Jupyter notebook 3 : facteurs individuels

Dans ce notebook, on va étudier le possibe impact sur le taux de suicide de facteurs qui prennent en compte l'état des individus, plus que de la société elle même.

Facteurs sélectionnés

- mortalité infantile
- chomage
- tabagisme
- alcoolisme
- consomation de drogues

On va chercher pour chaque facteur sa possible corélation avec le taux de suicide, pour à terme sélectionner les facteurs les plus représentatifs qu'on utilisera pour réaliser un modèle prédictif grace à sickyt learn.

Importation de la bibliothèque et des jeux de donné et des bibliotèques

In [9]:

```
import pandas as pd

dfMort = pd.read_csv('./Data/mortalite_moins5.csv', sep=',')
dfChom = pd.read_csv('./Data/chomage.csv', sep=',')
dfSmoke = pd.read_csv('./Data/smoking_part.csv', sep=',')
dfDrug = pd.read_csv('./Data/drug.csv', sep=',')
dfAlc = pd.read_csv('./Data/alcohol.csv', sep=',')

dfAS = pd.read_csv('./Data/age-standardized.csv', sep=',')
```

On crée un tableau regroupant l'ensemble de ces paramètres

In [10]:

```
dfMort = dfMort.drop('Country', axis = 1 )
df = dfChom.merge(dfMort, left_on = 'Country Code', right_on ='Country Cod
e')
df = df.merge(dfSmoke, left_on = 'Country Code', right_on ='Code')
df = df.drop('Code', axis = 1)
df = df.merge(dfDrug, left_on = 'Country Code', right_on ='Code')
df = df.drop('Code', axis = 1)
df = df.merge(dfAlc, left_on = 'Country Code', right_on ='Code')
df = df.drop('Code', axis = 1)
df = df.merge(dfAS, left_on = 'Country', right_on ='Country')
df = df.drop(103, axis = 0)
df = df.drop(59, axis = 0)
```

```
for i in range (4,df.shape[1]) :
    pd.to_numeric(df.iloc[:,i])
```

In [11]:

```
df = df.sort_index()
df.head(5)
```

Out[11]:

	Country	Country Code	Unemployement (%) 2016	Mortality under 5 for 1000 (2016)	Smoking (%)	Drug use disorders (%)	Alcohol (litres per capita)	sta su
0	Albania	ALB	15.220000	9.3	28.7	0.515036	7.5	
1	Algeria	DZA	10.202000	24.5	15.6	1.714025	0.9	
2	Argentina	ARG	8.016000	10.9	21.8	0.999099	9.8	
3	Armenia	ARM	17.617001	13.7	24.1	0.499895	5.5	
4	Australia	AUS	5.711000	3.8	14.7	2.237198	10.6	

Etude des facteurs

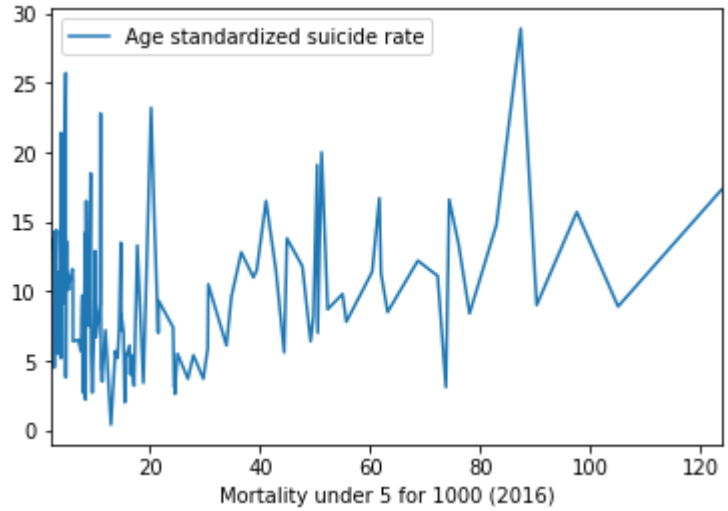
Maintenant, on réalise des courbes pour chercher d'éventuelles corrélations.

