R vs Python

Claude REN

10/22/2020

Introduction

In this .pdf you will find all the basics of R and its equivalence in Python. The purpose is to provide a document where you can easily find useful information when working on R and Python.

Basic operator

Calculate with R

```
# No import
10*(1+3-2.4)
```

```
[1] 16
```

```
10^2
```

```
[1] 100
```

```
10**2
```

```
[1] 100
```

```
sqrt(100)
```

```
[1] 10
```

```
pi
```

```
[1] 3.141593
```

```
cos(pi)
```

```
[1] -1
```

```
exp(1)
```

```
[1] 2.718282
```

```
log(1)
```

```
[1] 0
```

```
round(2.5435, 2)
```

```
[1] 2.54
```

```
a <- 100
print(a)
```

```
[1] 100
```

Calculate with Python

```
import numpy as np
10*(1+3-2.4)
```

```
16.0
```

```
10^2        # Not correct !
```

```
8
```

```
10**2
```

```
100
```

```
np.sqrt(100)
```

```
10.0
```

```
np.pi
```

```
3.141592653589793
```

```
np.cos(np.pi)
```

```
-1.0
```

```
np.exp(1)
```

```
2.718281828459045
```

```
np.log(1)
```

```
0.0
```

```
round(2.543534, 2)
```

```
2.54
```

```
a = 100
print(a)
```

```
100
```

Vector operations with R

```
v <- c(10, 20, 30)
v
```

```
[1] 10 20 30
```

```
length(v)
```

```
[1] 3
```

```
2*v+1
```

```
[1] 21 41 61
```

```
v**2
```

```
[1] 100 400 900
```

```
log(v)
```

```
[1] 2.302585 2.995732 3.401197
```

```
w <- c(1, 2, 3)
v-w
```

```
[1]  9 18 27
```

```
v*w
```

```
[1] 10 40 90
```

```
v/w
```

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

```
v%*%w
```

```
     [,1]
[1,]  140
```

```
sum(v)
```

```
[1] 60
```

```
mean(v)
```

```
[1] 20
```

```
min(v)
```

```
[1] 10
```

```
max(v)
```

```
[1] 30
```

```
sd(v)
```

```
[1] 10
```

```
median(v)
```

```
[1] 20
```

Vector operations with Python

```
v = np.array([10, 20, 30])
v
```

```
array([10, 20, 30])
```

```
len(v)
```

```
3
```

```
2*v+1
```

```
array([21, 41, 61])
```

```
v**2
```

```
array([100, 400, 900])
```

```
np.log(v).round(4)
```

```
array([2.3026, 2.9957, 3.4012])
```

```
w = np.array([1, 2, 3])
v-w
```

```
array([ 9, 18, 27])
```

```
v*w
```

```
array([10, 40, 90])
```

```
v/w
```

```
array([10., 10., 10.])
```

```
np.dot(v,w)
```

```
140
```

```
sum(v)
```

```
60
```

```
np.average(v)
```

```
20.0
```

```
min(v)
```

```
10
```

```
max(v)
```

```
30
```

```
np.std(v, ddof = 1)
```

```
10.0
```

```
np.median(v)
```

```
20.0
```

For the standard deviation, the formula used in numpy with std() is different from the one in R with sd().

$$\sigma_p = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2 - \overline{x}^2}$$

$$\sigma_R = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}x_i^2 - \overline{x}^2}$$

In order to find the same result in Python as in R, you have to precise ddof = 1 in Python.

Vector manipulation with R

```
u <- c(1, 2, 3, 4, 5)
u[2]
```

```
[1] 2
```

```
u[3:5]
```

```
[1] 3 4 5
```

```
u[5] <- 50
u[1:3] <- 1
u
```

```
[1]  1  1  1  4 50
```

```
v <- c(10,20,30,30,60,50)
w <- c(20,10,31,31,61,51)
u <- c(5 ,5 ,5 ,32,62,49)
```

