R Markdown with Other Engines

Yingqi Jing

December 27, 2019

Contents

List of Tables

List of Figures

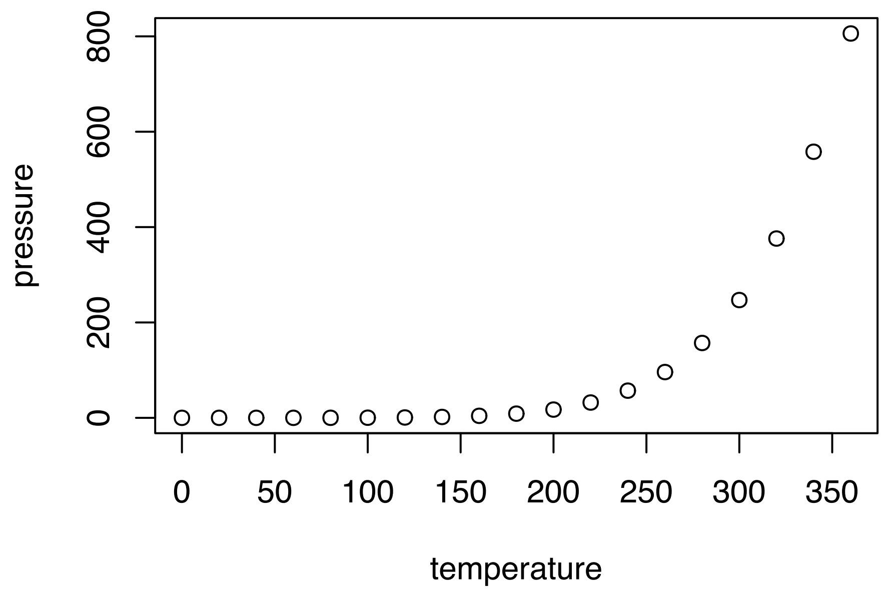

Figure S1: Relationship between temperature and presure

S1 R Markdown

S1.1 Including Plots

You can also embed plots, for example:

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

S1.1.1 subsubsection

```
summary(cars)
```

S2 Python code chunk

Note: `python.reticulate = T` can support variable inheritance across different chunk.

```
x = 'hello, python world!'
print(x.split(' '))
```

```
['hello,', 'python', 'world!']
```

```
x + " another chunk"
```

```
'hello, python world! another chunk'
```

S3 C++ code chunk

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
```

```
  return x * 2;
}
```

```
out = timesTwo(10) # test function in R chunk or console
out
```

```
[1] 20
```

S4 Julia code chunk

It seems that Julia code can be inherited across different chunks. This is one big advantage!

```
# the semicolon holds printing
list1 = ["Julia", "is fast!"];
println(list1)
```

```
["Julia", "is fast!"]
```

```
mystring = "my test sting for julia"
```

```
"my test sting for julia"
```

```
mystringnew = "$(mystring) new";
println(mystringnew)
```

```
my test sting for julia new
```

S5 Bash script

Note: Bash script in one chunk **cannot** be inherited by another chunk!

```
echo "Hello Bash"
```

```
Hello Bash
```

```
FILE='bash_name'
echo $FILE
```

```
bash_name
```

S6 Stan code chunk

We can assign the stan code to a variable (*model*1), and can use this later in the `R` code chunk.

```
parameters {
  real y[2];
}
model {
  y[1] ~ normal(0, 1);
  y[2] ~ double_exponential(0, 2);
}
```

```
fit <- sampling(model1, chains = 1)
```

```

SAMPLING FOR MODEL '6c400a2ac89dae0e85da9f7673dc5d1c' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 1.6e-05 seconds
```

```
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.013973 seconds (Warm-up)
Chain 1:                0.012085 seconds (Sampling)
Chain 1:                0.026058 seconds (Total)
Chain 1:
```

```r
print(fit)
```

```
Inference for Stan model: 6c400a2ac89dae0e85da9f7673dc5d1c.
1 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=1000.

      mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
y[1] -0.05    0.04 1.01 -1.88 -0.78 -0.01  0.60  1.96   610    1
y[2]  0.12    0.17 2.99 -6.06 -1.30  0.09  1.42  6.07   304    1
lp__ -1.55    0.08 1.33 -4.86 -2.12 -1.19 -0.62 -0.12   282    1

Samples were drawn using NUTS(diag_e) at Fri Dec 27 20:20:19 2019.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```