A chaque graphique, on affiche le taux de suicide en fonction du facteur pris en compte

Mortalité infantile

In [12]:

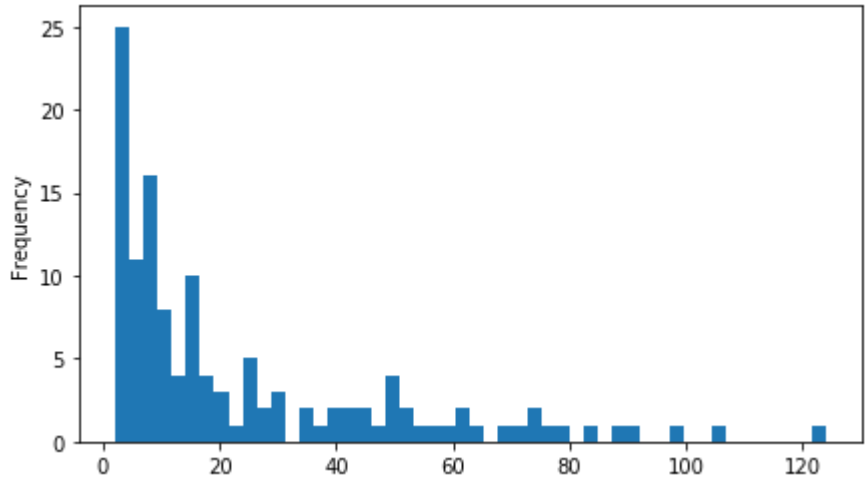
```
df = df.sort_values( by = 'Mortality under 5 for 1000 (2016)')
df.plot(x = 'Mortality under 5 for 1000 (2016)', y = 'Age standardized sui
cide rate');
```



Répartition en fréquence :

In [13]:

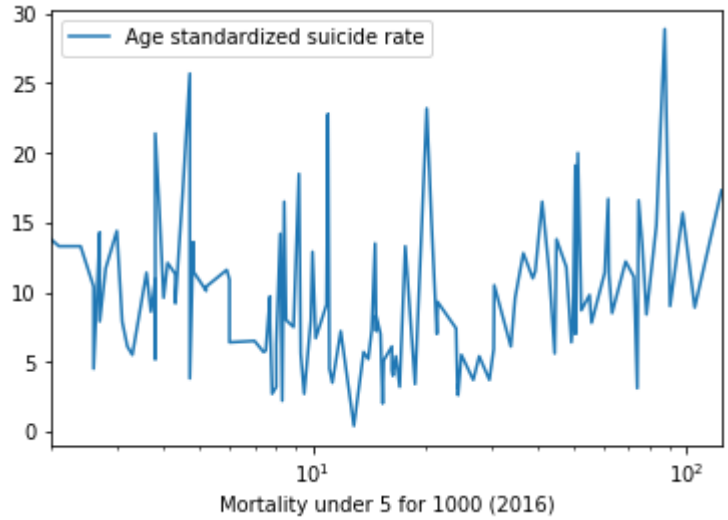
```
df.iloc[:,3].plot.hist( figsize=(7,4), bins = 50);
```



On observe que la majorité des pays possèdent une faible mortalité infantile. Cependant pour d'autres, la mortalité est dispropostionnée par rapport à la médiane, une échelle logarithmique est plus appropriée pour afficher les données.

In [14]:

```
df.plot(x = 'Mortality under 5 for 1000 (2016)', y = 'Age standardized sui
cide rate', logx = True);
```



Conclusion :

