Présentateur:
Mohammadsadegh Shokrizadeh

Analyste de Données
Larochelle

R – Python Interface

PLAN DE PRESENTATION:

- R reticulate
- Python rpy2
- Jupyter rmagic

Python Dans R

R reticulate:

- Un package pour utiliser les scripts de Python dans l'environnement de R
- Facilite l'utilisation des modules qui ne sont pas présents dans R
- Nous donne la puissance de Python dans R
- Nous permets de bénéficier d'avantages de programmation OOP en R
- La transformation des types de données de Python est maintenant plus facile avec R reticulate.
- Nous permets d'importer et exécuter les codes qu'on a déjà écrit en Python

Quelle Version de Python a Utiliser?

- Par défault, reticulate utilise la version de Python qui se trouve dans la variable d'environnement PATH de votre ordinateur.
- Avec la function "use_python()" on peut dire à R quelle version de Python on veut utiliser.

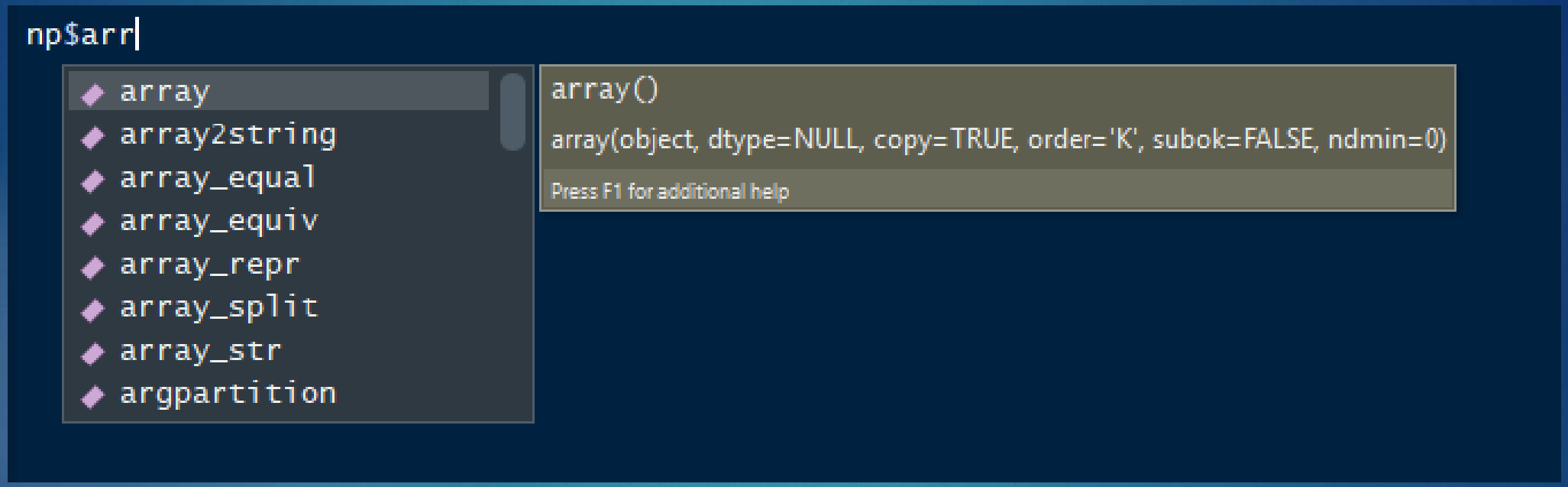
- Si on désire utiliser l'environement de conda, la fonction "use_conda()" nous aide à le faire.
- Même chose pour la version d'environnement virtuel de Python: "use_virtualenv()"

