

Use Python with R with reticulate : : CHEATSHEET



The **reticulate** package lets you use Python and R together seamlessly in R code, in R Markdown documents, and in the RStudio IDE.

Python in R Markdown

(Optional) Build Python env to use.

knitr versions ≥ 1.18 will automatically use the reticulate engine for Python chunks. See `?reticulate::eng_python` for a listing of supported knitr chunk options.

Suggest the Python environment to use, in your setup chunk.

Begin Python chunks with ````{python}`. Chunk options like **echo**, **include**, etc. all work as expected.

Use the **py** object to access objects created in Python chunks from R chunks.

Python chunks all execute within a **single** Python session so you have access to all objects created in previous chunks.

Use the **r** object to access objects created in R chunks from Python chunks.

Output displays below chunk, including matplotlib plots.

```
python.Rmd x
1 {r setup, include = FALSE}
2 library(reticulate)
3 virtualenv_create("fmri-proj")
4 py_install("seaborn", envname = "fmri-proj")
5 use_virtualenv("fmri-proj")
6
7
8 {python, echo = FALSE}
9 import seaborn as sns
10 fmri = sns.load_dataset("fmri")
11
12
13 {r}
14 f1 <- subset(py$fmri, region == "parietal")
15
16
17 {python}
18 import matplotlib as mpl
19 sns.lmplot("timepoint", "signal", data=r.f1)
20 mpl.pyplot.show()
21
```

```
python.R x
1 library(reticulate)
2 py_install("seaborn")
3 use_virtualenv("r-reticulate")
4
5 sns <- import("seaborn")
6
7 fmri <- sns$load_dataset("fmri")
8 dim(fmri)
9
10 # creates tips
11 source_python("python.py")
12 dim(tips)
13
14 # creates tips in main
15 py_run_file("python.py")
16 dim(py$tips)
17
18 py_run_string("print(tips.shape)")
19
```

Python in R

Call Python from R code in three ways:

IMPORT PYTHON MODULES

Use **import()** to import any Python module. Access the attributes of a module with **\$**.

- **import**(module, as = NULL, convert = TRUE, delay_load = FALSE) Import a Python module. If convert = TRUE, Python objects are converted to their equivalent R types. Also **import_from_path()**. import("pandas")
- **import_main**(convert = TRUE) Import the main module, where Python executes code by default. import_main()
- **import_builtins**(convert = TRUE) Import Python's built-in functions. import_builtins()

SOURCE PYTHON FILES

Use **source_python()** to source a Python script and make the Python functions and objects it creates available in the calling R environment.

- **source_python**(file, envir = parent.frame(), convert = TRUE) Run a Python script, assigning objects to a specified R environment. source_python("file.py")

RUN PYTHON CODE

Execute Python code into the **main** Python module with **py_run_file()** or **py_run_string()**.

- **py_run_string**(code, local = FALSE, convert = TRUE) Run Python code (passed as a string) in the main module. py_run_string("x = 10"); py\$x
- **py_run_file**(file, local = FALSE, convert = TRUE) Run Python file in the main module. py_run_file("script.py")
- **py_eval**(code, convert = TRUE) Run a Python expression, return the result. py_eval("1 + 1")

Access the results, and anything else in Python's **main** module, with **py**.

- **py** An R object that contains the Python main module and the results stored there. py\$x

Object Conversion

Tip: To index Python objects begin at 0, use integers, e.g. 0L

Reticulate provides **automatic** built-in conversion between Python and R for many Python types.

R	Python
Single-element vector	Scalar
Multi-element vector	List
List of multiple types	Tuple
Named list	Dict
Matrix/Array	NumPy ndarray
Data Frame	Pandas DataFrame
Function	Python function
NULL, TRUE, FALSE	None, True, False

Or, if you like, you can convert manually with

py_to_r(x) Convert a Python object to an R object. Also **r_to_py**().

tuple(..., convert = FALSE) Create a Python tuple. tuple("a", "b", "c")

dict(..., convert = FALSE) Create a Python dictionary object. Also **py_dict**() to make a dictionary that uses Python objects as keys. dict(foo = "bar", index = 42L)

np_array(data, dtype = NULL, order = "C") Create NumPy arrays. np_array(c(1:8), dtype = "float16")

array_reshape(x, dim, order = c("C", "F")) Reshape a Python array. x <- 1:4; array_reshape(x, c(2, 2))

py_func(f) Wrap an R function in a Python function with the same signature. py_func(xor)

py_main_thread_func(f) Create a function that will always be called on the main thread.

iterate(it, f = base::identity, simplify = TRUE) Apply an R function to each value of a Python iterator or return the values as an R vector, draining the iterator as you go. Also **iter_next**() and **as_iterator**().

py_iterator(fn, completed = NULL) Create a Python iterator from an R function. seq_gen <- function(x){ n <- x; function() {n <- n + 1; n}}; py_iterator(seq_gen(9))

Helpers

py_capture_output(expr, type = c("stdout", "stderr")) Capture and return Python output. Also **py_suppress_warnings**().

py_get_attr(x, name, silent = FALSE) Get an attribute of a Python object. Also **py_set_attr**(), **py_has_attr**(), and **py_list_attributes**().

py_help(object) Open the documentation page for a Python object. py_help(sns)

py_last_error() Get the last Python error encountered. Also **py_clear_last_error**() to clear the last error. py_last_error()

py_save_object(object, filename, pickle = "pickle", ...) Save and load Python objects with pickle. Also **py_load_object**(). py_save_object(x, "x.pickle")

with(data, expr, as = NULL, ...) Evaluate an expression within a Python context manager. py <- import_builtins(); with(py\$open("output.txt", "w") %as% file, { file\$write("Hello, there!") })



Python in the IDE

Requires reticulate plus RStudio v1.2+. Some features require v1.4+.

