reticulate walkthrough

Liam D. Bailey

Contents

We are going to use the package reticulate to run python in R. This will allow us to streamline code between the two languages.

Load reticulate and virtual environments

The package reticulate is used for running a python session in your R script.

```r
library(reticulate)

#Create and install modules into a virtual environment
#virtualenv_create("example_env")
#py_install("pandas", envname = "example_env")

use_virtualenv("example_env")
```

Specify the location of python

By default, R will search for python in the regular R search path. You can specify that python should be added to PATH when you install. Otherwise, we can specify the location of the python application.

We do this with the use_python function

```r
use_python("C:\\Users\\Liam\\AppData\\Local\\Programs\\Python\\Python37")
```

reticulate requires Python >=3.6.5. You can check that you're using the correct version (and also check library paths etc.) with the `py_discover_config()` function.

```
py_discover_config()
```

```
## python:         C:/Users/Liam/Anaconda3/python.exe
## libpython:      C:/Users/Liam/Anaconda3/python37.dll
## pythonhome:     C:/Users/Liam/Anaconda3
## version:        3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
## Architecture:   64bit
## numpy:          C:/Users/Liam/Anaconda3/Lib/site-packages/numpy
## numpy_version:  1.16.4
##
## python versions found:
##  C:/Users/Liam/Documents/.virtualenvs/example_env/Scripts/python.exe
##  C:/msys64/mingw64/bin/python3.exe
##  C:/Users/Liam/Anaconda3/python.exe
##  C:/Python38/python.exe
##  C:/Users/Liam/Anaconda3/envs/example_env/python.exe
##  C:/Users/Liam/Anaconda3/envs/gender_mob/python.exe
##  C:/Users/Liam/Anaconda3/envs/genmob/python.exe
```

Using python in R

Now that we have a session of python in our RMarkdown script, we can simply write and run python code in our RMarkdown document using the Python version and environment as specified in the configuration above.

Instead of starting our chunk with {r} we start it with {python}

```
print("Hello World!")
```

```
## Hello World!
```

Import python modules

You can import python modules into R using the `import` function. The python functions can then be used in regular R script. This is most useful if you just want access to standard python functions.

HOWEVER you need to save this as an object to call the functions within that module!

In this example, the os module has a function listdir() which lists all objects inside the directory.

```
os <- import("os")
os$listdir()
```

```
## [1] "add.py"                 "iris.csv"               "reticulate_intro.html"
## [4] "reticulate_intro.Rmd"   "return_object.py"
```

Importing python script

Python modules will be searched for in the file path where python is installed. However, you may want to write some functions of your own in python. These can be more easily called using `source_python`.

For example, in the directory there is a .py file called `add`.

We can call this script and it will return all objects defined in the .py script!

```
source_python('add.py')
add(5, 10)
```

```
## [1] 15
```

This will load all objects in the script, not just functions. See the code below in `return_object.py`.

When we run it, we will not have objects i and x defined in our R session!

```
source_python('return_object.py')
i + x
```

```
## [1] 300
```

We may not always want to load all objects into the global environment. Instead, we can use `py_run_file`.

```
#Remove objects from global to demonstrate example
rm(list = c("i", "x"))

#Run .py script. This time i and x are not loaded into the global environment
py_run_file("return_object.py")
```

The difference here is that any objects created in the R script are not assigned in the global environment and instead need to be called from within the `py` object.

```
i
```

```
## Error in eval(expr, envir, enclos): object 'i' not found
```

```
py$i + py$x
```

```
## [1] 300
```

This will also apply to parsing strings using `py_run_string`.

```
py_run_string("y = 300")
```

```
py$i + py$y
```

```
## [1] 400
```

Using R objects in python

Similarly, you can call an object which as been defined in R in a python chunk using the `r` object.

```
r.a + r.b
```

```
## 300.0
```

Conversion of objects between python and R

Different classes of objects will be converted when transferring between python and R. These are listed below.

