Notes on using Python within RStudio

Christopher Paciorek

November 2019

RStudio has allowed the use of Python chunks in R Markdown documents for a while. These can be run and rendered in the resulting output file by 'knitting' the document using the `Knit` button (which invokes `rmarkdown::render`) or by invoking `rmarkdown::render` yourself.

With the development of the `reticulate` package, the functionality of using Python within RStudio has been greatly enhanced. I believe much of this new functionality is in place as of versions of RStudio >= 1.2.

With the notebook (inline output) functionality turned off

If you run a Python chunk from an R Markdown document in RStudio it will run the code in a Python process and show the result in the Console window

In fact it will invoke `reticulate::repl_python` to give you a "read-eval-print loop" (REPL) interpreter interface to Python in the console. If you want, you can enter Python code directly in the console. In addition, execution of subsequent Python code chunks can use objects from earlier chunks. To exit from the Python REPL to return to the R REPL, you can either type exit at the Python prompt or simply execute an R chunk.

With the notebook (inline output) functionality turned on

In the RStudio preferences under `R Markdown` you can select `Show output inline for all R Markdown documents`. This causes the output from chunks to show up in the editor window (i.e., notebook style) rather than in the console.

In this case, running Python chunks will print the output in the editor window, just as running R chunks will.

Reticulate and working in both R and Python

`reticulate` allows you to work on objects in both R and Python, moving seamlessly between the two languages. Lots more detail in the reticulate documentation and online.

```
library(reticulate)
devs <- rnorm(5)
```

```
## access the R object in Python code
r.devs[0]
```

```
## 1.7024160843141927
```

```
pyvals = {"a": 0, "b": 1}
```

```
## access the Python object in R code
py$pyvals['a']
```

```
## $a
## [1] 0
```

There's lots more functionality. That's just a teaser of the cool things you can do.