Incorporating Python in R (Under Windows)

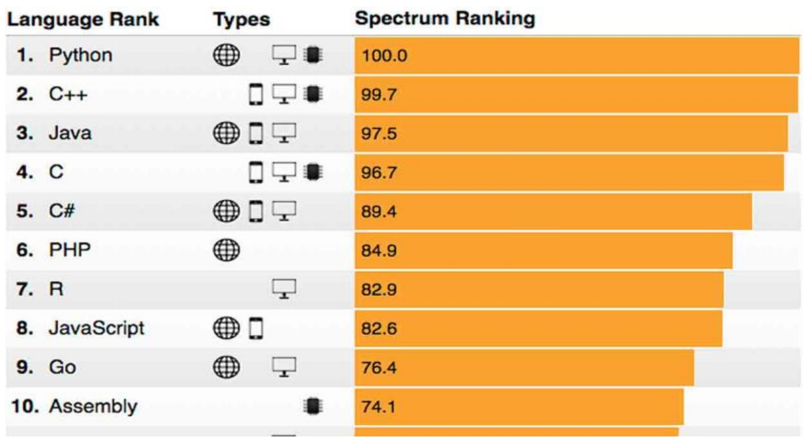
Pan Liu 11253308 HEC

I. Introduction

Python and R are both popular and powerful languages for data science. Their backgrounds decide that they have different characteristics and resources which could be complementary. It could be of interest to incorporate the two in the same platform.

1. Python vs. R

Python and R are two mainstream programming language for data science. Python is a versatile general-purpose language, widely used in machine learning and data science as well as other application scenarios. While R is a language specialized for data manipulation, statistical computing and graphics [1]. Both languages have gained immense popularity (Fig. 1(a) & 1(b)), largely thanks to the boom of data science and machine learning in recent years.



Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥 ▪	100.0
2. C++	📱 🖥 ▪	99.7
3. Java	🌐 📱 🖥	97.5
4. C	📱 🖥 ▪	96.7
5. C#	🌐 📱 🖥	89.4
6. PHP	🌐	84.9
7. R	🖥	82.9
8. JavaScript	🌐 📱	82.6
9. Go	🌐 🖥	76.4
10. Assembly	▪	74.1

Fig. 1 (a) Top 10 programming language of 2018, by IEEE SPECTRUM [2]

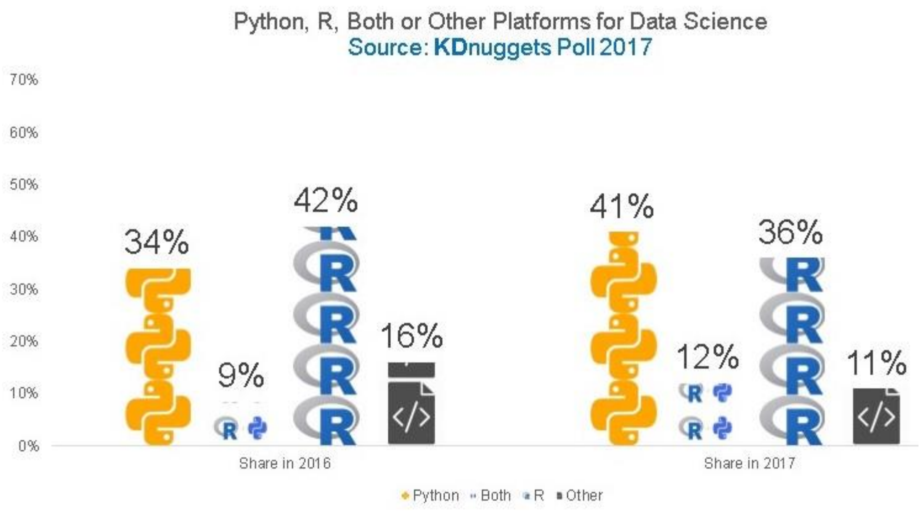


Fig. 1 (b) Language usage for data science [3]

History and development:

- **R:**

R was originated from the S language developed in Bell lab at 1976, which was designed for statistical computing. At early 1990s, two statisticians from University of Auckland, Robert Gentleman & Ross Ihaka, drew features from the old languages and developed the R language [4]. Until today, they are still active in maintaining the R project, together with the "R Core Team", composed of tens of researchers around the world mainly working in statistics-related fields [5].

R have a strong active community who contributes a huge growing number of packages with various functions, which makes the R environment more and more powerful. In the official repository Comprehensive R Archive Network (CRAN) there are already over 10,000 packages. Apart from CRAN, you can also find R resources on other platforms including R-Forge, Bioconductor and Git-hub.

- **Python:**

The creation of Python was largely influenced by the ABC language, a general-purpose language designed to be easily readable. In 1991, Guido van Rossum, a Dutch programmer who had worked in the ABC project, created and released the first Python version. Today the Python Software Foundation membered by tens of thousands of developers, is responsible for maintaining the Python.

Python also has a vast community who actively contributed to numerous packages that add to the increasing ability of Python. Especially in the machine learning field, Python is widely considered as the most resourceful platform. The official third-party repository for Python packages is the Python Package Index (PyPI) where over 100,000 packages are hosted.

