Introduction à Python

Fabien Dupuis

Table des matières

Chapitre 1

Structure de base numérique

Les notebooks du cours sont accessibles dans le zip accessible sur le lien suivant : https://1drv.ms/u/s!Am09h0q2OIX0bJxMkXN1DDmMIY4?e=P0n2zz.

Dans ce chapitre on considére que les listes, les tuples et les fonctions sont maitrisés. Le but de ce chapitre est

- introduire les packages numpy, scipy, matplotlib.
- construire des matrices et des vecteurs
- faire des simulations
- tracer des graphiques

1.1 Introduction à la librairie numpy : Matrices et vecteurs en python

On va créer la matrice suivante :

$$A = \begin{pmatrix} 1.1 & 2.6 & 5 \\ 3 & 3.6 & 9 \\ 5.5 & 4 & 4.9 \end{pmatrix}$$

```
In [1]: import numpy as np
        A = np.array([[1.1,2.6,5],[3,3.6,9],[5.5,4,4.9]])
        A
```

```
Out[1]: array([[1.1, 2.6, 5. ],
               [3. , 3.6, 9. ],
               [5.5, 4. , 4.9]])
```

On détermine les dimensions de A :

```
In [2]: A.shape
```

```
Out[2]: (3, 3)
```

On va céer le vecteur suivant : B = [1.2,3.8,5.9] :

```
In [3]: import numpy as np
        B = np.array([1.2,3.8,5.9])
        B
```

```
Out[3]: array([1.2, 3.8, 5.9])
```

On détermine les dimensions de B :

```
In [4]: B.shape
```

```
Out[4]: (3,)
```

Voici quelques fonctions utiles en statistique :

- np.sum : somme d'array
- np.cumsum : somme cumulée d'un array
- np.var : variance
- np.mean : moyenne
- np.size : taille

On utilise la fonction np.var sur la matrice A. Les autres fonctions s'utilisent de la même façon. Variance de chaque colonne :

```
In [7]: np.var(A,axis=0)
```

```
Out[7]: array([3.24666667, 0.34666667, 3.64666667])
```

```
In [0]: Variance de chaque ligne:
```

```
In [8]: np.var(A,axis=1)
```

```
Out[8]: array([2.58, 7.28, 0.38])
```

```
In [9]: np.var(A)
```

```
Out[9]: 4.42
```

Exercice 1

Calculer la moyenne sur chaque ligne, sur chaque colonnes de la matrice :

$$A = \begin{pmatrix} 5 & 2.6 & 5 & 9 \\ 4 & 3.6 & 9 & 10 \\ 5.5 & 4 & 6 & 7 \end{pmatrix}$$

1.2 Simulation avec Python

Le package numpy de Python permet de réaliser des objets array simulés.

- numpy.random.uniform(low=a, high=b, size=None) : simulation uniforme sur [a,b]
- numpy.random.normal(loc=a, scale=b, size=None) : simulation d'une va normal de moyenne a et d'écart type b
- numpy.random.poisson : simulation d'une va de poisson

On va simuler une matrice avec numpy.random.uniform :

```
In [16]: import numpy as np
         Uni = np.random.uniform(2,6,size=(3,2))
```

On va simuler une matrice et un vecteur avec numpy.random.normal :

```
In [19]: import numpy as np
         Norm = np.random.normal(2,3,size=(2,3))
         Norm
```

```
Out[19]: array([[-1.80259786,  3.91781508,  1.25704838],
                [ 4.64228033,  5.23646808, -0.12003757]])
```

```
In [22]: Nve = np.random.normal(6,2,size=5)
         Nve
```

```
Out[22]: array([7.05764819, 4.36788279, 5.63441866, 5.91012783, 8.30667172])
```

On va simuler le vecteur de probabilité v=[1/4,1/5,1/5,1-1/4-2*1/5] sur les états [a,b,c,d] :

```
In [24]: import numpy as np
         proba = np.array([1/4,1/5,1/5,1-1/4-2*1/5])
         etat = np.array(['a','b','c','d'])
         Simu = np.random.choice(etat, size=20, replace=True, p=proba)
         Simu
```

```
Out[24]: array(['a', 'c', 'a', 'd', 'c', 'a', 'c', 'b', 'd', 'b', 'b', 'b', 'a',
                'c', 'a', 'd', 'd', 'd', 'a', 'd'], dtype='<U1')
```

Exercice 2

Simuler un vecteur de probabilité sur l'ensemble des états ['voit','velo','trot'] dont les probabilitées de chaque état sont [1/6 , 2/6 , 3/6]. Simuler ensuite un objet numpy.array de taille 10 issu d'une loi normal de moyenne 4 et variance 10.

1.3 Fonctions particulières sur les objet array

On présente des fonctions permettant la modification, la construction d'objet array.

- numpy.arange([start,]stop, [step,]dtype=None) :
- numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)
- numpy.insert(arr, obj, values, axis=None)
- numpy.append(arr, values, axis=None)
- numpy.concatenate((a1, a2, ...), axis=0, out=None)
- numpy.stack(arrays, axis=0, out=None)
- numpy.ones

On construit V1 un vecteur comprenant des nombres compris entre 2 et 8 par pas de 2.

```
In [6]: import numpy as np
        V1 = np.arange(2,10,2)
        V1
```

```
Out[6]: array([2, 4, 6, 8])
```

On construit un vecteur V2 comprenant 5 valeurs comprises entre 6 et 8 :

```
In [11]: import numpy as np
         V2 = np.linspace(6,8,5)
         V2
```

```
Out[11]: array([6. , 6.5, 7. , 7.5, 8. ])
```

```
In [0]: Ici, on concatène V1 et V2:
```

```
In [12]: np.concatenate([V1,V2])
```

```
Out[12]: array([2. , 4. , 6. , 8. , 6. , 6.5, 7. , 7.5, 8. ])
```

On va concaténer une matrice (3,3) avec une matrice (3,2) selon l'axe horizontal :

```
In [14]: import numpy as np
         M1 = np.array([[1,2,6],[4,5,6],[8,8,8]])
         M2 = np.array([[6,5],[4,4],[2,1]])
         np.concatenate([M1,M2],axis=1)
```

```
Out[14]: array([[1, 2, 6, 6, 5],
                [4, 5, 6, 4, 4],
                [8, 8, 8, 2, 1]])
```

On va créer un vecteur ayant que des 1 excepté son 9 ième élément qui vaut 15 :

```
In [19]: V3 = np.ones(15)
         np.insert(V3,9,15)
```

```
Out[19]: array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1., 15.,  1.,  1.,  1.,
                 1.,  1.,  1.])
```

```
In [0]: On va utiliser append pour
```

```
In [25]: import numpy as np
         V1 = np.array([5,8,9])
         V2 = np.array([9,9,5,6.8])
         np.append(V1,V2)
```

```
Out[25]: array([5. , 8. , 9. , 9. , 9. , 5. , 6.8])
```

