Projet : Manipulation des données avec Pandas

Pandas est une librairie Python spécialisée dans l'analyse des données qui a beaucoup de succès. Dans ce projet pratique, nous nous intéresserons surtout aux fonctionnalités de manipulations de données qu'elle propose. Un objet de type "DataFrame" (tableau de données), permet de réaliser de nombreuses opérations de pré-traitements, de filtrage, de nettoyage, de construction de nouvelles variables à partir des existentes, etc.,... préalables à la modélisation statistique.

La librairie est très largement documentée. Il faut prendre le temps de consulter le Help et de s'exercer simplement ! Deux liens du Help sont incontournables, celui relative aux DataFrame (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame), celui relative aux Series que nous travaillerons également dessus (vecteur de données : http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html#pandas.Series).

Librairie Pandas - Options et version

Il faut charger la librairie et, éventuellement, la configurer selon vos attentes. Pensez à vérifier votre version, elle doit être (non seulement la même sur tous les postes mais aussi la plus récente).

In [2]:
```
# BRANIS AINOUZ
# LORY LETICEE
# FAYCAL BORDJAH
# MOUNIA LYAF
# EISI 21.3

# 1è étape : il faut charger Pandas
# R :
import pandas as pd
import numpy as np

# Vérifiez sa version ?
# R :
pd.show_versions()

# Quelle est la dernière version et MAJ la votre en ligne de commande (au cas
# R : la derniere version de pandas est la 1.1.2

# Rappelez les commandes qui permettent d'installer, MAJ, utiliser 1version m
# Reponses :
# Install : pip install pandas
# Upgrade : pip install --upgrade pandas
# Uninstall : pip uninstall pandas
# Other Versions : pip install pandas==0.19.2

# Modifiez le nb de lignes à afficher dans les print à 10. L'idée est d'évite
# à de multiples affichages de longs tableaux !
# R :
pd.options.display.max_rows = 10
```

```
# Modifiez le nb de colonnes à afficher dans les print à 10. L'idée est d'évi
# par un grand nombre de col !
# R :
pd.options.display.max_columns = 20

# Malgré le fait que ns avons fixé 1petit nb de col à afficher (à 10), les ré
# En d'autres termes, l'idée est d'améliorer l'affichage ds la console en évi
# Activez l'option nécessaire à ce besoin ?
# R :
pd.set_option('display.max_colwidth', None)
# Possible d'utiliser egalement pd.options.display.max_columns = None
pd.options.display.max_columns
```

```
INSTALLED VERSIONS
------------------
commit           : 2a7d3326dee660824a8433ffd01065f8ac37f7d6
python           : 3.8.5.final.0
python-bits      : 64
OS               : Darwin
OS-release       : 19.0.0
Version          : Darwin Kernel Version 19.0.0: Wed Sep 25 20:18:50 PDT 2019;
root:xnu-6153.11.26~2/RELEASE_X86_64
machine          : x86_64
processor        : i386
byteorder        : little
LC_ALL           : None
LANG             : fr_FR.UTF-8
LOCALE           : fr_FR.UTF-8

pandas           : 1.1.2
numpy            : 1.19.2
pytz             : 2020.1
dateutil         : 2.8.1
pip              : 20.1.1
setuptools       : 49.2.0
Cython           : None
pytest           : None
hypothesis       : None
sphinx           : None
blosc            : None
feather          : None
xlsxwriter       : None
lxml.etree       : None
html5lib         : None
pymysql          : None
psycopg2         : None
jinja2           : 2.11.2
IPython          : 7.18.1
pandas_datareader: None
bs4              : None
bottleneck       : None
fsspec           : None
fastparquet      : None
gcsfs            : None
matplotlib       : None
numexpr          : None
odfpy            : None
openpyxl         : None
pandas_gbq       : None
pyarrow          : None
pytables         : None
pyxlsb           : None
s3fs             : None
scipy            : None
sqlalchemy       : None
tables           : None
tabulate         : None
xarray           : None
xlrd             : None
```

```
xlwt            : None
numba           : None
```

Out[2]:
```
20
```

Après avoir réussi ces dernieres manip, créez 1script (startup.py) avec ces diff commandes et indiquez son path à comme PYTHON_STARTUP. L'idée est qu'à chaque lancement de Python, vos différents imports et options soient activées automatiquement !