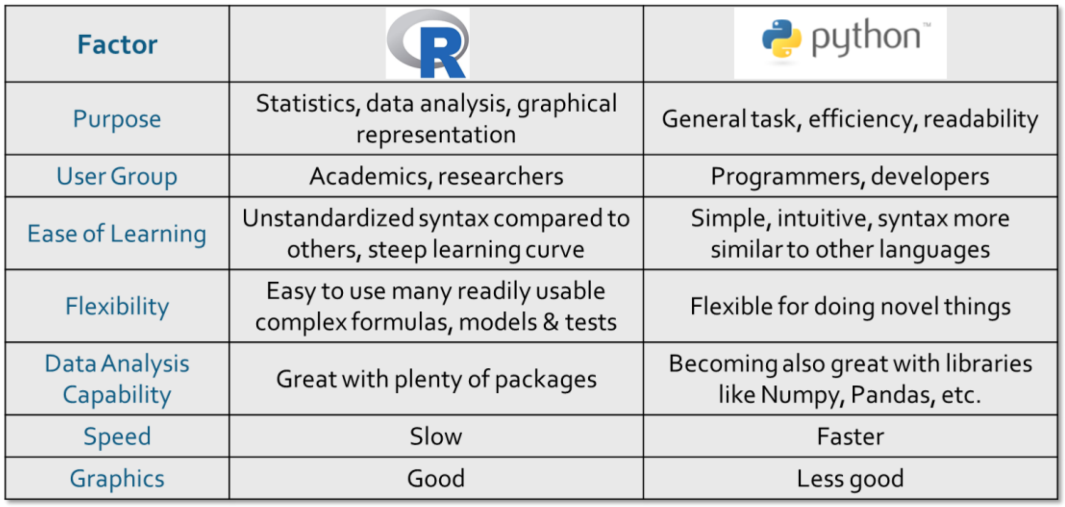
Characteristics:

Both R and Python have abundant dedicated packages for data science, and are excellent choices for reproducible research and analysis. While their different "backgrounds" decide there are still differences in their characteristics concerning data science (Fig. 2).

Broadly, R is more dedicated to statistical computing and visualization, you can directly apply many readily usable complex formulas, models and tests. While Python is a general-purpose language that can be versatile at different tasks, and more flexible for doing novel things.

Moreover, from their history we know that unlike Python, R is not developed by computer scientists or programmers, but rather by researchers who need tools to effectively design and perform statistical experiments and communicate results [6]. By design you can use readily usable simple R commands to perform complex operations. While this makes the syntax of R tends to be more concise yet less "standard" or readable, compared to Python. So, it could be more difficult for beginners to start with R programming than Python. And in the technical aspect, Python has better performance than R, in terms of computation speed and handling large data size (The speed issue in R is partly addressed by using C behind some computation intensive functions).

One more thing that we need to notice as a researcher: R tends to be more "professional" in statistics than Python. Because the authors are usually professionals who develop the package for their own research, so they are more likely to know the scientific stuff clearly. While Python package often has many contributors varying in their backgrounds [7].



Factor	R	python
Purpose	Statistics, data analysis, graphical representation	General task, efficiency, readability
User Group	Academics, researchers	Programmers, developers
Ease of Learning	Unstandardized syntax compared to others, steep learning curve	Simple, intuitive, syntax more similar to other languages
Flexibility	Easy to use many readily usable complex formulas, models & tests	Flexible for doing novel things
Data Analysis Capability	Great with plenty of packages	Becoming also great with libraries like Numpy, Pandas, etc.
Speed	Slow	Faster
Graphics	Good	Less good

Fig 2. Summary of some characteristics of R and Python

For more comprehensive comparison between R and Python, here is a nice article: [8].

2. Why incorporating Python in R

As we have seen, the two language have many **complementary qualities and resources**. You may want to combine the versatility and speed of Python with the statistics and visualization expertise of R. And sometimes you may come across certain applications where you get much handier packages and API support in Python than in R.

For example, we can imagine a workflow: one uses Python to scrap and clear data from the web, then applies some readily available functions in R to build and test prototype models on the sample data; at a certain point he feels like using the *scikit-learn* package to tune some models and performing analysis on a larger data set with Python. Finally, he uses R to generate fancy graphical representations of results.

Apart from complementarity, another obvious motivation for incorporating Python in R could be to **transfer your existing knowledge and code**. If you already proficient with Python, you can easily transfer your skills, even code, directly into R environment when you need to.

Of course, the application of Python and R can be done separately and consecutively in their own IDEs, but obviously it will be much more convenient if we are able to use the two languages in the same platform without manually transfer the data.

II. Structure and Methodology of Report

Having discussed why it's interesting to incorporate Python in R, in the following of this report, I would like to concretely introduce how to incorporate Python in R environment, mainly under Windows system.

Firstly, a brief review of relevant discussions and documentations will be given. Then, a summary of related R resources will be presented. For the main part, I will introduce the major approaches and key aspects of how to use Python in R, mainly empowered by the *reticulate* packages. And at the end, an exemplary data analysis will be performed to demonstrate the procedure.

