

# A short presentation about reticulate & disk.frame

wzzang

2020-08-04

---

## How to handle large datasets?

Eg. a 50gb sized csv file with 500k+ obs and prob. 10k+ vars while only a subset of vars is needed.

### Two solutions

- + Python (pandas/dask) + R (reticulate)
  - + R (disk.frame)
- 

## First, reticulate!

1. Install and load the package

```
if(!require(reticulate)){  
  install.packages("reticulate")  
}  
library(reticulate)
```

2. Check Python configuration

Note, if not specifying pre-existing Python env, prompt to install a `miniconda`  
`r-reticulate` conda env will be created using Python3.6 and Numpy  
`use_python()` to specify which Python to use!

```
py_config()  
### python:      /Users/wzzang/Library/r-miniconda/envs/r-reticulate/bin/python  
### libpython:   /Users/wzzang/Library/r-miniconda/envs/r-reticulate/lib/libpython3.6m.dylib  
### pythonhome:  /Users/wzzang/Library/r-miniconda/envs/r-reticulate:/Users/wzzang/Library/r-miniconda  
### version:     3.6.11 | packaged by conda-forge | (default, Jul 28 2020, 23:03:33) [GCC Clang 10.0.0]  
### numpy:       /Users/wzzang/Library/r-miniconda/envs/r-reticulate/lib/python3.6/site-packages/numpy  
### numpy_version: 1.19.1
```

3. Let's check if `pandas` is available.

```
conda_remove('r-reticulate', packages = 'pandas')  
py_module_available(module = 'pandas')  
### [1] FALSE
```

4. Before installing `pandas`, let's check available conda envs.

```
conda_list()
###          name                                     python
### 1  r-miniconda                                     /Users/wzzang/Library/r-miniconda/bin/python
### 2  r-reticulate /Users/wzzang/Library/r-miniconda/envs/r-reticulate/bin/python
```

5. install the package under the *r-reticulate* env

```
py_install(packages = "pandas", envname = "r-reticulate")
```

Now, let's check again.

```
py_module_available(module = 'pandas')
### [1] TRUE
```

6. New envs can be created using `conda_create(envname = "xx", packages = "pandas")`  
Then specify the use of the newly created env using `use_condaenv(condaenv = "xx")`

7. Now let's run Python in R  
First, let's import a csv file in Python.  
Here we use Beijing PM2.5 dataset.

```
import pandas as pd
filename=~"/R_Ladies/BJ_PM25.csv"
pdat = pd.read_csv(filename, sep = ',')
pdat.head()
###   No  year  month  day  hour  pm2.5  DEWP  TEMP  PRES  cbwd  Iws  Is  Ir
###  0   1  2010     1    1    0   NaN   -21  -11.0  1021.0  NW   1.79  0  0
###  1   2  2010     1    1    1   NaN   -21  -12.0  1020.0  NW   4.92  0  0
###  2   3  2010     1    1    2   NaN   -21  -11.0  1019.0  NW   6.71  0  0
###  3   4  2010     1    1    3   NaN   -21  -14.0  1019.0  NW   9.84  0  0
###  4   5  2010     1    1    4   NaN   -20  -12.0  1018.0  NW  12.97  0  0
pdat.shape
### (43824, 13)
```

8. Now, let's run the same code in R.

- There are a few options. First, we can run it in string.

```
py_run_string("import pandas as pd")
dt=py$pd$read_csv(py$filename, sep = ',')
head(dt)
###   No year month day hour pm2.5 DEWP TEMP PRES cbwd  Iws Is Ir
###  1  1 2010     1    1    0   NaN  -21  -11  1021  NW  1.79  0  0
###  2  2 2010     1    1    1   NaN  -21  -12  1020  NW  4.92  0  0
###  3  3 2010     1    1    2   NaN  -21  -11  1019  NW  6.71  0  0
###  4  4 2010     1    1    3   NaN  -21  -14  1019  NW  9.84  0  0
###  5  5 2010     1    1    4   NaN  -20  -12  1018  NW 12.97  0  0
###  6  6 2010     1    1    5   NaN  -19  -10  1017  NW 16.10  0  0
dim(dt)
### [1] 43824    13
```

- Alternatively, we can run a *.py* file.

```
cat ~/R_Ladies/demo_chunk.py
### #!/usr/bin/env python
### # coding: utf-8
###
### import pandas as pd
### filename='~/Downloads/BJ_PM25.csv'
### FileReader = pd.read_csv(filename, chunksize=100, sep = ',')
### dfList=[]
###
### for df in FileReader:
###     dfList.append(df)
###
### #concatenate all chunks
### chunkdf = pd.concat(dfList,sort=False)
###
### #chunkdf.head()
### #chunkdf.shape
```

