SASMarkdown

```r
colorize <- function(x, color) {
  if (knitr::is_latex_output()) {
    sprintf("\\textcolor{%s}{%s}", color, x)
  } else if (knitr::is_html_output()) {
    sprintf("<span style='color: %s;'>%s</span>", color,
      x)
  } else x
}
```

In a first code chunk, set up your SAS engine configuration.

This depends on your operating system, the version of SAS, and whether or not you have SAS installed in the default location. This example catches Windows SAS for me. In Macbook you just need to find the location of sas.exe and save the directory.

SASmarkdown has a very short memory, it could not save the previous code chunks in memory if you are using Rmarkdown editing anything. It only works when knitr works.

```r
# install.packages("SASmarkdown")
require(SASmarkdown)
sas_enginesetup(sashtml=sashtml)

if (file.exists("C:/Program Files/SASHome/SASFoundation/9.4/sas.exe")) {
  saspath <- "C:/Program Files/SASHome/SASFoundation/9.4/sas.exe"
} else {
  saspath <- "sas"
}

sasopts <- "-nosplash -ls 75"
knitr::opts_chunk$set(engine.path=list(sas=saspath, saslog=saspath),
                      engine.opts=list(sas=sasopts, saslog=sasopts))

saspath
```

```
## [1] "C:/Program Files/SASHome/SASFoundation/9.4/sas.exe"
```

Example 1

use the sashlep.class data.

```
/* SAS code for exmple1, we are in SAS now so every comment has to change */
proc means data=sashelp.class (keep = age);
run;
```

```
##                              The MEANS Procedure
##
##                           Analysis Variable : Age
##
##       N            Mean         Std Dev         Minimum         Maximum
##      -------------------------------------------------------------------
##      19      13.3157895       1.4926722      11.0000000      16.0000000
##      -------------------------------------------------------------------
```

Example 2

```
6          ods graphics off;
7          proc corr data=sashelp.class nosimple plots=matrix;
8          run;

WARNING: You must enable ODS graphics before requesting plots.
NOTE: PROCEDURE CORR used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

3 Variables:

Age Height Weight

Pearson Correlation Coefficients, N = 19 Prob > |r| under H0: Rho=0

Age

Height

Weight

Age

1.00000

0.81143

<.0001

0.74089

0.0003

Height

0.81143

<.0001

1.00000

0.87779

<.0001

Weight

0.74089

0.0003

0.87779

<.0001

1.00000

```
proc corr data=sashelp.class nosimple plots=matrix;
run;
```

3 Variables:

Age Height Weight

Pearson Correlation Coefficients, N = 19 Prob > |r| under H0: Rho=0

Age

Height

Weight

Age

1.00000

0.81143

<.0001

0.74089

0.0003

Height

0.81143

<.0001

1.00000

0.87779

<.0001

Weight

0.74089

0.0003

0.87779

<.0001

1.00000

```
help(package="SASmarkdown")
```

For this section the code must include the "collectcode = TRUE"

```
data class;
    set sashelp.class;
    keep age;
    run;
```

Without collectcode to link the code chunks, a later chunk that referenced the data in the WORK library would produce an error, but this now works. (No special option is needed for this later step.)

```
proc means data=class;
run;
```

Analysis Variable : Age

N

Mean

Std Dev

Minimum

Maximum

19

13.3157895

1.4926722

11.0000000

16.0000000

datasetp3

This code chunk and the previous one does the same work: you can either use the (r, engine='sashtml') or use the sashtml

```
proc means data=class;
run;
```

Analysis Variable : Age

N

Mean

Std Dev

Minimum

Maximum

19

13.3157895

1.4926722

11.0000000

16.0000000

You may run SAS (https://www.sas.com) code using the sas engine. You need to either make sure the SAS executable is in your environment variable PATH, or (if you do not know what PATH means) provide the full path to the SAS executable via the chunk option engine.path, e.g., engine.path = "C:\Program Files\SASHome\x86\SASFoundation\9.3\sas.exe".

```
filename myurl url "https://www.utsc.utoronto.ca/~butler/c32/soap.txt";

proc import
  datafile=myurl
  out=soap
  dbms=dlm
  replace;
  getnames=yes;
  delimiter=" ";
```

After that, proceed as you would in the SAS IDE (or on SAS Studio online), *without* `collectcode` on the top of the code chunk:

```
proc means;
  var scrap speed;
```

Variable

N

Mean

Std Dev

Minimum

Maximum

scrap

speed

27

27

315.4814815

210.1851852

82.9895129

63.4198689

140.0000000

100.0000000

470.0000000

320.0000000

This works because the "collected" chunk with the `proc import` in it is added to the top of this code, so that the data set is read in again, and because it "belongs" to this chunk, the variables `scrap` and `speed` will be found. We could also run a regression in the same way:

```
proc reg;
  model scrap=speed;
```

Model: MODEL1

Dependent Variable: scrap

Number of Observations Read

27

Number of Observations Used

27

Analysis of Variance

Source

DF

Sum ofSquares

MeanSquare

F Value

Pr > F

Model

1

149661

149661

127.23

<.0001

Error

25

29408

1176.31033

Corrected Total

26

179069

Root MSE

34.29738

R-Square

0.8358

Dependent Mean

315.48148

Adj R-Sq

0.8292

Coeff Var

10.87144

Parameter Estimates

Variable

DF

ParameterEstimate

StandardError

t Value

Pr > |t|

Intercept

1

64.03568

23.24876

2.75

0.0108

speed

1

1.19631

0.10606

11.28

<.0001

Model: MODEL1

Dependent Variable: scrap

(there are also supposed to be some plots which you won't see here) and once again the reading in of the data is added behind the scenes to the top of this code. In this case, as we suspected from the scatterplot, there is a significantly positive relationship between the speed of the production line and the amount of scrap produced.

You could also have a second chunk of "collected" code. For example, you might want to run a regression, saving an output data set (say, with the residuals in it), and, later, do something with the residuals. My example below saves the leverages (along with all the original variables). The `noprint` on the first line suppresses the regression output, which we saw before and don't want to see again:

```
proc reg noprint;
  model scrap=speed;
  output out=saved h=leverage;
```

Because I put `collectcode=T` in *this* code chunk header, our collection of code now includes (a) reading in the data and (b) running this regression, obtaining the output data set with the leverages in it. Thus, to display the leverages in order, I now only need to do this:

```
proc sort;
  by descending leverage;
proc print;
```

Obs

case

scrap

speed

line

leverage

1

1

218

100

a

0.15313

2

25

422

320

b

0.15236

3

16

140

105

b

0.14284

4

5

470

300

a

0.11418

5

2

248

125

a

0.10643

6

27

410

295

b

0.10583

7

14

425

290

a

0.09795

8

21

180

135

b

0.09109

9

13

275

140

a

0.08414

10

23

361

275

b

0.07721

11

9

410

270

a

0.07125

12

18

384

270

b

0.07125

13

11

241

155

a

0.06616

14

24

252

155

b

0.06616

15

15

367

265

a

0.06577

16

6

394

255

a

0.05624

17

19

341

255

b

0.05624

18

10

260

170

a

0.05248

19

8

321

175

a

0.04888

20

20

215

175

b

0.04888

21

12

331

190

a

0.04093

22

26

273

190

b

0.04093

23

7

332

225

a

0.03914

24

22

260

200

b

0.03803

25

3

360

220

a

0.03796

26

4

351

205

a

0.03729

27

17

277

215

b

0.03726

and everything will work. I sorted the leverages so that you can observe that the highest leverages go with the most extreme (highest or lowest) `speed` values.