A correlated tutorial, where those approaches of incorporating Python in R are exhibited with concrete instances, will be given in the appendix R Markdown file (source-code also available). Many technical aspects and potential issues that I don't go into details in this report, would be explained within the tutorial.

III. Literature Review

Since this is mainly a technical subject, in this "literature" review part I would mainly like to share some interesting articles and helpful tutorials and documentations that I personally found useful.

1. **Discussion and comparison concerning R and Python**:
   On the internet there is a heated debate about which of the 2 languages is the best for data science. You can google this subject are find many discussions. I read some of them and here I just want to share 2 articles that I found particularly informative.
   For more comprehensive comparison between R and Python, reference [8] is a nice article from the website DataCamp. The author made a beautiful infographic which exhibits the pros and cons of both language, concerning applications in data science.
   In another article [6], the examination of key strengths of R and Python was made, followed by a tutorial of integrating Python and R.

2. **Documentations and tutorials**:
   Ref. [9] is from Rstudio blog where the CEO of Rstudio announced *reticulate* package. In this announcement they gave a nice brief introduction and guide on the *reticulate*.
   The same introduction can be found on the homepage of ***reticulate* website: [10]**. On this official website we are provided with links towards detailed documentations about various sorts of topics concerning using *reticulate*. If you have problems on concrete technical issues, this might be the first place you want to check.
   Ref. [11]: This article is a short yet not very comprehensive tutorial on how to call or run python from R using Reticulate.
   Moreover, for Unix users, you may consider another package *rPython* that is only available for Unix at this moment. This is the official website with plenty of documentations: [12]
   Here is another short article on how to use rPython: [13].

IV. Review of R Resources

Reticulate: as we will see in the following of this report, the reticulate package provides convenient support for incorporating Python in R. It's officially supported by RStudio. The package is relatively young and we will still find some issues during the usage.

The package is installable via CRAN [14].

(Fun facts from Wiki: Reticulated python is a species of python found in Southeast Asia… [15])

Fig 3. Logo of *reticulate* [10].

rPython *(only available for unix)*: *rPython* is another package for the same aim of using Python in R environment. The stable version only supports Unix systems. It's available on CRAN [16] for Unix users.

While I do find there's an experimental version of *rPython* for Windows, available on Github [17]. But I didn't manage to make it work normally on my computer.

Apart from the *reticulate*, you will also need the *knitr* package if you are going to use the R Markdown, which we will discuss later.

V. Incorporating Python in R using *reticulate*

In this chapter, I will introduce the major approaches of using Python in R together with some key aspects to notice, by using the *reticulate* package. According to the apparent way of coding and interacting with Python, I personally categorize them into 3 approaches: 1) using Python code chunk in R Markdown, 2) reuse stand-alone Python script, and 3) embedding Python in R script. (One may also categorize them differently according to the different mechanisms on the background.)

Here I will make brief introduction on the usage and some key aspects. For exemplary code and more technical details please see the attached tutorial.

1. Using Python Code Chunk in R Markdown

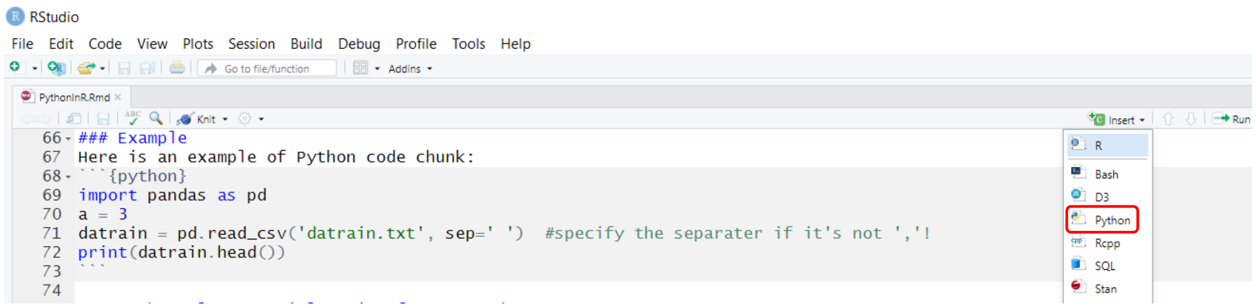
Personally, I think using Python Code chunk is the most convenient and straightforward approach of incorporating Python in R.

Resource required:

- *knitr* package v.1.18 or higher: R markdown is powered by the *knitr* package, which supports multiple languages including Python, Julia, C++, SQL, etc. After *knitr* v.1.18, the default Python engine is set to be from *reticulate*.
- *reticulate* package: Empower the incorporation of Python in R. It's also officially supported by the RStudio.
- RStudio, better be > v.1.2(preview): Since the *reticulate* is still a young project backed by RStudio, many things are under improvements. Some utilities, like passing data between environments which we will see later, only work normally with the newest preview version 1.2.
- Python "correctly" installed: in short, the *reticulate* need to be able to find the Python correctly registered in the PATH variable of your system. (This could be an issue for many Anaconda users like myself. Please see the tutorial for detailed solution.)

Usage and aspects to notice:

Python code chunks work exactly like R code chunks. Inside the Python code chunk (Fig 4), you can just write normal Python code directly: you can define variables, use Python packages, functions, etc.

