# reticulate

Nina Norgren

> **Note**
>
> These are exercises for practicing to use the `reticulate` package in R. Try to do the exercises yourself before looking at the answers. Some sections have more pure Python code than others, if you feel that your Python skills are rusty, feel free to look at the answers and try your best to follow along. We will be writing some Python code chunks, so use R Markdown for this exercise.
> Please download the following file and unzip the contents before the lab.
> raukr-reticulate.zip

## Setup

Load the following libraries else install them from CRAN.

```
library(reticulate)
library(ggplot2)
library(purrr)
library(stringr)
```

Create a virtual environment and install Python libraries.

```
py_require(c("pandas==2.2.1","sqlalchemy==2.0.30"))
```

Reticulate will automatically figure out which python version to use. If you want to manually specify the version you can use `use_python(<pythonpath>)`.

## IMdB

### Preparations

The International Movie Database is a large database containing all information about movies, TV series, actors, producers, etc, and the ratings they received. If you are not aware of it, check out their website imdb.com for more information.

You will be working on a smaller subset of some of the data listed, which consists of movies, ratings, and the principal actors playing in the movies. You will receive a file with python functions used to query this small database from R, where you will further process the data to answer questions related to different movies and actors. The underlying Python code uses the `sqlalchemy` library for querying the `sqlite` database.

In preparation for using the Python code in R, make sure that the following files are all located in your working directory:

1. `imdb.db`
2. `model.py`
3. `imdb_functions.py`

Start by loading all the python functions into R.

**Show code**

```
source_python("imdb_functions.py")
```

First, inspect which functions got imported when you sourced your python file. You can find them in the **Environment** table in RStudio. Some of the functions listed are part of the SQLAlchemy package used, but one example to look at is the function `get_actors()`.

As you can see, R creates a wrapper function in R, for calling the underlying Python function. This specific function takes a movie title as input, and returns the principal actors of the movie. You can further study what the function does by looking at the code in the `imdb_functions.py` file. You can see that it queries the database for a specific movie, and returns the principal actors in it.

### Get to know the data

Let's try out the `get_actors()` function. Get the principal actors for the movie **Gattaca**, and inspect the output type.

**Show code**

```
actors <- get_actors('Gattaca')
str(actors)
```

```
List of 1
 $ Gattaca: chr [1:4] "Ethan Hawke (Vincent,Jerome)" "Uma Thurman (Irene)" "Jude Law (Je
```

Next let's do the same with the function `get_movies()`. List movies that **Brent Spiner** has been in.

**Show code**

```
movies <- get_movies('Brent Spiner')
str(movies)
```

```
List of 1
 $ Brent Spiner: chr [1:3] "Star Trek: First Contact" "Star Trek: Insurrection" "Star Tr
```

For printing some basic information about a movie, without saving anything to an R object, use the `print_movie_info` function. Here, find out information about the **Avengers** movies.

**Show code**

```
print_movie_info('Avengers')
```

```
Title:  The Avengers
Year:  1998
Runtime (min):  89
Genres:  Action,Adventure,Sci-Fi
Average rating:  3.8
Number of votes:  41414

Title:  The Avengers
Year:  2012
Runtime (min):  143
Genres:  Action,Adventure,Sci-Fi
Average rating:  8.0
Number of votes:  1283281

Title:  Avengers: Age of Ultron
Year:  2015
Runtime (min):  141
Genres:  Action,Adventure,Sci-Fi
Average rating:  7.3
Number of votes:  769172

Title:  Avengers: Infinity War
Year:  2018
Runtime (min):  149
Genres:  Action,Adventure,Sci-Fi
Average rating:  8.4
Number of votes:  881191

Title:  Avengers: Endgame
Year:  2019
Runtime (min):  181
Genres:  Action,Adventure,Drama
Average rating:  8.4
Number of votes:  880234
```

Capture the output from the previous function and save it as a variable.

**Show code**

```
output <- py_capture_output(print_movie_info('Avengers'))
cat(output)
```

```
Title:  The Avengers
Year:  1998
Runtime (min):  89
Genres:  Action,Adventure,Sci-Fi
Average rating:  3.8
Number of votes:  41414

Title:  The Avengers
Year:  2012
Runtime (min):  143
Genres:  Action,Adventure,Sci-Fi
Average rating:  8.0
Number of votes:  1283281

Title:  Avengers: Age of Ultron
Year:  2015
Runtime (min):  141
Genres:  Action,Adventure,Sci-Fi
Average rating:  7.3
Number of votes:  769172

Title:  Avengers: Infinity War
Year:  2018
Runtime (min):  149
Genres:  Action,Adventure,Sci-Fi
Average rating:  8.4
Number of votes:  881191

Title:  Avengers: Endgame
Year:  2019
Runtime (min):  181
Genres:  Action,Adventure,Drama
Average rating:  8.4
Number of votes:  880234
```

Inspect the types of the variables `actors` and `movies`. What type are they? What type where

they converted from in Python?

