Use Python with R with reticulate : : CHEAT SHEET

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.

Add **knitr::knit_engines$set(python = reticulate::eng_python)** to the setup chunk to set up the reticulate Python engine (not required for knitr >= 1.18).

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

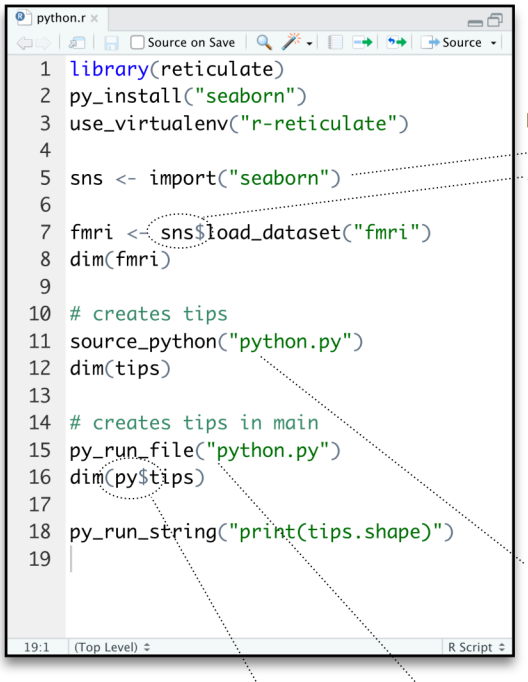
Begin Python chunks with **```{python}**. Chunk options like **echo**, **include**, etc. all work as expected.

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 code

Call Python from R 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. Also **py_call**. *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**. *py_to_r(x)*

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

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

iterate(..., convert = FALSE) 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**. *iterate(iter, print)*

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_capture_output("x")*

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_get_attr(x)*

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!")})*

RStudio