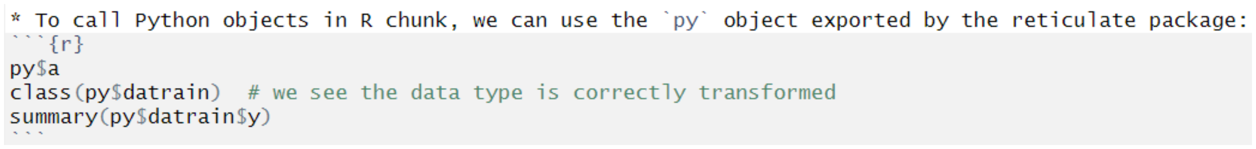


~~~
### Example
Here is an example of Python code chunk:
```{python}
import pandas as pd
a = 3
datrain = pd.read_csv('datrain.txt', sep=' ')  #specify the separater if it's not ','!
print(datrain.head())
```
~~~

Fig 4. Python code chunk in R Markdown within RStudio.

All the Python chunks work in the same Python session at background. When you execute one Python chunk, the objects (of public type) created will persist in the Python session environment. Thus, they will be directly accessible from other Python chunks.

The key point is: you can also access the Python objects within R chunks! This is the essence of incorporating two languages in the same platform, that they share freely the objects or data created. To access the Python objects in R, you will need to call the *py* object which "contains" all the Python objects. For example, write \`*py$a*\` in R chunk to access Python object *a* (Fig 5).



~~~
* To call Python objects in R chunk, we can use the `py` object exported by the reticulate package:
```{r}
py$a
class(py$datrain)  # we see the data type is correctly transformed
summary(py$datrain$y)
```
~~~

Fig 5. Call Python objects within R code chunk in R markdown

Vice-versa, you can pass R objects into Python chunk, via the *r* object. E.g. using `*r.b*` in Python chunk to access R object *b*.

Obviously, there is the problem of type compatibility when passing objects between different languages, since the objects types are defined differently. Luckily, Python and R have rather similar definitions concerning the data types. And the *reticulate* package offers seamless automatic transformation of data type between the two. We just need to call the object in the way we have just seen, without any extra step, and the type of the object will be automatically transformed to corresponding type in the target environment. E.g. *list* in Python corresponds to *vector* in R, *dictionary* in Python corresponds to *named list* in R, etc.

More details concerning Python plotting, type conversion etc. can be found in the tutorial.

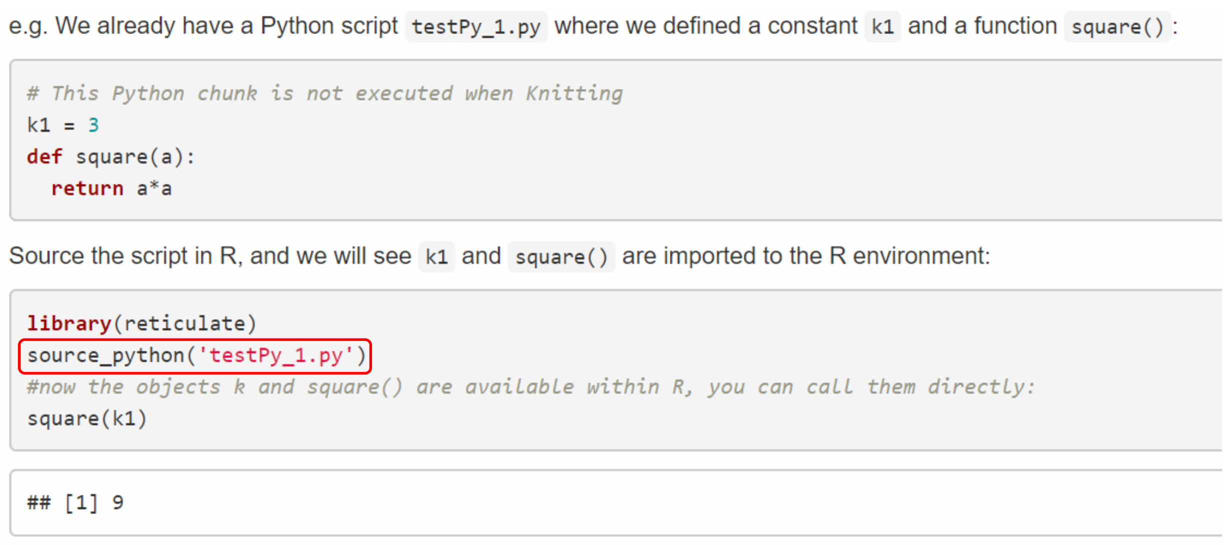
2. Reuse Stand-alone Python Script

Sometimes you may want to reuse your existing Python code; or prefer to edit Python in a stand-alone script with a full-featured IDE, instead of using the Python chunk in R Markdown. This is also feasible with the *reticulate* package!

Resource required: same as previous, except we may not need the *knitr* or newest RStudio version if we don't use R Markdown.

