## **2019 10 09**

Pandas DataFrame/Series

pd[[‘name’]] --- 2 dimentions DataFrame --- most regression programe take

pd[‘name’] ---- 1 Dimension Serie

Data = [{‘name’:Yang,’price’:1000}]

df[‘new colone’]=[324,34,5,34,5,3] ---- add new colone

df.columns

df.price + df.price

also df.[‘price’]

df.price > 3 ---- une serie de boolean

df.price > 3 AND df.price <5 wrong

(df.price > 3) & ( df.price <5 ) right

* pas de OR use I

l=[1,2,3]

s=pd.Series(l)

* list on peut mettre tous les types
* series un seul type

df3=pd.DataFrame[{}]

df[‘new colone’]=[324,34,5,34,5,3] ---- add new colone

bool\_series= df [‘price ’] >2

df[bool\_series]

⇒ df[df[‘price’>2]]

s.sort\_values(ascending=False) --- trier

df.sort\_values(‘price’,ascending=False) ----trier un column

df.sort\_values([‘type’,‘price’],ascending=False) ----trier by type then by price

df.sort\_index() ---tier by automatique index 12345

df.type.**str.**upper() ---- in order to use fonction to apply to all elements

dy.type.str.strip()

numbers=list(range(100))

df=pd.DataFramme []

df.head --- les first 5 lins

df.head(10)

df.set\_index(‘price’) ---change and define index, doubon allowed

df.loc[0:100] ----index localiser

df.loc[0:100,’price’] ----line 1-100

df.loc[:,’price’] --- all lines

df.loc[1:3), ‘name’:’price’] --- 1-4 lines , name to price

df.iloc[1:3,4:5] -----iloc take only numbers, the normal slicing of python; loc take name but sometimes index is also in number, it take the last element.

df[‘total’] = df.iloc[:,1:10].sum(axis=1)

df.set\_index([‘type’,’price’]) ---can have multi-index

df.reset\_index(drop=True)

pd.concat([df1,df2]) -----coller deux series?? by lines

pd.concat([df1,df2])

**pd.concat([df1,anotherDataFrame],axis=1)** --------coller par colone

df.set\_index(‘type’).index

for elem in df.colume --- parcourir column name

prinit(elem)

for elem in df.index

prnit(elem)

pd.read\_csv(‘output.csv’, index=False, skiprows=1,na\_value=’XX’)

s=pd.Series([1,2,None])

df.to\_csv(‘output.csv’,index=False,sep=’\t’) ---- export pandas DataFrame to csv file

df.to\_pickle / exl / hdf etcs

df.groupby([‘type’]).mean()

df.set\_index([‘price’,’name’]).unstack() --------reshaping, transformation DataFrame

df.fillna(0) -------replace NaN to 0

df.columns ------- all the columns name

for index, row in df.iterrows() ---- iteration

df.loc[df[‘type’]==male **& l** df[type2]==sfsfd]

df.loc[df[‘name’].**str.contains**(‘yan’)]

df.describe() --- count mean max …

df.sort\_value([‘name’,’tall’])

df.mean() ----- means of each colone

df[‘new column’]=df.col1+df.col2

df.drop(columns=’sfdsf’)

Very big Data -- pretraite, shrink the dataset

for df in pd.read\_csv(‘file’, chunksize = 10000 ):

results=df.groupby(‘time’).count()

newdf = concat([newdf, results])

## **2019 10 16**

## **Exercice**

Faire un programme qui:

* Trouve les 10 villes de France les + peuplées (e.g [https://fr.wikipedia.org/wiki/Liste\_des\_communes\_de\_France\_les\_plus\_peuplées](https://fr.wikipedia.org/wiki/Liste_des_communes_de_France_les_plus_peupl%C3%A9es))
* Pour chaque ville, trouve sa distance avec les autres.
  + par exemple, avec google maps api (nécessite de s'authentifier / créer un token) avec <https://github.com/googlemaps/google-maps-services-python> (cf. doc token dans le readme)
  + ou avec <https://fr.distance24.org/> (pas besoin de s'authentifier)
* Trouve les villes les plus proches

