Integrating R and Python in a single Quarto notebook environment

Hélène Langet

2024-11-25

Table of contents

1 Learning objectives

- Learn how to integrate Python in a Quarto notebook primarily using R code chunks ;
- Learn how to write and execute Python code chunks in a Quarto notebook ;
- Learn how to seamlessly share data frames and other objects between R and Python code chunks for a cohesive, mixed-language workflow.

2 Setup your R Environment for Python integration

To integrate Python within a Quarto notebook that primarily uses R code chunks (i.e., executed with the `knitr` engine), you will need the `reticulate` R package, which serves as a bridge between R and Python, allowing you to run Python code chunks alongside R and share objects between the two programming languages. This setup provides a powerful approach for combining R's statistical capabilities with Python's data manipulation and machine learning tools, enabling a robust, mixed-language workflow within Quarto.

Install `reticulate` if it is not already installed.

~~~
```{r}
install.packages("reticulate")
```
~~~

To ensure that R uses the correct Python executable, specify the path to Python using the `Sys.setenv()` function. This is especially important if you have multiple Python installations on your system or are using a specific environment, such as Anaconda. Replace *"C:/ProgramData/anaconda3/python.exe"* with the path to your Python executable.

> **Tip**
>
> You can also set environment variables permanently in your R environment variables by defining them in the `.Renviron` file.

~~~
```{r}
Sys.setenv(RETICULATE_PYTHON = "C:/ProgramData/anaconda3/python.exe")
```
~~~

> **! Important**
>
> Consistently using the same Python environment in `reticulate` helps preventing version conflicts and package incompatibilities.

Load `reticulate` at the beginning of your Quarto notebook.

~~~
```{r}
library(reticulate)
reticulate::use_python("C:/ProgramData/anaconda3/python.exe")
```
~~~

Use `py_config()` to verify that the correct Python version is active; this also allows you to confirm that required Python packages are available within your chosen environment.

~~~
```{r}
reticulate::py_config()
```
~~~

```
python:         C:/ProgramData/anaconda3/python.exe
libpython:      C:/ProgramData/anaconda3/python311.dll
```

```
pythonhome:     C:/ProgramData/anaconda3
version:        3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.19
Architecture:   64bit
numpy:          C:/ProgramData/anaconda3/Lib/site-packages/numpy
numpy_version:  1.24.3

NOTE: Python version was forced by RETICULATE_PYTHON_FALLBACK
```

3 Execute Python code and access R objects in Python

~~~
```{python}
import sys
print(f"Python executable path: {sys.executable}")
print(f"Python version: {sys.version}")
```
~~~

```
Python executable path: C:\PROGRA~3\ANACON~1\python.exe
Python version: 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.19
```

Below, we create a data frame in R using the built-in iris dataset, adding a new column called Sepal.Area to illustrate data manipulation in R. We use the dplyr package for simplicity and display the first few rows of the data frame using knitr::kable() for formatted output.

~~~
```{r}
#| label: tbl-1
#| tbl-cap: Data frame created, manipulated and displayed using R

library(dplyr)
df1 <- iris |>
  dplyr::mutate(Sepal.Area = Sepal.Length * Sepal.Width)
df1 |>
  head(5) |>
  knitr::kable()
```
~~~

Table 1: Data frame created, manipulated and displayed using R

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Sepal.Area
5.1	3.5	1.4	0.2	setosa	17.85
4.9	3.0	1.4	0.2	setosa	14.70
4.7	3.2	1.3	0.2	setosa	15.04
4.6	3.1	1.5	0.2	setosa	14.26
5.0	3.6	1.4	0.2	setosa	18.00

As illustrated below, you can access R objects (here the data frame `df1`) as a Python object by calling them through the `r.` prefix. In this example, we access the R data frame df1 and manipulate it in Python by creating a new column, Petal.Area. Then, use `{python}` as the chunk language to specify Python chunks within your Quarto notebook.

~~~
```{python}
import pandas as pd
df2 = r.df1
df2["Petal.Area"] = df2["Petal.Length"] * df2["Petal.Width"]
```
~~~

4 Access Python objects in R

As illustrated below, you can reciprocally access Python objects (here the data frame `df2`) as a R object by calling them through the `py$` prefix. Here, we access the Python data frame df2 in R and display the first few rows to confirm the changes.

~~~
```{r}
#| label: tbl-2
#| tbl-cap: Data frame read from a Python object and displayed using R

py$df2 |>
  head(5) |>
  knitr::kable()
```
~~~

Table 2: Data frame read from a Python object and displayed using R

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Sepal.Area	Petal.Area
5.1	3.5	1.4	0.2	setosa	17.85	0.28

Table 2: Data frame read from a Python object and displayed using R

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Sepal.Area	Petal.Area
4.9	3.0	1.4	0.2	setosa	14.70	0.28
4.7	3.2	1.3	0.2	setosa	15.04	0.26
4.6	3.1	1.5	0.2	setosa	14.26	0.30
5.0	3.6	1.4	0.2	setosa	18.00	0.28

In this example, df2 created in Python is now accessible in R. The py$ prefix provides a direct way to retrieve Python objects from the global environment into R. This allows for flexible back-and-forth interactions between R and Python without needing to export or save intermediate files.

4.1 Best practices

- Ensure that data structures are compatible. For example, R data frames are automatically converted to pandas data frames in Python, but other complex R objects may not be directly accessible in Python.
- Keep in mind that transferring large data frames between R and Python may introduce some overhead. Where possible, limit the size of data passed across languages.