Préparation de l'environnement de travail:

```
library(reticulate)

py_available()

py_config()

#To See Which Version of Python Will Be Used Without Actually Loading Python
py_discover_config()


conda_list(conda = "auto")
conda_version()

conda_create(envname = "myenv", conda = "auto")

use_condaenv(condaenv = "myenv")

conda_install(envname = "myenv", packages = "pandas")
conda_remove(envname = "myenv", packages = "pandas", conda = "auto")

# To Find The Location of MAin Conda Binary
conda_binary()


py_numpy_available()

py_module_available(module = "pandas")

py_install(packages = "pandas", envname = "myenv")

py_install(packages = "pands", envname = "myenv", method = "conda", conda = "auto")
```

Importation des modules de Python:

```
library(reticulate)

sklearn <- import("sklearn")

np <- import("numpy")


os <- import("os")

os$listdir("C:/")
```


np$arr
array
array2string
array_equal
array_equiv
array_repr
array_split
array_str
argpartition
array()
array(object, dtype=NULL, copy=TRUE, order='K', subok=FALSE, ndmin=0)
Press F1 for additional help

Exécuter des Codes

- Deux commandes importantes:

- py_run_string() que nous donne la capacité d'exécuter les commandes Python.

- py_run_file() que nous donne la capacité d'exécuter les fichiers Python.

- On peut aussi définir des variable en Python avec la fonction de py_run_string()
- En R, rappeler la variable defini en Python se fait en utilisant py$<non de variable>.

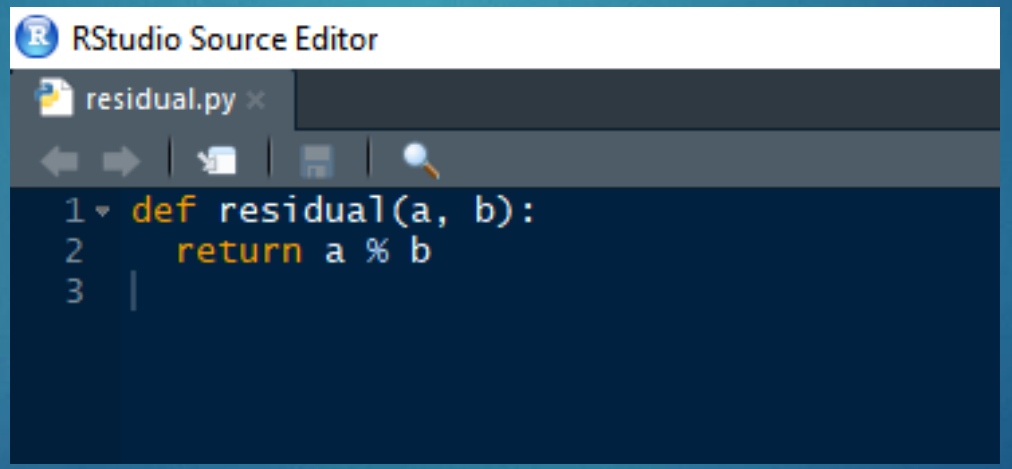
```r
x = 1500
print(paste("x in R is: ", x))


py_run_string("x = 15")
print(paste("x in Python is: ", py$x))
```

```
[1] "x in R is:  1500"
[1] "x in Python is:  15"
```

Source_python()

- Si on a un fichier de script de Python contenant des classes, des méthodes et des fonctions, en utilisant cette function on peut les utiliser.

- Exemple1: Notre fichier Python contient une function très simple qui calcule le résiduel d'une division.



RStudio Source Editor

residual.py

```
1 def residual(a, b):
2     return a % b
3
```

```
> source_python("residual.py")
>
>
> residual(2, 3)
[1] 2
>
>
> residual(15, 4)
[1] 3
```

Exemple 2:

- Dans notre fichier Python, “Employee Num Class.py”, on la class Employee qui contient des méthodes, class methodes et static methode.

```python
class Employee:

    num_of_emps = 0
    raise_amt = 1.04

    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.email = first + '.' + last + '@email.com'
        self.pay = pay

        Employee.num_of_emps += 1

    def fullname(self):
        return '{} {}'.format(self.first, self.last)

    def apply_raise(self):
        self.pay = int(self.pay * self.raise_amt)

    @classmethod
    def set_raise_amt(cls, amount):
        cls.raise_amt = amount

    @classmethod
    def from_string(cls, emp_str, splt):
        first, last, pay = emp_str.split(splt)
        return cls(first, last, pay)

    @staticmethod
    def is_workday(day):
        if day.weekday() == 5 or day.weekday() == 6:
            return False
        return True
```

