Use R and Python

Integrating R and Python in a single Quarto notebook environment

Hélène Langet

2025-01-27

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

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.

2.1 Install and load reticulate

If it is not already done, install the `reticulate` package using the function `install.packages()` in your **R console**:

```
install.packages("reticulate")
```

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

```
library(reticulate)
```

2.2 Verify your Python version

Use `py_config()` to verify what Python version is active. This also allows you to confirm that required Python packages are available within your chosen environment. By default, `reticulate` uses an isolated python virtual environment named `r-reticulate`.

```
reticulate::py_config()
```

```
Error in python_config_impl(python) :
  Error running "C:/Users/langhe/AppData/Local/r-miniconda/envs/r-reticulate/python310.dll.ex

python:         C:/Users/langhe/Documents/.virtualenvs/r-reticulate/Scripts/python.exe
```

```
libpython:      C:/Users/langhe/AppData/Local/r-miniconda/envs/r-reticulate/python310.dll
pythonhome:     C:/Users/langhe/Documents/.virtualenvs/r-reticulate
version:        3.10.16 | packaged by conda-forge | (main, Dec  5 2024, 14:07:43) [MSC v.194
Architecture:   64bit
numpy:          C:/Users/langhe/Documents/.virtualenvs/r-reticulate/Lib/site-packages/numpy
numpy_version:  2.2.2

NOTE: Python version was forced by VIRTUAL_ENV
```

> **!** Important
>
> If the `r-reticulate` does not exist, you can try to remove and recreate this environment using the following commands in your **R console**:
>
> ```
> reticulate::virtualenv_remove("r-reticulate", confirm = TRUE)
> reticulate::virtualenv_create("r-reticulate")
> ```
>
> If you encounter issues setting up your Python environment, please let us know.

2.3 Install Python packages to your active reticulate environment

Python packages will be installed in the active Python environment as set by the `RETICULATE_PYTHON_ENV` environment variable. If this variable is unset, then the `r-reticulate` environment will be used. You can install Python packages (e.g., the `pandas` package, which we will need later in this page) using the function `py_install()` in your **R console**:

```
reticulate::py_install("pandas")
```

2.4 Use of a specific Python executable

You can also use an alternate Python executable – if you already have one installed on your computer – by using the command `use_python()`:

```
reticulate::use_python("C:/ProgramData/anaconda3/python.exe")          ①
```

① Replace `"C:/ProgramData/anaconda3/python.exe"` with the path to your Python executable.

To ensure that R uses the correct Python executable, specify its path in the `RETICULATE_PYTHON` environment variable using the `Sys.setenv()` function. This is especially important if you have multiple Python installations on your system (e.g. those activated within Conda environments).

> **Tip**
>
> - Consistently using the same Python environment in `reticulate` helps preventing version conflicts and package incompatibilities.
> - To determine the location of your Python executable on Windows, you can open a command prompt, type the command `where python`, then press `Enter`.
> - Note that you can also set environment variables permanently in R or Quarto by defining them in files that allow you to store key-value pairs, such as `RETICULATE_PYTHON="C:/ProgramData/anaconda3/python.exe"`:
>
>   – the `_environment` file that Quarto loads automatically before rendering - see Quarto documentation;
>
>   – the `.Renviron` file that R loads automatically at the start of each session and applied for R scripts or Quarto notebooks. To open and edit your `.Renviron file`, you can use the `edit_r_environ()` function from the `usethis` package:
>
> ```
> usethis::edit_r_environ()
> ```

3 Execute Python code

You can execute Python code by specifying `{python}` as the chunk language at the beginning of a new code chunk, as is shown below:

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

```
Python executable path: C:\Users\langhe\DOCUME~1\VIRTUA~1\R-RETI~1\Scripts\python.exe
Python version: 3.10.16 | packaged by conda-forge | (main, Dec  5 2024, 14:07:43) [MSC v.194
```

4 Access R objects in Python

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 `mutate()` function from the `dplyr` package for simplicity and display the first five rows of the data frame, finally using the function `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 retrieve R objects (here the data frame `df1`) from the global environment as Python objects 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`.

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

5 Access Python objects in R

As illustrated below, you can reciprocally retrieve Python objects (here the data frame `df2`) from the global environment as R objects by calling them through the `py$` prefix. In this example, we use the Python data frame `df2` in R and display the first five rows to confirm the change.

````
```{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
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

This allows for flexible back-and-forth interactions between R and Python without needing to export or save intermediate files, and can be handy for workflows that encompass data wrangling, statistical analysis, machine learning, and visualization, by leveraging the strengths of both languages.

6 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.