A13-rPython

Verena Haunschmid

14 September 2016

rPython

Recently I have started using `Python` for data analysis. By coincidence I saw that someone on Twitter mentioned the R package rPython. Since sometimes it's convenient to mix languages in a project, I wanted to know more about it.

Here you can find lots of documentation.

Install & load rPython

```
install.packages("rPython")
```

```
library(rPython)
```

```
## Loading required package: RJSONIO
```

The package & methods

The package contains four main methods:

- `python.assign`
- `python.call`
- `python.exec`
- `python.load`

It also contains another method:

- `python.get`

If you call any of those methods it saves it to some internal workspace (I don't know any details about that but you can probably find out in the link above).

Using the package

In the following I'll show examples for all 4 methods.

python.assign

If we want to assign some values in our magic *environment* we can use this method.

```
python.assign('a', 5)
python.assign('b', 2)
python.assign('c', 3)
```

For my examples I will also assign the same values to variables with other names. Of course I could use the same names because this is a different workspace but I want the code below to be less confusing.

```
d <- 5
e <- 2
f <- 3
```

python.call

Assume I have a `python` function named `cool_python_fct` that I want to use but it takes too long to rewrite it in R. It takes 3 variables that I have in my R environment. There is a neat way to use the method anyway:

```
python.call('cool_python_fct', d, e, f)
```

```
## [1] 10
```

As you can see the result is printed directly!

We can also store it in an R variable.

```
res <- python.call('cool_python_fct', d, e, f)
res
```

```
## [1] 10
```

python.exec

There is another way to call/execute a function. If I don't want to pass any R objects, this is also fine. Note that you only get the printed output but if you want to save the computed value you need to assign it to a variable. With `python.get` you can extract it.

```
python.exec('x = cool_python_fct(a, b, c)')
python.get('x')
```

```
## [1] 10
```

This way I can also do other cool things, like importing python libraries. In the next example we check which python version is used:

```
python.exec("import sys")
python.exec("print sys.version")
```

Interestingly this is only printed when called in the R command line but not when I create the report. But this way it also works:

```
python.get("sys.version")
```

```
## [1] "2.7.10 (default, Oct 23 2015, 18:05:06) \n[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59
```

python.load

This method is really useful to load and execute code in a file. Let's say I have a file that contains all the packages I want to import.

```
python.load("imports.py")
```

Now you can use the `math` package.

```
python.get("math.pi")
```

```
## [1] 3.141593
```

What this means

Although the package only has 4 methods, it is actually really mighty. With `python.exec` and `python.call` we have two really powerful tools to combine R and Python. If you prefer to wrangle your data in R but want to use algorithms that are only available in Python, it's super easy now.