On va créer une matrice ayant 5 fois la même ligne et 1 ligne différente :

```
In [27]: import numpy as np
         V1 = np.array([[1,4,2.3,9]])
         V2 = np.array([[1,9.9,5.2,1]])
         np.concatenate([V1,V1,V2,V1,V1,V1],axis=0)
```

```
Out[27]: array([[1. , 4. , 2.3, 9. ],
                [1. , 4. , 2.3, 9. ],
                [1. , 9.9, 5.2, 1. ],
                [1. , 4. , 2.3, 9. ],
                [1. , 4. , 2.3, 9. ],
                [1. , 4. , 2.3, 9. ]])
```

Exercice 3

Créer la matrice suivante en utilisant np.concatenate, np.one, np.insert :

$$\begin{pmatrix} 2 & 2 & 5 & 2 & 2 \\ 2 & 2 & 5 & 2 & 2 \\ 2 & 2 & 5 & 2 & 2 \\ 2 & 2 & 5 & 2 & 2 \\ 2 & 2 & 5 & 2 & 2 \\ 2 & 2 & 5 & 2 & 2 \end{pmatrix}$$

```
In [10]: import numpy as np
         V1 = 2*np.ones(4,dtype=int)
         V2 = np.insert(V1,2,5)
         V2 = V2.reshape(1,5)
         M = np.concatenate([V2]*6,axis=0)
         M
```

```
Out[10]: array([[2, 2, 5, 2, 2],
                [2, 2, 5, 2, 2],
                [2, 2, 5, 2, 2],
```

```
       [2, 2, 5, 2, 2],
       [2, 2, 5, 2, 2],
       [2, 2, 5, 2, 2]])
```

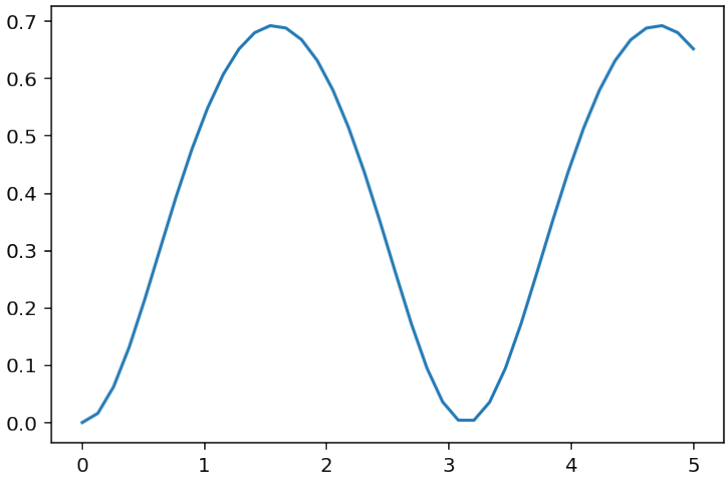
1.4 Courbe de fonction

La package numpy donne accés notamment aux fonctions suivantes numpy.cos, numpy.sin, numpy.log, numpy.exp.

On va tracer la courbe de$f(x) = \ln(1 + (sin(x))^2)$ sur [0,5]. On utilise le package matplotlib.pyplot pour tracer cette courbe.

```
In [11]: import numpy as np
         x = np.linspace(0,5,40)
         y = np.log(1+np.sin(x)**2)

         import matplotlib.pyplot as plt
         plt.plot(x,y)
         plt.show()
```

Out[11]:



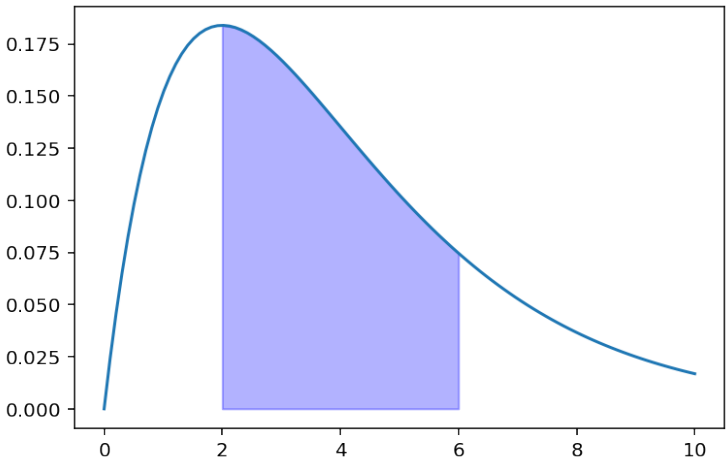
1.5 Exemple de courbe avec fonction spéciale

— Tracer la courbe de la fonction $f(x) = \frac{1}{4\Gamma(2)} x e^{-x/2}$ ou $\Gamma(t) = \int_0^{+\infty} x^{t-1} e^{-x} dx$ sur [0,10]. On utilise la fonction gamma fournit par package scipy.
— Mettre en évidence l'aire sous la courbe entre 2 et 6.
— Calculer $\int_2^4 \frac{1}{4\Gamma(2)} x e^{-x/2} dx$

```
In [1]: from scipy.special import gamma
        import numpy as np
        import matplotlib.pyplot as plt
```

```
x = np.linspace(0,10,100)
y = (1/(4*gamma(2)))*x*np.exp(-x/2)
plt.plot(x,y)
x1 = np.linspace(2,6,20)
y1 = (1/(4*gamma(2)))*x1*np.exp(-x1/2)
plt.fill_between(x1,y1, color='blue',alpha=0.3)
plt.show()
```

Out[1]:



On calcul l'integrale à laide d'une somme de Riemann : $\frac{b-a}{n}\sum_{i=0}^{n} f(a+i(b-a)/n)$

In [7]:
```
x = np.linspace(2,4,100)
y = (1/(4*gamma(2)))*x*np.exp(-x/2)
((4-2)/1000)*np.sum(y)
```

Out[7]: 0.0329647113167648

On verifie les résultats. On remarque l'utilisation du package scipy.

In [9]:
```
def f(x):
    y= (1/(4*gamma(2)))*x*np.exp(-x/2)
    return(y)

from scipy.integrate import quad
quad(f,2,4)
```

Out[9]: (0.32975303263304656, 3.660994092691789e-15)

Exercice 4

Soit $f(x) = \frac{1}{\Gamma(3)}x^2e^{-x}$. Tracer la courbe de f entre 0 et 20. Déssiner l'aire sous la courbe entre 10 et 15. Calculer $\int_{10}^{15} f(x)dx$.