- Comme on voit dans cette exemple, quand on importe le fichier Python dans l'espace de R, on peut appeler la classe Employee.
- Pour accéder aux méthodes et fonctions dans R, on doit utiliser $ après le nom de l'instance de la classe.

```r
source_python("Employee Num Class.py")

emp1 <- Employee("John", "Smith", 65000)

emp1$first

emp1$last

emp1$fullname()

emp1$email

emp1$pay

emp1$set_raise_amt(1.32)

emp1$raise_amt
```

```
[1] "John"
[1] "Smith"
[1] "John Smith"
[1] "John.Smith@email.com"
[1] 65000
[1] 1.32
```

Convertir Les Objets R/Python A Python/R

r_to_py(x, convert = TRUE/FALSE)

- Avec cette fonction, l'objet R va devenir comme un objet Python.
- L'option "convert" nous permets de décider si on veut convertir le type de l'objet automatiquement ou manuellement.

```{r}
x <- matrix(1:16, nrow=4)
x
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

```{r}
x_python <- r_to_py(x, convert = TRUE)
x_python
class(x_python)
```

```
[[ 1  5  9 13]
 [ 2  6 10 14]
 [ 3  7 11 15]
 [ 4  8 12 16]]
[1] "numpy.ndarray"          "python.builtin.object"
```

Py_to_r(x)

- Avec cette function, l'objet Python va devenir comme un objet R.

```{r}
x <- matrix(1:16, nrow=4)
x
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

```{r}
x_python <- r_to_py(x, convert = TRUE)
x_python
class(x_python)
```

```
[[ 1  5  9 13]
 [ 2  6 10 14]
 [ 3  7 11 15]
 [ 4  8 12 16]]
[1] "numpy.ndarray"          "python.builtin.object"
```

```{r}
x_r <- py_to_r(x_python)
x_r
class(x_r)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
[1] "matrix"
```

py_func()

- Pour convertir les fonctions créées en R, on utilise cette function.
- Elle va emballer, "wrape", la function créée en R afin qu'on ait une function Python avec la même signature.

```{r}
inspect <- import("inspect")

res <- function(a, b) a %/% b
res(7, 3)
```

```
[1] 2
```

```{r}
python_res <- py_func(res)
class(python_res)
python_res(7, 3)

inspect$getargspec(python_res)

```

```
[1] "python.builtin.function" "python.builtin.object"
[1] 2
ArgSpec(args=['a', 'b'], varargs=None, keywords=None, defaults=None)
```

Input/Output Les Objets:

- Avec les fonctions suivantes, on peut stocker et récupérer l'objet stocké.

py_save_object(objet, filename, pickle="pickle")

py_load_object(filename)

```{r}
py_save_object(x, filename = "x.pkl", pickle = "pickle")
```

```{r}
y <- py_load_object(filename = "x.pkl")
y
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
```

```
> x <- matrix(1:16, nrow = 4)
>
>
> repl_python()
Python 3.7.3 (C:\Users\MSHOKR~1\AppData\Local\CONTIN~1\ANACON~1\python.exe)
Reticulate 1.12 REPL -- A Python interpreter in R.
>>> r.x
array([[ 1,  5,  9, 13],
       [ 2,  6, 10, 14],
       [ 3,  7, 11, 15],
       [ 4,  8, 12, 16]])
>>> r.x = r.x * 100
>>> r.x
array([[ 100.,  500.,  900., 1300.],
       [ 200.,  600., 1000., 1400.],
       [ 300.,  700., 1100., 1500.],
       [ 400.,  800., 1200., 1600.]])
>>> exit
> x
     [,1] [,2] [,3] [,4]
[1,]  100  500  900 1300
[2,]  200  600 1000 1400
[3,]  300  700 1100 1500
[4,]  400  800 1200 1600
>
> repl_python()
Python 3.7.3 (C:\Users\MSHOKR~1\AppData\Local\CONTIN~1\ANACON~1\python.exe)
Reticulate 1.12 REPL -- A Python interpreter in R.
>>> dictionary = {'alpha': 1, 'beta': 2}
>>> dictionary
{'alpha': 1, 'beta': 2}
>>> exit
> py$dictionary
$alpha
[1] 1

$beta
[1] 2
```

