solution_2016

July 2, 2023

1 Solution - énoncé avril 2016

Solution de l'énoncé noté d'avril 2016 (lecture de gros fichiers avec pandas). Voir examens.

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')

from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

1.1 QCM

Les bonnes réponses sont en **gras**.

1.1.1 Que fait le programme suivant ?

- Il trie.
- **Il vérifie qu'un tableau est trié.**
- Rien car la boucle ne commence pas à 0.

[2]:
```
l = [0,1,2,3,4,6,5,8,9,10]
res = True
for i in range(1,len (l)) :
    if l[i-1] > l[i]:   # un tableau n'est pas trié si deux éléments consécutifs
      res = False       # ne sont pas dans le bon ordre
print(res)
```

```
False
```

1.1.2 La fonction suivante ne fonctionne pas sur ...

- Le nombre 0.
- La constante "123".
- **Les nombres strictement négatifs**

[3]:
```
def somme(n):
    return sum ( [ int(c) for c in str(n) ] )
    #  un signe moins aménera le calcul de int('-') qui edt invalide
```

```
somme(0), somme("123")
```

[3]: (0, 6)

1.1.3 Le programme suivant provoque une erreur. Quelle est l'exception qu'il va produire ?

- SyntaxError
- TypeError
- **IndexError**

[4]:
```
# déclenche une exception
li = list(range(0,10))
sup = [0,9]
for i in sup :
    del li [i]  # on supprime le premier élément
    # à ce moment le dernier élément est d'indice 8 et non plus 9
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-4-b9a66a0f5ab4> in <module>
      3 sup = [0,9]
      4 for i in sup :
----> 5     del li [i]  # on supprime le premier élément
      6     # à ce moment le dernier élément est d'indice 8 et non plus 9

IndexError: list assignment index out of range
```

1.1.4 Entourer ce que est vrai à propos de la fonction suivante ?

- **Elle est récursive.**
- Il manque une condition d'arrêt.
- **`fibo(4)` appelle récursivement 8 fois `fibo`: une fois `fibo(3)`, deux fois `fibo(2)`, trois fois `fibo(1)` et deux fois `fibo(0)`**

[5]:
```
def fibo (n) :
    print("fibo", n)
    if   n < 1  : return 0
    elif n == 1 : return 1
    else : return fibo (n-1) + fibo (n-2)

fibo(4)
```

```
fibo 4
fibo 3
fibo 2
fibo 1
fibo 0
fibo 1
fibo 2
fibo 1
fibo 0
```

[5]: 3

La fonction est évidemment récursive car elle s'appelle elle-même, elle fait même deux appels récursifs au sein de la même fonction ce qui explique les nombreux appels.

1.1.5 Combien de lignes comporte le dataframe `df2` ?

- **3**
- 4
- 5
- 6
- 7
- 8
- 9
- Aucun, le code provoque une erreur.

[6]:
```
import pandas
df = pandas.DataFrame([dict(x=1, t="e"), dict(x=3, t="f"), dict(x=4, t="e")])
df2 = df.merge(df, left_on="x", right_on="x")
df2
```

[6]:

	x	t_x	t_y
0	1	e	e
1	3	f	f
2	4	e	e

Le dataframe initial a 3 lignes. On le fusionne avec lui même avec une colonne qui ne contient des valeurs distinctes. Chaque ligne ne fusionnera qu'avec une seule ligne. Le résultat contient 3 lignes.

1.1.6 Combien de lignes comporte le dataframe `df3` ?

- 3
- 4
- **5**
- 6
- 7
- 8
- 9
- Aucun, le code provoque une erreur.

