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.

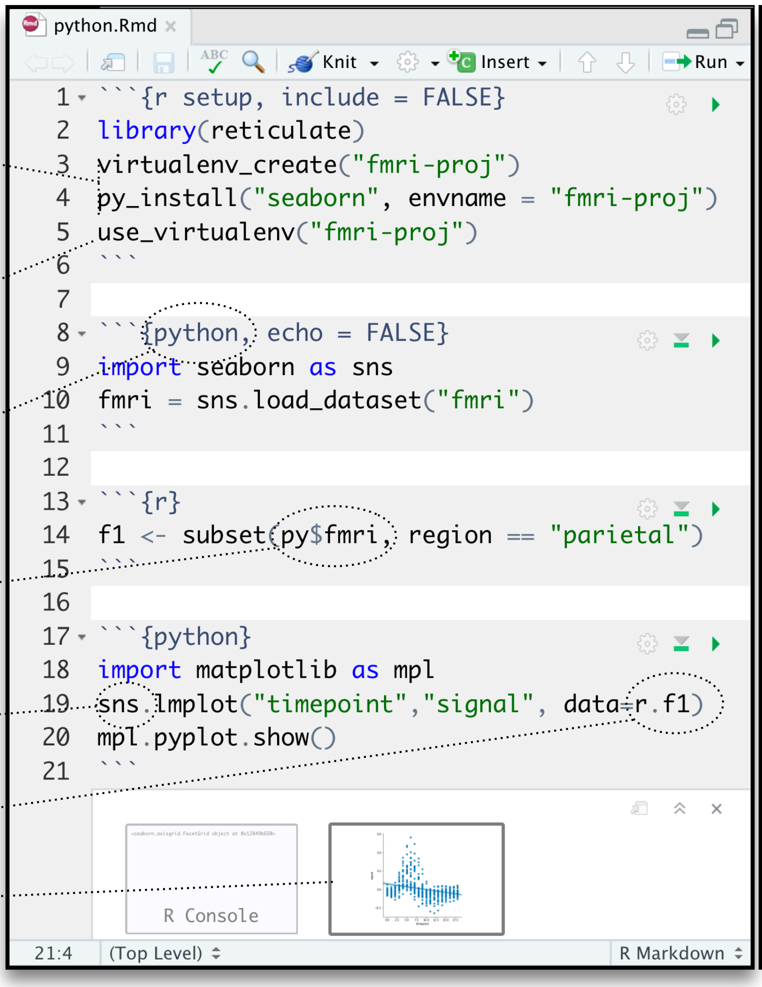
Begin Python chunks with **```{python}**. Chunk options like **echo**, **include**, etc. all work as expected.

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.

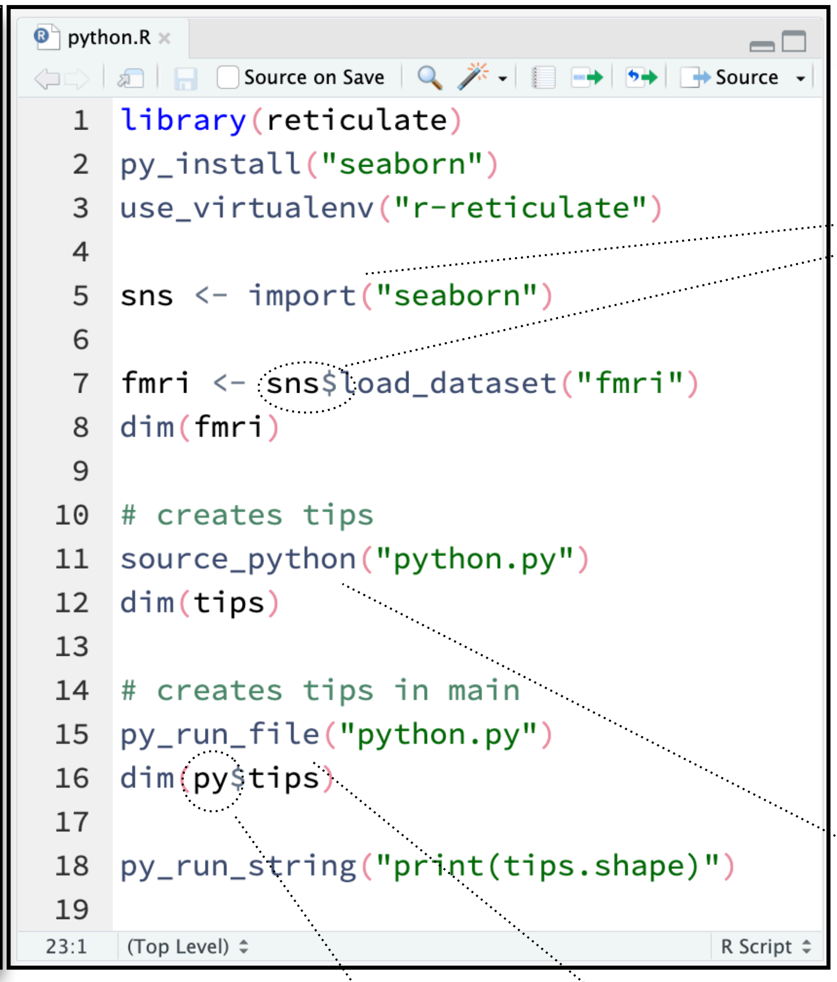


~~~~
```{r setup, include = FALSE}
library(reticulate)
virtualenv_create("fmri-proj")
py_install("seaborn", envname = "fmri-proj")
use_virtualenv("fmri-proj")
```

```{python, echo = FALSE}
import seaborn as sns
fmri = sns.load_dataset("fmri")
```

```{r}
f1 <- subset(py$fmri, region == "parietal")
```

```{python}
import matplotlib as mpl
sns.lmplot("timepoint","signal", data=r.f1)
mpl.pyplot.show()
```
~~~~



```
library(reticulate)
py_install("seaborn")
use_virtualenv("r-reticulate")

sns <- import("seaborn")

fmri <- sns$load_dataset("fmri")
dim(fmri)

# creates tips
source_python("python.py")
dim(tips)

# creates tips in main
py_run_file("python.py")
dim(py$tips)

py_run_string("print(tips.shape)")
```

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. 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(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()**. 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!")})

posit™

Python in the IDE

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

Syntax highlighting for Python scripts and chunks.

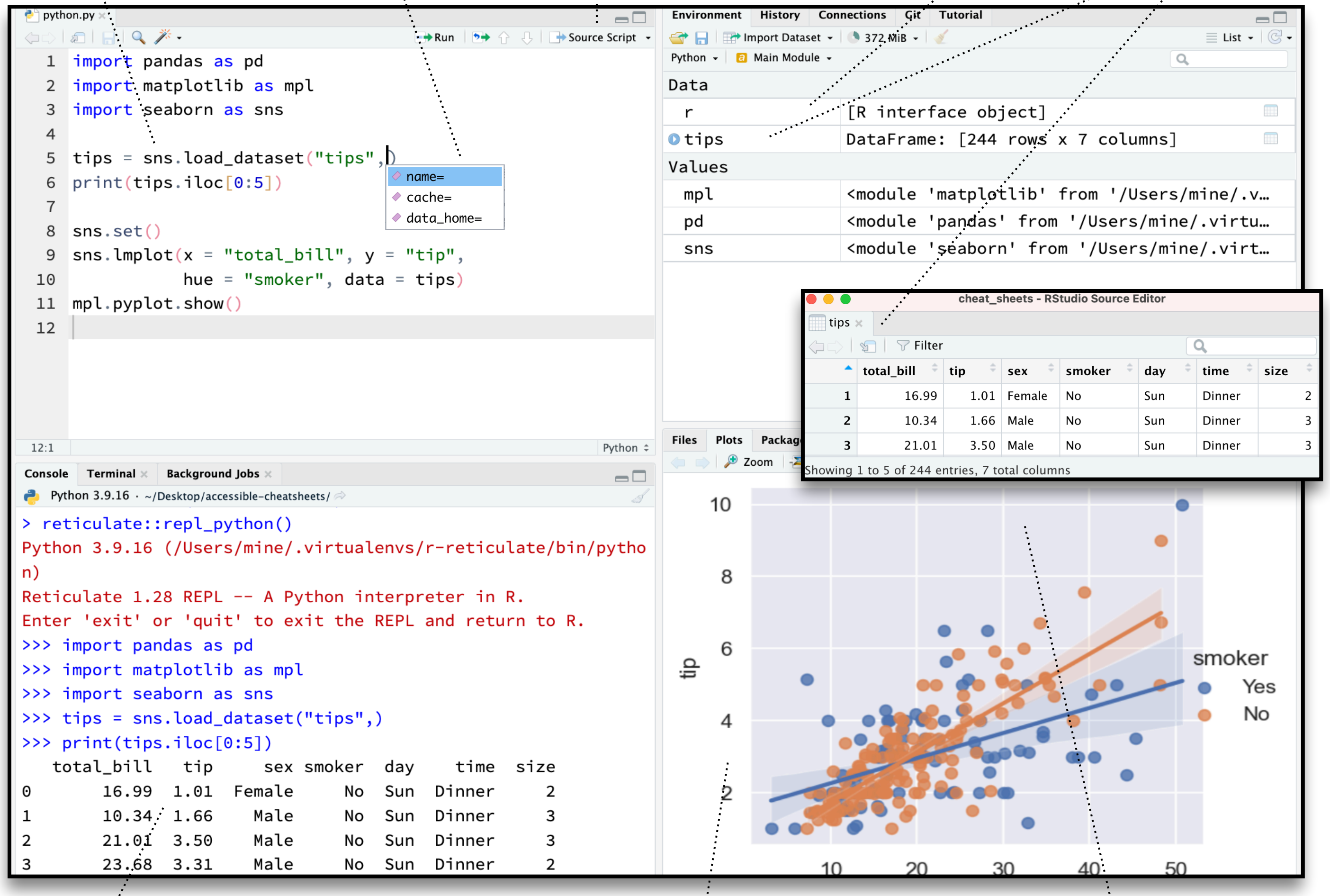
Tab completion for Python functions and objects (and Python modules imported in R scripts).

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.



```python
import pandas as pd
import matplotlib as mpl
import seaborn as sns

tips = sns.load_dataset("tips",)
print(tips.iloc[0:5])

sns.set()
sns.lmplot(x = "total_bill", y = "tip",
           hue = "smoker", data = tips)
mpl.pyplot.show()
```

```
Data
  r          [R interface object]
  tips       DataFrame: [244 rows x 7 columns]
Values
  mpl        <module 'matplotlib' from '/Users/mine/.v…
  pd         <module 'pandas' from '/Users/mine/.virtu…
  sns        <module 'seaborn' from '/Users/mine/.virt…
```

	total_bill	tip	sex	smoker	day	time	size
1	16.99	1.01	Female	No	Sun	Dinner	2
2	10.34	1.66	Male	No	Sun	Dinner	3
3	21.01	3.50	Male	No	Sun	Dinner	3

Showing 1 to 5 of 244 entries, 7 total columns

```
> reticulate::repl_python()
Python 3.9.16 (/Users/mine/.virtualenvs/r-reticulate/bin/pytho
n)
Reticulate 1.28 REPL -- A Python interpreter in R.
Enter 'exit' or 'quit' to exit the REPL and return to R.
>>> import pandas as pd
>>> import matplotlib as mpl
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips",)
>>> print(tips.iloc[0:5])
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
```

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

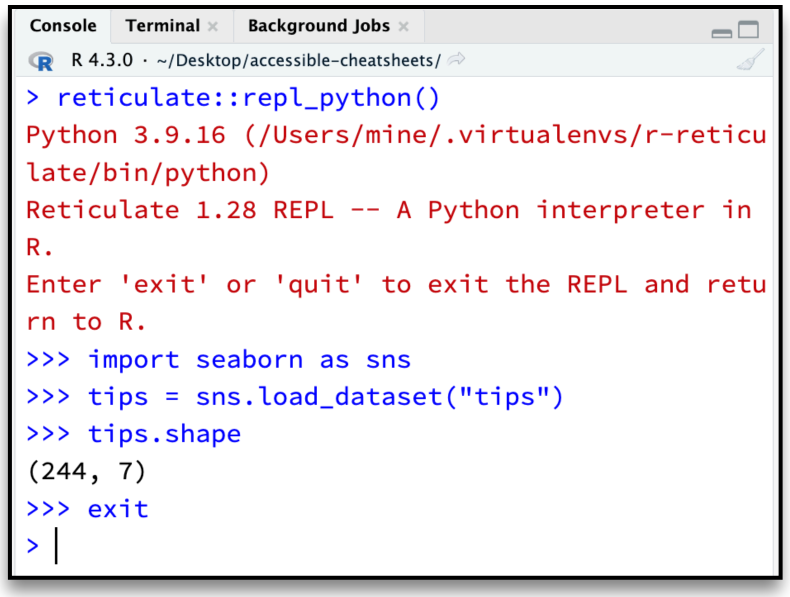
matplotlib plots display in plots pane.

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()
2. Type commands at **>>>** prompt.
3. Press **Enter** to run code.
4. Type **exit** to close and return to R console.



```
> reticulate::repl_python()
Python 3.9.16 (/Users/mine/.virtualenvs/r-reticu
late/bin/python)
Reticulate 1.28 REPL -- A Python interpreter in
R.
Enter 'exit' or 'quit' to exit the REPL and retu
rn to R.
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")
>>> tips.shape
(244, 7)
>>> exit
> |
```

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 all detected versions 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

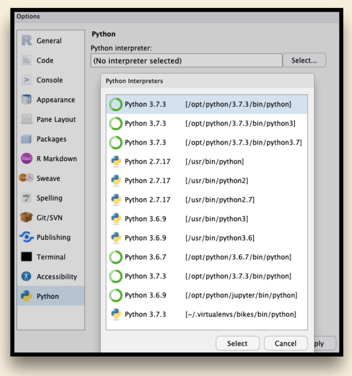
- **py_install(**packages, envname, ...**)** Installs Python packages into a Python env.
  py_install("pandas")
- **virtualenv_install(**envname, packages, ...**)** Install a package within a virtualenv. Also **virtualenv_remove().** virtualenv_install("r-pandas", packages = "pandas")
- **conda_install(**envname,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



```
Python 3.7.3    [/opt/python/3.7.3/bin/python]
Python 3.7.3    [/opt/python/3.7.3/bin/python3]
Python 3.7.3    [/opt/python/3.7.3/bin/python3.7]
Python 2.7.17   [/usr/bin/python]
Python 2.7.17   [/usr/bin/python2]
Python 2.7.17   [/usr/bin/python2.7]
Python 3.6.9    [/usr/bin/python3]
Python 3.6.9    [/usr/bin/python3.6]
Python 3.6.7    [/opt/python/3.6.7/bin/python]
Python 3.7.3    [/opt/python/3.7.3/bin/python]
Python 3.6.9    [/opt/python/jupyter/bin/python]
Python 3.7.3    [~/.virtualenvs/bikes/bin/python]
```

Otherwise, to choose an instance of Python to bind to, reticulate scans the instances on your computer in the following order, **stopping at the first instance that contains the module called by import()**.

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(). Will fail silently if called after import unless **required = TRUE**.
   - **use_python(**python, required = FALSE**)** Suggest a Python binary to use by path.
     use_python("/usr/local/bin/python")
   - **use_virtualenv(**virtualenv = NULL, required = FALSE**)** Suggest a Python virtualenv. use_virtualenv("~/myenv")
   - **use_condaenv(**condaenv = NULL, conda = "auto", required = FALSE**)** Suggest a conda env to use. use_condaenv(condaenv = "r-nlp", conda = "/opt/anaconda3/bin/conda")
3. Within virtualenvs and conda envs that carry the same name as the imported module. e.g. ~/anaconda/envs/nltk for import("nltk")
4. At the location of the Python binary discovered on the system PATH (i.e. Sys.which("python"))
5. At customary locations for Python, e.g. /usr/local/bin/python, /opt/local/bin/python...