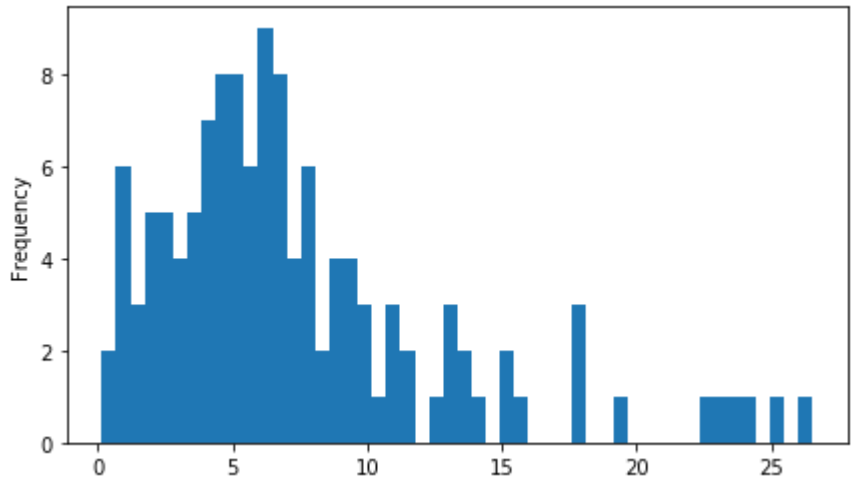
Pour le taux de mortalité infantile, on a une répartition inégales entre les pays. Beaucoup de pays ont un taux de mortalité faible (<10), tandis que les les autres pays voient leur taux augmenter de façon exponentielle. On observe difficilement un lien jusqu'à un taux de 20, mais au delà une tendance montante se dessine

On peut donc supposer une corélation entre en la mortalité infantile et le taux de suicide

Taux de chomage

In [15]:

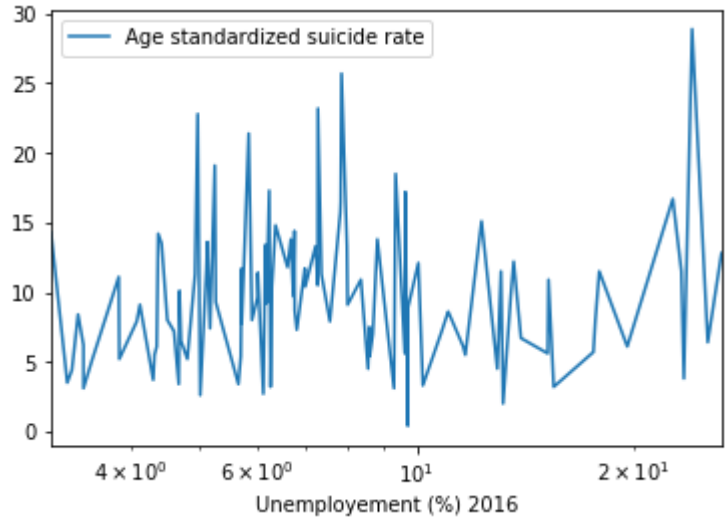
```
df.iloc[:,2].plot.hist( figsize=(7,4), bins = 50);
```



De même que précédemment, une échelle logarithmique est plus approrié en raison de la répartition des données

In [16]:

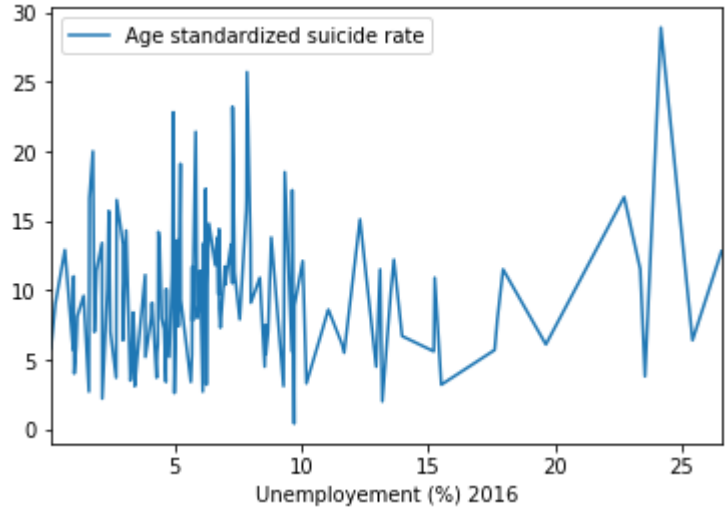
```
df = df.sort_values( by = 'Unemployement (%) 2016')
# On affiche pas les pays au taux de chomage inférieur à 3 (correspond à d
es pays en dévelopement à la population majoritairement rurale et vivrièr
e)
df[(df['Unemployement (%) 2016'] >= 3)].plot(x = 'Unemployement (%) 2016',
y = 'Age standardized suicide rate', logx = True );
```



Echelle non logaritmique

In [17]:

```
df.plot(x = 'Unemployement (%) 2016', y = 'Age standardized suicide rate'
);
```