REPL : Read-Eval-Print-Loop

repl_python()

Cette function nous permets d'avoir un environnement Python dans las session de R.

Pour avoir accès aux objets R dans l'environnement Python, on doit les appeler à l'aide de r.<nom d'objet>

Pour accéder aux objets créés dans l'espace de Python dans la session de R: py$<nom d'objet>

Exemple Pratique:

- Dans cet exemple, à l'aide de package sklearn de Python, on va créer un modèle de régression pour les données de dataset de diabetes de sklearn.

1. Premièrement, importer les packages nécessaires.
2. Préparer le dataset pour utiliser dans les fonctions.
3. Créer le modèle, ajuster le modèle et faire la prédiction.

```
library(reticulate)

sklearn <- import("sklearn")

np <- import("numpy")

datasets <- import("sklearn.datasets")
diabetes <- datasets$load_diabetes

linear_model <- sklearn$linear_model

metrics <- import("sklearn.metrics")
mean_squared_error <- metrics$mean_squared_error
r2_score <- metrics$r2_score




diabetes_X <- as.matrix(diabetes('data')[[1]])
dim(diabetes_X)

diabetes_X_train <- as.matrix(diabetes_X[1:422,])
diabetes_X_test <- as.matrix(diabetes_X[423:nrow(diabetes_X),])

diabetes_y_train = diabetes('data')[[2]][1:422]
diabetes_y_test = diabetes('data')[[2]][423:nrow(diabetes_X)]



regr <- linear_model$LinearRegression()

fit <- regr$fit(np$array(diabetes_X_train), np$array(diabetes_y_train))

print(regr$coef_)


prediction <- fit$predict(diabetes_X_test)

mean_squared_error(diabetes_y_test, prediction)
r2_score(diabetes_y_test, prediction)
```

Importer les packages

```
library(reticulate)

sklearn <- import("sklearn")

np <- import("numpy")

datasets <- import("sklearn.datasets")
diabetes <- datasets$load_diabetes

linear_model <- sklearn$linear_model

metrics <- import("sklearn.metrics")
mean_squared_error <- metrics$mean_squared_error
r2_score <- metrics$r2_score
```

Préparer Dataset

```
diabetes_X <- as.matrix(diabetes('data')[[1]])
dim(diabetes_X)

diabetes_X_train <- as.matrix(diabetes_X[1:422,])
diabetes_X_test <- as.matrix(diabetes_X[423:nrow(diabetes_X),])

diabetes_y_train = diabetes('data')[[2]][1:422]
diabetes_y_test = diabetes('data')[[2]][423:nrow(diabetes_X)]
```

Créer le modèle:

```
regr <- linear_model$LinearRegression()

fit <- regr$fit(np$array(diabetes_X_train), np$array(diabetes_y_train))

print(regr$coef_)
```

```
> print(regr$coef_)
 [1]    0.3034995 -237.6393153  510.5306054  327.7369804 -814.1317094  492.8145880  102.8484522  184.6064891  743.5196168   76.0951722
```

Faire La Prédiction:

```
prediction <- fit$predict(diabetes_X_test)

mean_squared_error(diabetes_y_test, prediction)
r2_score(diabetes_y_test, prediction)
```

```
> mean_squared_error(diabetes_y_test, prediction)
[1] 2004.568
> r2_score(diabetes_y_test, prediction)
[1] 0.5850753
```

R Dans Python: rpy2

rpy2 Module de Python

- Un package qui a été designé pour faciliter l'usage de R dans Python.
- Il crée un R intégré (embedded R, en anglais)
- Les objets de R sont exposés en tant qu'instances des classes implementées de Python. (Python-Implemented Classes)
- Le but de ce package est de minimiser la connaissance nécessaire de R pour les workarounds et profiter de l'environement de Python en même temps qu'on utilise R.