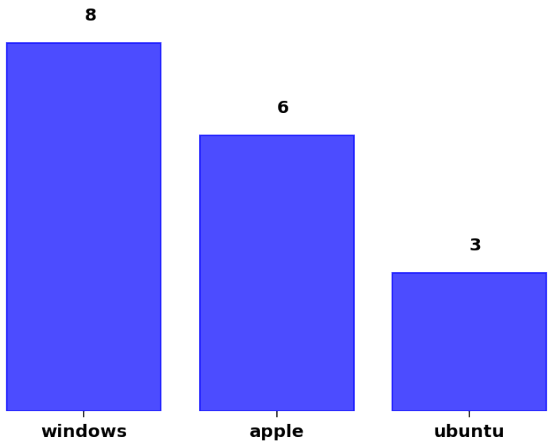
1.6 Barplot et histogramme

Exemple de Barplot

Parmi un ensemble de 17 étudiants, 8 sont équipés d'ordinateur windows, 6 d'apple et 3 d'ubuntu. On va représenter ces données avec un barplot.

In [23]:
```
import numpy as np
import matplotlib.pyplot as plt


#les noms de bars
bars = np.array(['windows','apple','ubuntu'])
y_pos = np.arange(len(bars))

#la hauteur des bars
height = np.array([8,6,3])


barlist=plt.bar(y_pos, height,edgecolor='blue',alpha=0.
↪7,color=['blue','blue','blue']) #
plt.xticks(y_pos,bars,color='black',fontweight='bold')
#plt.tick_params(top='off', bottom='off', left='off', right='off',␣
↪labelleft='off', labelbottom='on')
#for spine in plt.gca().spines.values():
    #spine.set_visible(False)
plt.box(False)
plt.yticks([])
for i in range(0,len(height)):
    plt.text(y_pos[i],height[i]+0.
↪5,str(height[i]),color='black',fontweight='bold')

plt.show()
```

Out[23]:



Exercice 5

Dans la liste python ci dessous on 5 modalitées : A,B,C,D. Représenter ces données à l'aide d'un diagramme en bar. Chaque bar représente un pourcentage.

In [30]:
```
#executer la cellule
import numpy as np
```

```
np.random.seed(1998)

proba = [0.1,0.2,0.3,0.4]
etat = np.array(['A','B','C','D'])
Simu = np.random.choice(etat, size=1000, replace=True, p=proba)
Simu[0:6]
```

Out[30]: array(['C', 'D', 'B', 'D', 'C', 'D'], dtype='<U1')

Exemple d'histogramme

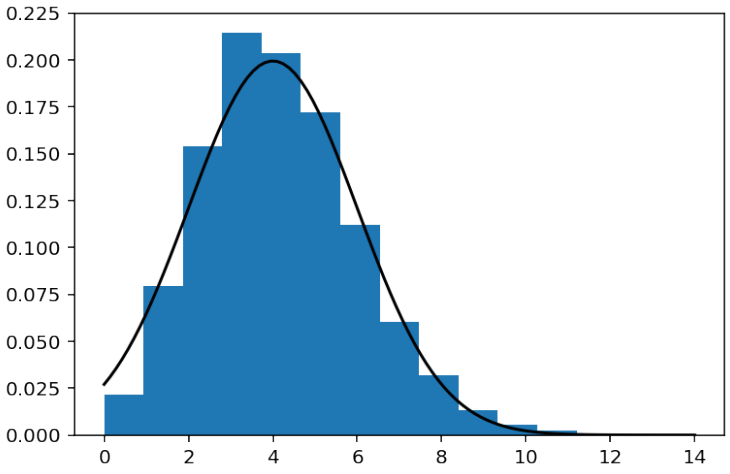
- Simuler un échantillon de Poisson de paramètre 4 et de taille 10 000.
- Tracer son histogramme.
- Tracer dans la même figure la courbe $f(x) = \frac{1}{2\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-4}{2}\right)^2}$ entre 0 et 15.

On simule l'échantillon de loi de Poisson de paramètre 4 et de taille 10 000.

```
In [3]: import numpy as np
        np.random.seed(1998)
        pois = np.random.np.random.poisson(4, 10000)
```

On trace l'histogrammme en utilisant plt.hist. Ensuite on trace la courbe

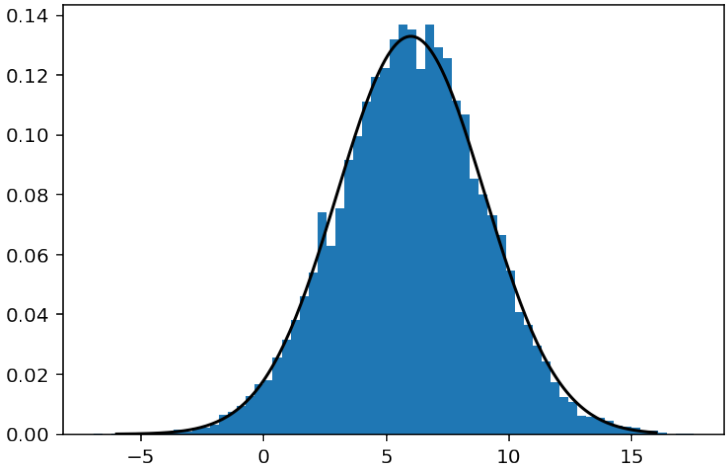
```
In [17]: import numpy as np
         import matplotlib.pyplot as plt
         plt.hist(pois,bins='sturges',density=True)  #sturges
         x = np.linspace(0,14,100)
         y = (1/(2*np.sqrt(2*np.pi)))*np.exp(-(1/2)*((x-4)/2)**2)
         plt.plot(x,y,color='black')
         plt.show()
```



Exercice 6 : histogramme

- Simuler un échantillon de taille 10000 de loi normal de moyenne 6 et d'écart type 3.
- Faire l'histogramme de cet échantillon en utilisant plt.hist avec l'option density=True et bins='sturges' ou 'auto'.
- Tracer la courbe $f(x) = \frac{1}{\sqrt{2\pi \times 9}} e^{-\frac{1}{2}\left(\frac{x-6}{3}\right)^2}$ sur le même graphique dans l'interval [-6,16].

Voici un graphique possible que l'on peut obtenir.



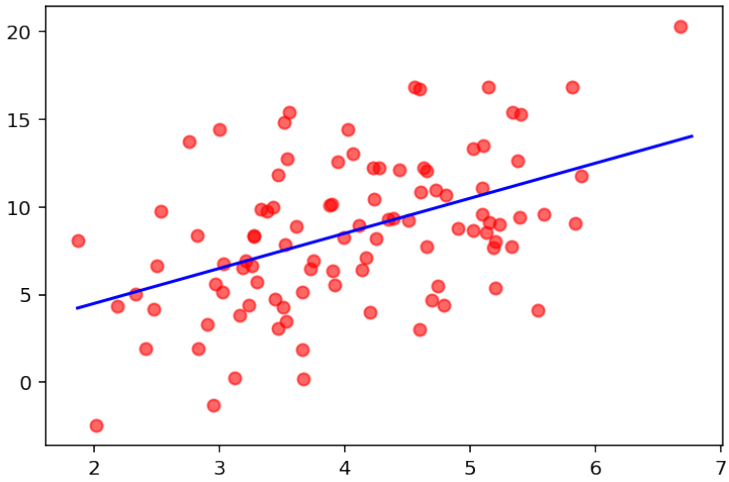
1.7 Nuage de points

On va simuler un nuage de point provenant d'un modèle linéaire Y = 0.5 + 2X. On trace le nuage de point à l'aide de la fonction plt.scatter. Ensuite on trace la droite d'équation Y = 0.5 + 2X sur le même grpahique.

