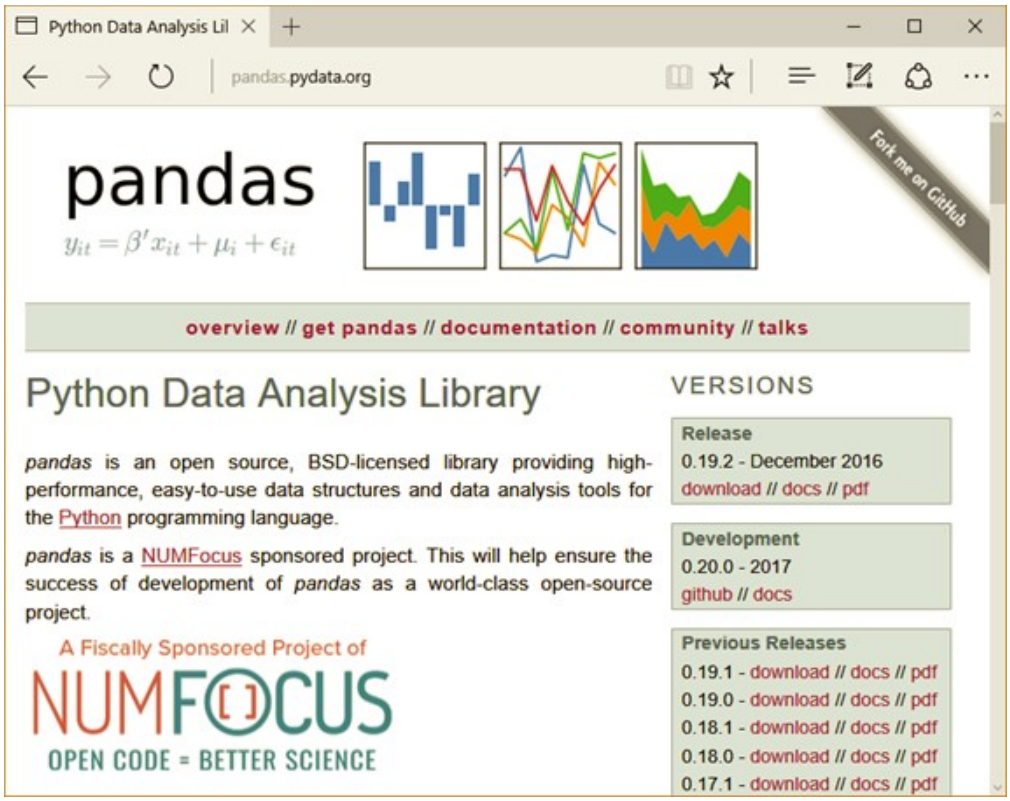
Python : Manipulation des données avec Pandas



Pandas est une librairie Python spécialisée dans l'analyse des données. Dans ce support, nous nous intéresserons surtout aux fonctionnalités de manipulations de données qu'elle propose. Un objet de type "data frame", bien connu sous R, permet de réaliser de nombreuses opérations de filtrage, prétraitements, etc., préalables à la modélisation statistique.

La librairie est très largement documentée. Il faut prendre le temps de la parcourir simplement (ce n'est pas le plus évident). Deux références sont incontournables, celle relative aux DataFrame (tableau de données : http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame), celle relative aux Series (vecteur de données, une variable dans notre contexte : http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html#pandas.Series).

Chargement et description des données

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 raccord (la même ou plus récente) avec ce qui est proposé dans ce tutoriel.

In [1]:
```
#Première étape : il faut charger la librairie Pandas
import pandas

#par convenance pure, nous modifions le nombre de lignes
#à afficher dans les print. L'idée est d'éviter que le tutoriel
#se résume à de multples affichages de longs tableaux

#vous pouvez modifier cette option à votre guise
pandas.options.display.max_rows = 10

#vérifier la version
print(pandas.__version__)
```

```
0.19.2
```

Structure DataFrame

Une matrice DataFrame correspond à une matrice individus-variables où les lignes correspondent à des observations, les colonnes à des attributs décrivant les individus.

Concernanr notre fichier "heart.txt" : la première ligne correspond aux noms des champs (des variables) ; à partir de la seconde ligne, nous disposons des valeurs pour chaque enregistrement (individu) ; le caractère tabulation "\t" fait office de séparateur de colonnes.

