# 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.



(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.

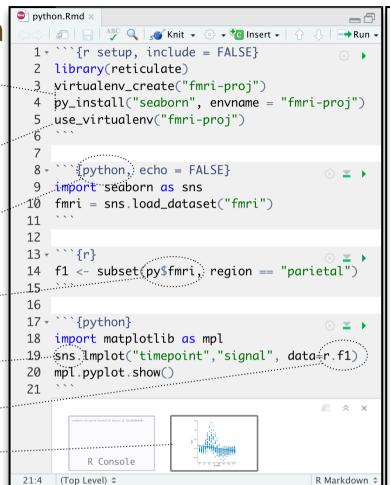
Begin Python chunks with ```{python}. Chunk options like echo, include, etc. all work as expected.

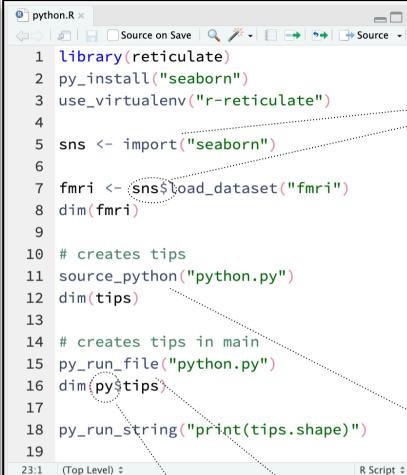
Use the **py** object to access objects created in Python chunks from R chunks.

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 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")

### **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.

Python
Scalar
List
Tuple
Dict
NumPy ndarray
Pandas DataFrame
Python function
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))</pre>

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))</pre>

#### 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")

#### **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



Python in the IDE Requires reticulate plus RStudio v1.2+. Some features require v1.4+.

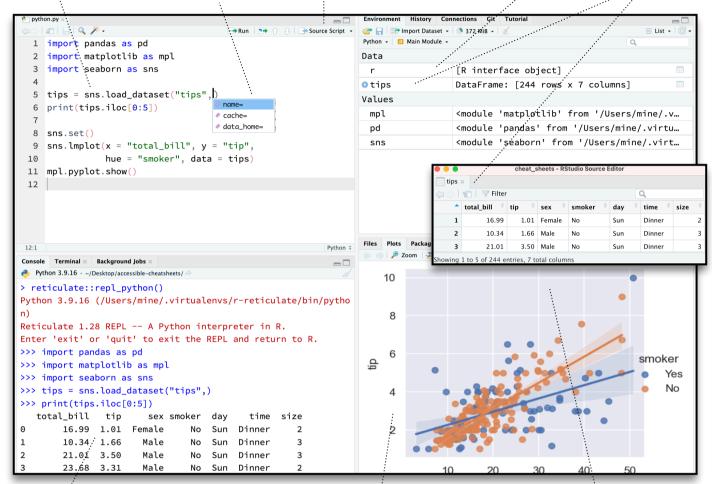
highlighting for Python scripts and chunks.

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

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

View Python objects in the Environment Pane.

View Python objects in the Data Viewer.



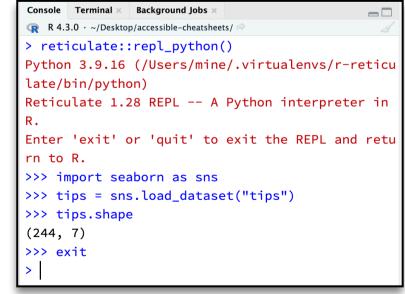
A Python REPL opens in the console when you run Python code with a keyboard shortcut. Type **exit** to close.

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.

- 1. 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.



# **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 virtualeny. Also virtualenv\_remove(). virtualenv\_install("rpandas", 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:

- 1. 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")
- 2. 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")
- 3. A virtual env found in the current working directory: "./.venv"
- 4. Environments that are named after the imported module. e.g. ~/.virtualenvs/r-scipy/ for import("scipy")
- 5. The package default virtualenv, "r-reticulate".
- 6. At the location of the Python binary discovered on the system PATH (i.e. Sys.which("python"))