[7]:
```
import pandas
df = pandas.DataFrame([dict(x=1, t="e"), dict(x=3, t="f"), dict(x=4, t="e")])
df3 = df.merge(df, left_on="t", right_on="t")
df3.head()
```

[7]:

	x_x	t	x_y
0	1	e	1
1	1	e	4
2	4	e	1
3	4	e	4
4	3	f	3

Le dataframe initial a 3 lignes. On le fusionne avec lui même avec une colonne qui contient des valeurs non distinctes. Il y a 2 `'e'` et 1 `'f'`. La clé unique `'f'` fusionnera avec elle-même, les clés `'e'` fusionneront les unes avec les autres soit 2x2 = 4 lignes. Résultat : $1 + 4 = 5$.

1.2 Dataframes

On suppose qu'on a un fichier de données trop gros pour être chargé en mémoire. On veut produire des statistiques simples. Pour tester votre code, vous pourrez utiliser le fichier *data.txt* construit comme suit :

[8]:
```
import pandas
from urllib.error import URLError
url_ = "https://archive.ics.uci.edu/ml/machine-learning-databases/00350/"
name = "default%20of%20credit%20card%20clients.xls"
url = url_ + name
try:
    df = pandas.read_excel(url, skiprows=1)
except URLError:
    # backup plan
    url_ = "http://www.xavierdupre.fr/enseignement/complements/"
    url = url_ + name
    df = pandas.read_excel(url, skiprows=1)
df.to_csv("data.txt", encoding="utf-8", sep="\t", index=False)
```

1.2.1 Q1

Ecrire une fonction qui agrège un dataset par **AGE** et calcule le mininum, maximum et la moyenne en une seule fois pour les variables **LIMIT_BAL**, **default payment next month** et qui calcule le nombre d'observations partageant le même **AGE**.

[9]:
```
import numpy
res = df.groupby("AGE").agg({"LIMIT_BAL": (min, max, numpy.mean),
                            "ID":len,
                            "default payment next month": (min, max, numpy.mean)})
res.head()
```

[9]:
```
    LIMIT_BAL                          ID default payment next month      \
          min     max           mean   len                         min max
AGE
21      10000   60000   23283.582090    67                           0   1
22      10000  200000   37928.571429   560                           0   1
23      10000  500000   59752.953813   931                           0   1
24      10000  420000   75661.047028  1127                           0   1
25      10000  500000  102731.871838  1186                           0   1


         mean
AGE
21   0.208955
22   0.301786
23   0.265306
24   0.266193
25   0.254637
```

1.2.2 Q2

Lire la documentation de read_csv. On veut charger un fichier en plusieurs morceaux et pour chaque morceau, calculer l'agrégation ci-dessus. Le nom des colonnes n'est présent qu'à la première ligne du programme.

[10]:
```
aggs = []
step = 10000
columns = None
for i in range(0, df.shape[0], step):
    part = pandas.read_csv("data.txt", encoding="utf-8", sep="\t",
                        skiprows=i,
                        nrows=step,
                        header=0 if columns is None else None,
                        names=columns)
    agg = part.groupby("AGE").agg({"LIMIT_BAL": (min, max, numpy.mean),
                        "ID":len,
                        "default payment next month": (min, max, numpy.mean)})
    aggs.append(agg)
    if columns is None:
        columns = part.columns

tout = pandas.concat(aggs)
tout.head()
```

[10]:
```
    LIMIT_BAL                          ID default payment next month      \
          min     max           mean  len                         min max
AGE
21      10000   60000   23846.153846   26                           0   1
22      10000  150000   34720.812183  197                           0   1
23      10000  500000   63718.750000  320                           0   1
24      10000  400000   71879.518072  415                           0   1
25      10000  440000  100143.540670  418                           0   1


         mean
AGE
21   0.192308
22   0.279188
23   0.268750
24   0.306024
25   0.277512
```

Les points importants :

- on utilise la fonction read_csv pour lire le fichier par morceaux avec *skip_rows* et *nrows*
- on calcul les statistiques sur chaque morceau
- le nom des colonnes n'apparaît qu'à la première ligne, donc il faut les conserver pour les ajouter lorsqu'on charge le second morceau du fichier (et les suivant)

Le troisième point est plus élégamment traité avec le paramètre *iterator*. Cette solution est meilleure car la boucle n'utilise pas l'information `df.shape[0]` : cela revient à lire deux fois le fichier, une fois pour avoir le nombre de lignes, une autre pour lire le contenu. La seconde solution ne lit qu'une seule fois le fichier.