Dans ce qui suit, nous chargeons le fichier de données et nous procédons à quelques vérifications.

In [2]:
```
#chargement du fichier
#df est le nom de l'objet de type data frame créé
#sep spécifie le caractère séparateur de colonnes
#header = 0 : la ligne numéro 0 = aux noms des champs
#éventuellement decimal permet d'indiquer le point décimal
df = pandas.read_table("heart.txt",sep = '\t',header = 0)

#vérifions le type de df
print(type(df))
```

```
<class 'pandas.core.frame.DataFrame'>
```

Le type DataFrame est bien reconnu.

Voyons maintenant l'architecture de la structure DataFrame.

In [3]:
```
#dimensions : nombre de lignes, nombre de colonnes
#la ligne d'en-tête n'est pas comptabilisée
#dans le nombre de lignes
print(df.shape)
```

```
(270, 8)
```

In [4]:
```
#afficher les premières lignes du jeu de données
print(df.head())
```

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

In [5]:
```
#énumération des colonnes
print(df.columns)
```

```
Index(['age', 'sexe', 'typedouleur', 'sucre', 'tauxmax', 'angine',
       'depression', 'coeur'],
      dtype='object')
```

In [6]:
```
#type de chaque colonne
print(df.dtypes)
```

```
age             int64
sexe           object
typedouleur    object
sucre          object
tauxmax         int64
angine         object
depression      int64
coeur          object
dtype: object
```

In [7]:
```
#informations sur les données
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 8 columns):
age            270 non-null int64
sexe           270 non-null object
typedouleur    270 non-null object
sucre          270 non-null object
tauxmax        270 non-null int64
angine         270 non-null object
depression     270 non-null int64
coeur          270 non-null object
dtypes: int64(3), object(5)
memory usage: 17.0+ KB
None
```

In [8]:
```
#description des données
print(df.describe(include='all'))
```

```
                age      sexe typedouleur sucre     tauxmax angine  depression
\
count   270.000000       270         270   270  270.000000    270       270.0
unique         NaN         2           4     2         NaN      2         NaN
top            NaN  masculin           D     A         NaN    non         NaN
freq           NaN       183         129   230         NaN    181         NaN
mean     54.433333       NaN         NaN   NaN  149.677778    NaN        10.5
...            ...       ...         ...   ...         ...    ...         ...
min      29.000000       NaN         NaN   NaN   71.000000    NaN         0.0
25%      48.000000       NaN         NaN   NaN  133.000000    NaN         0.0
50%      55.000000       NaN         NaN   NaN  153.500000    NaN         8.0
75%      61.000000       NaN         NaN   NaN  166.000000    NaN        16.0
max      77.000000       NaN         NaN   NaN  202.000000    NaN        62.0

          coeur
count       270
unique        2
top     absence
freq        150
mean        NaN
...         ...
min         NaN
25%         NaN
50%         NaN
75%         NaN
max         NaN

[11 rows x 8 columns]
```

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.

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 [9]:
```
#accès à une colonne
print(df['sexe'])
```

```
0      masculin
1       feminin
2      masculin
3      masculin
4       feminin
         ...
265    masculin
266    masculin
267     feminin
268    masculin
269    masculin
Name: sexe, dtype: object
```

In [10]:
```
#autre manière d'accéder à une colonne avec le .
#sous R nous utiliserions le $
print(df.sexe)
```

```
0      masculin
1       feminin
2      masculin
3      masculin
4       feminin
         ...
265    masculin
266    masculin
267     feminin
268    masculin
269    masculin
Name: sexe, dtype: object
```

In [11]:
```
#accéder à un ensemble de colonnes
print(df[['sexe','sucre']])
```

```
         sexe sucre
0    masculin     A
1     feminin     A
2    masculin     A
3    masculin     A
4     feminin     A
..        ...   ...
265  masculin     B
266  masculin     A
267   feminin     A
268  masculin     A
269  masculin     A

[270 rows x 2 columns]
```

In [12]:
```
#une colonne est un vecteur (Series en terminologie Pandas)
#affichage des premières valeurs
print(df['age'].head())
```

