Python interoperability

If you havn't installed any python environment in your computer, I recommand you to install anaconda with python 3.7. The installer could be install form https://www.anaconda.com/distribution/#download-section

Step 1

To use python in R, we need to install the R package `reticulate`.

```
library(reticulate)
```

Step 2

You could pick your own favourite python environment

```
py_discover_config()
```

```
## python:         /usr/bin/python
## libpython:      /System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/config/libpyth
## pythonhome:     /System/Library/Frameworks/Python.framework/Versions/2.7:/System/Library/Frameworks/P
## version:        2.7.16 (default, Dec 13 2019, 18:00:32)  [GCC 4.2.1 Compatible Apple LLVM 11.0.0 (cla
## numpy:          /System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python/numpy
## numpy_version:  1.8.0
##
## python versions found:
##  /Users/Randy/miniconda3/bin/python
##  /usr/bin/python
##  /usr/local/bin/python
##  /usr/bin/python3
##  /usr/local/bin/python3
##  /Users/Randy/miniconda3/envs/py27/bin/python
##  /Users/Randy/miniconda3/envs/r-tensorflow/bin/python
```

Try to locate a python 3 binary.

```
# I am picking up this python
use_python("~/miniconda3/bin/python", required = TRUE)
# or if you are using a conda env
# use_condaenv("myenv", conda = "~/miniconda3/bin/conda", required = TRUE)
```

Check again your python setup

```
py_config()
```

```
## python:         /usr/bin/python
## libpython:      /System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/config/libpyt
## pythonhome:     /System/Library/Frameworks/Python.framework/Versions/2.7:/System/Library/Frameworks/
## version:        2.7.16 (default, Dec 13 2019, 18:00:32)  [GCC 4.2.1 Compatible Apple LLVM 11.0.0 (cl
## numpy:          /System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python/numpy
## numpy_version:  1.8.0
##
## python versions found:
##  /usr/bin/python
##  /usr/local/bin/python
##  /usr/bin/python3
##  /usr/local/bin/python3
##  /Users/Randy/miniconda3/envs/py27/bin/python
##  /Users/Randy/miniconda3/envs/r-tensorflow/bin/python
##  /Users/Randy/miniconda3/bin/python
```

There are a variety of ways to integrate Python code into your R projects:

- Python in R Markdown

```
import random
x = random.random()
```

```
# to access Python objects in R
py$x
```

```
## [1] 0.520044
```

- Importing Python module

The `import()` function enables you to import any Python module and call its functions directly from R.

```
random <- import("random")
y <- random$random()
```

```
# to access R object in python
r.y
```

```
## 0.1478298927403513
```

The `import_builtins()` function enables to access the built in functions.

```
builtins <- import_builtins()
a <- builtins$range(5L)

builtins <- import_builtins()
builtins$len(a)
```

```
## [1] 5
```

- Sourcing Python scripts: The `source_python()` function enables you to source a Python script the same way you would source() an R script (Unless `envir = NULL`, Python functions and objects defined within the script become directly available to the R session).

```
source_python("script.py", envir = NULL)
py$z
```

```
## [1] 0.6952688
```

```
# I personally don't recommand it
source_python("script.py")
z
```

```
## [1] 0.1274193
```

- Python REPL: `repl_python()`

Type conversions

https://rstudio.github.io/reticulate/#type-conversions

By default when Python objects are returned to R they are converted to their equivalent R types.

```
random <- import("random")
(x <- random$random())
```

```
## [1] 0.6796256
```

```
class(x)
```

```
## [1] "numeric"
```

However, if you'd rather make conversion from Python to R explicit and deal in native Python objects by default you can pass convert = FALSE to the import function

```
random <- import("random", convert = FALSE)
(x <- random$random())
```

```
## 0.510719797769
```

```
class(x)
```

```
## [1] "python.builtin.float"  "python.builtin.object"
```

We cannot work with native Python objects directly.

```
x + 1
```

```
## Error in x + 1: non-numeric argument to binary operator
```

The function `py_to_r` converts native Python objects to R objects.

```
# convert x to R object first
py_to_r(x) + 1
```

```
## [1] 1.51072
```

Let's check another numpy example.

```
np <- import("numpy", convert = FALSE)

# do some array manipulations with NumPy
a <- np$array(c(1:4))
sum <- a$cumsum()

# convert to R explicitly at the end
py_to_r(sum)
```

```
## [1]  1  3  6 10
```

R Objects are converted to Python objects when they are passed to Python functions.

```
import sys
def abssum(x):
    print("received a {} object\n".format(type(x)))
    return sum((abs(z) for z in x))
```

```
x <- rnorm(10)
# x is implicitly to a native Python list
py$abssum(x)
```

```
## [1] 6.498798
```