- rpy2 est composé des sous packages suivants:

1. rpy2.rinterface: interface low-level de package -> quand on a besoin de vitesse et flexibilité.
2. Rpy2.robjects: interface high-level de package -> quand on désire "ease-of-use"
3. rpy2.interactive: interface high-level, base sur rpy2.robjects -> interactive work
4. rpy2.rpy_classic: interface high-level
5. Rrpy2.rlike: pour imiter les objets R en Python

- Importer des librairies de R:

```
from rpy2.robjects.packages import importr

base = importr('base')

utils = importr('utils')
```

- Installer des packages:

```
import rpy2.robjects.packages as rpackages

utils = rpackages.importr('utils')

utils.chooseCRANmirror(ind=1)

packnames = ('ggplot2')

from rpy2.robjects.vectors import StrVector

names_to_install = [x for x in packnames if not rpackages.isinstalled(x)]

if len(names_to_install) > 0:
    utils.install_packages(StrVector(names_to_install))
```

- Créer une fonction de R:

```
import rpy2.robjects as robjects
robjects.r('''

        f <- function(r, verbose=FALSE) {
            if (verbose) {
                cat("I am calling f().\n")
            }
            2 * pi * r
        }

        ''')
```

```
R object with classes: ('function',) mapped to:
<SignatureTranslatedFunction - Python:0x000001D08D0EAAC8 / R:0x000001D08D53E2D8>
```

```
r_f = robjects.r['f']
```

```
res = r_f(3)
res
```

```
FloatVector with 1 elements.

18.849556
```

- Vecteur de R:

```
import rpy2.robjects as robjects
```

```
res = robjects.StrVector(['abc', 'def'])
print(res)
```

```
[1] "abc" "def"
```

```
res = robjects.IntVector([1,2,3])
res.r_repr()
print(res)
```

```
[1] 1 2 3
```

```
res = robjects.FloatVector([1.1,2.2,3.3])
print(res)
```

```
[1] 1.1 2.2 3.3
```

```
v = robjects.FloatVector([1.1,2.2,3.,4.4,5.5,6.6])
m = robjects.r['matrix'](v, nrow=2)
print(m)
```

```
     [,1] [,2] [,3]

[1,]  1.1  3.0  5.5

[2,]  2.2  4.4  6.6
```

- Utiliser les fonctions de R:

```
rsum = robjects.r['sum']

rsum(robjects.IntVector([1,2,3]))[0]
```

```
6
```

```
rsort = robjects.r['sort']
rsort(robjects.IntVector([1,2,3]), decreasing=True)
```

```
IntVector with 3 elements.

3  2  1
```

Exemple:

```
import array

from rpy2.robjects import IntVector, Formula
from rpy2.robjects.packages import importr

stats = importr('stats')

x = IntVector(range(1, 11))
y = x.ro + stats.rnorm(10, sd=0.2)

fmla = Formula('y ~ x')
env = fmla.environment
env['x'] = x
env['y'] = y

fit = stats.lm(fmla)
fit
```

ListVector with 12 elements.

coefficients FloatVector with 2 elements.

0.222956 0.963990

residuals FloatVector with 10 elements.

-0.079480 0.088825 -0.078115 -0.017890 ... 0.027435 0.174804 0.067588 -0.190627

effects FloatVector with 10 elements.

