Using `reticulate` to access CPTAC data in R

`reticulate` (https://rstudio.github.io/reticulate/) is an R package that allows you to use R create Python objects and convert them into R objects. You can then work with them like any other object in your R environment. Thus, even though `cptac` is a Python package, with the help of `reticulate` you can use it to load and work with the CPTAC datasets in R. All `cptac` features are available through `reticulate` just like they would be in a Python environment, including the `cptac.utils` module and the join functions.

Environment setup

First, you need to load the `reticulate` package, and then tell it which Python environment you want to use to access the `cptac` package.

```r
# Install reticulate if necessary
if (!require(reticulate)) install.packages("reticulate")

# Load the package
library(reticulate)

# Specify to use the environment where the cptac package is installed. Replace
# "dev" with the name of your environment. If you use an environment manager
# besides conda, you'll want this command: use_virtualenv("myenv")
use_condaenv("dev", required = TRUE)
```

Import cptac and load data

Now we will import `cptac` and load the dataset we want.

```r
# Import the package. We pass convert = FALSE so that objects won't be
# converted from Python to R until we explicitly ask for it. This is necessary
# to properly prepare a multiindex dataframe for conversion to R (see below).
cptac <- import("cptac", convert = FALSE)

# Load the dataset
cc <- cptac$Ccrcc()
```

Get a dataframe and convert it to an R object

This dataframe just has a regular column index (not a multiindex), so we can directly convert it into an R object after accessing it.

When `reticulate` converts a `pandas.DataFrame` into R, it converts it to the R `data.frame` type. For a complete list of the type conversion conventions `reticulate` uses between Python and R, see this section of the `reticulate` documentation.

```r
# Load the table
transcript_py <- cc$get_transcriptomics()

# Convert into R
transcript <- py_to_r(transcript_py)
```

```r
print(transcript[1:10, 1:6])
```

```
##                A1BG       A1CF      A2M      A2ML1    A3GALT2    A4GALT
## C3L-00004 0.9953363 16.6778283 353.2634 0.04663386 0.03102656 17.196885
## C3L-00010 0.6793996 16.6827122 359.0784 0.07735049 0.06861737 13.560508
## C3L-00011 0.3545485  0.2456055 222.0754 0.06073557 0.27353611  1.321499
## C3L-00026 2.5437746 16.3475325 228.2823 0.08568399 0.15201998  7.868391
## C3L-00079 4.3552054  4.8589582 275.0902 0.10635897 0.00000000  6.863003
## C3L-00088 1.1142561 13.6544686 452.1893 0.06128475 0.19027928 16.849545
## C3L-00096 1.6246968  8.1072767 213.0509 0.02908392 0.07740076  9.544136
## C3L-00097 1.0602006  4.5412927 496.9853 0.02619470 0.27884676 16.115351
## C3L-00103 1.2943166  1.8534188 738.1870 0.05181419 0.13789259 11.346960
## C3L-00183 1.0913716  6.2932201 566.8782 0.08785637 0.26721294 19.886189
```

Load a multiindex dataframe

If you're loading a dataframe that has a column multiindex, you need to flatten the multiindex before converting it to an R object. We will use the provided `cptac.utils` function for this. See tutorial 4 for more information on column multiindexes.

```r
# Load the table
prot_py_multiindex <- cc$get_proteomics()

# Load cptac.utils so we can access a helper function to convert the multiindex
# to a single level index.
utils <- import("cptac.utils", convert = FALSE)

# Convert the multiindex to a single level index. This works because we passed
# convert = FALSE when we imported the cptac package, so the dataframe is still
# a Python object at this point (it  wasn't automatically converted into an R
# object on loading) and can be passed to this Python function.
prot_py_single_index <- utils$reduce_multiindex(prot_py_multiindex, flatten = TRUE)

# Convert to R
prot <- py_to_r(prot_py_single_index)

print(prot[1:10, 1:3])
```

```
##           A1BG_NP_570602.2 A1CF_NP_620310.1 A2M_NP_000005.2
## C3L-00004      -0.30430193        0.6414471   -2.482087e-05
## C3L-00010       1.19591477        0.1946199    1.360294e+00
## C3L-00011      -0.28615529       -0.7804553   -1.010889e-01
## C3L-00026       0.13573040        0.4042856    2.613837e-01
## C3L-00079      -0.12395875       -0.6777732   -3.625469e-01
## C3L-00088       0.42754207        0.3102494    1.308011e+00
## C3L-00096      -0.24210674       -0.1287324   -2.562006e-01
## C3L-00097       0.50646874       -0.5132426   -4.952483e-01
## C3L-00103       0.72083588       -1.1358590    4.887083e-01
## C3L-00183       0.08294611       -0.1280677    1.555405e-01
```

Use another `cptac.utils` function

As you saw in the previous example, we can access functions from `cptac.utils` through `reticulate`. In fact, all functions provided in `cptac.utils` are available through `reticulate`. Here we demonstrate a call to the get_frequently_mutated function.

```
# Note that we already imported the utils module above when we were using
# the reduce_multiindex function. If we hadn't already done that, we'd
# need to run the command on the following line:
# utils <- import("cptac.utils", convert = FALSE)

# Call the function
freq_mut_py <- utils$get_frequently_mutated(cc)

# Convert to R
freq_mut <- py_to_r(freq_mut_py)

print(freq_mut)
```

```
##    Gene Unique_Samples_Mut Missense_Mut Truncation_Mut
## 1  BAP1          0.1545455   0.06363636     0.09090909
## 2 KDM5C          0.1727273   0.03636364     0.14545455
## 3 PBRM1          0.4000000   0.07272727     0.33636364
## 4 SETD2          0.1363636   0.01818182     0.11818182
## 5   TTN          0.1181818   0.09090909     0.03636364
## 6   VHL          0.7454545   0.30000000     0.44545455
```

Use a join function

Using reticulate, you can also access all the table joining functions provided by cptac.

```
# Call the join function
transcript_and_mut_py <- cc$join_omics_to_mutations(
  omics_df_name = "transcriptomics",
  mutations_genes = "VHL",
  omics_genes = "PBRM1",
  quiet = TRUE
)

# Convert to R
transcript_and_mut <- py_to_r(transcript_and_mut_py)

print(transcript_and_mut[1:10,])
```

```
##           PBRM1_transcriptomics      VHL_Mutation VHL_Location
## C3L-00004              8.842076 Missense_Mutation      p.V130F
## C3L-00010              8.847511 Nonsense_Mutation      p.R161*
## C3L-00011             12.055053 Missense_Mutation      p.V155M
## C3L-00026              8.077332    Frame_Shift_Ins p.A149Cfs*25
## C3L-00079             15.106487    Frame_Shift_Del  p.P61Afs*69
## C3L-00088             10.739359 Missense_Mutation       p.L89P
## C3L-00096              9.011007 Missense_Mutation      p.L178P
## C3L-00097             11.617714 Nonsense_Mutation       p.E70*
## C3L-00103             11.084372 Missense_Mutation       p.L85P
## C3L-00183             12.484973     Wildtype_Tumor  No_mutation
##           VHL_Mutation_Status Sample_Status
## C3L-00004     Single_mutation         Tumor
## C3L-00010     Single_mutation         Tumor
## C3L-00011     Single_mutation         Tumor
## C3L-00026     Single_mutation         Tumor
## C3L-00079     Single_mutation         Tumor
## C3L-00088     Single_mutation         Tumor
## C3L-00096     Single_mutation         Tumor
## C3L-00097     Single_mutation         Tumor
## C3L-00103     Single_mutation         Tumor
## C3L-00183      Wildtype_Tumor         Tumor
```

Note that if we wanted to pass an empty list to the `mutations_filter` parameter in order to use the default mutation filter, we can't just do something like `mutations_filter = r_to_py(c())`. The reason is that the `r_to_py` function converts and empty R vector into a `None` in Python, not an empty list. However, we can solve our problem by first creating a non-empty vector, converting it to a Python list, and then calling the list's `clear` method:

```r
empty_filter <- r_to_py(c(0, 0)) # Vector must have more than one element
empty_filter$clear()
```

`empty_filter` is now an empty Python list, and can be passed to `join_omics_to_mutations` to get the default mutations filter.

If you're using R Markdown, another solution is to just execute the command in a Python chunk in your R Markdown document, passing an empty list directly using Python's syntax, and then access the output from the next R code chunk. This is demonstrated in the next section.

`reticulate`'s Python engine for R Markdown

If you're using R Markdown, `reticulate` provides a Python engine that allows you to run Python code chunks, and then access objects within them from R through an object called `py`. Similarly, R objects are accessible from Python chunks through an object named `r`.

Here is a Python code chunk that performs the same join function as the previous example, but with an empty list as the mutations filter:

```python
import cptac
cc = cptac.Ccrcc()
```

```python
transcript_and_mut = cc.join_omics_to_mutations(
  omics_df_name="transcriptomics",
  mutations_genes="VHL",
  omics_genes="PBRM1",
  mutations_filter=[],
  quiet=True
)
```

And here we access the output in an R chunk, where we could then perform further analysis.

```r
transcript_and_mut_2 <- py$transcript_and_mut
print(transcript_and_mut_2[1:10,])
```

```
##           PBRM1_transcriptomics      VHL_Mutation VHL_Location
## C3L-00004              8.842076 Missense_Mutation      p.V130F
## C3L-00010              8.847511 Nonsense_Mutation      p.R161*
## C3L-00011             12.055053 Missense_Mutation      p.V155M
## C3L-00026              8.077332   Frame_Shift_Ins p.A149Cfs*25
## C3L-00079             15.106487   Frame_Shift_Del  p.P61Afs*69
## C3L-00088             10.739359 Missense_Mutation       p.L89P
## C3L-00096              9.011007 Missense_Mutation      p.L178P
## C3L-00097             11.617714 Nonsense_Mutation       p.E70*
## C3L-00103             11.084372 Missense_Mutation       p.L85P
## C3L-00183             12.484973    Wildtype_Tumor  No_mutation
##           VHL_Mutation_Status Sample_Status
## C3L-00004     Single_mutation         Tumor
## C3L-00010     Single_mutation         Tumor
## C3L-00011     Single_mutation         Tumor
## C3L-00026     Single_mutation         Tumor
## C3L-00079     Single_mutation         Tumor
## C3L-00088     Single_mutation         Tumor
## C3L-00096     Single_mutation         Tumor
## C3L-00097     Single_mutation         Tumor
## C3L-00103     Single_mutation         Tumor
## C3L-00183      Wildtype_Tumor         Tumor
```