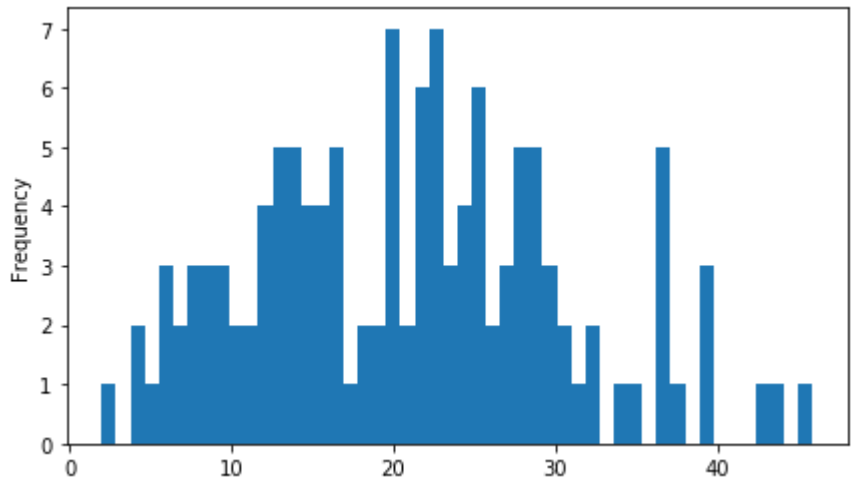
Conclusion :

Pour le taux de chomage, on observe aucune tendance. On pourrait extrapoler sur les valeurs extrèmes (taux de chomage supérieur à 10) en disant qu'une montée du taux de suicide est observable pour un très fort taux de chomage. Cependant le nombre de données sur cette plage est trop faible pour pouvoir affirmer quoi que ce soit.

Tabagisme

In [18]:

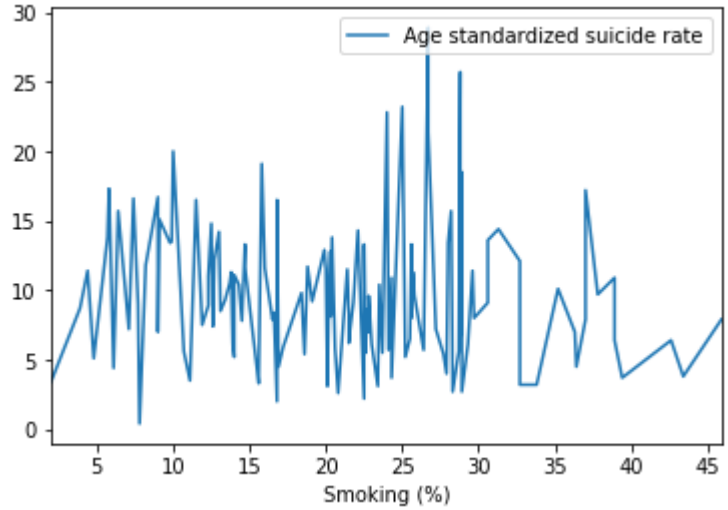
```
df.iloc[:,4].plot.hist( figsize=(7,4), bins = 50);
```



On obtient une répartition gaussienne, une échelle normale est donc adaptée

In [19]:

```
df = df.sort_values( by = 'Smoking (%)')
df.plot(x = 'Smoking (%)', y = 'Age standardized suicide rate');
```



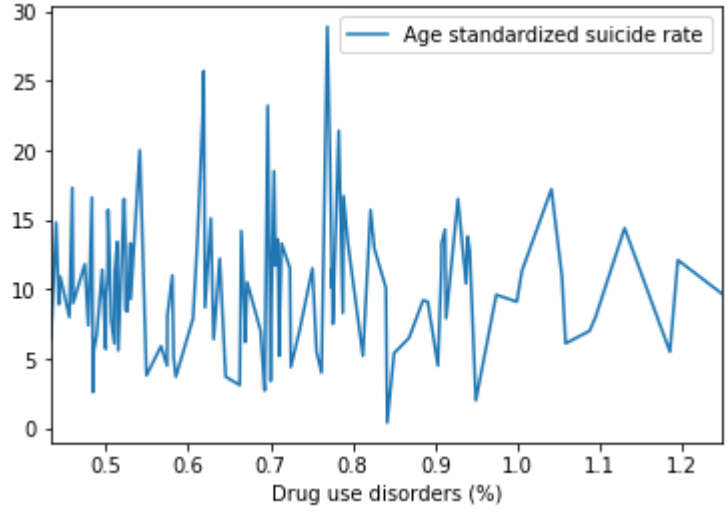
Conclusion

La courbe liant le taux de suicide et le tabagisme est relativement constante, on peut supposer qu'il n'y a pas de lien entre les deux.