In [2]:
```
import numpy as np
np.random.seed(seed=1998)
x = np.random.normal(loc=4,scale=1,size=100)
e = np.random.normal(loc=0,scale=4,size=100)
y = 0.5+2*x+e

M = max(x)
m = min(x)
x_trend = np.arange(m,M+0.1,0.1)
y_trend = 0.5+2*x_trend
```

In [3]:
```
#On trace le nuage de point
import matplotlib.pyplot as plt
plt.scatter(x,y,color="red",alpha=0.6)
plt.plot(x_trend,y_trend,color="blue")
plt.show()
```

Out[3]:



1.8 Exemple d'equation différentiel stochastique

Soit l'EDS $dX_t = a(X_t)dt + b(X_t)dW_t$, a est le terme drift, W_t est un processus de Winner et X_t le terme de diffusion. On va résoudre cette équation en utilisant un schéma d'Euler Maruyama.

— On discrétise le temps en N intervalles de longueur Δt

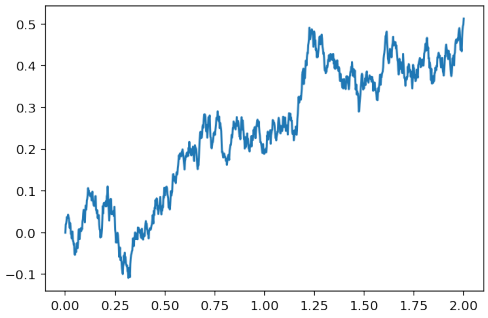
— $Y_n = Y_{n-1} + a(Y_n)\Delta t + b(Y_{n-1})\Delta W$ ou $\Delta W \sim \mathcal{N}(0, \Delta t)$

Soit l'EDS, $dX_t = \Theta(\mu - X_t)dt + \sigma dW_t$. voici un code de résolution avec $\Theta = 1.1, \mu = 0.8, \sigma = 0.3$ sur l'intervalle de temps $[0, 2]$ en 1000 pas.

```
In [7]: import numpy as np
        import matplotlib.pyplot as plt

        t_0 = 0 # define model parameters
        t_end = 2
        length = 1000
        theta = 1.1
        mu = 0.8
        sigma = 0.3

        t = np.linspace(t_0,t_end,length) # define time axis
        dt = np.mean(np.diff(t))

        y = np.zeros(length)
        y0 = np.random.normal(loc=0.0,scale=1.0)

        drift = lambda y,t: theta*(mu-y) # define drift term, google to learn about␣
↪lambda
        diffusion = lambda y,t: sigma # define diffusion term
        noise = np.random.normal(loc=0.0,scale=1.0,size=length)*np.sqrt(dt)

        for i in range(1,length):
            y[i] = y[i-1] + drift(y[i-1],i*dt)*dt + diffusion(y[i-1],i*dt)*noise[i]
```

```
        plt.plot(t,y)
        plt.show()
```

Out[7]:

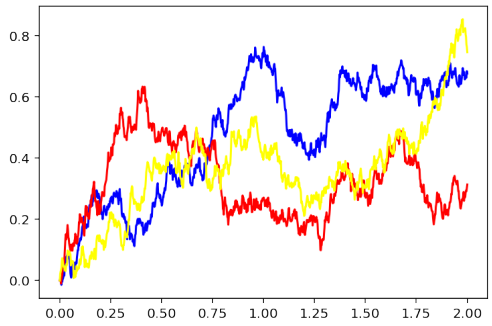


In [0]: On modifie le code pour avoir plusieurs réalisations:

In [11]:
```
        import numpy as np
        import matplotlib.pyplot as plt

        t_0 = 0 # define model parameters
        t_end = 2
        length = 1000
        theta = 1.1
        mu = 0.8
        sigma = 0.3

        t = np.linspace(t_0,t_end,length) # define time axis
        dt = np.mean(np.diff(t))

        y = np.zeros(length)
        #y0 = np.random.normal(loc=0.0,scale=1.0)
        y0='R'
        ybis = np.zeros(length)
        ytier = np.zeros(length)

        drift = lambda y,t: theta*(mu-y) # define drift term, google to learn about␣
↪lambda
        diffusion = lambda y,t: sigma # define diffusion term
        noise = np.random.normal(loc=0.0,scale=1.0,size=length)*np.sqrt(dt)
        noisebis = np.random.normal(loc=0.0,scale=1.0,size=length)*np.sqrt(dt)
        noisetier = np.random.normal(loc=0.0,scale=1.0,size=length)*np.sqrt(dt)

        for i in range(1,length):
            y[i] = y[i-1] + drift(y[i-1],i*dt)*dt + diffusion(y[i-1],i*dt)*noise[i]
            ybis[i] = ybis[i-1] + drift(ybis[i-1],i*dt)*dt +␣
↪diffusion(ybis[i-1],i*dt)*noisebis[i]
            ytier[i] = ytier[i-1] + drift(ytier[i-1],i*dt)*dt +␣
↪diffusion(ytier[i-1],i*dt)*noisetier[i]

        plt.plot(t,y,color='blue')
```

```
plt.plot(t,ybis,color='red')
plt.plot(t,ytier,color='yellow')
plt.show()
```

Out[11]:



Exercice 7

Résoudre une équation différentielle stochastique comme dans l'exemple précédent. Vous devrez indiquer clairement le schéma de résolution utilisé à l'aide de latex.

1.9 Exercice de synthése

Exercice 8

Le poids des hommes suit une loi normal de paramètre de moyenne 77.4kg et d'écart type 12kg. Le poids des femmes suit une loi normal de paramètre de moyenne 62.4 et d'écart type 10.9. En France, il y a 32 455 859 hommes pour 34 534 967 femmes au 1er janvier 2017. Une compagnie maritime organise en Corse des expéditions pouvant accueillir 100 personnes par bateau. Selon les normes de sécurité en vigueur, un bateau ne peut accueillir une charge dépassant les 7.2 tonnes. A l'aide d'une simulation, calculer le risque que cette normes ne soient pas respectées ? (On ferra l'hypothèse que les touristes sont adultes et voyagent sans bagage)

Chapitre 2

DATAFRAME PYTHON

Les notebooks du cours sont accessibles dans le zip accessible sur le lien suivant : https://1drv.ms/u/s!Am09h0q2OIX0bJxMkXN1DDmMIY4?e=P0n2zz.