Usage and aspects to notice: technically there are 2 ways:

1. **Sourcing Python script**: to execute a Python script and import all the Python objects (of public type) created in the Python session directly into the R environment. By "importing" I mean those objects will be transformed into R type and existing in the R environment. So, you can call them directly in R without the *py* object. This is accomplished by applying the `*source_python()*` function to the target Python script file (Fig 6).



e.g. We already have a Python script `testPy_1.py` where we defined a constant `k1` and a function `square()`:

```
# This Python chunk is not executed when Knitting
k1 = 3
def square(a):
  return a*a
```

Source the script in R, and we will see `k1` and `square()` are imported to the R environment:

```
library(reticulate)
source_python('testPy_1.py')
#now the objects k and square() are available within R, you can call them directly:
square(k1)
```

```
## [1] 9
```

Fig 6. Sourcing a Python script in R

2. **Executing Python script**: to execute a Python script. Similar to previous one, but by this way it doesn't import those objects created directly into R. You need to use the *py* object to access them as usual. This is accomplished by applying the `*py_run_file()*` function to the target Python script file.

3. Embedding Python in R Script

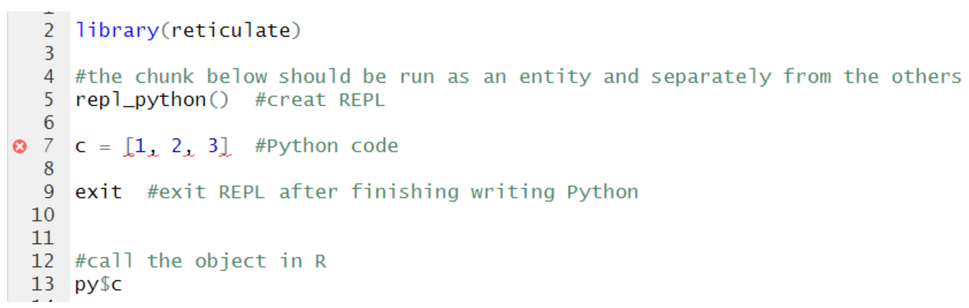
If you don't like Python chunk in R markdown but still want to occasionally use Python code in R, there are ways to run Python code within R script. Also, you can get access to Python packages in R, with simple command.

Resource required: same as previous.

Usage and aspects to notice:

2 ways of executing Python within R, and also a way of importing Python packages.

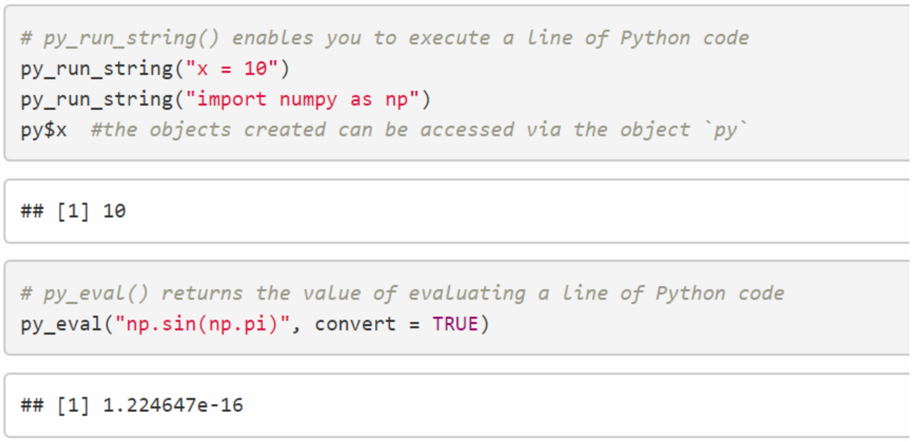
1. **Creating an interactive shell (REPL)**: A read–eval–print loop (REPL) [18] for Python in the R session is a shell where you can use Python interactively. It can be created with the *repl_python()* function (Fig 7). The Python code within the REPL could be executed, and the objects created could be accessed via the *py* object. (Be aware that for unknown reason this doesn't work normally in R Markdown, while it works in ordinary R script.)



```
library(reticulate)

#the chunk below should be run as an entity and separately from the others
repl_python()  #creat REPL

c = [1, 2, 3]  #Python code

exit  #exit REPL after finishing writing Python


#call the object in R
py$c
```

Fig 7. Using REPL in R script

2. **Run snippets of Python code:**
   You can also run snippets of code using `*py_run_string()*` or `*py_eval()*` function. The former could execute a line of Python code and create objects, the later could evaluate a line of Python code and return the value (fig 8). Personally I don't find this approach convenient nor intuitive.



```
# py_run_string() enables you to execute a line of Python code
py_run_string("x = 10")
py_run_string("import numpy as np")
py$x  #the objects created can be accessed via the object `py`
```