-17.471265 8.755870 -0.072333 -0.002725 ... 0.070748 0.227501 0.129668 -0.119165

... ...

call Vector with 2 elements.

SignatureTranslatedFunc... Formula

terms y ~ x attr(,"variables") list(y, x) attr(,"factors") x y 0 x 1 attr(,"term.labels") [1] "x" attr(,"order") [1] 1 attr(,"intercept") [1] 1 attr(,"response") [1] 1 attr(,".Environment") attr(,"predvars") list(y, x) attr(,"dataClasses") y x "numeric" "numeric"

model R/rpy2 DataFrame (10 x 2)

y	x
1.107466	1
2.239760	2
3.036810	3
4.061025	4
...	...
6.998318	7
8.109677	8
8.966451	9
9.672225	10

Jupyter rmagic

Jupyter Notebook Magic Commands

- Jupyter Notebook est une application open-source web pour créer et partager les documents contenant les codes, visualization, etc..
- Jupyter est normalement une composante de Anaconda.
- Jupyer peut prendre en charge plusieurs langages de programmation.
- Avec Jupyter Magics on peut faire plusieurs tâches compliquées très facilement.
- Jupyter Magics nous donne la capacité d'exécuter différents kernels dans un même notebook.
- Et encore plus …!

- Liste des magics:

```
In [1]: %lsmagic

Out[1]: Available line magics:
%alias  %alias_magic  %autoawait  %autocall  %automagic  %autosave  %bookmark  %cd  %clear  %cls  %colors  %conda  %config  %co
nnect_info  %copy  %ddir  %debug  %dhist  %dirs  %doctest_mode  %echo  %ed  %edit  %env  %gui  %hist  %history  %killbgscripts
%ldir  %less  %load  %load_ext  %loadpy  %logoff  %logon  %logstart  %logstate  %logstop  %ls  %lsmagic  %macro  %magic  %matpl
otlib  %mkdir  %more  %notebook  %page  %pastebin  %pdb  %pdef  %pdoc  %pfile  %pinfo  %pinfo2  %pip  %popd  %pprint  %precisio
n  %prun  %psearch  %psource  %pushd  %pwd  %pycat  %pylab  %qtconsole  %quickref  %recall  %rehashx  %reload_ext  %ren  %rep
%rerun  %reset  %reset_selective  %rmdir  %run  %save  %sc  %set_env  %store  %sx  %system  %tb  %time  %timeit  %unalias  %unl
oad_ext  %who  %who_ls  %whos  %xdel  %xmode

Available cell magics:
%%!  %%HTML  %%SVG  %%bash  %%capture  %%cmd  %%debug  %%file  %%html  %%javascript  %%js  %%latex  %%markdown  %%perl  %%prun
%%pypy  %%python  %%python2  %%python3  %%ruby  %%script  %%sh  %%svg  %%sx  %%system  %%time  %%timeit  %%writefile

Automagic is ON, % prefix IS NOT needed for line magics.
```

Exemple: Voir le contenant d'un fichier Python dans Jupyter.

```
In [2]: %pycat Temperature.py
```

```
from tkinter import *
from tkinter import ttk


#================================================

import pandas as pd
import numpy as np
#import matplotlib.pyplot as plt
import seaborn as sns
#%matplotlib inline

from urllib.request import urlopen
from bs4 import BeautifulSoup


url = "https://meteo.gc.ca/canada_e.html"

html = urlopen(url)

soup = BeautifulSoup(html, 'lxml')

rows = soup.find_all('tbdoy')

tbl = list()
for tr in soup.find_all('tr'):
    t = list()
```

- La commande nécessaire dans le "Anaconda prompt" afin de préparer notre environnement Jupyter pour R:

  conda install –c r r-essentials

- Il faut que rpy2 package soit installé sur la machine et aussi pour Anaconda.