2.1 Création de dataframe

Les dataframes python sont traités avec la librairie pandas. On peut créer des dataframes de plusieurs façon :

- à partire d'un dictionnaire
- à partir de liste
- à partire d'un fichier json
- à partire d'une matrice numpy.array
- à partire d'une ou plusieurs séries pandas
- à partir d'un ou plusieur dataframe pandas
- à partire de fichier importé : fichier de type csv, excel, txt, json, page HTML (webscrapping), xml

Les dataframes python sont composés

- d'un index perméttant d'identifier les lignes
- d'un ensemble de colonnes
- des données

Chaque colonne d'un dataframe est un objet de type pandas.Series.

2.1.1 Exemple (création d'un dataframe à partir d'un dictionnaire)

On va créer un dataframe à partir d'un dictionnaire. Ce dataframe sera le tableau suivant :

	*COL*0	*COL*1	*COL*2
*id*0	*voiture*	16	52
*id*1	*vlo*	18	44
*id*2	*moto*	24	23
*id*3	*voiture*	44	11
*id*4	*moto*	10	32
*id*5	*vlo*	3	8

[19]:
```
import pandas as pd
Data1 = pd.DataFrame({'COL0':['voiture','vélo','moto','voiture','moto','vélo'],
                      'COL1':[16,18,24,44,10,3],'COL2':[52,44,23,11,32,8]},\
                     index=['id0','id1','id2','id3','id4','id5'])
```

[13]:
```
Data1
```

[13]:
```
        COL0  COL1  COL2
id0  voiture    16    52
id1     vélo    18    44
id2     moto    24    23
id3  voiture    44    11
id4     moto    10    32
id5     vélo     3     8
```

[14]:
```
type(Data1['COL0'])
```

[14]:
```
pandas.core.series.Series
```

2.1.2 Exemple (création d'un dataframe à partir d'une liste)

On va créer un dataframe à partir d'une liste de liste. Ce dataframe sera le tableau suivant :

	COL0	*COL1*	*COL2*	*COL3*
l0	*Paris*	16	52	55
l1	*Grenoble*	18	44	11
l2	*Nancy*	24	23	44
l3	*Dijon*	44	11	12
l4	*Grenoble*	10	32	71

[8]:
```
import pandas as pd
Data2 = pd.DataFrame([['Paris',16,52,55],['Grenoble',18,44,11],['Nancy',24,23,44],\
                      ['Grenoble',10,32,71]],\
                     columns=['COL0','COL1','COL2','COL3'],index=['l0','l1','l2','l3'])
Data2.head(2)
```

[8]:
```
        COL0  COL1  COL2  COL3
l0     Paris    16    52    55
l1  Grenoble    18    44    11
```

2.1.3 Exemple (création d'un dataframe à partir d'un ensemble de série pandas)

On reprend le dataframe précédent. On va créer un dataframe en concaténant 4 series pandas :

[7]:
```
import pandas as pd

S0 = pd.Series(['Paris', 'Grenoble','Nancy','Dijon','Grenoble'], \
               index=['l0', 'l1','l2','l3','l4'], name='COL0')
S1 = pd.Series([16, 18,24,44,10], index=['l0', 'l1','l2','l3','l4'], name='COL1')
S2 = pd.Series([52, 44,23,11,32], index=['l0', 'l1','l2','l3','l4'], name='COL2')
```

```
S3 = pd.Series([55, 11,44,12,72], index=['l0', 'l1','l2','l3','l4'], name='COL3')
Data3 = pd.concat([S0,S1,S2,S3],axis=1)
```

2.1.4 Exemple : création d'un dataframe à partir d'un np.array

[10]:
```
import numpy as np
ar = np.array([[1.1, 2, 3.3, 4], [2.7, 10, 5.4, 7], [5.3, 9, 1.5, 15]])
df = pd.DataFrame(ar, index = ['a1', 'a2', 'a3'], columns = ['A', 'B', 'C', 'D'])
df.head(2)
```

[10]:

	A	B	C	D
a1	1.1	2.0	3.3	4.0
a2	2.7	10.0	5.4	7.0

2.1.5 Exercice

Créer le dataframe suivant de de 2 ou 3 manières différentes.

	Classe1	Classe2	Classe3	classe4
machine learning	10	9	4	8
stochastic integral	15	15	9	4
python	20	24	23	16
LaTeX	6	44	11	44
c++	12	10	32	11

2.2 Colonne, index, ligne, dimension, filtre

Dans cette section on montre comment s'informer sur les colonnes, les lignes, les dimensions du tableau. Nous donnons dans le tableau quelques attributs des dataframes :

columns	The column labels of the DataFrame.
index	The index (row labels) of the DataFrame.
shape	Return a tuple representing the dimensionality of the DataFrame.
dtypes	Return the dtypes in the DataFrame.
iloc	Purely integer-location based indexing for selection by position.
loc	Access a group of rows and columns by label(s) or a boolean array.
values	Return a Numpy representation of the DataFrame.

2.2.1 Exemple

[3]:
```
import pandas as pd

S0 = pd.Series(['Paris', 'Grenoble','Nancy','Dijon','Grenoble'], \
               index=['l0', 'l1','l2','l3','l4'], name='COL0')
S1 = pd.Series([16, 18,24,44,10], index=['l0', 'l1','l2','l3','l4'], name='COL1')
S2 = pd.Series([52, 44,23,11,32], index=['l0', 'l1','l2','l3','l4'], name='COL2')
S3 = pd.Series([55, 11,44,12,72], index=['l0', 'l1','l2','l3','l4'], name='COL3')
Data3 = pd.concat([S0,S1,S2,S3],axis=1)
```

```
Data3
```

[3]:

	COL0	COL1	COL2	COL3
l0	Paris	16	52	55
l1	Grenoble	18	44	11
l2	Nancy	24	23	44
l3	Dijon	44	11	12
l4	Grenoble	10	32	72

1) On détermine le nom des colonnes du dataframe que l'on stocke dans l'objet DATA3_COL

[8]:
```
DATA3_COL = Data3.columns
DATA3_COL
```

[8]:
```
Index(['COL0', 'COL1', 'COL2', 'COL3'], dtype='object')
```

2) On détermine l'index du dataframe Data3. L'index représente les lignes d'un dataframe

[10]:
```
DATA3_ROW = Data3.index
DATA3_ROW
```

[10]:
```
Index(['l0', 'l1', 'l2', 'l3', 'l4'], dtype='object')
```

3) On extrait la ligne 3 de Data3 de deux façons différentes.