Vector manipulation with Python

```
u = np.array([1, 2, 3, 4, 5])
u[1]
```

```
2
```

```
u[2:5]
```

```
array([3, 4, 5])
```

```
u[4] = 50
u[0:3] = 1
u
```

```
array([ 1,  1,  1,  4, 50])
```

```
v = np.array([10,20,30,30,60,50])
w = np.array([20,10,31,31,61,51])
u = np.array([5 ,5 ,5 ,32,62,49])
```

Equivalence str() R in Python : The function str() in R show you the structure of your variable, it can be very useful and you'll will find no equivalent function in Python. The only way to get the same information is to create a function :

```
str(u)
```

```
 num [1:6] 5 5 5 32 62 49
```

```
import pandas as pd
du = pd.DataFrame(u)
print(str(du.info()) + "\n"+ str(u))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       6 non-null      int64
dtypes: int64(1)
memory usage: 176.0 bytes
None
[ 5  5  5 32 62 49]
```

You can see above that the str() function used in the python chunks is not at all the same function as in R. In Python it is used to transform non string type to string character.

The one below is for data.frame, keep it for future usage :

```
def rstr(df):
    structural_info = pd.DataFrame(index=df.columns)
    structural_info['unique_len'] = df.apply(lambda x: len(x.unique())).values
    structural_info['unique_val'] = df.apply(lambda x: [x.unique()]).values
    print(df.shape)
    return structural_info
rstr(df)
```

Vector manipulation with R

```
options(width = 30)
sum(is.na(u))
```

```
[1] 0
```

```
u_ <- c(NA,u,NA,NA)
u_
```

```
[1] NA  5  5  5 32 62 49 NA NA
```

```
sum(is.na(u_))
```

```
[1] 3
```

```
range(u)
```

```
[1]  5 62
```

```
range(u_ , na.rm = TRUE)
```

```
[1]  5 62
```

```
quantile(u)
```

```
   0%   25%   50%   75%  100%
 5.00  5.00 18.50 44.75 62.00
```

```
summary(u)
```

```
   Min. 1st Qu.  Median
   5.00    5.00   18.50
   Mean 3rd Qu.    Max.
  26.33   44.75   62.00
```

```
sd(u_, na.rm = TRUE)
```

```
[1] 25.23225
```

```
cor(v,w)
```

```
[1] 0.9433573
```

```
sort(v)
```

```
[1] 10 20 30 30 50 60
```

```
sort(v, decreasing = TRUE)
```

```
[1] 60 50 30 30 20 10
```

```
order(w)
```

```
[1] 2 1 3 4 6 5
```

```
rank(w, ties.method="min")
```

```
[1] 2 1 3 3 6 5
```

```
rank(w, ties.method="max")
```

```
[1] 2 1 4 4 6 5
```

```
pmax(v,w,u)
```

```
[1] 20 20 31 32 62 51
```

```
pmin(v,w,u)
```

```
[1]  5  5  5 30 60 49
```

Vector manipulation with Python

```
np.set_printoptions(
    suppress=True,linewidth=40)
du.isna().sum()
```

```
0    0
dtype: int64
```

```
u_ = np.append(u,np.nan)
u_ = np.append(np.nan, u_)
u_ = np.append(u_, np.nan)
u_
```

```
array([nan,  5.,  5.,  5., 32., 62.,
       49., nan, nan])
```

```
du_ = pd.DataFrame(u_)
du_.isna().sum()
```

```
0    3
dtype: int64
```

```
print(str(min(u)) + " " + str(max(u)))
```

```
5 62
```

```
print(str(np.nanmin(u_)) +
  " " + str(np.nanmax(u_)))
```

```
5.0 62.0
```

```
np.quantile(u, q = 0.5) # q represent the %
```

```
18.5
```

```
du.describe()   # It works like summary()
```

```
np.nanstd(u_, ddof = 1)
```

```
25.232254490367417
```

```
np.corrcoef(v,w)
```

```
array([[1.        , 0.9433573],
       [0.9433573, 1.       ]])
```

```
np.sort(v)
```

```
array([10, 20, 30, 30, 50, 60])
```

```
-np.sort(-v)
```

```
array([60, 50, 30, 30, 20, 10])
```

```
np.argsort(w)
```

```
array([1, 0, 2, 3, 5, 4])
```

```
[sorted(w).index(x) for x in w]
# No easy equivalence for the "max"
```

```
[1, 0, 2, 2, 5, 4]
```

```
z = np.maximum(u,v)
np.maximum(z,w) # Same with minimum()
```

```
array([20, 20, 31, 32, 62, 51])
```