- Now, we run *.py* script in R and check the *py* object in R.

```
source_python(file="~/R_Ladies/demo_chunk.py")
head(py$chunkdf)
###   No year month day hour pm2.5 DEWP TEMP PRES cbwd   Iws Is Ir
### 1  1 2010     1   1    0   NaN  -21  -11 1021   NW  1.79  0  0
### 2  2 2010     1   1    1   NaN  -21  -12 1020   NW  4.92  0  0
### 3  3 2010     1   1    2   NaN  -21  -11 1019   NW  6.71  0  0
### 4  4 2010     1   1    3   NaN  -21  -14 1019   NW  9.84  0  0
### 5  5 2010     1   1    4   NaN  -20  -12 1018   NW 12.97  0  0
### 6  6 2010     1   1    5   NaN  -19  -10 1017   NW 16.10  0  0
```

- Or, in interactive REFL (Python read-eval-print loop)  

```
filename="~/R_Ladies/BJ_PM25.csv" repl_python()
import pandas as pd
refl_df = pd.read_csv(r.filename, sep = ',')
refl_df.head()
exit
```

`options(reticulate.traceback=T)` to print *Python stack traces for errors*

---

## Now, `disk.frame`!

- make use of `fst` and `data.table` packages
- process data larger than RAM but smaller than Disk
- create a folder (`disk.frame` object) containings chunked files (in *.fst* format)

1. install and load the package

```
if(!require(disk.frame)){
  install.packages('disk.frame')
}
library(disk.frame)
```

2. configure the package setup to allow parallel processing.  
By default, `setup_disk.frame()` uses all available CUP cores.

```
setup_disk.frame(workers = 2)
```

3. load the same BJ PM2.5 dataset.

- Note, directories (disk.frame object) must have `.df` extension

```
otdir = '~/R_Ladies/tmp.df'
if(!dir.exists(otdir)){
  dir.create(otdir)
}

disk.df = as.disk.frame(df = py$pdmat,
  outdir = otdir,
  overwrite = TRUE)
```

- There should be a series of `.fst` files/chunks in the `.df` folder  
Let's take a look.

```
list.files(otdir, all.files = T)
## [1] "."      ".."      ".metadata" "1.fst"  "2.fst"  "3.fst"
```

- `.df` also contains a meta file

```
cat ~/R_Ladies/tmp.df/.metadata/meta.json
### {"nchunks": [3], "shardkey": [""], "shardchunks": [3], "compress": [50]}
```

- `in_chunk_size` can be used to specify row numbers of each reading in case of large dataset  
“If `disk.frame` deems your CSV to be small enough to fit in RAM it will use `data.table`'s `fread`. If the file is too large, it will use `bigreadr` to SPLIT the file into smaller files and then read the smaller files simultaneously. In practice this was found to be the fastest approach.”

```
otdir = '~/R_Ladies/temp.df'
if(!dir.exists(otdir)){
  dir.create(otdir)
}

disk2.df = csv_to_disk.frame(py$filename,
  in_chunk_size = 10000,
  outdir = otdir,
  overwrite = TRUE,
```

```

backend = "data.table", #or 'readr'
chunk_reader = "bigreadr")

list.files(otdir)
### [1] "1.fst" "2.fst" "3.fst"

```

- File split for larger sized data as shown below

```

The number of workers available for disk.frame is 5
Stage 1 of 2: splitting the file ~/Downloads/ukb41049.csv into smaller files:
Destination: /var/folders/4k/hpin0brs7k18_f4vygr7mm00000gn/T//RtmpdaaGGS/r1reee0b3c5ee71e
Stage 1 of 2 took: 00:06:51 elapsed (12.1s cpu)
Stage 2 of 2: Converting the smaller files into disk.frame
csv_to_disk.frame: Reading multiple input files.
Please use 'colClasses = ' to set column types to minimize the chance of a failed read
-----
-- Converting CSVs to disk.frame -- Stage 1 of 2:
Converting 6 CSVs to 69 disk.frames each consisting of 69 chunks
Progress: ----- 100%
-- Converting CSVs to disk.frame -- Stage 1 or 2 took: 00:23:25 elapsed (6.549s cpu)
-----
-- Converting CSVs to disk.frame -- Stage 2 of 2:
Row-binding the 69 disk.frames together to form one large disk.frame:
Creating the disk.frame at ~/R_Ladies/test.df