[11]:
```
def agg_exo(df):
    gr = df.groupby("AGE").agg({
            #'LIMIT_BAL': {'LB_min': 'min','LB_max': 'max', 'LB_avg': 'mean'},
            'LIMIT_BAL': ['min', 'max', 'mean'],
            'default payment next month': ['min', 'max', 'mean'],
            #'ID': {'len': 'count'}
            'ID': ['count'],
```

```
        })
    gr.columns = ['LB_min', 'LB_max', 'LB_avg',
                  'dpnm_min', 'dpmn_max', 'dpmn_avg',
                  'len']
    return gr

params = {'filepath_or_buffer': "data.txt",
          'encoding': "utf-8", 'sep':"\t" ,
          'iterator': True, 'chunksize':10001}
tout2 = pandas.concat([agg_exo(part) for part in pandas.read_csv(**params)], axis=0)
tout2.head()
```

[11]:

	LB_min	LB_max	LB_avg	dpnm_min	dpmn_max	dpmn_avg	len
AGE							
21	10000	60000	23846.153846	0	1	0.192308	26
22	10000	150000	34720.812183	0	1	0.279188	197
23	10000	500000	63718.750000	0	1	0.268750	320
24	10000	400000	71879.518072	0	1	0.306024	415
25	10000	440000	100143.540670	0	1	0.277512	418

1.2.3 Q3

Le dataframe `tout` est la concaténation de deux dataframes contenant des informations aggrégées pour chaque morceau. On veut maintenant obtenir les mêmes informations agrégrées pour l'ensemble des données uniquement à partir du dataframe `tout`. Ecrire le code qui fait cette agrégation.

[12]:
```
tout[("LIMIT_BAL","w")] = tout[("LIMIT_BAL", "mean")] * tout[("ID", "len")]
# faire attention aux pondérations ici
tout[("default payment next month","w")] = tout[("default payment next month",␣
 ↪"mean")] * tout[("ID", "len")]
toutm = tout.reset_index()

tout_agg = toutm.groupby("AGE").agg({ ("LIMIT_BAL", "min"): min,
                                      ("LIMIT_BAL", "max"): max,
                                      ("LIMIT_BAL", "w"): sum,
                                      ("default payment next month", "min"): min,
                                      ("default payment next month", "max"): max,
                                      ("default payment next month", "w"): sum,
                                      ("ID", "len"):sum,
                                     })
# et là
tout_agg[("LIMIT_BAL", "mean")] = tout_agg[("LIMIT_BAL", "w")] / tout_agg[("ID",␣
 ↪"len")]
tout_agg[("default payment next month", "mean")] = tout_agg[("default payment next␣
 ↪month", "w")] / tout_agg[("ID", "len")]
tout_agg = tout_agg [ sorted(tout_agg.columns)]
tout_agg.head()
```

[12]:

	ID	LIMIT_BAL				\
	len	max	mean	min	w	
AGE						
21	67	60000	23283.582090	10000	1560000.0	
22	560	200000	37928.571429	10000	21240000.0	

```
23    931    500000   59752.953813  10000   55630000.0
24   1127    420000   75661.047028  10000   85270000.0
25   1186    500000  102731.871838  10000  121840000.0

    default payment next month
                            max      mean min      w
AGE
21                            1  0.208955   0   14.0
22                            1  0.301786   0  169.0
23                            1  0.265306   0  247.0
24                            1  0.266193   0  300.0
25                            1  0.254637   0  302.0
```

Calculer une moyenne sur des observations est assez facile mais cela se complique quand on fait une moyenne de moyennes. Il faut retenir le nombre d'observations que représente chaque moyenne sinon la moyenne finale sera fausse. Cela explique la ligne 3.