Vector manipulation with R

```
options(width = 30)
cumsum(v)
```

```
[1]  10  30  60  90 150 200
```

```
cumprod(v)
```

```
[1] 1.00e+01 2.00e+02 6.00e+03
[4] 1.80e+05 1.08e+07 5.40e+08
```

```
cummax(w)
```

```
[1] 20 20 31 31 61 61
```

```
cummin(w)
```

```
[1] 20 10 10 10 10 10
```

Boolean operation with R

```
a <- 1
b <- 2
(a == 1)
```

```
[1] TRUE
```

```
(a == b)
```

```
[1] FALSE
```

```
(a <= b)
```

```
[1] TRUE
```

```
A <- c(TRUE,TRUE,FALSE,FALSE)
B <- c(TRUE,FALSE,TRUE,FALSE)
A & B
```

```
[1]  TRUE FALSE FALSE FALSE
```

```
A | B
```

```
[1]  TRUE  TRUE  TRUE FALSE
```

```
!A
```

```
[1] FALSE FALSE  TRUE  TRUE
```

```
c <- (a > b)
c
```

```
[1] FALSE
```

```
v <- c(10,20,30,30,60,50)
t <- (v > 30)
t
```

```
[1] FALSE FALSE FALSE FALSE
[5]  TRUE  TRUE
```

```
w <- v[(v>30)]
w
```

```
[1] 60 50
```

```
which(v == 30)
```

```
[1] 3 4
```

Vector manipulation with Python

```
np.cumsum(v)
```

```
array([ 10,  30,  60,  90, 150, 200])
```

```
np.cumprod(v)
```

```
array([       10,       200,      6000,
        180000,  10800000, 540000000])
```

```
dw = pd.DataFrame(w)
dw.cummax().transpose()
```

```
    0   1   2   3   4   5
0  20  20  31  31  61  61
```

```
dw.cummin().transpose()
```

```
    0   1   2   3   4   5
0  20  10  10  10  10  10
```

Boolean operations with Python

```
bool(a == 1)
```

```
True
```

```
bool(a == b)
```

```
False
```

```
bool(a <= b)
```

```
True
```

```
A = np.array([True,True,False,False])
B = np.array([True,False,True,False])
np.logical_and(A, B)
```

```
array([ True, False, False, False])
```

```
np.logical_or(A, B)
```

```
array([ True,  True,  True, False])
```

```
np.logical_not(A)
```

```
array([False, False,  True,  True])
```

```
c = np.array(a > b)
c
```

```
array(False)
```

```
v = np.array([10,20,30,30,60,50])
t = np.array([v > 30])
t
```

```
array([[False, False, False, False,
         True,  True]])
```

```
w = v[(v>30)]
w
```

```
array([60, 50])
```

```
np.where(v == 30)
```

```
(array([2, 3]),)
```

Boolean operations with R

```
which(v == max(v))
```

```
[1] 5
```

```
which(v == min(v))
```

```
[1] 1
```

```
s <- 1*t
s
```

```
[1] 0 0 0 0 1 1
```

```
v <- c(10,20,70,30,60,50)
all(v > 5)
```

```
[1] TRUE
```

```
any(v < 5)
```

```
[1] FALSE
```

Boolean operations with Python

```
np.where(v == max(v))
```

```
(array([4]),)
```

```
np.where(v == min(v))
```

```
(array([0]),)
```

```
s = 1*t
s
```

```
array([[0, 0, 0, 0, 1, 1]])
```

```
v = np.array([10,20,70,30,60,50])
all(v > 5)
```

```
True
```

```
any(v < 5)
```

```
False
```

Conclusion

This conclude the document for now, I'll try to update it if I find useful tips. Hope this will help you in the future.