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

In [13]:
```
#affichage des dernières valeurs
print(df['age'].tail())
```

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

In [14]:
```
#statistique descriptive. Pour plus de détails, voir :
#http://pandas.pydata.org/pandas-docs/stable/basics.html#summarizing-data-describe
print(df['age'].describe())
```

```
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 [15]:
```
#calculer explicitement la moyenne
print(df['age'].mean())
```

```
54.43333333333333
```

In [16]:
```
#comptage des valeurs
print(df['typedouleur'].value_counts())
```

```
D    129
C     79
B     42
A     20
Name: typedouleur, dtype: int64
```

In [17]:
```
#un type Series est un vecteur, il est possible d'utiliser des indices
#première valeur
print(df['age'][0])
```

```
70
```

In [18]:
```
#3 premières valeurs
print(df['age'][0:3])
```

```
0    70
1    67
2    57
Name: age, dtype: int64
```

In [19]:
```
#ou bien donc
print(df.age[0:3])
```

```
0    70
1    67
2    57
Name: age, dtype: int64
```

In [20]:
```
#trier les valeurs d'une variable de manière croissante
print(df['age'].sort_values())
```

```
214    29
174    34
138    34
224    35
81     35
       ..
15     71
255    71
4      74
73     76
199    77
Name: age, dtype: int64
```

La plus petite valeur est 29, elle correspond à l'observation n°214.

In [21]:
```
#nous pouvons aussi obtenir les indices des valeurs triées
print(df['age'].argsort())
```

```
0      214
1      174
2      138
3      224
4       81
      ...
265     15
266    255
267      4
268     73
269    199
Name: age, dtype: int64
```

214 est le numéro de l'individu portant la plus petite valeur de la variable age, puis vient le n°174, etc. Ces résultats sont complètement cohérents avec ceux

In [22]:
```
#le tri peut être généralisé aux DataFrame
#par exemple : trier le tableau de données selon l'âge
#puis affichage des premières lignes avec head()
print(df.sort_values(by='age').head())
```

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

Itérations sur les variables

Les itérations sur les variables peuvent se faire via une boucle, ou via l'utilisation de fonctions callback appelée à l'aide d'une fonction .apply(). Ce qui n'est pas sans rappeler les fonctions sapply() et lapply() de R.

In [23]:
```
#boucler sur l'ensemble des colonnes
for col in df.columns:
    print(df[col].dtype)
```

```
int64
object
object
object
int64
object
int64
object
```

In [24]:
```
#passage par la librairie numpy
import numpy

#fonction call back
def operation(x):
    return(x.mean())

#appel de la fonction sur l'ensemble des colonnes du DataFrame
#axis = 0 ==> chaque colonne sera transmise à la fonction operation()
#la selection select_dtypes() permet d'exclure les variables non numériques
resultat = df.select_dtypes(exclude=['object']).apply(operation,axis=0)
print(resultat)
```

```
age            54.433333
tauxmax       149.677778
depression     10.500000
dtype: float64
```

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 [25]:
```
#accès à la valeur située en (0,0)
print(df.iloc[0,0])
```

```
70
```

In [26]:
```
#valeur située en dernière ligne, première colonne
#utilisation de l'indiçage négatif
print(df.iloc[-1,0])
```

```
67
```

In [27]:
```
#valeur située en dernière ligne, première colonne
#shape[0] renvoie le nombre de lignes (première dimension)
#il faut réduire de -1 parce le premier indice est égal à 0
#sinon on déborde
print(df.iloc[df.shape[0]-1,0])
```

```
67
```

In [28]:
```
#5 premières valeurs de toutes les colonnes
#lignes => 0:5 (0 à 5 [non inclus])
#colonnes = : (toutes les colonnes)
print(df.iloc[0:5,:])
```

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

In [29]:
```
#avec l'indiçage négatif, on peut facilement accéder aux 5 dernières lignes
print(df.iloc[-5:,:])
```

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

In [30]:
```
#5 premières lignes et deux premières colonnes
print(df.iloc[0:5,0:2])
```

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

In [31]:
```
#5 premières lignes et colonnes 0, 1 et 4
#on a une liste d'indices en colonne
print(df.iloc[0:5,[0,2,4]])
```

