SAS and Markdown documents

Using SASpy with R markdown and knitr

Frederico Muñoz

October 2021

Introduction

Markdown is a document markup language that is widely used to produce documentation that can be easily edited, tracked through version control and transformed into many different output formats.

One popular application of Markdown is R Markdown, as used by the package `knitr` and that allows the inclusion and execution of R code in markdown documents: it's a similar approach to Jupyter notebooks, but based on Markdown documents and that implements literate programming.

It should be stressed that R Markdown documents are not limited to the R language: the `knitr` library is in R and R is obviously a first-class citizen, but it supports many different *engines*, one of them being Python, and a single document can contain code in different languages: as we will see we can use Python, R and SAS using SASpy.

The challenge

SASpy output of non-tabular results is HTML, which makes sense considering that all the target environments where it is used are browser-based: Jupyter, Zeppelin and Databricks are the available options at the time of writing. Markdown documents, however, do not render HTML: the output of the code chunk should be in markdown, and this is what then makes possible for tools like Pandoc to convert to target formats.

This highlights a good use-case for using SASpy with markdown: producing PDFs that can be highly customisable and include all the tables and images (something currently not possible by exporting a Jupyter notebook, for example).

Making it work

To make SASpy work transparently in a `.Rmd` document we will take the following approach:

- For *tables* we can take advantage of the default format that SASpy uses, Pandas, that support markdown.
- For everything else, including plots, we will parse the HTML and convert it to markdown.

We start by establishing the session, exactly like we would do in a Jupyter notebook, with a twist: we activate batch mode to have access to the HTML code, instead of the iPython object.

```
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=41e0ab9c-5409-406a-b563-4e0213976d38-ses0000
```

With batch mode activated we always get a dictionary with two keys, `LOG` and `LST`, the latter containing the HTML code (or Pandas object for tabular data).

```
sas.set_batch(True)
```

For Pandas we can make use of the `to_markdown` method directly: the utility function `p2m` simply wraps this to print the markdown table.

For HTML we get the body of the HTML (using BeautifulSoup), and convert it to markdown through the `markdownify` library. The `h2m` function encapsulates this logic.

```
from bs4 import BeautifulSoup as BS
from markdownify import markdownify as md

def p2m (p):
    print(p.to_markdown(tablefmt="github"))

def h2m (d):
    html = d['LST']
    soup = BS(html)
    body = soup.find('body')
    print(md(str(body)).strip())
```

With this functions defined the code is almost identical to what would be the equivalent Jupyter notebook: only the last command changes since we want to use the utility functions to output markdown.

The `columnInfo` method of SASpy will produce a markdown table (which will then be correctly displayed in a PDF, through LaTeX).

```
iris = sas.sasdata('iris','sashelp')
p2m(iris.columnInfo())
```

	Member	Num	Variable	Type	Len	Pos	Label
0	SASHELP.IRIS	4	PetalLength	Num	8	16	Petal Length (mm)
1	SASHELP.IRIS	5	PetalWidth	Num	8	24	Petal Width (mm)
2	SASHELP.IRIS	2	SepalLength	Num	8	0	Sepal Length (mm)
3	SASHELP.IRIS	3	SepalWidth	Num	8	8	Sepal Width (mm)
4	SASHELP.IRIS	1	Species	Char	10	32	Iris Species

The same applies to plots, in this case a heatmap (which runs `proc sgplot heatmap` in the backend):

```
h2m(iris.heatmap('PetalLength', 'PetalWidth'))
```