Chargement, structure DataFrame et description des données

1DataFrame correspond à 1tableau (lignes x cols) que je noterai à partir de maintenant "df". Concernant notre fichier "sante.txt" : La 1è ligne correspond aux noms des vars. A partir de la 2è ligne, ns disposons des valeurs pr chaque enregistrement (individu). Le caractère tabulation "\t" fait office de séparateur de cols. Ns allons vérifier tt ça ds 1 1er tps en lignes de commandes Linux et ds un 2sd temps sur Python.En ligne de commande Linux : 1) Créez 1répertoire "data" (où vs stockerez d'1façon permanente ts les objets que ns travaillerons dessus). 2) Télécharez à partir du google-drive-datartisan le fichier "sante.txt" (secteur de la sante). # R : mkdir data Dans le terminal linux (et non ds 1éditeur de texte), déterminez : 3) la taille du fichier, le nombre de lignes et affichez les 5 1è (lignes). # L'idée est d'avoir un aperçu du fichier ET surtout de déterminer : # Le fait s'il tiens en mémoire (données massives ou non), # Le type de séparateur (tabulation, virgule, pipe,...), # La présence ou non des noms des vars. # Le tout pour préparer la commande Python de création de df (à partir de ce fichier) ! # R : Taille du fichier : du -h data/sante.txt Nb de ligne : wc -l data/sante.txt Affichez les 5 premiere ligne : head -n 5 data/sante.txt

In [50]:
```
# Refaites ces diff manip strictement à l'aide de commandes Python : création
# R : indice import os, open()
!mkdir data

# Revenez sur le script "startup.py" et y mettre les commandes qui vs permett
# Créer le rép "data" (s'il n'existe pas) ?
# Se positinner dessus automatiquement (à chaque démarrage).
# R :
```

```
mkdir: data: File exists
```

In [47]:
```
# Chargez le fichier en tant qu'objet DataFrame (nom = df_sante)
# R :
path = "data/sante.txt"
dt = pd.read_csv(path, sep='\t', low_memory=False)
# Vérifiez que le type obtenu est bien un 'pandas.core.frame.DataFrame'
# R :
type(dt)
```

Out[47]:
```
pandas.core.frame.DataFrame
```

Le type DataFrame est bien reconnu. Voyons maintenant sa structure. Faire un print(df) où vs vérifiez à l'oeil nu, dans la console, ligne par ligne la structure du df n'a pas de sens (contexte Big Data) ! Croyez moi, ça ne va pas vs faire avancer (inversement, au bon entendeur !). Ns préferons alors déterminer la dimension du df, afficher un bout des 1ères et dernières lignes (et comparer avec les résultats déjà obtenu avec le fichier ".txt").

In [5]:
```
# Déterminez les dimensions : nb de lignes et nb de colonnes ?
# R :
dt.shape
# Vérifier que ce sont les bonnes dimensions avec le fichier ".txt"
# Rq : la ligne d'en-tête n'est pas comptabilisée dans le nombre de lignes !
```

Out[5]:
```
(270, 11)
```

In [6]:
```
# Affichez les premières lignes du jeu de données ?
# R :
dt.head()
```

Out[6]:

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11	c
0	70	masculin	D	A	109	non	24	presence	NaN	NaN	
1	67	feminin	C	A	160	non	16	absence	NaN	NaN	
2	57	masculin	B	A	141	non	3	presence	NaN	NaN	
3	64	masculin	D	A	105	oui	2	absence	NaN	NaN	
4	74	feminin	B	A	121	oui	2	absence	NaN	NaN	

In [45]:
```
# Affichez les dernières lignes du jeu de données ?
# R :
dt.tail()
```

Out[45]:

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11
265	52	masculin	C	B	162	non	5	absence	NaN	NaN
266	44	masculin	B	A	173	non	0	absence	NaN	NaN
267	56	feminin	B	A	153	non	13	absence	NaN	NaN
268	57	masculin	D	A	148	non	4	absence	NaN	NaN
269	67	masculin	D	A	108	oui	15	presence	NaN	NaN

Nous passons beaucoup de notre temps à charger des df et à afficher le head (tail) pour vérifier que tt est OK ! Il vaut mieux en faire de ça 1fonc 1fois pr tte que vs appelez au besoin ?

In [36]:
```
# Ecrivez cette fonc où vs géréz les exceptions (contexte script en productio
# R :
def loadDataFrames(path):
    try:
        dt = pd.read_csv(path, sep='\t', low_memory=False)
        return dt.head()

    except FileNotFoundError as e:
        print(e)

path = "data/sante.txt"
loadDataFrames(path)
```

Out[36]:

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11	c
0	70	masculin	D	A	109	non	24	presence	NaN	NaN	
1	67	feminin	C	A	160	non	16	absence	NaN	NaN	
2	57	masculin	B	A	141	non	3	presence	NaN	NaN	
3	64	masculin	D	A	105	oui	2	absence	NaN	NaN	
4	74	feminin	B	A	121	oui	2	absence	NaN	NaN	

In [37]:
```
dt = pd.read_csv(path, sep='\t', low_memory=False)
dt.head()
```

Out[37]:

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11	c
0	70	masculin	D	A	109	non	24	presence	NaN	NaN	

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11	c
1	67	feminin	C	A	160	non	16	absence	NaN	NaN	
2	57	masculin	B	A	141	non	3	presence	NaN	NaN	
3	64	masculin	D	A	105	oui	2	absence	NaN	NaN	
4	74	feminin	B	A	121	oui	2	absence	NaN	NaN	

Les colonnes ?

In [38]:
```
# Lister les colonnes (les noms des vars) ?
# R :
dt.columns.values.tolist()
```

Out[38]:
```
['age',
 'sexe',
 'typedouleur',
 'sucre',
 'tauxmax',
 'angine',
 'depression',
 'coeur col9',
 'col10',
 'col11',
 'col13']
```

In [39]:
```
# Supposons que ce dernier affichage ne vous conviens pas (à l'horizontal, su
# Mettez en place 1boucle "for" qui vous permez d'afficher le résultat (à la
# R :
def showInVertical(dataframe,nbCol):
    i = 1
    listCol = dataframe.columns.values.tolist()
    for val in listCol :
        print(val)
        if i >= nbCol:
            break
        i += 1

showInVertical(dt,5)
```

```
age
sexe
typedouleur
sucre
tauxmax
```

In [52]:
```
# Quelle est le type de chaque colonne : int64, object,... ?
# R : Le type est String
dt = pd.read_csv(path, sep='\t', low_memory=False)
dt.dtypes
```

Out[52]:
```
age              int64
sexe            object
typedouleur     object
sucre           object
tauxmax          int64
                ...
depression       int64
coeur col9      object
col10          float64
col11          float64
col13          float64
Length: 11, dtype: object
```

Il y'a combien type ? (3) Que veut dire le type "object" ? Question à revenir dessus plus tard (après avoir travaillé sur les Series), combien de type (en ligne de commande) ? # R :

In [6]:
```
# Au lieu de taper ces 2 dernières commandes,
# il existe une commande qui vous permet de retrouver ces 2 derniers résultat
# Laquelle ?
# R :
```

Pandas a tendance d'abuser de la mémoire lors de chargement des df. Aussi, il est vivement recommandé de bien affecter les bons types à chaque variable ne serait-ce que pour éviter de faire des opérations douteuses, par exemple arithmétiques sur des vars de type "object". Ds notre cas, vérifiez que notre fichier ne comporte pas de nombres avec des chiffres après la virgule ? Malgré cela, Pandas a stocké les cols concernées en int64 au lieu de int8 (gaspillage) !

In [66]:
```
# Par souci d'optimisation, mettez les cols qui sont en int64 en int8 ?
# R :

dt = pd.read_csv(path, sep='\t', low_memory=False)
dt.astype({'age': 'int8', 'depression':'int8', 'tauxmax':'int8'}).dtypes
```

Out[66]:
```
age              int8
sexe           object
typedouleur    object
sucre          object
tauxmax          int8
                ...
depression       int8
coeur col9     object
col10         float64
col11         float64
col13         float64
Length: 11, dtype: object
```

In [ ]:
```
Q à traiter plus tard : Comment faire la même manip (sans relire le fichier)
```

Supposons que ns avons à travailler sur un fichier avec des nombres avec des chiffres après la virgule (ou point pr spécifier un décimal (US Vs. EU) Trouvez l'option ds pd.read_table() qui permet de le spécifier ? # R :

Description des données

In [70]:
```
# Trouvez une commande pr présenter les statistiques descriptives de tt le df
# R :

dt.describe(include='all')
```

Out[70]:

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9
count	270.000000	270	270	270	270.000000	270	270.0	270
unique	NaN	2	4	2	NaN	2	NaN	2
top	NaN	masculin	D	A	NaN	non	NaN	absence
freq	NaN	183	129	230	NaN	181	NaN	150
mean	54.433333	NaN	NaN	NaN	149.677778	NaN	10.5	NaN
...	...	...	...	...	...	...	...	...
min	29.000000	NaN	NaN	NaN	71.000000	NaN	0.0	NaN
25%	48.000000	NaN	NaN	NaN	133.000000	NaN	0.0	NaN
50%	55.000000	NaN	NaN	NaN	153.500000	NaN	8.0	NaN
75%	61.000000	NaN	NaN	NaN	166.000000	NaN	16.0	NaN

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9
max	77.000000	NaN	NaN	NaN	202.000000	NaN	62.0	NaN

11 rows × 11 columns

Explication : Certains indicateurs statistiques ne sont valables que pour : # les variables numériques (ex. moyenne, min, etc. pour age, tauxmax,...), # et inversemment pour les non-numériques (ex. top, freq, etc. pour sexe, typedouleur, ...), # d'où les NaN dans certaines situations. Est-ce que ce résultat vs convient ?! # Pr plus de clarté, ns souhaitons afficher les stats pr les vars num dans un 1er tps (1è commande ?) # Et ds un 2sd tps les vars de type objects (2è commande ?) # Comment faire ?

In [69]:
```
# Stats desc pr les vars num ?
dt.describe(include=[np.number])
```

Out[69]:

	age	tauxmax	depression	col10	col11	col13
count	270.000000	270.000000	270.000000	0.0	0.0	0.0
mean	54.433333	149.677778	10.500000	NaN	NaN	NaN
std	9.109067	23.165717	11.452098	NaN	NaN	NaN
min	29.000000	71.000000	0.000000	NaN	NaN	NaN
25%	48.000000	133.000000	0.000000	NaN	NaN	NaN
50%	55.000000	153.500000	8.000000	NaN	NaN	NaN
75%	61.000000	166.000000	16.000000	NaN	NaN	NaN
max	77.000000	202.000000	62.000000	NaN	NaN	NaN

In [71]:
```
# Stats desc pr les vars non-num ?
dt.describe(include=['object'])
```

Out[71]:

	sexe	typedouleur	sucre	angine	coeur col9
count	270	270	270	270	270
unique	2	4	2	2	2
top	masculin	D	A	non	absence
freq	183	129	230	181	150

Manipulation des variables

Accès aux variables

Il est possible d'accéder explicitement aux variables. Dans un premier temps, nous utilisons directement les noms des champs (les noms des variables, en en-tête de colonne).

In [74]:
```
# Accès à une colonne de votre choix ?
# R :
s=dt['age']
print(s)
```

```
0      70
1      67
2      57
3      64
4      74
```

```
..
265    52
266    44
267    56
268    57
269    67
Name: age, Length: 270, dtype: int64
```

In [83]:
```
# Autre manière d'accéder à une colonne avec le ".  ?
# R :
a = dt.loc[ : , 'age' ]
```

```
<class 'pandas.core.series.Series'>
```

In [79]:
```
# Accéder à un ensemble de colonnes ?
# R :
dt.loc[ : , ['age', 'tauxmax'] ]
```

Out[79]:

	age	tauxmax
0	70	109
1	67	160
2	57	141
3	64	105
4	74	121
...	...	...
265	52	162
266	44	173
267	56	153
268	57	148
269	67	108

270 rows × 2 columns

In [88]:
```
# Une colonne est un vecteur (Series en terminologie Pandas)
# Affichage des premières valeurs
# R :
dt['age'].head()
```

Out[88]:
```
0    70
1    67
2    57
3    64
4    74
Name: age, dtype: int64
```

In [86]:
```
# Affichage des dernières valeurs
# R :
dt['age'].tail()
```

Out[86]:
```
265    52
266    44
267    56
268    57
269    67
Name: age, dtype: int64
```

In [89]:
```
# Statistique descriptive d'une col. Pour plus de détails, voir :
```

```
# http://pandas.pydata.org/pandas-docs/stable/basics.html#summarizing-data-de
# R :
dt['age'].describe()
```

Out[89]:
```
count    270.000000
mean      54.433333
std        9.109067
min       29.000000
25%       48.000000
50%       55.000000
75%       61.000000
max       77.000000
Name: age, dtype: float64
```

In [90]:
```
# Calculer explicitement la moyenne, par ex col "age"
# R :
dt['age'].mean()
```

Out[90]:
```
54.43333333333333
```

In [91]:
```
# Comptage des valeurs, par ex col "typedouleur"
# R :
dt['typedouleur'].count()
```

Out[91]:
```
270
```

In [93]:
```
# Un type Series est un vecteur, il est possible d'utiliser des indices
# Première valeur ?
# R :
dt['age'][0]
```

Out[93]:
```
70
```

In [94]:
```
# 3 premières valeurs ?
# R :
dt['age'][range(3)]
```

Out[94]:
```
0    70
1    67
2    57
Name: age, dtype: int64
```

In [95]:
```
# Triez les valeurs d'une variable de manière croissante ?
# R :
dt['age'].sort_index(ascending=False)
```

Out[95]:
```
269    67
268    57
267    56
266    44
265    52
       ..
4      74
3      64
2      57
1      67
0      70
Name: age, Length: 270, dtype: int64
```

In [99]:
```
# le tri peut être généralisé aux DataFrame
# par exemple : triez le df selon l'âge puis afficher les 1è lignes
# R :
dt.sort_values(by='age').head()
```

Out[99]:

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11
214	29	masculin	B	A	202	non	0	absence	NaN	NaN
174	34	masculin	A	A	174	non	0	absence	NaN	NaN
138	34	feminin	B	A	192	non	7	absence	NaN	NaN
224	35	feminin	D	A	182	non	14	absence	NaN	NaN
81	35	masculin	D	A	130	oui	16	presence	NaN	NaN

Accès indicé aux données d'un DataFrame

On peut accéder aux valeurs du DataFrame via des indices ou plages d'indice. La structure se comporte alors comme une matrice. La cellule en haut et à gauche est de coordonnées (0,0).

Il y a différentes manières de le faire, l'utilisation de .iloc[,] constitue une des solutions les plus simples. N'oublions pas que Shape permet d'obtenir les dimensions (lignes et colonnes) du DataFrame.

In [105…
```
# Accèdez à la valeur située en (0,0) ?
# R :
dt.iloc[0][0]
```

Out[105… 70

In [106…
```
# Accédez à la valeur située en dernière ligne, première colonne ?
# indice : utilisez l'indiçage négatif
# R :
dt.iloc[-1][0]
```

Out[106… 67

In [113…
```
# Accédez la valeur située en dernière ligne, première colonne ?
# indice : shape[0] renvoie le nombre de lignes (1è dimension) qu'il faut réd
# sinon on déborde
# R :
nb_ligne = dt.shape[0]-1
dt.iloc[nb_ligne][0]
```

Out[113… 67

In [125…
```
# Accédez aux 5 premières valeurs de toutes les colonnes ?
# lignes => 0:5 (0 à 5 [non inclus])
# colonnes = : (toutes les colonnes)
dt.iloc[0:5]
```

Out[125…

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11	c
0	70	masculin	D	A	109	non	24	presence	NaN	NaN	
1	67	feminin	C	A	160	non	16	absence	NaN	NaN	
2	57	masculin	B	A	141	non	3	presence	NaN	NaN	
3	64	masculin	D	A	105	oui	2	absence	NaN	NaN	
4	74	feminin	B	A	121	oui	2	absence	NaN	NaN	

In [127…
```
# Accéder aux 5 dernières lignes du df ? avec l'indiçage négatif, on le peut
dt.iloc[-5: 0]
```

Out[127…

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11	col13

In [128…
```
# Accédez aux 5 premières lignes et deux premières colonnes ?
dt.iloc[0:5, 0:2]
```

Out[128…

	age	sexe
0	70	masculin
1	67	feminin
2	57	masculin
3	64	masculin
4	74	feminin

In [132…
```
# Accédez aux 5 1è lignes et colonnes 0, 1 et 4 ?
# indice : on a une liste d'indices en colonne
dt.iloc[0:5, [0,1,4]]
```

Out[132…

	age	sexe	tauxmax
0	70	masculin	109
1	67	feminin	160
2	57	masculin	141
3	64	masculin	105
4	74	feminin	121

In [135…
```
# faites la même chose autrement ?
# indice : remarquez le rôle de 2 dans 0:5:2
dt.iloc[0:5:2]
```

Out[135…

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11	c
0	70	masculin	D	A	109	non	24	presence	NaN	NaN	
2	57	masculin	B	A	141	non	3	presence	NaN	NaN	
4	74	feminin	B	A	121	oui	2	absence	NaN	NaN	

Restrictions avec les conditions - Les requêtes

Nous pouvons isoler les sous-ensembles d'observations répondant à des critères définis sur les champs. Nous utiliserons préférentiellement la méthode .loc[,] dans ce cadre.

In [143…
```
# Listez les individus présentant une douleur de type A
dt.loc[dt['typedouleur'] == "A"]
```

Out[143…

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11
13	61	masculin	A	A	145	non	26	presence	NaN	NaN
18	64	masculin	A	A	144	oui	18	absence	NaN	NaN
19	40	masculin	A	A	178	oui	14	absence	NaN	NaN
37	59	masculin	A	A	125	non	0	presence	NaN	NaN
63	60	feminin	A	A	171	non	9	absence	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...
198	69	feminin	A	A	151	non	18	absence	NaN	NaN
205	52	masculin	A	B	178	non	12	absence	NaN	NaN
210	59	masculin	A	A	159	non	2	presence	NaN	NaN
228	58	feminin	A	B	162	non	10	absence	NaN	NaN
229	52	masculin	A	A	190	non	0	absence	NaN	NaN

20 rows × 11 columns

In [138…
```
# Nous constatons que l'on indexe avec un vecteur de booléens si on va dans l
print(dt['typedouleur'] == "A")
```

```
0      False
1      False
2      False
3      False
4      False
       ...
265    False
266    False
267    False
268    False
269    False
Name: typedouleur, Length: 270, dtype: bool
```

Seules les observations correspondant à True sont repris par .loc[,]. Nous pouvons les comptabiliser :

In [139…
```
# Comptez le nombre d'individus qui présente une douleur de type "A" ?
dt[dt['typedouleur'] == "A"].shape[0]
```

Out[139… 20

In [140…
```
# pour un ensemble de valeurs de la même variable,
# nous utilisons isin()
print(dt.loc[dt['typedouleur'].isin(['A','B']),:])
```

```
     age      sexe typedouleur sucre  tauxmax angine  depression coeur col9  \
2     57  masculin           B     A       141    non           3   presence
4     74   feminin           B     A       121    oui           2    absence
13    61  masculin           A     A       145    non          26   presence
18    64  masculin           A     A       144    oui          18    absence
19    40  masculin           A     A       178    oui          14    absence
..   ...       ...         ...   ...       ...    ...         ...        ...
262   58  masculin           B     A       160    non          18   presence
263   49  masculin           B     A       171    non           6    absence
264   48  masculin           B     A       168    non          10   presence
266   44  masculin           B     A       173    non           0    absence
267   56   feminin           B     A       153    non          13    absence
```

```
     col10  col11  col13
2      NaN    NaN    NaN
4      NaN    NaN    NaN
13     NaN    NaN    NaN
18     NaN    NaN    NaN
19     NaN    NaN    NaN
..     ...    ...    ...
262    NaN    NaN    NaN
263    NaN    NaN    NaN
264    NaN    NaN    NaN
266    NaN    NaN    NaN
267    NaN    NaN    NaN

[62 rows x 11 columns]
```

Des opérateurs logiques permettent de combiner les conditions. Nous utilisons respectivement : & pour ET, | pour OU, et ~ pour la négation.

In [145…
```
# Listez les individus présentant une douleur de type A et angine == oui ?
dt.loc[(dt['typedouleur'] == 'A') & (dt['angine'] == 'oui')]
```

Out[145…

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11
18	64	masculin	A	A	144	oui	18	absence	NaN	NaN
19	40	masculin	A	A	178	oui	14	absence	NaN	NaN
143	51	masculin	A	A	125	oui	14	absence	NaN	NaN
160	38	masculin	A	A	182	oui	38	presence	NaN	NaN

In [146…
```
# Liste les personnes de moins de 45 ans, de sexe masculin, présentant une ma
dt.loc[(dt['age'] < 45) & (dt['sexe'] == 'masculin') & (dt['coeur col9'] == '
```

Out[146…

	age	sexe	typedouleur	sucre	tauxmax	angine	depression	coeur col9	col10	col11
40	40	masculin	D	A	181	non	0	presence	NaN	NaN
47	44	masculin	D	A	177	non	0	presence	NaN	NaN
50	42	masculin	D	A	125	oui	18	presence	NaN	NaN
81	35	masculin	D	A	130	oui	16	presence	NaN	NaN
147	40	masculin	D	A	114	oui	20	presence	NaN	NaN
…	…	…	…	…	…	…	…	…	…	…
182	41	masculin	D	A	158	non	0	presence	NaN	NaN
193	35	masculin	D	A	156	oui	0	presence	NaN	NaN
231	39	masculin	D	A	140	non	12	presence	NaN	NaN
237	43	masculin	D	A	120	oui	25	presence	NaN	NaN
252	44	masculin	D	A	153	non	0	presence	NaN	NaN

11 rows × 11 columns

In [147…
```
# On peut n'afficher qu'une partie des colonnes
# On définit la projection dans une liste
colonnes = ['age','sexe','coeur col9','tauxmax']
# que l'on utilise en paramètre dans .loc[]
# pour la même restruction que précédemment
print(dt.loc[(dt['age'] < 45) & (dt['sexe'] == "masculin") & (dt['coeur col9'
```

```
     age      sexe coeur col9  tauxmax
40    40  masculin   presence      181
47    44  masculin   presence      177
50    42  masculin   presence      125
81    35  masculin   presence      130
147   40  masculin   presence      114
..   ...       ...        ...      ...
182   41  masculin   presence      158
193   35  masculin   presence      156
231   39  masculin   presence      140
237   43  masculin   presence      120
252   44  masculin   presence      153

[11 rows x 4 columns]
```

Calculs récapitulatifs - Croisement des variables

A la manière des tableaux croisés dynamiques (TCD) d'Excel, nous pouvons procéder à des croisements et opérer des calculs récapitulatifs, qui vont du comptage simple aux calculs statistiques mettent en jeu d'autres variables.

In [151…
```
# Fréquences selon sexe et coeur (tri croisé) ?
# voir : http://pandas.pydata.org/pandas-docs/stable/generated/pandas.crossta
colonnes = ['sexe','coeur col9']
dt[colonnes].value_counts()
```

Out[151…
```
sexe      coeur col9
masculin  presence      100
          absence        83
feminin   absence        67
          presence       20
dtype: int64
```

In [155…
```
# Le même tri croisé mais avec un pourcentage en ligne ?
# indice : ns pouvons demander un post-traitement après la commande précedent
dt[colonnes].value_counts(normalize=True) * 100
```

Out[155…
```
sexe      coeur col9
masculin  presence      37.037037
          absence       30.740741
feminin   absence       24.814815
          presence       7.407407
dtype: float64
```

In [164…
```
# Calculez la moyenne d'âge selon le sexe et la maladie ?
# indice : ns utilisons la fonction mean() de la classe Series de la librairi
colonnes = ['sexe', 'coeur col9']
dt.groupby(colonnes)['age'].mean()
```

Out[164…
```
sexe      coeur col9
feminin   absence       54.582090
          presence      59.350000
masculin  absence       51.192771
          presence      56.040000
Name: age, dtype: float64
```

In [165…
```
# une autre manière de faire avec la commande pivot_table() pour exactement l
print(dt.pivot_table(index=['sexe'],columns=['coeur col9'],values=['age'],agg
```

```
                age
coeur col9    absence presence
sexe
feminin     54.582090    59.35
masculin    51.192771    56.04
```

L'utilisation de groupby() permet d'accéder aux sous-DataFrame associés à chaque item de la variable de regroupement. Il est dès lors possible d'appliquer explicitement d'autres traitements sur ces sous-ensembles de données.

In [166…
```
# Scission des données selon le sexe
g = dt.groupby('sexe')
print(g)
# Calculer la dimension du sous-DataFrame associé aux hommes
print(g.get_group('masculin').shape)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f0283b8cfd0>
(183, 11)
```

In [171…
```
# Calculez la moyenne de l'âge chez les hommes.
g = dt.groupby('sexe')
g.get_group('masculin')['age'].mean()
```

Out[171… 53.84153005464481

In [172…
```
# On peut appliquer différentes fonctions
# agg() permet de revenir sur quelque chose qui ressemble au crosstab()
print(g[['age','depression']].agg([pd.Series.mean, pd.Series.std]))
```

```
                age                depression
               mean       std          mean        std
sexe
feminin   55.678161  9.626144    8.885057  11.332630
masculin  53.841530  8.818189   11.267760  11.459408
```

In [175…
```
# Nous pouvons itérer sur les groupes
for groupe in g:
    # groupe est un tuple
    print(groupe[0]) #étiquette du groupe
    # accès à la variable 'age' du groupe concerné
    print(pd.Series.mean(groupe[1]['age']))
```

```
feminin
55.67816091954023
masculin
53.84153005464481
```

Construction de variables calculées

Les calculs sont vectorisés pour les vecteurs de type Series de Pandas. Ce qui évite de passer par des boucles fastidieuses pour manipuler les valeurs des vecteurs.

In [42]:
```
# Création d'une variable tauxnet (qui n'a aucune signification médicale)
# Utilisation de la libraire numpy (log = logarithme népérien)
```

In [43]:
```
# Ns cherchons à la concaténer au DataFrame
```

La construction d'une variable ex-nihilo est également possible. Par ex., nous souhaitons créer une indicatrice pour la variable sexe, 1 pour masculin, 0 pour féminin.

In [52]:
```
# Création d'une Série de 0 de la même longueur
# que notre DataFrame(nombre de lignes)
# nous utilisons la méthode de numpy pour cela
code = pandas.Series(numpy.zeros(df.shape[0]))
print(code.shape)
```

```
(270,)
```

In [53]:
```
#les "sexe = masculin" sont codés 1
#de fait, "sexe = feminin" est codé zéro puisque le
#vecteur a préalablement été créé avec des valeurs 0
code[df['sexe']=='masculin'] = 1
print(code.value_counts())
```

```
1.0    183
0.0     87
dtype: int64
```

In [54]:
```
#une autre solution plus simple, mais il faut connaître eq()
codebis = df['sexe'].eq('masculin').astype('int')
print(codebis.value_counts())
```

```
1    183
0     87
Name: sexe, dtype: int64
```

Graphiques

Passer par matplotlib permet de réaliser des graphiques performants (http://matplotlib.org/). Mais il faut connaître les procédures de la librairie, ce qui nécessite un apprentissage supplémentaire qui n'est pas toujours évident.

Heureusement, Pandas propose des commandes simples qui encapsulent l'appel à ces procédures et nous simplifie grandement la vie. Il faut importer matplotlib pour que l'ensemble fonctionne correctement.