```
   age typedouleur  tauxmax
0   70           D      109
1   67           C      160
2   57           B      141
3   64           D      105
4   74           B      121
```

In [32]:
```
#ou encore, pour exactement la même chose
#remarquez le rôle de 2 dans 0:5:2
print(df.iloc[0:5,0:5:2])
```

```
   age typedouleur  tauxmax
0   70           D      109
1   67           C      160
2   57           B      141
3   64           D      105
4   74           B      121
```

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 [33]:
```
#liste des individus présentant une douleur de type A
print(df.loc[df['typedouleur']=="A",:])
```

```
     age      sexe typedouleur sucre  tauxmax angine  depression     coeur
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
37    59  masculin           A     A      125    non           0  presence
63    60   feminin           A     A      171    non           9   absence
..   ...       ...         ...   ...      ...    ...         ...       ...
198   69   feminin           A     A      151    non          18   absence
205   52  masculin           A     B      178    non          12   absence
210   59  masculin           A     A      159    non           2  presence
228   58   feminin           A     B      162    non          10   absence
229   52  masculin           A     A      190    non           0   absence

[20 rows x 8 columns]
```

In [34]:
```
#nous constatons que l'on indexe avec un vecteur de
#booléens si on va dans le détail. En effet,
print(df['typedouleur']=="A")
```

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

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

In [35]:
```
print((df['typedouleur']=="A").value_counts())
```

```
False    250
True      20
Name: typedouleur, dtype: int64
```

In [36]:
```
#pour un ensemble de valeurs de la même variable,
#nous utilisons isin()
print(df.loc[df['typedouleur'].isin(['A','B']),:])
```

```
     age      sexe typedouleur sucre  tauxmax angine  depression     coeur
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

[62 rows x 8 columns]
```

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

In [37]:
```
#liste des individus présentant une douleur de type A et angine == oui
print(df.loc[(df['typedouleur']=="A") & (df['angine'] == "oui"),:])
```

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

In [38]:
```
#liste des personnes de moins de 45 ans, de sexe masculin, présentant une maladi
e cardiaque
print(df.loc[(df['age'] < 45) & (df['sexe'] == "masculin") & (df['coeur'] == "pr
esence"),:])
```

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

[11 rows x 8 columns]
```

In [39]:
```
#on peut n'afficher qu'une partie des colonnes
#on définit la projection dans une liste
colonnes = ['age','sexe','coeur','tauxmax']
#que l'on utilise en paramètre dans .loc[]
#pour la même restruction que précédemment
print(df.loc[(df['age'] < 45) & (df['sexe'] == "masculin") & (df['coeur'] == "pr
esence"),colonnes])
```

```
     age      sexe     coeur  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 [40]:
```
#fréquences selon sexe et coeur - cela ressemnble à table() de R
#voir : http://pandas.pydata.org/pandas-docs/stable/generated/pandas.crosstab.ht
ml
print(pandas.crosstab(df['sexe'],df['coeur']))
```

```
coeur     absence  presence
sexe
feminin        67        20
masculin       83       100
```

In [41]:
```
#nous pouvons demander un post-traitement, à la différence de table() de R
#par exemple, un pourcentage en ligne
print(pandas.crosstab(df['sexe'],df['coeur'],normalize='index'))
```

```
coeur      absence  presence
sexe
feminin   0.770115  0.229885
masculin  0.453552  0.546448
```

In [42]:
```
#nous pouvons aussi insérer un champ calculé, comme avec les TCD d'Excel
#ex. moyenne d'âge selon le sexe et la maladie
#nous utilisons la fonction mean() de la classe Series de la librairie Pandas
#cela nous rapproche plus des fonctions tapply() ou aggregate() de R
print(pandas.crosstab(df['sexe'],df['coeur'],values=df['age'],aggfunc=pandas.Ser
ies.mean))
```

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

In [43]:
```
#une autre manière de faire avec la commande pivot_table()
#pour exactement le même résultat
print(df.pivot_table(index=['sexe'],columns=['coeur'],values=['age'],aggfunc=pan
das.Series.mean))
```

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

In [44]:
```
#multiplier les critères est possible
#mais comme avec les TCD, la lecture devient compliquée
print(pandas.crosstab([df['sexe'],df['sucre']],df['coeur']))
```