Syntax highlighting for Python scripts and chunks.

Tab completion for Python functions and objects (and Python modules imported in R scripts).

Source Python scripts.

Execute Python code line by line with **Cmd + Enter** (**Ctrl + Enter**).

View Python objects in the Environment Pane.

View Python objects in the Data Viewer.

The screenshot shows the RStudio interface with several panels. The top-left pane contains Python code for loading data and plotting. The top-right pane shows the Environment pane with loaded modules like pandas, matplotlib, and seaborn. The bottom-left pane shows the console with the output of the Python code. The bottom-right pane shows a scatter plot of tip vs total_bill with a regression line. A small cheat_sheets - RStudio Source Editor window is also visible.

A Python REPL opens in the console when you run Python code with a keyboard shortcut. Type **exit** to close.

matplotlib plots display in plots pane.

Press **F1** over a Python symbol to display the help topic for that symbol.

Python REPL

A REPL (Read, Eval, Print Loop) is a command line where you can run Python code and view the results.

- Open in the console with **repl_python()**, or by running code in a Python script with **Cmd + Enter** (**Ctrl + Enter**).
 - repl_python(module = NULL, quiet = getOption("reticulate.repl.quiet", default = FALSE), input = NULL)** Launch a Python REPL. Run **exit** to close. **repl_python()**
- Type commands at **>>>** prompt.
- Press **Enter** to run code.
- Type **exit** to close and return to R console.

The screenshot shows the RStudio console with the output of the Python REPL. It displays the R version, the Python version, and the Reticulate version. It also shows the prompt for entering Python code and the output of the code.

Configure Python

Reticulate binds to a local instance of Python when you first call **import()** directly or implicitly from an R session. To control the process, find or build your desired Python instance. Then suggest your instance to reticulate. **Restart R to unbind.**

Find Python

- install_python(version, list = FALSE, force = FALSE)** Download and install Python. **install_python("3.9.16")**
- py_available(initialize = FALSE)** Check if Python is available on your system. Also **py_module_available()** and **py_numpy_module()**. **py_available()**
- py_discover_config()** Return the detected installation of Python. Use **py_config()** to check which version has been loaded. **py_config()**
- virtualenv_list()** List all available virtual environments. Also **virtualenv_root()**. **virtualenv_list()**
- conda_list(conda = "auto")** List all available conda environments. Also **conda_binary()** and **conda_version()**. **conda_list()**

Create a Python env

- virtualenv_create(envname = NULL, ...)** Create a new virtual environment. **virtualenv_create("r-pandas")**
- conda_create(envname = NULL, ...)** Create a new conda environment. **conda_create("r-pandas", packages = "pandas")**

Install Packages

Install Python packages with R (below) or the shell:

pip install SciPy
conda install SciPy

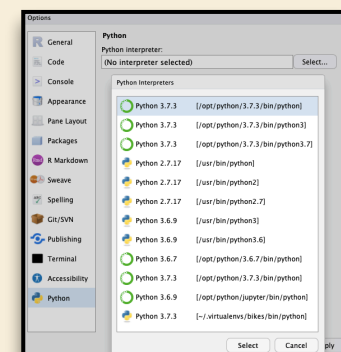
- py_install(packages, envname, ...)** Installs Python packages into a Python env. **py_install("pandas")**
- virtualenv_install(envname, packages, ...)** Install a package within a virtualenv. Also **virtualenv_remove()**. **virtualenv_install("r-pandas", packages = "pandas")**
- conda_install(envname, packages, ...)** Install a package within a conda env. Also **conda_remove()**. **conda_install("r-pandas", packages = "plotly")**

Suggest an env to use

Set a default Python interpreter in the RStudio IDE Global or Project Options.

Go to **Tools > Global Options... > Python** for Global Options.

Within a project, go to **Tools > Project Options... > Python**.



Otherwise, reticulate scans the instances on your computer in the following order:

- The instance referenced by the environment variable **RETICULATE_PYTHON** (if specified). **Tip: set in .Renviron file.**
 - Sys.setenv(RETICULATE_PYTHON = PATH)** Set default Python binary. Persists across sessions! Undo with **Sys.unsetenv()**. **Sys.setenv(RETICULATE_PYTHON = "/usr/local/bin/python")**
- The instances referenced by **use_** functions if called before **import()**.
 - use_python(python)** Path to a Python binary. **use_python("/usr/local/bin/python")**
 - use_virtualenv(virtualenv)** Path to or name of a Python virtualenv. **use_virtualenv("~/myenv")** **use_virtualenv("r-keras")**
- A virtual env found in the current working directory: **./.venv**
- Environments that are named after the imported module. e.g. **~/virtualenvs/r-scipy/** for **import("scipy")**
- The package default virtualenv, "r-reticulate".
- At the location of the Python binary discovered on the system PATH (i.e. **Sys.which("python")**)