Youtube: comment use API

exo:distance d24

html:

find(‘table’).find\_all (tr)[1:]

for r in rows

cells = ifindall(tr)

city = cell[1].find(a).text.strip() -----strip eleve espace blanc

cities.append(city)

return cites[:limite]

API:

[http://fr.distance24.org/route.json?stops=Hamburg|Berlin](https://fr.distance24.org/route.json?stops=Hamburg%7CBerlin)

url= f””http “

data=requests.get(url).json()

return data[‘distance’]

import itertools

itertools.combinations[cities,2]

creat dataframe: (distances)

df = pd.DataFrame(combinations,columns=(‘origin, dest))

df.sort\_values(‘distance’)

df.set\_index(‘origin’’dest’).unstack

Gmap:

gmap=googlemap.Client(key=open).read()

rows[0]

list.

itertools.combinations(CITIES,2) ------ combination of 2 cities

df.pd.read\_csv(people)

df.set\_index(id)

df.loc(0149014) ---- use 1 id to list the lines

df.loc(01,02)[‘name’’sex’]

mask=df.gender==M ----use mask

df.loc[mask] ------ all male people

.loc ----- can use value to localise

.iloc ------ only take position (numbers)

homework 3

top ten contributers

L'exercice pour le cours prochain est le suivant:

* Récupérer via crawling la liste des 256 top contributors sur cette page <https://gist.github.com/paulmillr/2657075>
* En utilisant l'API github (<https://developer.github.com/v3/>) récupérer pour chacun de ces users le nombre moyens de stars des repositories qui leur appartiennent. Pour finir, classer ces 256 contributors par leur note moyenne.﻿

Comme l'API github dispose de restrictions d'accès (limitation du nombre de requêtes), vous aurez besoin de vous authentifier via un token: <https://help.github.com/en/articles/creating-a-personal-access-token-for-the-command-line>.

OAuth2 token

curl

resp=request.() ----------JSON

Cours: Cleaning the data

csv ---- virgule

line vide, celule vide,

people = pd.read\_csv(xxxx)

people.shape --- (234,7)

.**dtypes** ------ panda automaticly guess the type of the data like objet, int etcs

columns = people.columns ----------rename columns

columns.str.replace(‘ ’ , ’\_’)

people.rename(columns={‘email adress’:’email\_adress’}

people.gender.replace(‘Fameale’:F, ‘Male’:’M’)

empty line

df.name.isna()

df=df[df.name.notna()] ------- delete all lines with empty names

df.dropna(subset=[‘name’]) --------------drop all the lines with NULL(considering these datas not valide) we can all parameter, eg we can like delete lines with at least 3 null

df2 = df[df.x.notna()]

df.duplicated(subset=[‘id’],keep=False)

df.drop\_duplicates() -----delete doublons

df.drop\_duplicates(subset=’name’) -----delete doublons when identical name, delete the whole line

df.age.replace({A:B}) --------------- age into numbers

df[age] = pd.to\_numeric(df.age, errors=’coerce’) ------------ par default errors = raise , exception quand c’est pas des chiffre, coerce transform not chiffe to Nan.

df.head().age.astype(int)

pd.to\_datetime(df.date) ------------transform date format

data\_col.dt.day\_name()

pd.to\_datetime(df.date,format=’%Y-%d-%m’)

df.last\_seen(last seen time) -----------Convert epoch to human-readable date and vice versa

**Unix time** (also known as **Epoch time**, **POSIX time**,[[1]](https://en.wikipedia.org/wiki/Unix_time#cite_note-1) **seconds since the Epoch**,[[2]](https://en.wikipedia.org/wiki/Unix_time#cite_note-single-unix-spec-4.16-2) or **UNIX Epoch time**[[3]](https://en.wikipedia.org/wiki/Unix_time#cite_note-3)) is a system for describing a [point in time](https://en.wikipedia.org/wiki/Timestamp). It is the number of [seconds](https://en.wikipedia.org/wiki/Second) that have elapsed since the **Unix epoch**, that is the time 00:00:00 [UTC](https://en.wikipedia.org/wiki/Coordinated_Universal_Time) on 1 January 1970, minus [leap seconds](https://en.wikipedia.org/wiki/Leap_second). Leap seconds are ignored,[[4]](https://en.wikipedia.org/wiki/Unix_time#cite_note-single-unix-spec-rationale-4.16-4) with a leap second having the same Unix time as the second before it, and every day is treated as if it contains exactly 86400 seconds.[[2]](https://en.wikipedia.org/wiki/Unix_time#cite_note-single-unix-spec-4.16-2) Due to this treatment Unix time is not a true representation of UTC.

Unix time is widely used in [operating systems](https://en.wikipedia.org/wiki/Operating_system) and [file formats](https://en.wikipedia.org/wiki/File_format). In [Unix-like](https://en.wikipedia.org/wiki/Unix-like) operating systems, date is a command which will print or set the current time; by default, it prints or sets the time in the system time zone, but with the -u flag, it prints or sets the time in UTC and, with the TZ environment variable set to refer to a particular time zone, prints or sets the time in that time zone.[[5]](https://en.wikipedia.org/wiki/Unix_time#cite_note-5)

**UNIX时间**，或称**POSIX时间**是[UNIX](https://zh.wikipedia.org/wiki/UNIX)或[类UNIX](https://zh.wikipedia.org/wiki/%E9%A1%9EUNIX)系统使用的时间表示方式：从[UTC](https://zh.wikipedia.org/wiki/%E5%8D%94%E8%AA%BF%E4%B8%96%E7%95%8C%E6%99%82)1970年1月1日0时0分0秒起至现在的总秒数，不考虑[闰秒](https://zh.wikipedia.org/wiki/%E9%96%8F%E7%A7%92)[[1]](https://zh.wikipedia.org/wiki/UNIX%E6%97%B6%E9%97%B4#cite_note-1)。 在多数Unix系统上Unix时间可以透过date +%s指令来检查。示例：**1571466118** ([ISO 8601](https://zh.wikipedia.org/wiki/ISO_8601):2019-10-19T06:21:58Z)

当这个页面生成时的Unix时间

import time

timer.time

pd.to\_datatime(df.last\_seen,unit=’s’’)

<https://www.epochconverter.com/>

df.fillna()

df.regestration.combine\_first( ) ???

df.query(‘age>48’ and age >34’)

df.first\_name+ “ ” + df.last\_name

df.adress.str.split(‘, ‘).str[0]

df.name.str() ?????

df.money ------$34,01

df.money.str[1:].str.replace(‘,’,’.’) -----, to .

pd.to\_numeric(df.money)

df.currency=df.money.str[0]

df.loc[df.currency==’€’,’money’]=df.money[df.currency==’€’,’money’]\*1.10 ..

Check Email : nope@thankyou

ragex101 -<https://regex101.com/>

<https://www.debuggex.com/cheatsheet/regex/pcre>

df[ - df.email.str.contains(‘@’)]

df[ - df.email.str.contains(‘.+@ [0-9a-zA-Z]\.\-\_] + \.\w(2,)’)]

General Email Regex (RFC 5322 Official Standard) <https://emailregex.com/>

(?:[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])\*")@(?:(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]\*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])

%%timeit

x = df[ df.preference.str.contains(‘dessert’) ] ------ look for string is slower than numbers

x = df[ df.preference.str.contains(‘yaourt’) ]

x = df[ df.preference.str.split(\/)

preference.get\_dummies(sept=’/’) ------------------ make a binary mask

dessert\_col= preference.get\_dummies(sept=’/’).dessert ------- get a binary mask on dessert

df[‘dessert\_col’] = df.set\_index(dessert\_col)

<https://engineering.upside.com/a-beginners-guide-to-optimizing-pandas-code-for-speed-c09ef2c6a4d6>

df=pd.DataFrame({‘x’:[1,2,np.nan,np.nan,4,5] })

df.x.interpolate() ------automoticly rempli les valeurs (lineair)

df.set\_index

df.reindex (df2.index, method=’ffill’) ?? ------- make a complet index, if ancien index doesnot existe, make NULL values.

JSON的基本数据类型：

* 对象（object）：一个无序的“键-值对”(pair)，其中键是字符串。建议但不强制要求对象中的键是独一无二的。对象以花括号**{**开始，并以**}**结束。键-值对之间使用逗号分隔。键与值之间用冒号**:**分割。
* 数值：十进制数，不能有前导0，可以为负数，可以有小数部分。还可以用**e**或者**E**表示指数部分。不能包含非数，如NaN。不区分整数与浮点数。JavaScript用双精度浮点数表示所有数值。
* 字符串：以双引号**""**括起来的零个或多个[Unicode](https://zh.wikipedia.org/wiki/Unicode)[码位](https://zh.wikipedia.org/wiki/%E7%A0%81%E4%BD%8D)。支持[反斜杠](https://zh.wikipedia.org/wiki/%E5%8F%8D%E6%96%9C%E6%9D%A0)开始的[转义字符序列](https://zh.wikipedia.org/wiki/%E8%BD%AC%E4%B9%89%E5%AD%97%E7%AC%A6%E5%BA%8F%E5%88%97)。
* 布尔值：表示为true或者false。
* 值的有序列表（array）：有序的零个或者多个值。每个值可以为任意类型。序列表使用方括号**[**，**]**括起来。元素之间用逗号**,**分割。形如：[value, value]
* null类型：值写为null

<http://sametmax.com/un-gros-guide-bien-gras-sur-les-tests-unitaires-en-python-partie-1/>

Le test unitaire le plus bête qu’on puisse avoir en Python :

|  |
| --- |
| *# Fichier de code*  **def** fonction\_a\_tester(param1, param2):  **return** param1 + param2 |

|  |
| --- |
| *# Fichier de test*    **from** fichier\_de\_code **import** fonction\_a\_tester    **assert** fonction\_a\_tester(1, 1) == 2 *# test de l'addition*  **assert** fonction\_a\_tester(1, -1) == 0 *# test avec chiffre négatif*  **assert** fonction\_a\_tester(4, 2) == 6 *# test avec autre chose que des 1*  **assert** fonction\_a\_tester(4.5, 2) == 6.5 *# test avec des floats* |

Deux constats :

* C’est parfaitement chiant. Les tests unitaires sont dans 99% des cas des tautologiques super ennuyeuses.
* On teste le même code plusieurs fois, avec plusieurs cas de figure, pour être certain que ça se comporte comme prévu.

assert est un mot clé qui lève l’exception AssertionError quand l’expression évaluée ne retourne pas True. L’utilisation d’assert n’est pas le sujet de l’article, ici on s’en sert pour faire un test unitaire tout simplement parce que la première ligne qui ne renverra pas True fera planter le programme. C’est le test unitaire du pauvre.

Un test unitaire, ce n’est que ça. Un répétition bête et emmerdante de vérifications généralement très connes.

En l’essence, c’est ça l’intérêt des tests unitaires : vous faire sauter au yeux quand quelque chose casse. On appelle ça des “tests de régression”, et c’est l’usage le plus courant.

<http://sametmax.com/un-gros-guide-bien-gras-sur-les-tests-unitaires-en-python-partie-2/>

Dans jupyterlab, pensez à utiliser <Tab> pour l'autocomplétion et pour découvrir quelles méthodes un type possède. <Shift> + <Tab> pour afficher la documentation d'une fonction.

Je vous invite aussi à survoler les fonctions présentes de base en python:

* <https://docs.python.org/3/library/functions.html>

Ainsi que les modules présents de base (stdlib):

* <https://docs.python.org/3/library/>

Et pour tout le reste il y a pypi:

* [http://pypi.org](http://pypi.org/)