Single-element Vector(R) <-> Scalar(py)

Multi-element Vector(R) <-> List(py)

List of mutliple types(R) <-> Tuple(py)

Named list(R) <-> Dict(py)

Matrix(R) <-> NumPy ndarray(py)

Data frame(R) <-> Pandas DataFrame(py)

Function(R) <-> Function(py)

NULL(R) <-> None(py)

TRUE/FALSE(R) <-> True/False(py)

Help

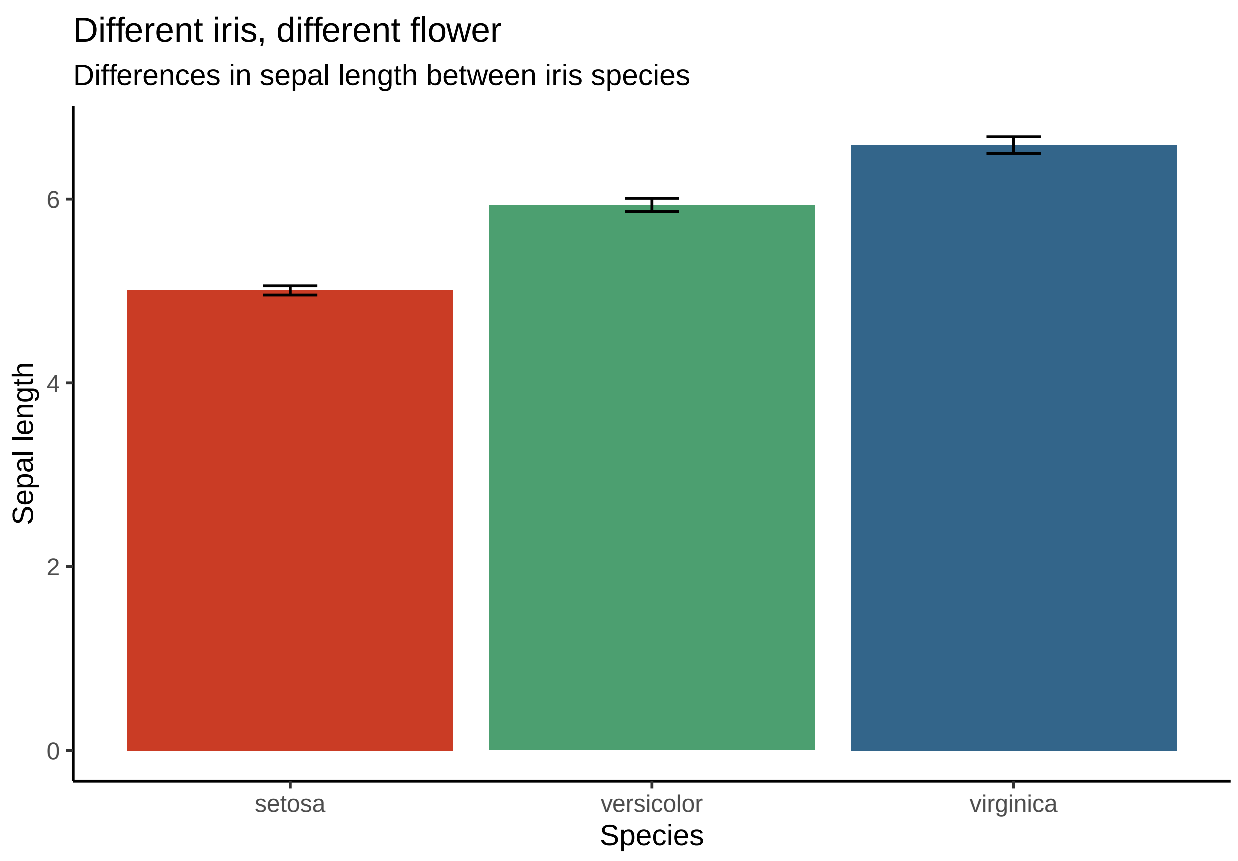
If you need help on a python function, you can use the `py_help` function. This will open a .txt file with a description of the function.

```r
py_help(os$chdir)
```

Worked example: Data wrangling in pandas and plotting in ggplot

```python

import pandas as pd

iris_data = pd.read_csv("./iris.csv")

summary_data = (iris_data
.groupby(by = "Species")["Sepal.Length"]
.agg(["sem", "mean"])
.reset_index())
```

```r
library(ggplot2)

ggplot(data = py$summary_data) +
  geom_col(aes(x = Species, y = mean, fill = Species)) +
  geom_errorbar(aes(x = Species, ymin = mean - sem, ymax = mean + sem), width = 0.15) +
  scale_y_continuous(name = "Sepal length") +
  scale_fill_manual(values = c("#CA3C25", "#4C9F70", "#33658A"), guide = NULL) +
  labs(title = "Different iris, different flower", subtitle = "Differences in sepal length between iris
  theme_classic()
```


Different iris, different flower
Differences in sepal length between iris species
Sepal length
0
2
4
6
setosa
versicolor
virginica
Species