- Préparer Jupyter Notebook pour Rmagics:

```
import rpy2.ipython
%load_ext rpy2.ipython
```

- %R

%R [-I INPUT] [-o OUTPUT] [-w WIDTH] [-h HEIGHT] [-d DATAFRAME] …

- Avec %R on peut exécuter les codes de R et ensuite avoir les résultats dans Python namespace et aussi comme un objet de Python.

```
%R X=c(1,4,5,7)
```

```
array([1., 4., 5., 7.])
```

```
%R X=c(1,4,5,7); sd(X); mean(X)
```

```
array([4.25])
```

- Exemple1: Envoyer un objet de Python sans avoir de données manquantes à R et générer la moyenne.

```
import numpy as np

Z = np.array([1,4,5,10])
Z
```

```
array([ 1,  4,  5, 10])
```

```
%R -i Z mean(Z)
```

```
array([5.])
```

- Exemple 2: Envoyer un objet de Python avec données manquantes à R et générer la moyenne.

```
Z = np.array([1,4,5,10,np.nan])
Z
```

```
array([ 1.,  4.,  5., 10., nan])
```

```
%R -i Z mean(Z, na.rm=TRUE)
```

```
array([5.])
```

- Voir la valeur d'un objet R dans la format R: %Rget

```
%R X = c(1,4,5,7)
X
```

```
<rpy2.rinterface.FloatSexpVector - Python:0x000002D97F117FC0 / R:0x000002D977956B08>
```

```
%Rget X
```

FloatVector with 4 elements.

1.000000	4.000000	5.000000	7.000000

- Transférer objet de Python a R Avec %Rpush

```
Z = np.array([1,4,5,10,np.nan])
%Rpush Z
```

```
%R d = mean(Z, na.rm=TRUE)
```

```
array([5.])
```

```
%Rget d
```

FloatVector with 1 elements.

5.000000

- Exemple:

```
%R X = c(1,4,5,7)
%R Y = c(2,4,3,9)
%R summary(lm(Y~X))
```

ListVector with 11 elements.

call	Vector with 2 elements. RObject Vector
terms	Y ~ X attr(,"variables") list(Y, X) attr(,"factors") X Y 0 X 1 attr(,"term.labels") [1] "X" attr(,"order") [1] 1 attr(,"intercept") [1] 1 attr(,"response") [1] 1 attr(,".Environment") attr(,"predvars") list(Y, X) attr(,"dataClasses") Y X "numeric" "numeric"
residuals	FloatVector with 4 elements. 0.880000 -0.240000 -2.280000 1.640000
...	...
adj.r.squared	FloatVector with 1 elements. 0.548966
fstatistic	FloatVector with 3 elements. 4.651376 1.000000 2.000000
cov.unscaled	Matrix with 4 elements. 1.213333 -0.226667 -0.226667 0.053333

- Exemple:
- Générer le dataset:

```
age = np.random.uniform(0, 18, size=50)
height = 22 + 8.6 + np.random.normal(size=50, scale=10)

growing = pd.DataFrame({'age': age,
                        'height': height})

growing.head()
```

	age	height
0	4.603793	33.373657
1	5.847403	23.118891
2	12.703409	24.330741
3	3.867384	30.571136
4	11.773677	28.225791

- Exemple – continue …
- Générer la model:

```
%%R -i age,height -o coefficients

fit <- lm(height ~ age)

coefficients <- coef(fit)
```

```
print(coefficients)
```

```
(Intercept)         age

 31.6602712  -0.1009285
```

