

Statistics 360: Advanced R for Data Science

Lecture 11

Brad McNeney

R and Python: References

- ▶ The reticulate website: <https://rstudio.github.io/reticulate/>
- ▶ Python setup: <https://docs.python.org/3/using/index.html>
- ▶ Python tutorial: <https://docs.python.org/3/tutorial/>

Calling Python from R

- ▶ Use cases:
 - ▶ Tap into a growing set of tools for data science, such as scikit-learn <https://scikit-learn.org/stable/> , keras <https://keras.io/> , ...
 - ▶ Workflow requires substantial computations in **both** languages; e.g., fit a neural network in Python, plot results using ggplot2
- ▶ Not recommended:
 - ▶ Using R/RStudio as a development environment for Python.
 - ▶ Better to go all-in with Python and use Jupyter Notebooks as your IDE.

Prerequisites

- ▶ Install and load the reticulate package

```
library(reticulate)
```

- ▶ Install Python
 - ▶ See the Python setup link on the references slide
- ▶ Optional: Install the conda package manager
 - <https://conda.io/projects/conda/en/latest/user-guide/install/index.html>
 - ▶ I'll use conda for creating environments and installing packages.
 - ▶ Users more familiar with virtual environments may prefer `venv` and `pip`.

Python packages and environments

- ▶ Python packages are like R packages.
 - ▶ Can be installed from the command line with `conda install <package>`.
- ▶ Python environments, like RStudio projects, are used to compartmentalize your work with Python.
 - ▶ A complete Python installation, including its own Python executable and packages.
 - ▶ Create from the command line with `conda create <env_name>`
 - ▶ Then “activate” with `conda activate <env_name>` and “de-activate” with `conda deactivate`.

Installing packages with reticulate

- ▶ See [https:](https://rstudio.github.io/reticulate/articles/python_packages.html)

[//rstudio.github.io/reticulate/articles/python_packages.html](https://rstudio.github.io/reticulate/articles/python_packages.html)

```
library(reticulate)
use_python("/home/mcneney/miniconda3/bin/python") # required on my comp
#conda_create("r-reticulate") # commented out to avoid re-doing every t
# install packages into this environment
#conda_install("r-reticulate", "pandas") # commented out to avoid re-do
```

Using a conda environment

- ▶ For each R session in which you want to use your conda environment:

```
library(reticulate)
use_python("/home/mcneney/miniconda3/bin/python") # if necessary
use_condaenv("r-reticulate")
```


Python embedded in RMarkdown

- ▶ Example from https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html

```
# code chunk header is ``{python} rather than ``{r}  
import numpy as np  
import pandas as pd  
df = pd.DataFrame(np.random.randn(3,4), columns=['A', 'B', 'C', 'D'])  
df
```

```
##           A           B           C           D  
## 0  1.276903 -0.423408 -0.580627  2.106952  
## 1 -2.155920  0.151673  0.421038 -1.558741  
## 2 -1.702611 -0.998947  0.812479 -1.214097
```

Importing Python packages (modules)

- ▶ You can also import Python packages into R and call their functions directly.

```
npr <- import("numpy.random")
pd <- import("pandas") # import is from reticulate
df <- pd$DataFrame(npr$randn(3L,4L),columns=c('A','B','C','D'))
df
```

##		A	B	C	D
## 1		-0.52114256	-0.0299745	-1.3750306	-2.475675
## 2		-0.01294737	0.8607080	0.6807521	-1.012529
## 3		2.32345329	1.2391246	0.9176832	0.584932

Notes

- ▶ Access Python functions from an imported package with \$.
- ▶ The `randn()` function requires integer arguments – have to use 3L and 4L to pass integers.
 - ▶ reticulate converts R vectors of length 1 to Python scalars.
 - ▶ In general reticulate will try to convert to/from appropriate data types. See the list at <https://rstudio.github.io/reticulate/index.html#type-conversions>
- ▶ I used the numpy random number generator, but passed column names as an R character vector.
 - ▶ reticulate converts this to a python list.

Sourcing Python scripts

- ▶ Source with `source_python()` and retrieve objects from an object named `py`.
 - ▶ `py` appears to be implemented as an environment, but behaves more like a list.

```
source_python("lec11_1.py")  
ls(py) # is.environment(py)
```

```
## [1] "convert" "pyobj"
```

```
names(py)
```

```
## [1] "df" "np" "pd" "r" "R" "sys"
```

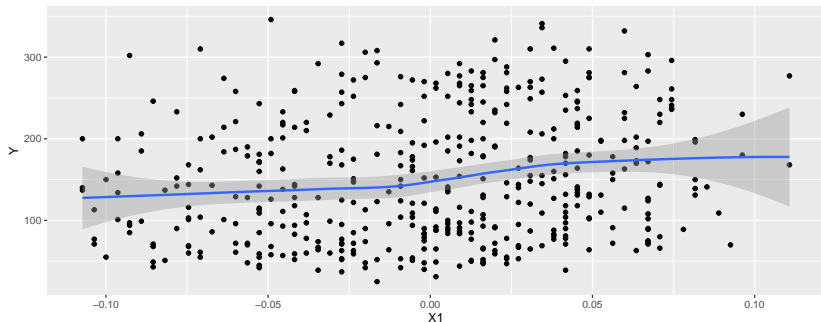
```
py$df
```

```
##           A           B           C           D  
## 1 -0.9346697  0.60271639 0.8536789 -1.1278611  
## 2  0.7975562  1.11655368 0.9269830 -0.9174114  
## 3  0.3762885 -0.06047241 0.5804783 -0.9096843
```

Another example

```
source_python("lec11_2.py")  
py$MSE
```

```
## [1] 2859.69  
ddat <- data.frame(Y=py$diabetes_y,py$diabetes_X)  
library(ggplot2)  
ggplot(ddat,aes(x=X1,y=Y)) + geom_point() + geom_smooth()
```



Python REPL

- ▶ You can also start the Python interpreter and compute interactively.
 - ▶ Useful for debugging your Python scripts

```
# repl_python()  
# type your Python commands  
# objects will be available in R through py object  
# exit() to quit
```