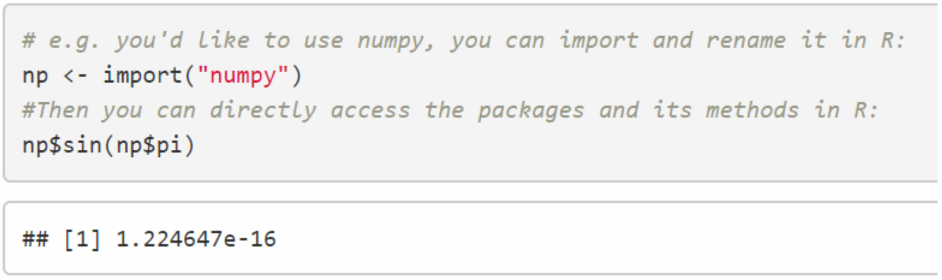
```
## [1] 10
```

```
# py_eval() returns the value of evaluating a line of Python code
py_eval("np.sin(np.pi)", convert = TRUE)
```

```
## [1] 1.224647e-16
```

Fig 8. Run snippets of Python code

3. **Importing Python packages**:
   If you are not a fan of Python code chunk in R Markdown or coding python script, but you want to use certain nice Python packages, then here is a simple way that you can directly import the packages into R: simply apply the `*import()*` function (Fig 9). After a package is imported, you can directly access it, and call its functions.



```
# e.g. you'd like to use numpy, you can import and rename it in R:
np <- import("numpy")
#Then you can directly access the packages and its methods in R:
np$sin(np$pi)
```

```
## [1] 1.224647e-16
```

Fig 9. Importing Python packages into R

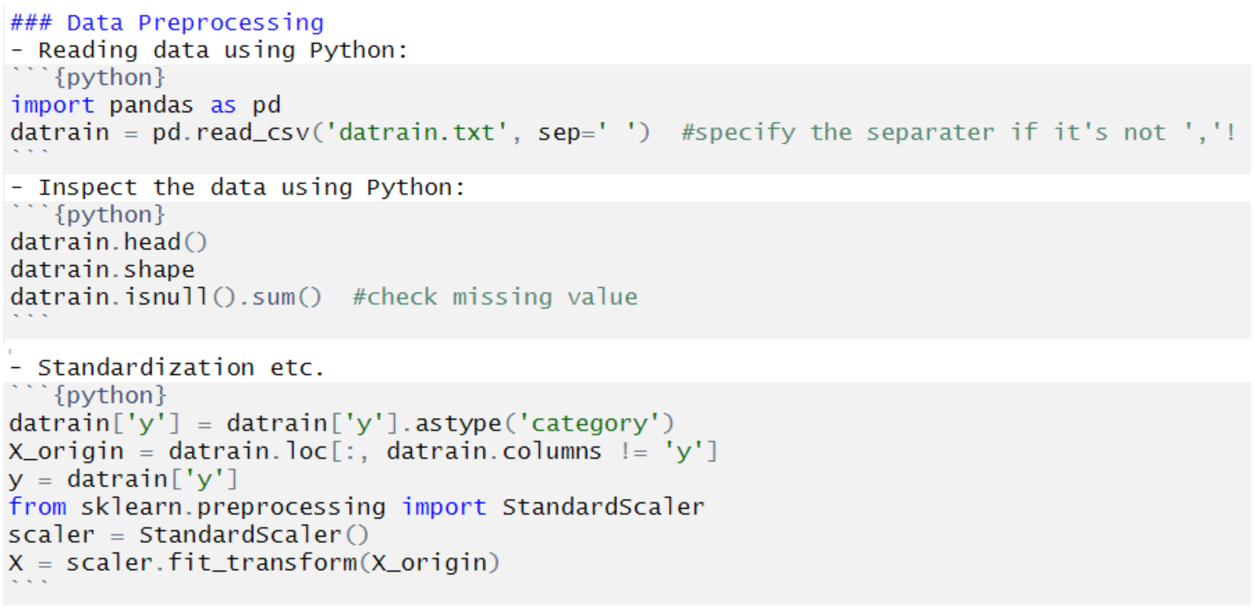
VI. Demonstrative Data Analysis Case

In this chapter we will see a data analysis example during which we demonstrate parts of the procedure of incorporating Python in R, and do a little comparison in the meanwhile. All the codes are written within the R Markdown. We will see that we can use Python and R interchangeably within the R Studio platform. Here we use the Wine Quality data (given in the attachment) that we have studied for the course homework.

(This part is also included in the tutorial, you can check that for the output results).

1. **Data Preprocessing**:

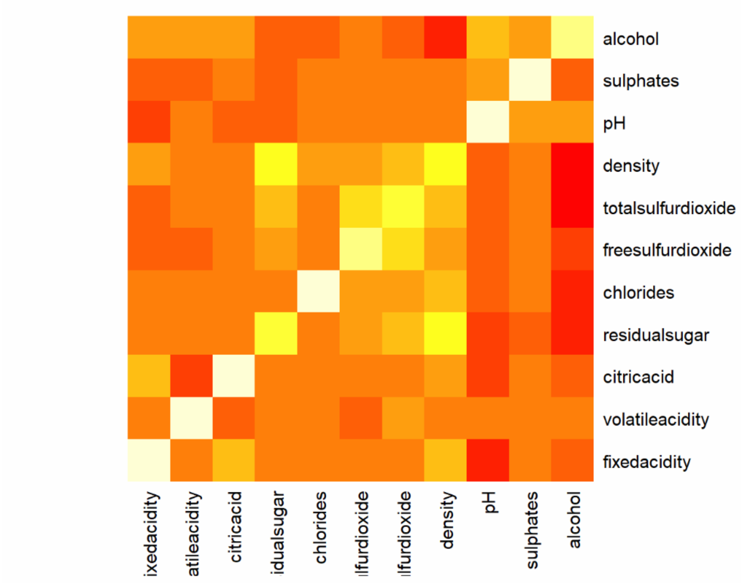
For the data loading and inspection operations, it would be quite similar (even in syntax level) for both Python and R. Here for example we use Python (Fig 10).



~~~
### Data Preprocessing
- Reading data using Python:
```{python}
import pandas as pd
datrain = pd.read_csv('datrain.txt', sep=' ')  #specify the separater if it's not ','!
~~~
- Inspect the data using Python:
```{python}
datrain.head()
datrain.shape
datrain.isnull().sum()  #check missing value
```

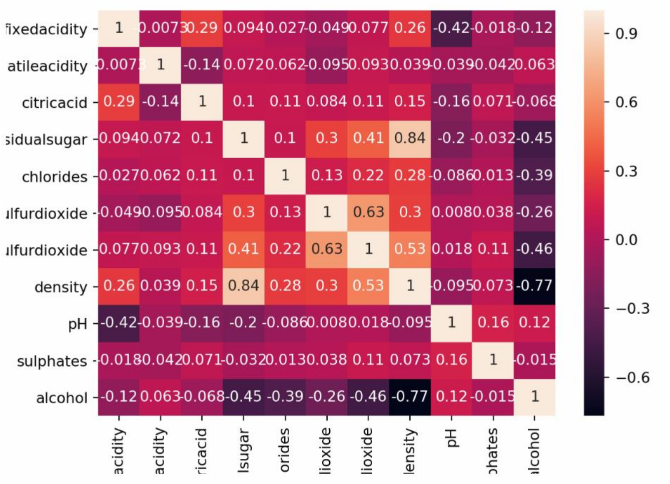
- Standardization etc.
```{python}
datrain['y'] = datrain['y'].astype('category')
X_origin = datrain.loc[:, datrain.columns != 'y']
y = datrain['y']
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X_origin)
```
```

Fig 10. Data Preprocessing within Python code chunk

2. **Graph:**

R is designed to be good at graphical representation and visualization of data. You could write brief code to obtain fancy graphs. Meanwhile, Python is also catching up in graphics, there are nice packages for visualization, like *seaborn*.

For example, let’s plot the correlation matrix in form of heatmap:
```

```
heatmap(cor(py$X_origin), Colv = NA, Rowv= NA)
```



(a)

```
import seaborn as sns
sns.heatmap(datrain.corr(), annot=True)
```



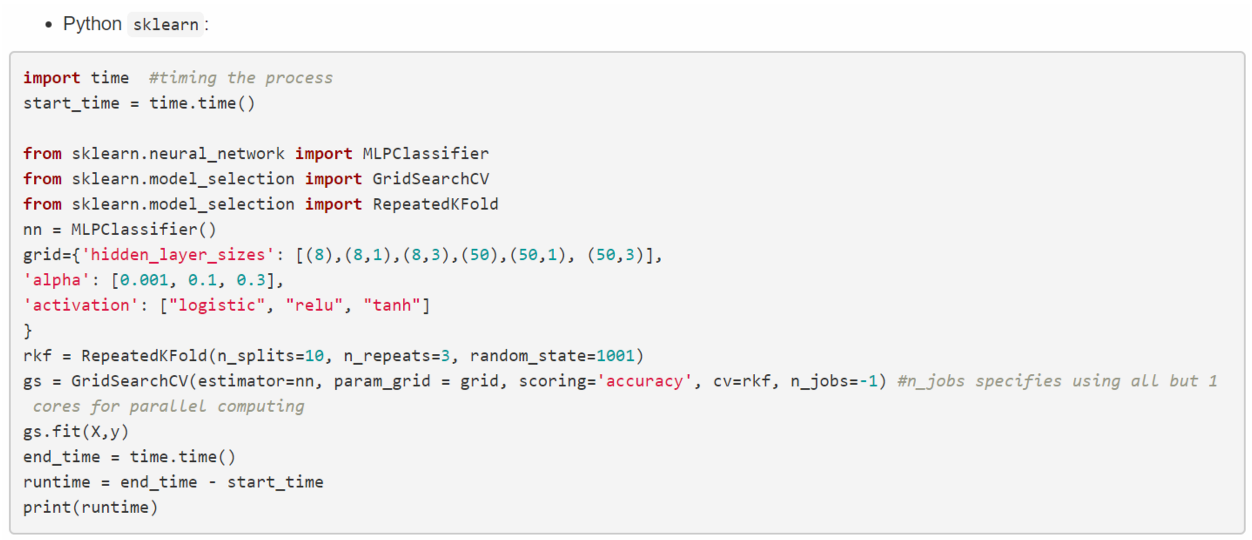
(b)

Fig 11. Plot heatmap with (a) R and (b) Python.

3. Model tuning and fitting:

Both Python and R have nice packages for ML. Python is probably more dominant in this field, and it may have more powerful packages for ML tasks. So, with the tools we have seen, we can use those Python packages within R if needed.

As an example, here we take a mainstream package related to ML model tuning from each: Python *sklearn* vs. R *caret*.



- Python `sklearn`:

```python
import time  #timing the process
start_time = time.time()

from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold
nn = MLPClassifier()
grid={'hidden_layer_sizes': [(8),(8,1),(8,3),(50),(50,1), (50,3)],
'alpha': [0.001, 0.1, 0.3],
'activation': ["logistic", "relu", "tanh"]
}
rkf = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1001)
gs = GridSearchCV(estimator=nn, param_grid = grid, scoring='accuracy', cv=rkf, n_jobs=-1) #n_jobs specifies using all but 1
 cores for parallel computing
gs.fit(X,y)
end_time = time.time()
runtime = end_time - start_time
print(runtime)
```

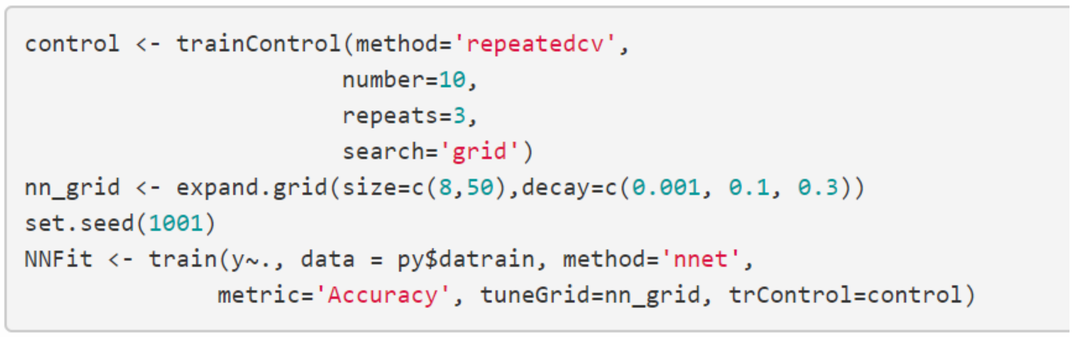
Fig 12. Using Python *sklearn* within Python code chunk to tune neural network model

With sklearn, we can tune various hyperparameters of the ANN model (Activation function, alpha, hidden layer sizes, etc.). The above takes ~262 sec (run in Spyder IDE with 5 cores).

However, the *reticulate* seem to have problem with the parallel processing applied in Python, using whichever approach of the above 3. (Here I used parallel computing for the GridSearchCV() function in the above example).

So, I run another simpler grid-search without parallel computing setting, the code take ~157 sec in the R markdown (if use 5 cores, take ~28 sec in Spyder).

Then, let's run the same simple grid-search using *caret* (in fact I find the tunable hyperparameters are very limited for many ML models in R, using *caret*. For example, here we tune the neural network model from *nnet* package, only network size and alpha are tunable):



```r
control <- trainControl(method='repeatedcv',
                        number=10,
                        repeats=3,
                        search='grid')
nn_grid <- expand.grid(size=c(8,50),decay=c(0.001, 0.1, 0.3))
set.seed(1001)
NNFit <- train(y~., data = py$datrain, method='nnet',
               metric='Accuracy', tuneGrid=nn_grid, trControl=control)
```

Fig 12. Using R *caret* to tune neural network model

In R this grid-search take ~51 sec, still using 5 cores for parallel computing. We see that it's slower than normally in Python, which took 28 sec.

Reference:

[1] https://www.r-project.org/about.html.

[2] https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages.

[3] https://dzone.com/articles/r-or-python-data-scientists-delight.

[4] Ihaka, R., & Gentleman, R. (1996). *R: A Language for Data Analysis and Graphics*. Journal of Computational and Graphical Statistics, 5(3), 299-314. doi:10.2307/1390807.

[5] https://www.stat.auckland.ac.nz/~ihaka/downloads/Massey.pdf.

[6] https://www.business-science.io/business/2018/10/08/python-and-r.html.

[7] https://www.reddit.com/r/statistics/comments/8de54s/is_r_better_than_python_at_anything_i_started/.

[8] https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis.

[9] https://blog.rstudio.com/2018/03/26/reticulate-r-interface-to-python/

[10] https://rstudio.github.io/reticulate/.

[11] https://www.r-bloggers.com/run-python-from-r/.

[12] https://rpython.readthedocs.io/en/latest/.

[13] http://www.programmingr.com/content/calling-python-r-rpython/

[14] https://CRAN.R-project.org/package=reticulate.

[15] https://en.wikipedia.org/wiki/Reticulated_python.

[16] https://cran.r-project.org/web/packages/rPython/index.html.

[17] https://github.com/cjgb/rPython-win.

[18] https://en.wikipedia.org/w/index.php?title=Read%E2%80%93eval%E2%80%93print_loop&oldid=885750456.

Appendix

1. Tutorial: *PythonInR.html* and *PythonInR.rmd*.
2. Code scripts used in the tutorial for tests purpose: *testPy_1.py*, *testPy_1.py*, *testREPL.R*.
3. Data file for the analysis example: *datrain.csv*.