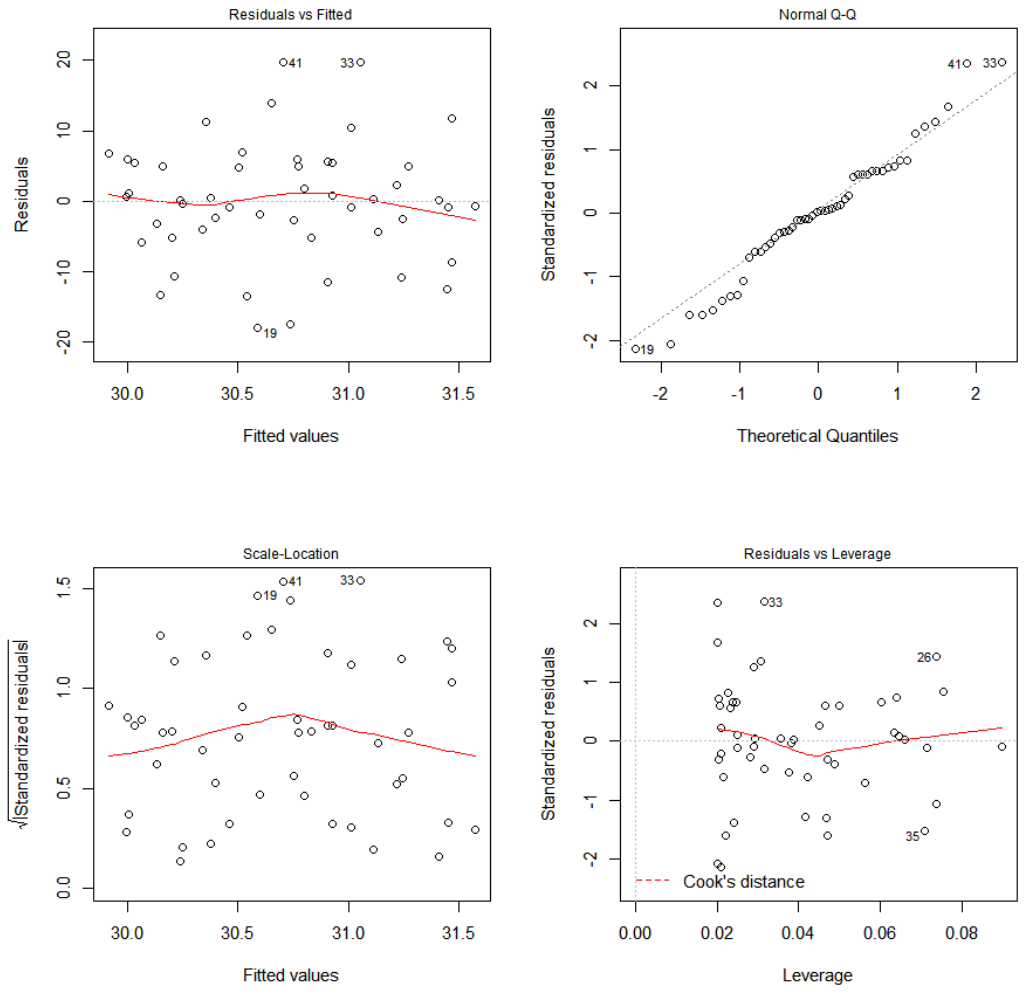
```
%R summary(fit)
```

ListVector with 11 elements.

call	Vector with 2 elements. RObject Vector
terms	height ~ age attr(,"variables") list(height, age) attr(,"factors") age height 0 age 1 attr(,"term.labels") [1] "age" attr(,"order") [1] 1 attr(,"intercept") [1] 1 attr(,"response") [1] 1 attr(,".Environment") attr(,"predvars") list(height, age) attr(,"dataClasses") height age "numeric" "numeric"
residuals	Array with 50 elements. -1.829459 5.521018 5.447837 -4.396086 ... -0.882839 13.965464 -3.206481 -10.648939
...	...
adj.r.squared	FloatVector with 1 elements. -0.017572
fstatistic	FloatVector with 3 elements. 0.153845 1.000000 48.000000
cov.unscaled	Matrix with 4 elements. 0.104327 -0.008826 -0.008826 0.000924

- Exemple – continue …
- Générer le graphique.

```
%%R -w 900 -h 450

par(mfrow = c(1,2), cex = 1.25)

plot(fit)
```



Remerciements:

Merci à Larochelle Groupe Conseil pour tout le support qui m'a été fourni pour cet événement.