[14]:
```
#première méthode
Data3.loc['l3'].values
```

[14]:
```
array(['Dijon', 44, 11, 12], dtype=object)
```

[13]:
```
#deuxième façon
Data3.iloc[3,:].values
```

[13]:
```
array(['Dijon', 44, 11, 12], dtype=object)
```

4) On détermine les dimensions de Data3

[19]:
```
dim = Data3.shape
print(str(dim))
nb_ligne = dim[0]
nb_col = dim[1]

print("nombre de ligne:" + str(nb_ligne))
print("nombre de colonne:" + str(nb_col))
```

```
(5, 4)
nombre de ligne:5
nombre de colonne:4
```

5) On crée un dataframe Data3bis en filtrant sur les lignes tel que COL1>10 et COL2>11

[25]:
```
Data3bis = Data3.loc[(Data3['COL1']>10) & (Data3['COL2']>11)]
Data3bis
```

[25]:

	COL0	COL1	COL2	COL3
10	Paris	16	52	55
11	Grenoble	18	44	11
12	Nancy	24	23	44

6) On crée un dataframe Data3bis_comp complémentaire de Data3bis

[27]:
```
Data3bis_comp = Data3.loc[~((Data3['COL1']>10) & (Data3['COL2']>11))]
Data3bis_comp
```

[27]:

	COL0	COL1	COL2	COL3
13	Dijon	44	11	12
14	Grenoble	10	32	72

7) On crée un dataframe Data3tier en filtrant sur les lignes de Data3 tel que COL0 soit égale à Nancy ou Dijon.

[29]:
```
Data3tier = Data3.loc[Data3['COL0'].isin(['Nancy','Dijon'])]
```

2.2.2 Exercice

— 1) Déterminer le nom des colonnes du dataframe data_house
— 2) Déterminer l'index du datframe data_house
— 3) Extraire la ligne 100 de deux façons
— 4) Trouver le nombre de lignes que telle data_house['CRIM']>3.6 et data_house['AGE']>68.
— 5) Changer l'index de data_house. Cet index doit être comme cela : ['l0', 'l1', 'l2', 'l3', 'l4', 'l5', 'l6', 'l7', 'l8', 'l9',...,'l501', 'l502', 'l503', 'l504','l505']. Extraire la ligne 200

Construction du dataframe data_house.

[58]:
```
import numpy as np
import pandas as pd

from sklearn.datasets import load_boston
data_build = load_boston()
#print(data_build.feature_names)
#print(data_build.DESCR)

datavect=np.concatenate((data_build.data,data_build.target.reshape(506,1)),axis=1)
name_v=list(data_build.feature_names)+['MEDV']

data_house = pd.DataFrame(datavect,columns=name_v)
```

2.3 Création colonne, fonction

[1]:
```
import pandas as pd
import numpy as np
Data1 = pd.DataFrame({'COL0':['voiture','vélo','moto','voiture','moto','vélo'],
                      'COL1':[16,18,24,44,10,3],'COL2':[52,44,23,11,32,8],\
                     'COL3':[np.pi,2*np.pi,1.5*np.pi,1.2*np.pi,5.7*np.pi,8.14*np.pi]},\
                     index=['id0','id1','id2','id3','id4','id5'])
```

2.3.1 Exemple

On créer une colonne Data1['new1'] qui est la somme de Data1['COL1']+Data1['COL3'] :

[75]:
```
Data1['new1'] = Data1['COL1'] + Data1['COL3']
Data1.head(3)
```

[75]:

	COL0	COL1	COL2	COL3	new1
id0	voiture	16	52	3.141593	19.141593
id1	vélo	18	44	6.283185	24.283185
id2	moto	24	23	4.712389	28.712389

2.3.2 Exemple

On créer une colonne Data1['COS'] qui est le cosinus de Data1['COL3']. On utilise la fonction cos du package numpy.

[82]:
```
import numpy as np
Data1['COS'] = np.cos(Data1['COL3'])
Data1['COS'].astype(str)
```

[82]:
```
id0     -1.0
id1      1.0
id2    -1.83
id3    -0.80
id4    0.587
id5    0.904
Name: COS, dtype: object
```

2.3.3 Exemple

On créer une colonne Data1['MAX'] qui est le maximum des colonnes Data1['COL1'], Data1['COL2'], Data1['COL3']. Vous remarquerez l'emploi de axis=1 dans la fonction max.

[85]:
```
Data1['MAX'] = Data1[['COL1','COL2','COL3']].max(axis=1)
Data1.head(3)
```

[85]:

	COL0	COL1	COL2	COL3	new1	COS	MAX
id0	voiture	16	52	3.141593	19.141593	-1.000000e+00	52.0
id1	vélo	18	44	6.283185	24.283185	1.000000e+00	44.0

```
id2     moto    24    23  4.712389  28.712389 -1.836970e-16  24.0
```

2.3.4 Exemple

On créer un dataframe MAXIM dont la seule colonne est 'MAX'. Cette colonne est le maximum des colonnes Data1['COL1'], Data1['COL2'], Data1['COL3'].

[95]:
```
MAXIM = pd.DataFrame(Data1[['COL1','COL2','COL3']].max(axis=1),columns=['MAX'])
```

2.3.5 Exemple

On créer une colonne Data1['EXTRACT'] qui est l'extraction des 3 premiers caractères de Data1['COL0'].

[6]:
```
Data1['EXTRACT'] = Data1['COL0'].str[0:3]
Data1['EXTRACT']
```

[6]:
```
id0    voi
id1    vél
id2    mot
id3    voi
id4    mot
id5    vél
Name: EXTRACT, dtype: object
```

2.3.6 Exercice

Le but de l'exercice est d'ajouter des colonnes au dataframe Datasim1 que l'on crée ci-dessous.

1. Créer une colonne Datasim1['EXTRA'] qui est l'extraction des 2 premiers caractères de colonne Datasim1['provenance']
2. Créer une colonne Datasim1['MED'] qui est la médianne des variables Datasim1['var1'], Datasim1['var2'] et Datasim1['var3'].
3. Créer une colonne Datasim1['SINUS'] qui est le sinus de la colonne Datasim1['var3'].

[1]:
```
from scipy.stats import norm
from numpy.random import randint, seed
import numpy as np
import pandas as pd
seed(seed=1998)

CLASSE = ['group1','group2','group3']
PROVE = ['PREPA','UNIV']

Datasim1 = {'group':[CLASSE[randint(0,3)] for i in range(0,20)],\
            'provenance':[PROVE[randint(0,2)]for i in range(0,20)],\
           'var1':norm.rvs(loc=10,scale=2,size=20),'var2':norm.
 ↪rvs(loc=11,scale=3,size=20),\
            'var3':norm.rvs(loc=11,scale=3,size=20)}

Datasim1 = pd.DataFrame(Datasim1)
```

[3]:
```
Datasim1.head(2)
```

[3]:
```
    group provenance       var1       var2       var3
0  group1       UNIV  13.049084  14.802893  15.965563
1  group2       UNIV  10.605237   9.197587  17.776026
```

2.4 Logique apply, et np.where

— La logique apply permet d'appliquer une fonction python sur un dataframe.
— np.where permet de créer une colonne à partir de conditions appliquées à d'autres colonnes.