```
coeur           absence  presence
sexe     sucre
feminin  A           61        15
         B            6         5
masculin A           66        88
         B           17        12
```

In [45]:
```
#il reste que les possibilités sont étendues
print(pandas.crosstab([df['sexe'],df['sucre']],df['coeur'],normalize='index'))
```

```
coeur            absence  presence
sexe     sucre
feminin  A      0.802632  0.197368
         B      0.545455  0.454545
masculin A      0.428571  0.571429
         B      0.586207  0.413793
```

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 [46]:
```
#scission des données selon le sexe
g = df.groupby('sexe')

#calculer la dimension du sous-DataFrame associé aux hommes
print(g.get_group('masculin').shape)
```

```
(183, 8)
```

In [47]:
```
#calculer la moyenne de l'âge chez les hommes.
print(g.get_group('masculin')['age'].mean())
```

```
53.84153005464481
```

In [48]:
```
#on peut appliquer différentes fonctions
#agg() permet de revenir sur quelque chose qui ressemble au crosstab()
print(g[['age','depression']].agg([pandas.Series.mean,pandas.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 [49]:
```
#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(pandas.Series.mean(groupe[1]['age']))
```

```
feminin
55.67816091954023
masculin
53.84153005464481
```

Construction de variables calculées

Comme sous Numpy (comme sous R), 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 [50]:
```
#création d'une variable tauxnet (qui n'a aucune signification médicale)
#utilisation de la libraire numpy (log = logarithme népérien)
import numpy
tauxnet = df['tauxmax']*numpy.log(df['age'])
print(tauxnet)
```

```
0      463.085981
1      672.750819
2      570.070229
3      436.682724
4      520.791876
          ...
265    640.101482
266    654.664807
267    615.878809
268    598.371588
269    454.106803
dtype: float64
```

In [51]:
```
#laquelle variable peut être concaténée au DataFrame
newdf = pandas.concat([df,tauxnet],axis=1)
print(newdf.shape)
```

```
(270, 9)
```

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

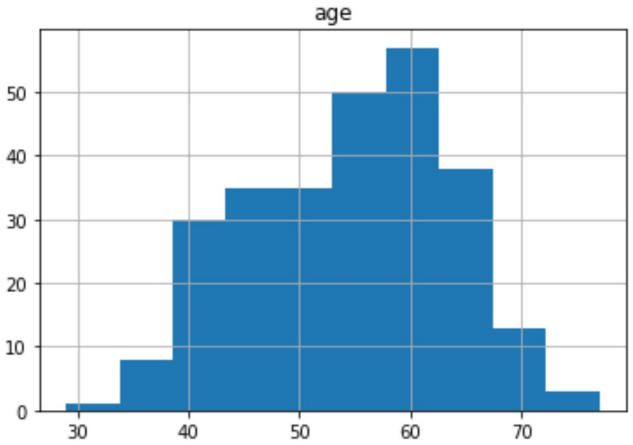
Passer par matplotlib permet de réaliser des graphiques performants (http://matplotlib.org/). Mais il faut connaître les procédures de la librairie, ce qui nécessite un apprentissage supplémentaire qui n'est pas toujours évident.

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 [55]:
```
#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
```

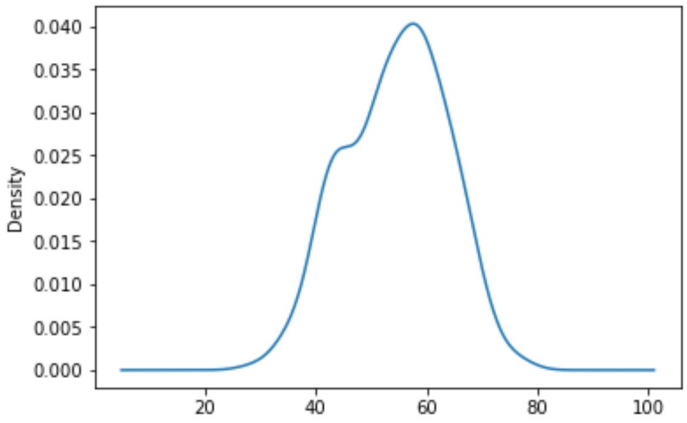
In [56]:
```
#histogramme de l'âge
df.hist(column='age')
```