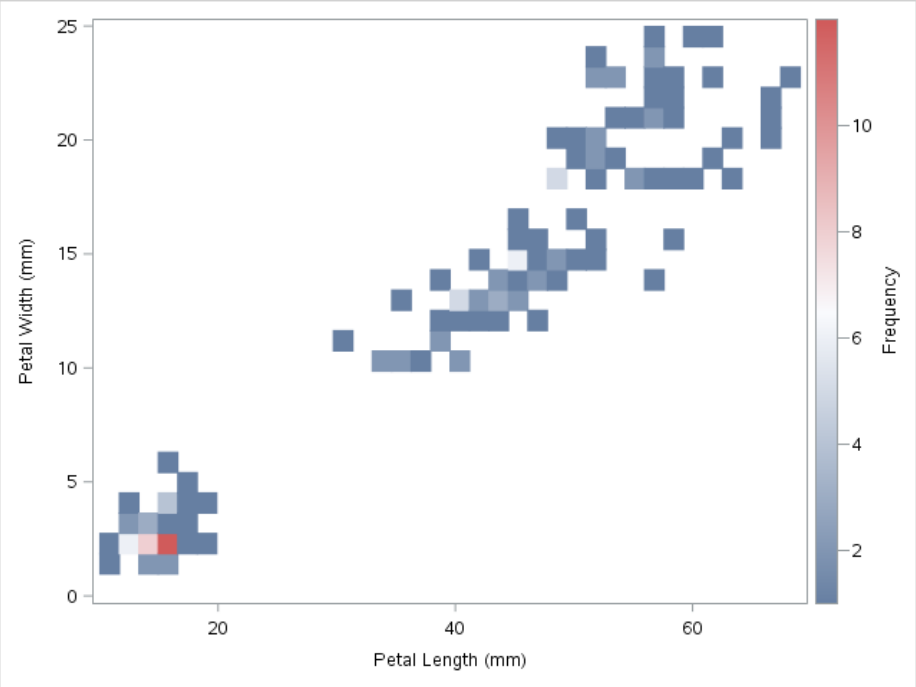

Figure 1: The SGPlot Procedure

Submitting SAS code works the same way: here we use `proc sgplot histogram` directly via sas.submit:

```
h2m(sas.submit("""
proc sgplot data=sashelp.iris ;
    histogram SepalLength;
run;
"""))
```

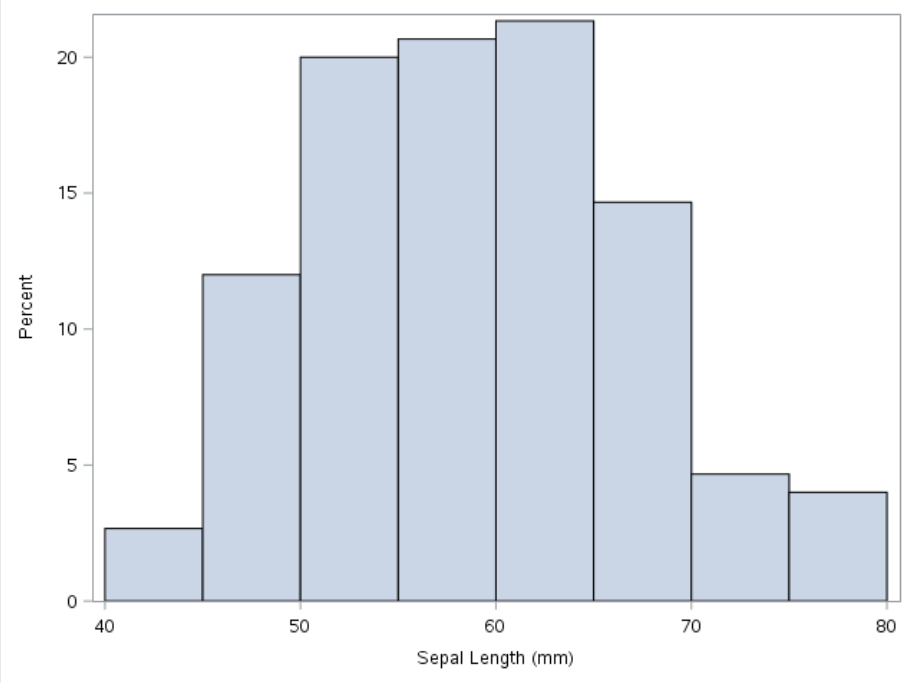

Figure 2: The SGPlot Procedure

From Rmd to PDF

R Markdown can be converted into standard markdown in several ways, but `knitr` is the most common one and is integrated into RStudio, for example.

In the shell the following command can be used (depends on having R and `knitr` installed, of course):

```
$ R -e 'library(knitr); knit("SASpy_with_R_markdown.Rmd",
"SASpy_with_R_markdown.md")'
```

To produce the PDF there are again many different options, with `pandoc` being one of the most common. There are quite a few options to modify the output, namely the ability to specify a LaTeXtemplate – check pandoc's documentation for details.

```
$ pandoc SASpy_with_R_markdown.md -o SASpy_with_R_markdown.pdf
```

Other markup formats

The same approach describe in this document can be applied to other formats, like Asciidoc or reStructuredText, as long as the utility functions are adapted to output the correct markup.