```

- Now, let's subset data by column names, using `dyplr` functions

```

sub.df = disk.df %>%
  srckeep(c("year", "pm2.5", "TEMP"))

sub.df %>% head
###   No year month day hour pm2.5 DEWP TEMP PRES cbwd   Iws Is Ir
### 1:  1 2010      1   1    0   NaN  -21  -11 1021   NW  1.79  0  0
### 2:  2 2010      1   1    1   NaN  -21  -12 1020   NW  4.92  0  0
### 3:  3 2010      1   1    2   NaN  -21  -11 1019   NW  6.71  0  0
### 4:  4 2010      1   1    3   NaN  -21  -14 1019   NW  9.84  0  0
### 5:  5 2010      1   1    4   NaN  -20  -12 1018   NW 12.97  0  0
### 6:  6 2010      1   1    5   NaN  -19  -10 1017   NW 16.10  0  0

```

- using `srckeep` only load selected cols into memory whereas `select` will load all cols before filtering
- We can get summary descriptions by cols

```

pm25.stats = disk.df %>%
  group_by(TEMP) %>%
  summarise(avrPM2.5 = mean(pm2.5, na.rm = T),
            medPM2.5 = median(pm2.5, na.rm = T)) %>%
  collect #computation executed

pm25.stats
### # A tibble: 64 x 3
###   TEMP avrPM2.5 medPM2.5
###   <dbl>   <dbl>   <dbl>
### 1   -19     27.5     27.5
### 2   -18     46.8      NA
### 3   -17     99.4     81.8
### 4   -16    105.      85
### 5   -15     92.9     65

```

```
### 6 -14 83.7 56.8
### 7 -13 73.4 66
### 8 -12 80.5 60
### 9 -11 76.4 43
### 10 -10 82.8 48
### # ... with 54 more rows
```

4. Add new data as a chunk to the existing disk frame.

```
add_chunk(disk2.df, dt) #dt: r data frame
nchunks(disk2.df)
### [1] 4
list.files(otdir)
### [1] "1.fst" "2.fst" "3.fst" "4.fst"
```

5. Apply functions to all chunks using *cmap*.

- Remember to specify `lazy=F` or to use `collect` for actual computation.

```
result = disk2.df %>%
  cmap(function(chk) {
    chk[1, ] #showing the first row of each chunk
  }, lazy=FALSE)

result
### [[1]]
###   No year month day hour pm2.5 DEWP TEMP PRES cbwd Iws Is Ir
### 1: 1 2010      1  1    0    NA -21 -11 1021  NW 1.79  0  0
###
### [[2]]
###   No year month day hour pm2.5 DEWP TEMP PRES cbwd Iws Is Ir
### 1: 3334 2010      5 19   21   32  -2  28 1000  SE 8.04  0  0
###
### [[3]]
###   No year month day hour pm2.5 DEWP TEMP PRES cbwd Iws Is Ir
### 1: 6667 2010     10  5   18   80  13  20 1012  SE 13.86  0  0
###
### [[4]]
###   No year month day hour pm2.5 DEWP TEMP PRES cbwd Iws Is Ir
### 1: 1 2010      1  1    0   NaN -21 -11 1021  NW 1.79  0  0
```

6. Save intermediate data as a new *.df*

```
otdir = '~/R_Ladies/selected.df'
if(!dir.exists(otdir)){
  dir.create(otdir)
}

disk.df %>%
  srckeep(c("year", "pm2.5", "TEMP")) %>%
```

```

write_disk.frame(outdir = otdir, overwrite = TRUE)

list.dirs('~ /R_Ladies')
### [1] "/Users/wzzang/R_Ladies"
### [2] "/Users/wzzang/R_Ladies/.metadata"
### [3] "/Users/wzzang/R_Ladies/selected.df"
### [4] "/Users/wzzang/R_Ladies/temp.df"
### [5] "/Users/wzzang/R_Ladies/temp.df/.metadata"
### [6] "/Users/wzzang/R_Ladies/tmp.df"
### [7] "/Users/wzzang/R_Ladies/tmp.df/.metadata"

list.files('~ /R_Ladies/selected.df/')
### [1] "1.fst" "2.fst" "3.fst"

file.info('~ /R_Ladies/tmp.df/1.fst')['size']
###
size
### ~ /R_Ladies/tmp.df/1.fst 317061

file.info('~ /R_Ladies/selected.df/1.fst')['size']
###
size
### ~ /R_Ladies/selected.df/1.fst 78665

```

7. More information about `disk.frame` can be found [here](#)