Just like how Thanos claimed to be inevitable in The Avengers, the direct or indirect use of python has become inevitable for R users in recent years. Fret not R users, you don’t have to abandon your favourite IDE, Rstudio, when using python. With the reticulate package you can use python in Rstudio and even have a mixture of R and python code running in the same session. If you blog with blogdown, you don’t have to migrate to another platform to write about your python projects. With the help of reticulate, you can continue publishing content on your blogdown site. An analogy of reticulate will be like a translator between R and python.

library(tidyverse)

library(reticulate)

**Setup**

[reticulate has a default approach to instruct R where to find python, which environment and version to use. There are three approaches to manually configure this](https://rstudio.github.io/reticulate/articles/versions.html).

1. virtualenv where you specify the directory of python virtual environment
2. use\_python where you specify the path where your ‘python’ resides.
3. use\_condaenv where you specify the name of the specific Conda environment to use. You can access the name(s) of the available environments via conda\_list()[[1]]

conda\_list()[[1]] %>% use\_condaenv()

Let’s check which python version, environment and configuration has been bind to this R session.

py\_config() #not run to maintain privacy

**Running python**

Amend {r} in your code chunk to {python} to run python code. For this post, I will add #{python/r} in my code chunks to make it explicit that I ran the code as a python or a r code chunk.

#{python}

Plist= [11,22,33,44,55,66]

print(Plist)

## [11, 22, 33, 44, 55, 66]

#{python}

def Psq\_fun (x):

value= x\*x

return(value)

print(Psq\_fun(9))

## 81

Alternatively, you can execute python scripts in your r chunks using the function py\_run\_string. You will need to wrap your python scripts within the quotation marks “”.

#{r}

py\_run\_string("def Pten (x):

value= x\*10

return(value)")

Let’s run the above in a python code chunk.

#{python}

Pten(2)

## 20

Do note that python objects/functions are not explicitly displayed in your global environment (you are after all working in an R global environment by default).

#{r}

ls()

## character(0)

Nevertheless, be assured that you can still access them in future python and r chunks.

#{python}

print(Psq\_fun(9))

## 81

#{python}

print(Plist)

## [11, 22, 33, 44, 55, 66]

**Accessing python**

You can access previously ran python functions/objects in your r chunks using the prefix py combined with the R’s dollar sign syntax $.

#{r}

py$Psq\_fun(4)

## [1] 16

#{r}

py$Plist

## [1] 11 22 33 44 55 66

Alternatively, you can directly evaluate previous python objects/functions in r chunks using the py\_eval function.

#{r}

py\_eval("Plist")

## [1] 11 22 33 44 55 66

#{r}

py\_eval("Psq\_fun(2)")

## [1] 4

**Accessing R**

Likewise, you can access R objects/functions in your python chunks using the prefix r combined with a punctuation mark .

#{r}

(Rvec<-c(11,22,33,44,55,66))

## [1] 11 22 33 44 55 66

#{python}

r.Rvec

## [11.0, 22.0, 33.0, 44.0, 55.0, 66.0]

#{r}

Rroot\_fun<-function(x){

value= x^.5

print(value)

}

#{python}

r.Rroot\_fun(81)

## 9.0

**Converting objects between languages**

Besides accessing a R object in python via r., you can convert the R object into a python object while still running R with r\_to\_py.

Rvec %>% r\_to\_py() %>% class()

## [1] "python.builtin.list" "python.builtin.object"

Previously, I mentioned that python objects do not exist in your global R environment when you run the python script directly inside {python} code chunks or with python\_run\_string. However, when you create python objects in {r} code chunks, the python object is saved in the R environment.

#{r}

# convert R object into python

Python\_in\_Renv<- Rvec %>% r\_to\_py()

# check if the converted object lives in the `R` environment. <https://stackoverflow.com/questions/1169248/test-if-a-vector-contains-a-given-element>

"Python\_in\_Renv" %in% ls()

## [1] TRUE

For python objects living in the R global environment, you can convert it back to a R object with py\_to\_r.

Python\_in\_Renv %>% py\_to\_r() %>% class()

## [1] "numeric"

**Layout**

At times you may decide to display R and python code side by side .

**Html document**

**static**

The first outlay is static where the page is divided into half and content appears on either half of the page. You will [sandwich text and code chunks meant to be on the left and right with the respective html/css code](https://stackoverflow.com/questions/31753897/2-column-section-in-r-markdown?noredirect=1&lq=1).

#Start of column partition

#Left column

Text: `R` code will be on the left

Chunk of R code ``{r}``

#Right column

Text: `python` code will be on the right

Chunk of Python code ``{python}``

# End of column partition

**Dynamic**

[An alternative will be a where different languages are compartmentalized into their respective tabs and the tabs are displayed next to each another. Users decide which language they wish to view and therefore which tab to highlight](https://bookdown.org/yihui/rmarkdown/html-document.html#tabbed-sections). You will use the [{.tabset} function with headers to create the tabs and end the tab section with ##](https://stackoverflow.com/questions/38062706/rmarkdown-how-to-end-tabbed-content).

## Level 2 heading {.tabset}

### Level 3 heading (first tab i.e. left tab)

Text: R code will be on the left tab

Chunk of R code ``{r}``

### Level 3 heading (next tab i.e. right tab)

Text: python code will be on the right tab

Chunk of python code ``{python}``

##

**Blog**

Unfortunately, the above do not work when building a site with blogdown despite trying several Hugo themes.

Text: R code will be on the left

Code: ``{r}``

Text: python code will be on the left

Code: ``{python}``

R code on the left

#{r}

Rroot\_fun

## function(x){

## value= x^.5

## print(value)

## }

python code on the right

#{python}

print(Pten)

##