1.2.4 Q4

Tracer un histogramme avec la valeur moyenne de la variable `LIMIT_BAL`, on ajoutera deux lignes pour les valeurs *min* et *max*. Quelques indications : How to align the bar and line in matplotlib two y-axes chart?.

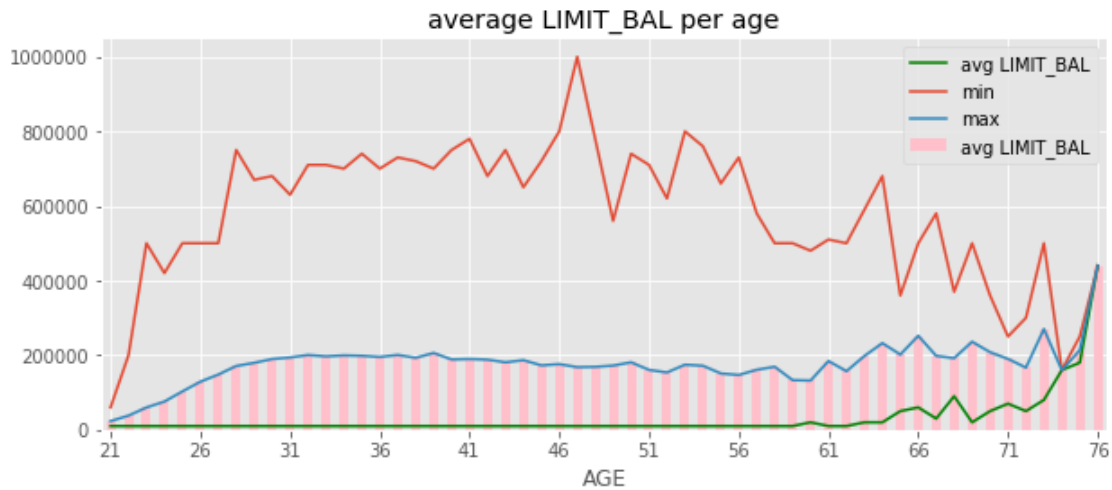
[13]:
```
tout_agg.head()
```

[13]:
```
        ID LIMIT_BAL                                        \
       len       max           mean    min             w
AGE
21      67     60000   23283.582090  10000    1560000.0
22     560    200000   37928.571429  10000   21240000.0
23     931    500000   59752.953813  10000   55630000.0
24    1127    420000   75661.047028  10000   85270000.0
25    1186    500000  102731.871838  10000  121840000.0

    default payment next month
                            max      mean min      w
AGE
21                            1  0.208955   0   14.0
22                            1  0.301786   0  169.0
23                            1  0.265306   0  247.0
24                            1  0.266193   0  300.0
25                            1  0.254637   0  302.0
```

[14]:
```
data = tout_agg
f, ax = plt.subplots(figsize=(10,4))
data.plot.bar(y=("LIMIT_BAL", "mean"), label="avg LIMIT_BAL", ax=ax, color="pink")
data.reset_index(drop=True).plot(y=("LIMIT_BAL", "min"), label="min", kind="line",␣
 ↪ax=ax, color="green")
data.reset_index(drop=True).plot(y=("LIMIT_BAL", "max"), label="max", kind="line",␣
 ↪ax=ax)
data.reset_index(drop=True).plot(y=("LIMIT_BAL", "mean"), label="avg LIMIT_BAL",␣
 ↪kind="line", ax=ax)
x = ax.get_xticks()
ax.xaxis.set_ticks(x[::5])
ax.xaxis.set_ticklabels(x[::5]+min(data.index))
```

```
ax.set_title("average LIMIT_BAL per age");
```



[15]:

```
```