```
plot(1:10)
```

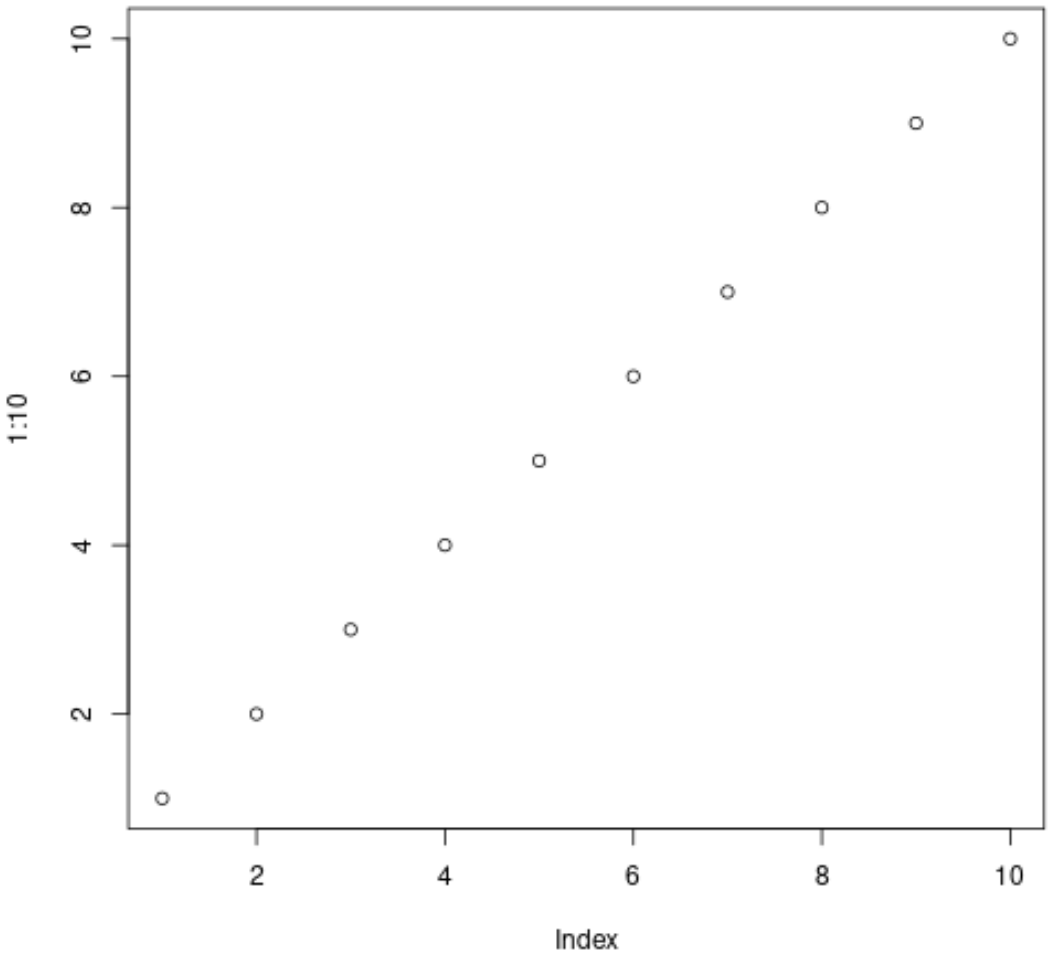

Figure 0.3: plot of chunk rcode

```
hist(rnorm(1000))
```

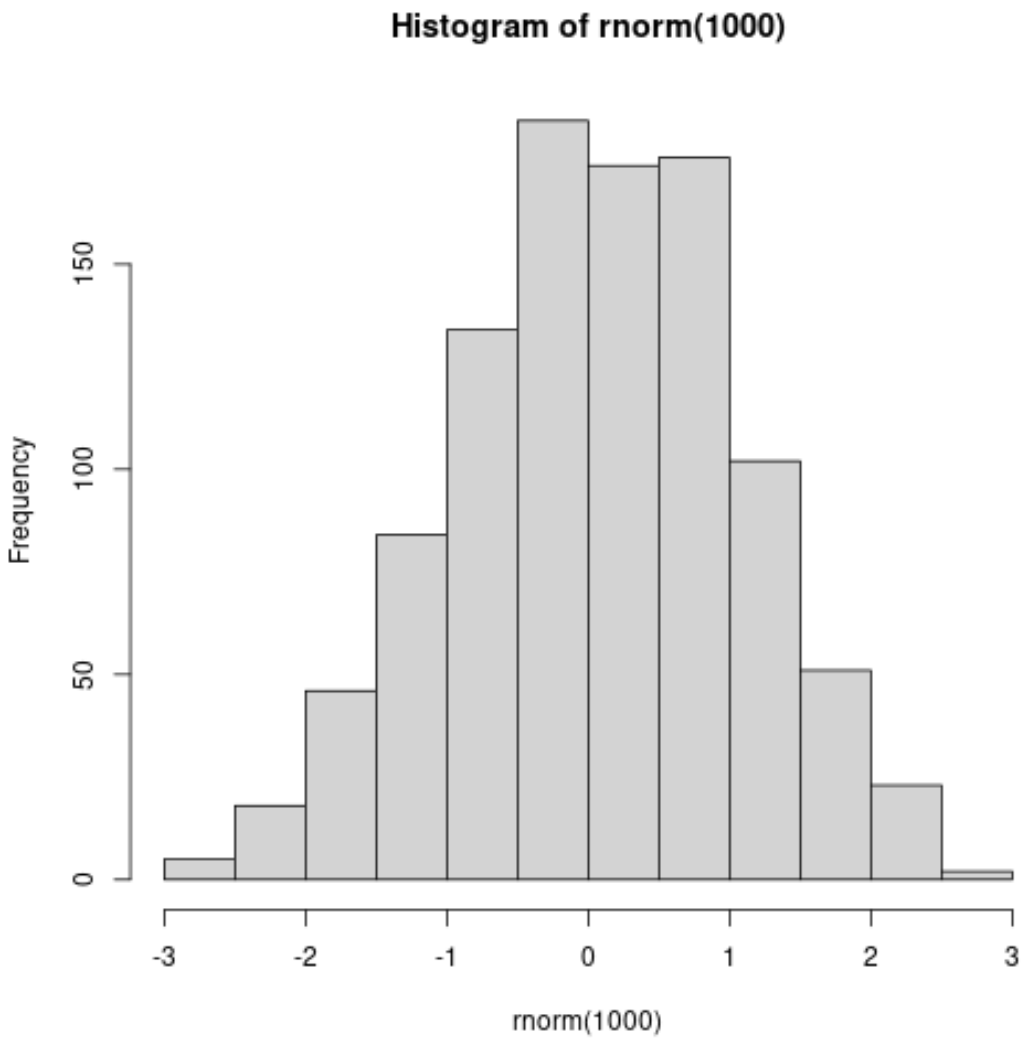

Figure 0.4: plot of chunk rcode