Out[56]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001C472A89940>]]
, dtype=object)
```



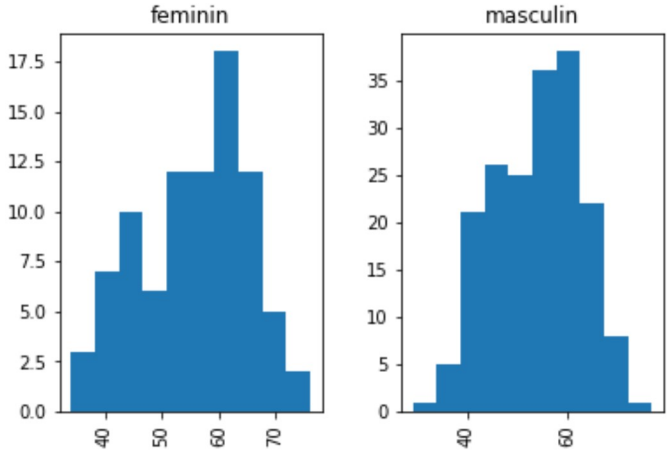
In [57]:
```
#density plot
df['age'].plot.kde()
```

Out[57]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c472ed2048>
```



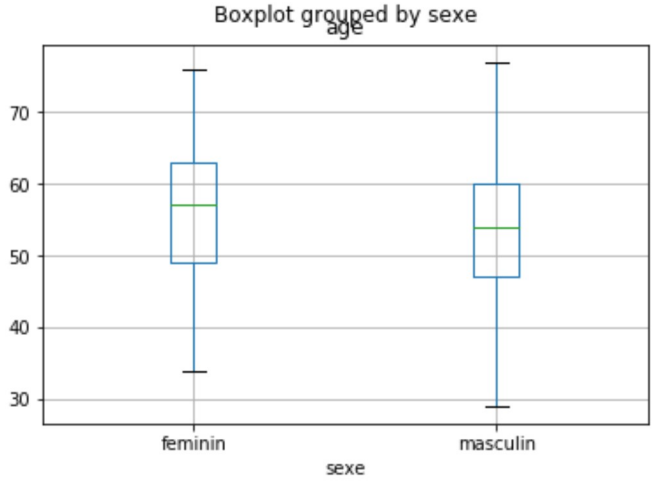
In [58]:
```
#histogrammes de l'âge selon le sexe
df.hist(column='age',by='sexe')
```

Out[58]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000001C473784668>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001C47381D160>],
dtype=object)
```



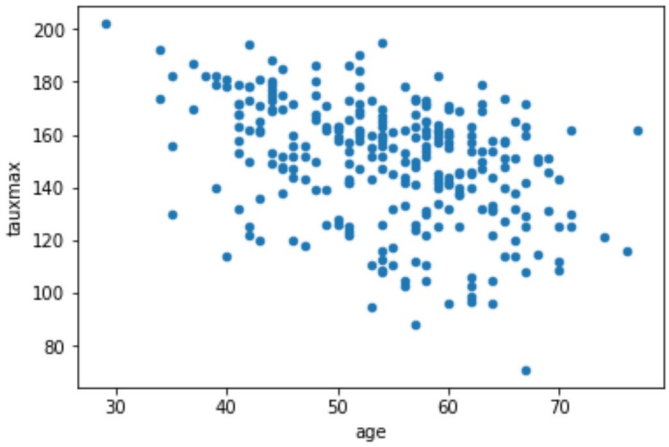
In [59]:
```
#comparaison des distributions avec un boxplot
df.boxplot(column='age',by='sexe')
```

Out[59]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4738f7198>
```