In [181…

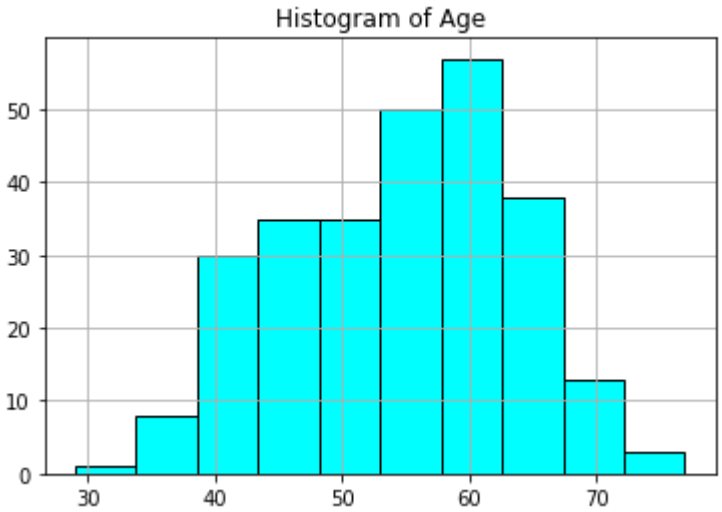
```
Collecting matplotlib
  Using cached matplotlib-3.3.2-cp36-cp36m-manylinux1_x86_64.whl (11.6 MB)
Collecting pillow>=6.2.0
  Using cached Pillow-7.2.0-cp36-cp36m-manylinux1_x86_64.whl (2.2 MB)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.6/dist-pa
ckages (from matplotlib) (1.19.2)
Requirement already satisfied: python-dateutil>=2.1 in /usr/lib/python3/dist-p
ackages (from matplotlib) (2.6.1)
Collecting certifi>=2020.06.20
  Using cached certifi-2020.6.20-py2.py3-none-any.whl (156 kB)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-p
ackages (from matplotlib) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/d
ist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in /ho
me/administrateur/.local/lib/python3.6/site-packages (from matplotlib) (2.4.7)
Requirement already satisfied: six in /home/administrateur/.local/lib/python3.
6/site-packages (from cycler>=0.10->matplotlib) (1.14.0)
Installing collected packages: pillow, certifi, matplotlib
  Attempting uninstall: pillow
    Found existing installation: Pillow 5.1.0
    Uninstalling Pillow-5.1.0:
ERROR: Could not install packages due to an EnvironmentError: [Errno 13] Permi
ssion non accordée: 'XpmImagePlugin.py'
Consider using the `--user` option or check the permissions.
```

In [182…
```
#indiquer que l'on veut voir apparaître les graphiques dans le notebook
#/!\ très important, sinon on ne verrait rien
%matplotlib inline

#importation de la librairie
import matplotlib.pyplot as plt
```

In [187…
```
# Tracer 1histogramme de l'âge ?
dt.hist(column='age', color='cyan', edgecolor='black')
plt.title('Histogram of Age')
```

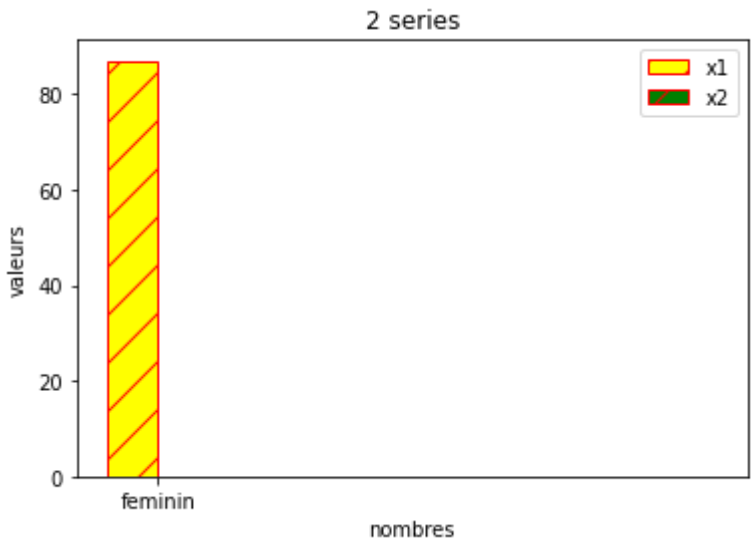
Out[187… `Text(0.5, 1.0, 'Histogram of Age')`



In [45]:
```
# Tracer 1density plot ?
```

In [200…
```
# Tracer l'histogramme de l'âge selon le sexe ?
bins = [x + 0.5 for x in range(0, 6)]
plt.hist([dt['sexe'], dt['age']], bins=bins, color = ['yellow', 'green'],
            edgecolor = 'red', hatch = '/', label = ['x1', 'x2'],
            histtype = 'bar') # bar est le defaut
plt.ylabel('valeurs')
plt.xlabel('nombres')
plt.title('2 series')
plt.legend()
```

Out[200… `<matplotlib.legend.Legend at 0x7f0284434da0>`



In [47]:
```
# Comparaison des distributions avec un boxplot ?
```

In [48]:
```
# Tracer 1scatterplot : age vs. tauxmax ?
```

In [49]:
```
# Tracer 1scatterplot (age vs. tauxmax) en distinguant les points ?
# (niveau de gris) selon les valeurs de dépression
```

In [51]:
```
# Tracer 1 scatterplot (age vs. tauxmax) en distinguant les points selon les
```

```
# indice : nécessite un recodage de coeur - ici en 0/1
# afficher le graphique en spécifiant la couleur (blue = 0, green = 1)
```

In [52]:
```
# Tracer 1diagramme à secteurs - comptage de sexe ?
```

In [53]:
```
# TRacez le scatterplot des variables pris deux à deux
# Cela n'a d'intérêt que pour les variables quantitatives bien évidemment
```