Consomation de drogues

In [20]:

```
df = df.sort_values( by = 'Drug use disorders (%)')
# on n'affiche pas les valeurs extrèmes pour avoir une meilleur visibilité.
df[df['Drug use disorders (%)'] < 1.25].plot(x = 'Drug use disorders (%)',
y = 'Age standardized suicide rate');
```



Conclusion

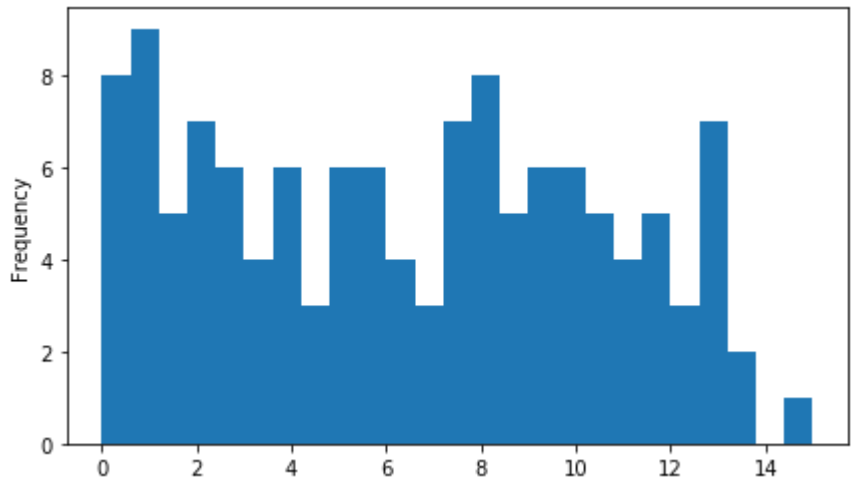
Rien non plus

.

Alcoolisme

In [21]:

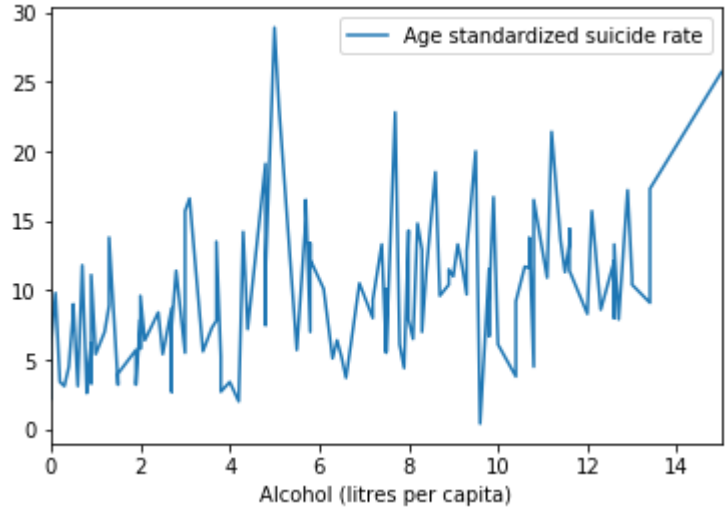
```
df.iloc[:,6].plot.hist( figsize=(7,4), bins = 25);
```



répartition homogène

In [22]:

```
df = df.sort_values( by = 'Alcohol (litres per capita)')
df.plot(x = 'Alcohol (litres per capita)', y = 'Age standardized suicide r
ate',);
```



Conclusion :

On observe clairement une tendance montante et linéaire entre la consomation d'alcool et le taux de suicide !