**Show code**

```
str(actors)
```

```
List of 1
 $ Gattaca: chr [1:4] "Ethan Hawke (Vincent,Jerome)" "Uma Thurman (Irene)" "Jude Law (Je
```

```
str(movies)
```

```
List of 1
 $ Brent Spiner: chr [1:3] "Star Trek: First Contact" "Star Trek: Insurrection" "Star Tr
```

```
# Both are of the R type `named list`, which is the type a Python `dictionary` gets conv
```

Source the python file again, but set `convert=FALSE`. What are the types now?

**Show code**

```
source_python("imdb_functions.py", convert = FALSE)

actors <- get_actors('Gattaca')
class(actors)
```

```
[1] "python.builtin.dict"   "python.builtin.object"
```

```
movies <- get_movies('Brent Spiner')
class(movies)
```

```
[1] "python.builtin.dict"   "python.builtin.object"
```

```
# Now actors and movies are both of the python type dictionary
```

Convert the types manually back to R types.

**Show code**

```
actors.r <- py_to_r(actors)
str(actors.r)
```

```
List of 1
 $ Gattaca: chr [1:4] "Ethan Hawke (Vincent,Jerome)" "Uma Thurman (Irene)" "Jude Law (Je
```

```
movies.r <- py_to_r(movies)
str(movies.r)
```

```
List of 1
 $ Brent Spiner: chr [1:3] "Star Trek: First Contact" "Star Trek: Insurrection" "Star Tr
```

### Working with Dataframes

In the following sections we will be working with pandas dataframes in R. The answers we show will mostly be using the Python `pandas` library from R, but there are of course pure R ways of doing the following exercises once we have converted the output from the python functions. You are free to choose how you solve the following exercises, either only python in R, a mix, or pure R. But we encourage you to mix, as you will then practice the type conversions and usages of the `reticulate` library, especially for those of you that are more fluent in Python.

### The highest ranked movie

The function `get_all_movies()` from the file `imdb_functions.py` can be used to retrieve all movies, either within a specified time period, or all of the movies in the database. If the imported function has a docstring, you can view the help documentation with:

```
py_help(get_all_movies)
```

Start by importing all movies into a pandas dataframe, by sourcing the python functions into R. Do *not* convert the result into an R dataframe.

**Show code**

```
source_python("imdb_functions.py", convert = FALSE)
movies_py <- get_all_movies()
class(movies_py)
```

```
[1] "pandas.core.frame.DataFrame"         "pandas.core.generic.NDFrame"
[3] "pandas.core.base.PandasObject"       "pandas.core.accessor.DirNamesMixin"
[5] "pandas.core.indexing.IndexingMixin" "pandas.core.arraylike.OpsMixin"
[7] "python.builtin.object"
```

Inspecting the `movies_py` variable we can see that it is of the type `pandas.dataframe`.

Now we are ready to answer our first question:

**Which movie/movies are the highest ranked of all times?**

We will try to answer this with a pandas method *directly* in a Python chunk. To do this we first have to make our `movies_py` variable visible to Python. Even though it is a Python object, since it was created within a R code chunk, Python code chunks cannot directly access them. To make R variables accessible in Python code chunks we use the `r` object. Remember that to access a Python variable from R, we used `py$`, to do the opposite we use `r.`. The $ and the . denotes the different ways in which Python and R represents methods.

Use the method `.max()` from the `pandas` module to find and filter out the top movie/movies.

**Show code**

```
# the code below is python code written in a python code chunk
movies = r.movies_py

# inspect what columns are present
movies.columns
```

```
Index(['id', 'tconst', 'titleType', 'primaryTitle', 'originalTitle',
       'startYear', 'endYear', 'runtimeMinutes', 'genres', 'averageRating',
       'numVotes'],
      dtype='object')
```

```
# find movies that has the highest averageRating
top_movies = movies[movies.averageRating == movies.averageRating.max()]

top_movies['primaryTitle']
```

```
3822    The Shawshank Redemption
5450               The Chaos Class
Name: primaryTitle, dtype: object
```

Above we are using pure pandas code directly in our RMarkdown document.

After dipping our toes in Python territory, we now go back to using normal R chunks:

Save `top_movies` as an R object, and find out from what years these movies are, and how many votes they got.

**Show code**

```
movies_r <- py$top_movies
df <- data.frame(movies_r$primaryTitle, movies_r$startYear, movies_r$numVotes)
df
```

```
     movies_r.primaryTitle movies_r.startYear movies_r.numVotes
1 The Shawshank Redemption               1994           2399394
2          The Chaos Class               1975             38290
```

So the answer to which are the highest ranked movies of all times is **The Shawshank Redemption** and **The Chaos Class**. Although, The Chaos Class did not get as many votes as The Shawshank Redemption.

### Average ratings over time

Next we want to explore how the average ratings for movies has changed over time. This one we will solve in normal R chunks, by importing the required python functions from the file `imdb_functions.py`, and also load `pandas` into R. As we will be using pandas in R, import the Python file without converting it.

Get all movies and save into a pandas dataframe.

**Show code**

```
source_python("imdb_functions.py", convert = FALSE)
movies_py <- get_all_movies()
class(movies_py)
```

```
[1] "pandas.core.frame.DataFrame"         "pandas.core.generic.NDFrame"
[3] "pandas.core.base.PandasObject"       "pandas.core.accessor.DirNamesMixin"
[5] "pandas.core.indexing.IndexingMixin" "pandas.core.arraylike.OpsMixin"
[7] "python.builtin.object"
```

Import pandas into R

**Show code**

```
pandas <- import("pandas")
```

Use pandas to group the data by startYear, and calculate the average ratings. Next, convert the result back into an R dataframe.

**Show code**

```
# use pandas to group columns by startYear
movies_grouped <- movies_py$groupby('startYear')['averageRating']$mean()

# convert to r dataframe
movies_grouped_r <- py_to_r(movies_grouped)
movies_grouped_r <- as.data.frame(movies_grouped_r)
```

In this case the conversion seems to have done something to our columns. To fix this, add startYear column back into the dataframe, using R.

**Show code**

```
# add Year column back to dataframe, and rename columns
movies_grouped_r <- cbind(rownames(movies_grouped_r), movies_grouped_r)
colnames(movies_grouped_r) <- c("startYear","averageRating")
movies_grouped_r[1:4,]
```

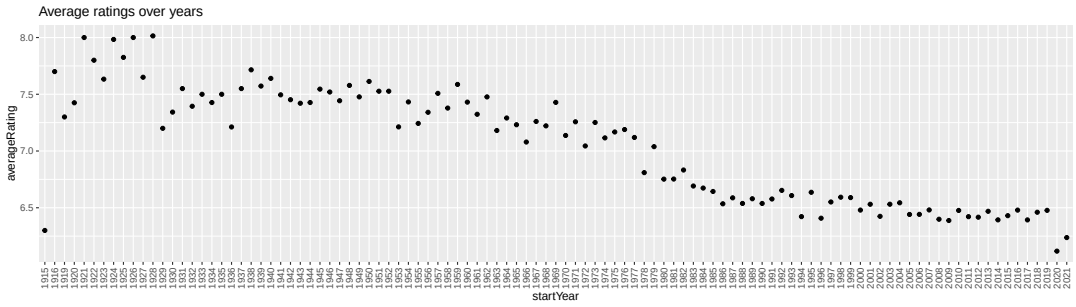
```
     startYear averageRating
1915      1915         6.300
1916      1916         7.700
1919      1919         7.300
1920      1920         7.425
```

Make sure to inspect that the dataframe looks like it is supposed to, and that the values make sense. Once we are sure we have managed to transform the data, we can proceed.

Plot the average ratings for each year.

**Show code**

```
ggplot(movies_grouped_r, aes(x=startYear, y=averageRating)) +
                                  geom_point() +
                                  theme(axis.text.x = element_text(angle = 90, vjust = 0.5,
                                  ggtitle('Average ratings over years')
```




## Bonus exercises

Below is 2 bonus exercises if you have time left in the end of the exercise. Do either one or both if you have the time.

### Find overlapping actors

Which actors have played together with both Ian McKellen and Patrick Stewart, but when they were in separate movies? Or rephrased, which actor has played with Ian McKellen Lee in one movie, and Patrick Stewart in another movie?

For example:

1. Actor 1 has played with IM in movie a, and with PS in movie b. PS was not in movie a, and IM was not in movie b
2. Actor 2 has played with IM in movie c, and with PS in movie c.

Scenario 1 would count, while scenario 2 would not, as IM and PS was in this movie both together.

To solve this one you need to think in several steps. There are of course several solutions, and you are free to approach this exercise however you want. We will give you a suggestion to one approach that could be used below:

Tip

- Get a list of movies where Ian McKellen has played
- Get a list of movies where Patrick Stewart has played
- Remove intersections
- Get all actors for all movies that Ian McKellen was in
- Get all actors for all movies that Patrick Stewart was in
- Remove duplicates
- Get intersection of actors

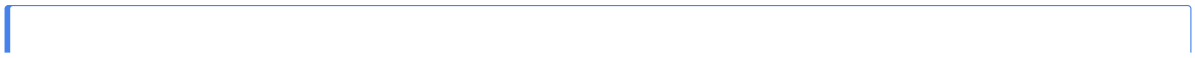
Tip

Remember that this database only has the PRINCIPAL actor of movies, meaning you might have results where an actor has a minor role and is not listed here. If you are unsure if your results are correct, we provide you with a Python function to check your results.

To find out if your answer is correct, your can import and use the function `check_results` from the `imdb_functions.py` file. Replace `'Actor Name'` with the name of the actor that you think is the answer to the question above.

```
source_python("imdb_functions.py")
res_actor <- 'Actor Name'
check_results(res_actor, 'Ian McKellen', 'Patrick Stewart')
```

And if you want to see one suggested solution to this problem:



**Show code**

```
source_python("imdb_functions.py", convert = FALSE)
act1 <- 'Ian McKellen'
act2 <- 'Patrick Stewart'

# get movies for Patrick Stewart
act1_movies <- get_movies(act1)
act1_movies
## {'Ian McKellen': ['The Keep', 'Six Degrees of Separation', 'Richard III', 'Apt Pupil'
movies1_lst <- py_to_r(act1_movies[act1])

# get movies for Ian McKellen
act2_movies <- get_movies(act2)
act2_movies
## {'Patrick Stewart': ['Star Trek: Generations', 'Star Trek: First Contact', 'Conspirac
movies2_lst <- py_to_r(act2_movies[act2])

# get movies both has played in
overlap <- intersect(movies1_lst, movies2_lst)

# remove overlap from each movielist
new_movies1_lst <- setdiff(movies1_lst, overlap)
new_movies2_lst <- setdiff(movies2_lst, overlap)

# get all actors that has played in those movies
# below we do things the functional way for the first
# movie list
actors_lst <- purrr::map(new_movies1_lst,
             ~ .x %>%
             get_actors() %>%
             py_to_r() %>%
```

Try some other actors and see what you find. For example, try actors that have played with *Johnny Depp* and *Helena Bonham Carter*.

### Try out examples from slides

In the slides we looked at a few examples of how we can use reticulate to use python libraries in R:

1. Random forest classification using Scikit-learn
2. Getting gene information from ENSEMBL's API
3. Do Natural language processing using Hugging Face models

Go back to the slides and try some of them out! Here you would have to read the documentation for the different libraries. Remember these are Python libraries that you import in R, so the syntax has to be updated as we have discussed previously.

## Session

Click here

```
sessionInfo()
```

```
R version 4.5.1 (2025-06-13)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04.2 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so;  LAPACK version 3

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: UTC
tzcode source: system (glibc)

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base
```

```
other attached packages:
[1] stringr_1.5.1     purrr_1.0.4       ggplot2_3.5.2     reticulate_1.42.0

loaded via a namespace (and not attached):
 [1] Matrix_1.7-0       gtable_0.3.6       jsonlite_2.0.0     dplyr_1.1.4
 [5] compiler_4.5.1     renv_1.0.9         tidyselect_1.2.1   Rcpp_1.0.14
 [9] scales_1.4.0       png_0.1-8          yaml_2.3.10        fastmap_1.2.0
[13] lattice_0.22-6     here_1.0.1         R6_2.6.1           labeling_0.4.3
[17] generics_0.1.3     knitr_1.50         tibble_3.2.1       rprojroot_2.0.4
[21] pillar_1.10.2      RColorBrewer_1.1-3 rlang_1.1.6        stringi_1.8.7
[25] xfun_0.52          cli_3.6.5          withr_3.0.2        magrittr_2.0.3
[29] digest_0.6.37      grid_4.5.1         lifecycle_1.0.4    vctrs_0.6.5
[33] evaluate_1.0.3     glue_1.8.0         farver_2.1.2       rmarkdown_2.29
[37] tools_4.5.1        pkgconfig_2.0.3    htmltools_0.5.8.1
```