Création de Datasim2

[4]:
```
from scipy.stats import uniform, norm
import numpy as np
import pandas as pd
from numpy.random import seed, randint
seed(seed=1998)
#somme sur une liste, extraction de l'élement 2 et 3
Datasim2 = {'var1':norm.rvs(loc=10,scale=2,size=10),'var2':norm.
 ↪rvs(loc=13,scale=1,size=10)}
Datasim2 = pd.DataFrame(Datasim2)
Datasim2['var3'] = [uniform.rvs(loc=1,scale=6,size=20) for o in range(0,Datasim2.
 ↪shape[0])]
Datasim2['var4'] = [np.round(uniform.rvs(loc=1,scale=6,size=20)).tolist()\
                    for o in range(0,Datasim2.shape[0])]
```

[5]:
```
Datasim2.head(2)
```

[5]:
```
        var1       var2                                               var3  \
0  13.078688  12.297775  [5.08783643854, 4.31929530444, 5.60855798844, ...
1  10.699885  13.118363  [4.38565516955, 5.08968885244, 2.14772642592, ...

                                                var4
0  [5.0, 4.0, 2.0, 2.0, 6.0, 4.0, 6.0, 3.0, 3.0, ...
1  [2.0, 3.0, 4.0, 2.0, 2.0, 2.0, 4.0, 3.0, 1.0, ...
```

2.4.1 Exemple (utilisation de np.where)

On créer une variable Datasim2['var2cond'] :

— Si Datasim2['var2']<13 alors Datasim2['var2cond']='ok'
— Si Datasim2['var2']<13 alors Datasim2['var2cond']='ko'

[35]:
```
import numpy as np
Datasim2['var2cond'] = np.where(Datasim2['var2']<13,'ok','ko')
Datasim2[['var2','var2cond']].head(3)
```

[35]:
```
        var2 var2cond
0  12.297775       ok
1  13.118363       ko
```

2.4.2 Exemple (utilisation de apply)

On créer une variable Datasim2[‘var2condbis’] :

— Si Datasim2[‘var2’]<13 alors Datasim2[‘var2condbis’]=‘ok’

— Si Datasim2[‘var2’]<13 alors Datasim2[‘var2condbis’]=‘ko’

[39]:
```
#lambda x: True if x % 2 == 0 else False
Datasim2['var2condbis'] = Datasim2['var2'].apply(lambda x: 'ok' if x<13 else 'ko')
Datasim2[['var2','var2cond','var2condbis']].head(3)
```

[39]:

	var2	var2cond	var2condbis
0	12.297775	ok	ok
1	13.118363	ko	ko
2	14.383082	ko	ko

Exemple (utilisation de apply)

On créer une variable Datasim2[‘var2condtier’] :

— Si Datasim2[‘var2’]<13 alors Datasim2[‘var2condtier’]=‘ok’

— Si Datasim2[‘var2’]<13 alors Datasim2[‘var2condtier’]=‘ko’

[43]:
```
def cond(x):
    if x<13:
        y='ok'
    else :
        y='ko'
    return(y)

Datasim2['var2condtier'] = Datasim2['var2'].apply(lambda x:cond(x))
```

2.4.3 Exemple apply

On extrait le troisième élément de chaque liste stocker dans Datasim2[‘var4’]

[52]:
```
Datasim2['var4extract'] = Datasim2['var4'].apply(lambda x:x[3])
Datasim2.head(3)
```

[52]:

	var1	var2	var3	\
0	13.078688	12.297775	[5.08783643854, 4.31929530444, 5.60855798844, ...	
1	10.699885	13.118363	[4.38565516955, 5.08968885244, 2.14772642592, ...	
2	11.577937	14.383082	[1.76799108326, 4.63829521568, 6.18488309772, ...	

	var4	var2cond	var2condbis	\
0	[5.0, 4.0, 2.0, 2.0, 6.0, 4.0, 6.0, 3.0, 3.0, ...	ok	ok	
1	[2.0, 3.0, 4.0, 2.0, 2.0, 2.0, 4.0, 3.0, 1.0, ...	ko	ko	
2	[1.0, 4.0, 4.0, 7.0, 5.0, 7.0, 3.0, 5.0, 6.0, ...	ko	ko	

	var2condtier	var1cond	var1condbis	sumvar3	var4extract
0	ok	non	non	85.916857	2.0
1	ko	non	non	79.394726	2.0
2	ko	non	non	83.174359	7.0

Exercice

On travail à partir du dataframe Datasim2. (n'oublier pas de créer Datasim2')

1. Créer une variable Datasim2['var1cond'] : oui si $8 < Datasim2['var1'] < 10$, non dans le cas contraire (on utilisera np.where)
2. Créer une variable Datasim2['var1cond'] : oui si $8 < Datasim2['var1'] < 10$, non dans le cas contraire (on utilisera apply)
3. Créer une variable Datasim2['sumvar3'] qui est la somme des np.array stocker dans var3.

2.5 Agrégation de donnée, groupby

Voici 2 liens utiles :

— https://pandas.pydata.org/pandasdocs/stable/generated/pandas.core.groupby.DataFrameGroupBy.agg.html

— https://pandas.pydata.org/pandasdocs/stable/comparison_with_sql.html

[6]:
```
#url = 'https://raw.github.com/pandas-dev/pandas/master/pandas/tests/data/tips.csv'
#tips = pd.read_csv(url)
semaine = ['lundi','mardi','mercredi','jeudi','vendredi']
a = 2
moyenne = [10,10+a,10+2*a,10+3*a,10+4*a]
sex = ['H','F']
repas = ['midi','soir']
from scipy.stats import norm
from numpy.random import randint, choice, seed
import pandas as pd
seed(seed=1998)


HF = choice(sex,replace=True,p=[1/3,2/3],size=100)
SOIR_MIDI = choice(repas,replace=True,p=[1/2,1/2],size=100)
#JOUR_FACT = [[randint(0,5)] for i in range(0,100)]
JOUR_FACT = []
for i in range(0,100):
    pos = randint(0,5)
    el = [semaine[pos],norm.rvs(loc=moyenne[pos],scale=2,size=1)[0]]
    JOUR_FACT.append(el)

JOUR = [o[0] for o in JOUR_FACT]
FACT = [o[1] for o in JOUR_FACT]

Data_a_aggrege = {'sex' : HF,'repas':SOIR_MIDI,'jour': JOUR,'total':FACT,'PB':
 ↪randint(0,3,size=100)}
Data_a_aggrege = pd.DataFrame(Data_a_aggrege)
```

[7]:
```
Data_a_aggrege.head(3)
```

[7]:

	PB	jour	repas	sex	total
0	1	vendredi	midi	F	17.163936
1	2	jeudi	midi	F	12.491058
2	0	lundi	soir	H	8.332165

2.5.1 Exemple

1) On va chercher les valeurs unique de la colonne repas. On convertit le résulat en list
2) On va chercher le maximum de la colonne total.

[81]:
```
Data_a_aggrege['repas'].unique().tolist()
```

[81]:
```
['soir', 'midi']
```

[7]:
```
max_total = Data_a_aggrege['total'].max()
max_total
```

[7]:
```
21.963736630043087
```

2.5.2 Exercice

1) Chercher la médianne de la colonne Data_a_aggrege[‘total’].
2) Chercher les valeur unique de la colonne Data_a_aggrege[‘jour’]. Mettre les résultats dans une list.