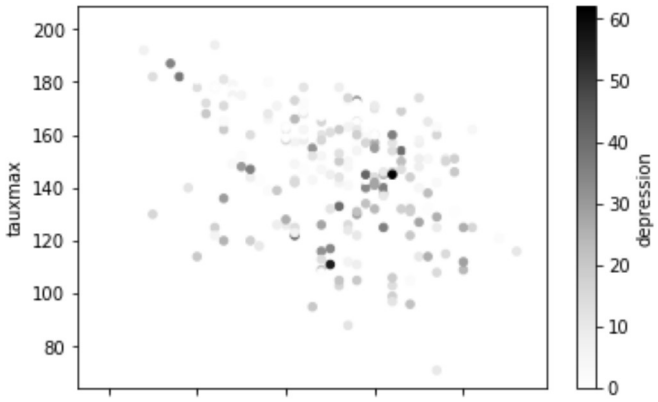
In [60]:
```
#scatterplot : age vs. tauxmax
df.plot.scatter(x='age',y='tauxmax')
```

Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x1c4738c90b8>



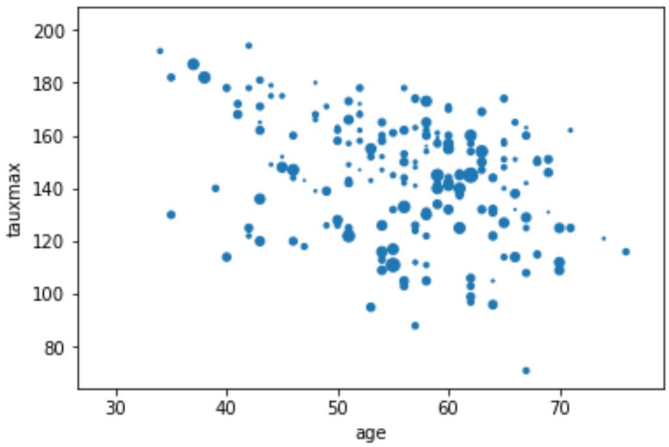
In [61]:
```
#scatterplot (age vs. tauxmax) en distinguant les points
#(niveau de gris)selon les valeurs de dépression
df.plot.scatter(x='age',y='tauxmax',c='depression')
```

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x1c4739ee400>



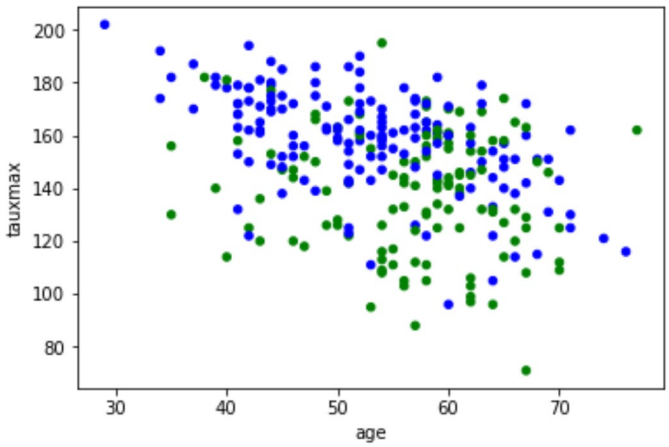
In [62]:
```
#scatterplot (age vs. tauxmax) en distinguant les points
#(taille des points) selon les valeurs de dépression
df.plot.scatter(x='age',y='tauxmax',s=df['depression'])
```

Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x1c474b1fc50>



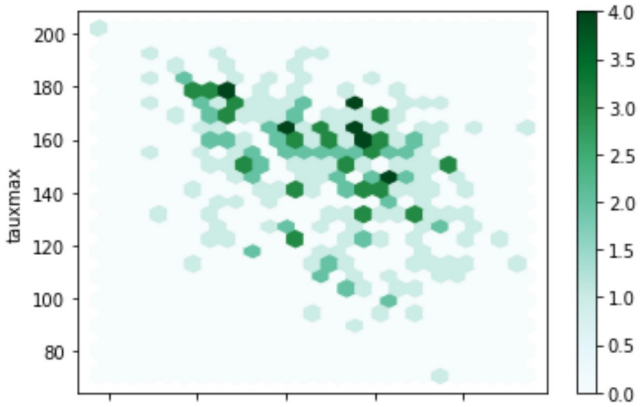
In [63]:
```
#scatterplot (age vs. tauxmax) en distinguant les points selon les valeurs
#de coeur : nécessite un recodage de coeur - ici en 0/1
code_coeur = df['coeur'].eq('presence').astype('int')
#afficher le graphique en spécifiant la couleur (blue = 0, green = 1)
#les habitués de R reconnaîtront l'astuce
df.plot.scatter(x='age',y='tauxmax',c=pandas.Series(['blue','green'])[code_coeur
])
```

Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x1c473a589e8>



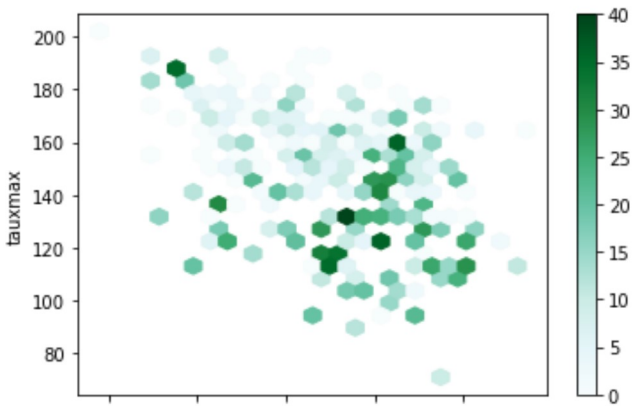
In [64]:
```
#grille à la carte de Kohonen - permet de voir la densité des points ici
df.plot.hexbin(x='age',y='tauxmax',gridsize=25)
```

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x1c474c050f0>



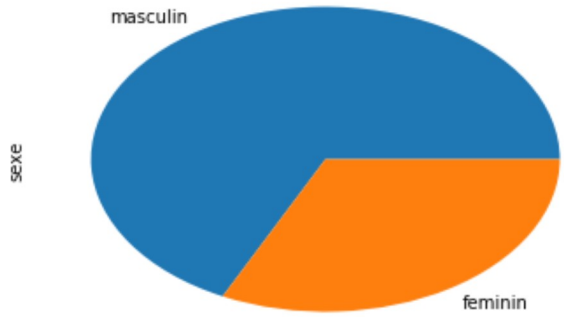
In [65]:
```
#calcul de la moyenne pour un vecteur
def moyenne(v):
    return(numpy.mean(v))

#grille à la carte de Kohonen où la couleur dépend de la moyenne de depression
df.plot.hexbin(x='age',y='tauxmax',C='depression',reduce_C_function=moyenne,grid
size=25)
```

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x1c473a8eeb8>



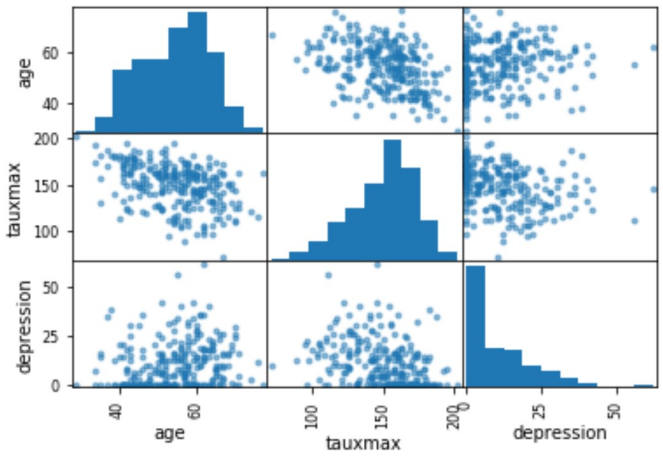
In [66]:
```
#diagramme à secteurs - comptage de sexe
df['sexe'].value_counts().plot.pie()
```

Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x1c474e7b898>



In [67]:
```
#l'équivalent de pairs de R - scatterplot des variables pris deux à deux
#cela n'a d'intérêt que pour les variables quantitatives bien évidemment
pandas.tools.plotting.scatter_matrix(df.select_dtypes(exclude=['object']))
```

Out[67]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001C474E8A4A8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001C474F8ACC0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001C474FF9208>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001C47504E908>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001C4750B4C18>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001C4750B4C50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001C47517B7B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001C4751D75C0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001C47523F710>]]
, dtype=object)
```



Conclusion

Les possibilités de Pandas sont vraiment immenses. Je comprends le succès de la librairie. Des ouvrages entiers lui sont consacrés. Le plus difficile finalement est d'explorer la documentation qui est particulière fournie, au point que l'on peut s'y perdre un peu parfois.

In [ ]:
```
```