Modèle Prédictif

Maintenant, on va créer un modèle prédictif à partir des critères précédents

On utilise des modèles de prédiction par régression qui sont adaptés lorsqu'il faut réaliser une prédiction sur une valeur. On commence par utliser un modèle par arbre de décision pour déterminer

In [26]:

```
import numpy as np
import statistics as st
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import train_test_split
from sklearn import linear_model



from sklearn import tree
from sklearn import linear_model

#modele = linear_model.LinearRegression()
modele = tree.DecisionTreeRegressor()
#modele = linear_model.Lasso(alpha=0.1)

a = np.zeros((1000,5))

#X = df.iloc[0:,[4, 6]]
X = df.iloc[0:,2:7]
y = df.iloc[0:,7]

for i in range(0,1000) :

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
3)

    transformer = Normalizer().fit(X_train)  # fit does nothing.
    transformer
    Normalizer()
    transformer.transform(X_train)

    transformer = Normalizer().fit(X_test)  # fit does nothing.
    transformer
    Normalizer()
    transformer.transform(X_test)


    modele = modele.fit(X_train,y_train)

    for j in range(0,5) :

        a[i][j] = modele.feature_importances_[j]

poid = [0,0,0,0,0]

for i in range(0,5) :
    for j in range(0,1000) :
        poid[i] = poid[i] + (moy[j][i])
    poid[i] = poid[i] / 1000
poid
```

```
--------------
ValueError                                Traceback (most rec
ent call last)
<ipython-input-26-a5a7750df81e> in <module>
     29     transformer.transform(X_train)
     30
---> 31     transformer = Normalizer().fit(X_test)  # fit doe
s nothing.
     32     transformer
     33     Normalizer()

~\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py i
n fit(self, X, y)
   1721         X : array-like
   1722         """
-> 1723         check_array(X, accept_sparse='csr')
   1724         return self
   1725

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
check_array(array, accept_sparse, accept_large_sparse, dtype,
order, copy, force_all_finite, ensure_2d, allow_nd, ensure_mi
n_samples, ensure_min_features, warn_on_dtype, estimator)
    540         if force_all_finite:
    541             _assert_all_finite(array,
--> 542                                allow_nan=force_all_fi
nite == 'allow-nan')
    543
    544     if ensure_min_samples > 0:

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in
_assert_all_finite(X, allow_nan)
     54                 not allow_nan and not np.isfinite(X).
all()):
     55             type_err = 'infinity' if allow_nan else
'NaN, infinity'
---> 56             raise ValueError(msg_err.format(type_err,
X.dtype))
     57     # for object dtype data, we only check for NaNs
 (GH-13254)
     58     elif X.dtype == np.dtype('object') and not allow_
nan:

ValueError: Input contains NaN, infinity or a value too large
for dtype('float64').
```

Le calcul du poid de chaque facteur dans les choix de l'arbre de régression montre le facteur le plus important est le 5ème (alcool), et les 2 et 3èmes (mortalité et tabagisme) sont aussi très pris en compte. Ce qui rejoint nos observations précédentes.

Cependant le modèle par arbre de décision permet juste de déterminer l'importance de nos critères, mais n'est pas le plus efficaces (on a essayé). Celui qui donne le meilleur score est la régression linéaire dans lequel on entre seulement les critères présélectionnés:

In [25]:

```
import numpy as np
import statistics as st
from sklearn.preprocessing import Normalizer
```

```
from sklearn.model_selection import train_test_split
from sklearn import linear_model



from sklearn import tree
from sklearn import linear_model

modele = linear_model.LinearRegression()
#modele = tree.DecisionTreeRegressor()
#modele = linear_model.Lasso(alpha=0.1)

moy = []

X = df.iloc[0:,[3,4, 6]]
#X = df.iloc[0:,2:7]
y = df.iloc[0:,7]

for i in range(0,1000) :

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
3)

    transformer = Normalizer().fit(X_train)  # fit does nothing.
    transformer
    Normalizer()
    transformer.transform(X_train)

    transformer = Normalizer().fit(X_test)  # fit does nothing.
    transformer
    Normalizer()
    transformer.transform(X_test)


    modele = modele.fit(X_train,y_train)

    moy.append(modele.score(X_test,y_test))

st.mean(moy)
```

Out[25]:

```
0.24686637750126825
```

In [ ]: