基礎ゼミ02

巨大データの処理(Python,R)

朝倉研究室 修士1年 小泉大哉

2018/4/27

本日の内容

- Pythonの型を説明
- 内包表記を使いこなす
- 高速化モジュールを知る

- dplyrの実践

（有効なパッケージの紹介）

オブジェクト

- Pythonには，入力する際に「オブジェクト」を考慮する必要がある

→型を間違えると，エラーを誘発することがある

→本日の内容では，主に「リスト」の型を主に使う

	型	表記	例
数値	整数	int	int(2) = 2
数値	小数(浮動小数点)	float	float(2) = 2.000000000
数値	文字列	str	str(2) = “2” または str(2) = ‘2’
配列	リスト	list	[1,2,3]
配列	タプル	tuple	(1,2,3)
配列	辞書	dict	{“Chips”:1,”Chocolate”:2,”beverage”:2}
配列	ファイル	file	csv

内包表記の使用

0以上10未満の整数を取り出したい！

- 通常表記

```
python_list = []
for i in range(10):
    python_list.append(i)
python_list
#>>> [0,1,2,3,4,5,6,7,8,9]
```

※下記参照

- 内包表記

```
python_list = [i for i in range(10)]
python_list
#>>>[0,1,2,3,4,5,6,7,8,9]
```

※ range(a,b) : a以上b未満の範囲

※ for i in range(a,b) : a以上b未満の範囲のいかなるiで

2種類の方法での時間を比較

- Jupyterのセルマジック%%timeitを使って計算

%%timeit ⇐ 入力

```
python_list = []
for i in range(10000):
    python_list.append(i)
```

#>>> 653 µs ± 5.66 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

2種類の方法での時間を比較

- Jupyterのセルマジック%%timeitを使って計算

%%timeit ⇐ 入力

```
python_list = [i for i in range(10000)]
```

#>>> 224 µs ± 3.19 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

→ 実際，通常表記より約3倍早くなる

If文が入っている場合の比較

0以上10未満の偶数を取り出したい！

- 通常表記

```
python_list2 = []
for i in range(10):
    if i%2 == 0:
    python_list2.append(i)
python_list2
```

条件の書き込み

- 内包表記

```
python_list2 = [i for i in
range(10) if i%2 == 0]
python_list
#>>>[0,2,4,6,8]
```

後置修飾

#>>> 653 µs ± 5.66 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

numpy, pandasについて

- 要素成分が多い場合、numpyを使うと速い
- データの処理はpandasで行うと速い

高速化モジュールについて

- Cython

　→PythonでC言語の拡張モジュールを割く際する際に使用するプログラミング言語

　→文字の定義がpythonと異なっている(C言語的記述が存在する)が、他はPython記法で計算することが出来る

- numba

　→JITともいう

　→Cythonと同様、高速化処理が必要な場合に使われる

Python まとめ

- Pythonで出てくる関数には便利なものがある

→書きたいコードが分からなかったら，基礎ゼミ第1回平林さんのスライドやインターネットなどで調べよう

- 書き上げたコードが回らなかったら

→ファイルや関数の型、括弧の有無、ファイルの指定場所などを確認する

- 大規模なデータを扱うにはPythonが便利

→Rのみの場合、処理が出来ないことがあうかもしれない

”dyplr”を使ってみよう

- #Match the directory to the current place
- #Session -> set working directory -> To source file location

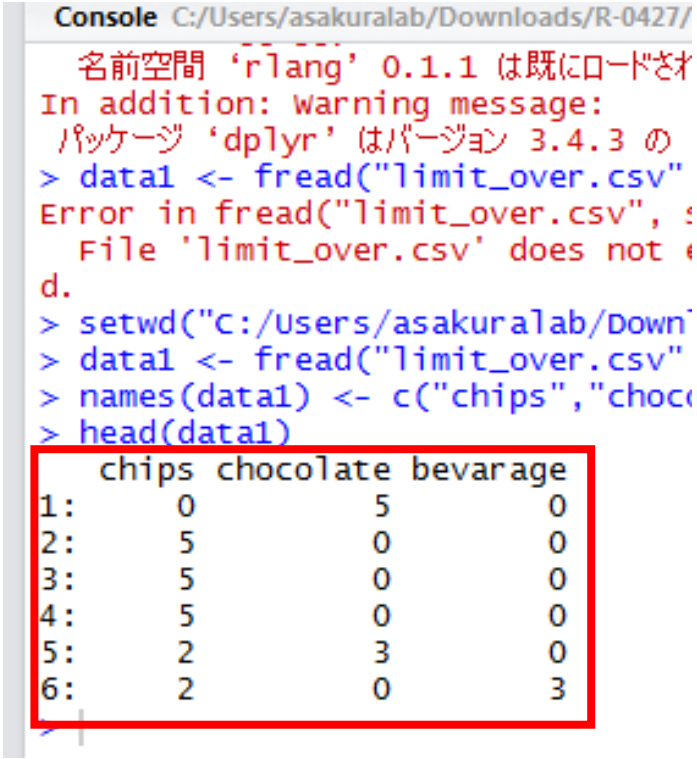
- #installing packages
- install.packages("data.table")
- install.packages("dplyr")
- #utilizing the packages
- library(data.table)
- library(dplyr)

”dyplr”を使ってみよう

- csv fileを読み込む

```
data1 <-
fread("limit_over.csv",stringsAsFactors=FALSE)
```

”dyplr”を使ってみよう




```
names(data1) <-
c("chips","chocolate","
beverage")
head(data1, n=10)
```

”dyplr”を使ってみよう

- data2 <- data1 %>% dplyr::filter(chips == 0)
- head(data2,n=200)
- data2A <- filter(data2,chocolate == 0)
- data2B <- select(data2A, beverage)
- data2C <- data1 %>% arrange(desc(chocolate))

- data3 <- data1 %>% dplyr::select(beverage)
- result <- data3 %>% summarise(max=max(beverage),min=min(beverage),mean=mean(beverage))
- head(result)

”dyplr”を使ってみよう

- write on the csv file (named as “limit_2.csv”)

fwrite(data2, "limit_2.csv",quote=FALSE, row.name

s=FALSE)

※2番目の成分は, csvファイルに書き込む際のセル内のdouble quotationの有無を表す(TRUE:書き込む FALSE:書き込まない)

※3番目の成分は, csvファイルの1行目における行数の書き込みの有無を表す(TRUE:書き込む FALSE:書き込まない)

Rまとめ

- Rには多くの便利な関数およびパッケージがある

→Python同様、書きたいコードは調べるか聞く

→統計するにはPythonより便利

- 処理の関数を覚えるだけではなく、どのような処理をしているかに関心をはらうと面白い

- 書き上げたコードにエラーが発生したら

→ファイルや関数の型、括弧の有無を確認する