2.5.3 Exemple : calcul d’agrégats

On va calculer la moyenne de la colonne Data_a_aggrege[‘total’] par Data_a_aggrege[‘jour’] et Data_a_aggrege[‘sexe’]

[8]:
```
import numpy as np
agg1 = Data_a_aggrege.groupby(['jour','sex']).agg({'total':np.mean,'PB':np.max})
agg1.head(2)
```

[8]:
```
               total  PB
jour  sex
jeudi F    16.415232   2
      H    16.796861   2
```

2.5.4 Exemple : calcul d’agrégats

Ici on calcule la médianne et le max de Data_a_aggrege[‘total’] groupé par jour et sex. On calcule également le minimum et la moyenne de la variable Data_a_aggrege[‘PB’] groupé par jour et sex.

[9]:
```
agg2 = Data_a_aggrege.groupby(['jour','sex']).agg({'total':['median','max'],'PB':
 ↪['min','mean']})
agg2.head(2)
```

[9]:
```
               total             PB
              median        max min      mean
jour  sex
jeudi F    16.628906  21.168633   0  0.863636
      H    17.102278  17.739241   0  1.500000
```

2.5.5 Exemple : calcul d'agrégats

Ici on regarde la répartition du nombre de ligne du dataframe Data_a_aggrege par jour,repas et sex :

[10]:
```
agg3 = Data_a_aggrege.groupby(['jour','repas','sex']).count()
agg3.head(3)
```

[10]:
```
                PB  total
jour  repas sex
jeudi midi  F    12     12
            H     3      3
      soir  F    10     10
```

Exercice

1) Calculer la somme et la moyenne de la variable Data_a_aggrege['PB'] groupé par repas
2) Calculer la somme et la moyenne de la variable Data_a_aggrege['total'] groupé par repas
3) Faire les calcule 1) et 2) en une ligne de code

2.6 Export et import de dataframe

Nous donnons dans le tableau quelques fonction d'export d'un dataframe :

to_csv([path_or_buf, sep, na_rep	Write DataFrame to a comma-separated values (csv) file
to_dict([orient, into])	Convert the DataFrame to a dictionary
to_excel(excel_writer[, sheet_name, na_rep	Write DataFrame to an excel sheet
to_pickle(path[, compression, protocol])	Pickle (serialize) object to file.

Nous donnons dans le tableau quelques fonction d'importation d'un dataframe :

pandas.read_csv	importation d'un fichier csv
pandas.read_excel	importation d'un fichier excel
pandas.read_pickle	importation d'un fichier pickle

2.6.1 Exemple (exportation de fichier)

1) Exporter le dataframe Data_a_aggrege dans un fichier excel
2) Exporter le dataframe Data_a_aggrege dans un fichier csv
3) Exporter le dataframe Data_a_aggrege dans un fichier pickle

[41]:
```
Data_a_aggrege.to_csv("/home/fabien/Bureau/Python Dauphine/export/data.csv",sep=";
 ↪",index=False)

Data_a_aggrege.to_excel("/home/fabien/Bureau/Python Dauphine/export/data.
 ↪xlsx",sheet_name="data",index=False)

Data_a_aggrege.to_pickle("/home/fabien/Bureau/Python Dauphine/export/data.pkl")
```

2.6.2 Exemple (importation de fichier)

Ci-dessous, nous importons 3 fichiers. Ces importations produisent des dataframes pandas.

[46]:
```
import pandas as pd
Dataimport1 = pd.read_csv("/home/fabien/Bureau/Python Dauphine/export/data.csv",sep=";
 ↪",encoding='latin_1')

Dataimport2 = pd.read_excel("/home/fabien/Bureau/Python Dauphine/export/data.
 ↪xlsx",sheet_name="data")

Dataimport3 = pd.read_pickle("/home/fabien/Bureau/Python Dauphine/export/data.pkl")
```

[47]:
```
Dataimport3.head(3)
```

[47]:

	PB	jour	repas	sex	total
0	1	vendredi	midi	F	17.163936
1	2	jeudi	midi	F	12.491058
2	0	lundi	soir	H	8.332165

[45]:
```
Dataimport2.head(3)
```

[45]:

	PB	jour	repas	sex	total
0	1	vendredi	midi	F	17.163936
1	2	jeudi	midi	F	12.491058
2	0	lundi	soir	H	8.332165

2.7 Exercice à rendre

Dans cet exercice, on travail avec le fichier mpg.txt (Miles Per Gallon). Ce fichier est téléchargeable :

— https://www.dropbox.com/sh/3sfu75df0lytgqk/AADLDVhlbnLtyFlyhRzt6yJta?dl=0

— https://www.dropbox.com/s/s4v4wfllmdhaqtd/mpg.txt?dl=0

Attention ce fichier n'a pas de nom de colonne. Voici les attributs et l'ordre dans lequel ces attributs apparaissent dans le fichier.

1. mpg :continuous
2. cylinders : multi-valued discrete
3. displacement :continuous
4. horsepower : continuous
5. weight : continuous
6. acceleration : continuous
7. model year : multi-valued discrete
8. origin :multi-valued discrete
9. car name : string (unique for each instance)

1) Importer ce fichier dans un dataframe pandas s'appelant mpg. Les noms des colonnes de ce fichiers doivent être : [mpg,cylinders,displacement,horsepower,weight,acceleration,model year,origin,car name]

2) Quelle est la plus petite valeur de la variable mpg. Cette la valeur sera mise dans une variable min_mpg et sera afficher à l'aide d'un print.

3) Certaint élément de la colonne horsepower ont la valeur '?'. Créer un dataframe mpgbis à partir de mpg en supriment les ligne de la colonne horsepower ayant pour valeur '?'. Convertir ensuite la colonne mpgbis['horspower'] en np.float64 à l'aide de astype.

4) Dans la suite de l'exercice on utilise le dataframe mpgbis. Trouver la moyenne de la colonne horsepower. Trouver ensuite la quantité de voiture dont la valeur de horsepower est supérieur à cette moyenne. Cette quantité sera miss dans une variable nb_voiture et sera afficher à l'aide d'un print.

5) Donner la moyenne et la medianne de mpgbis['weight'] et de mpgbis['acceleration'] pour chaque valeur de mpgbis['model year'] et mpgbis['origin'] (il faut faire un groupby 'model year' et 'origin'). Le résulat de cette requête sera stocker dans un dataframe agg_car. Afficher les 3 première ligne de agg_car.

6) Trouver les valeurs unique de mpgbis['origin'] que l'on stockera dans une variable origine_unique. A partir de mpgbis créer 3 dataframe comme suit :

— créer un dataframe dont la colonne mpgbis['origin'] vaut 1,
— créer un dataframe dont la colonne mpgbis['origin'] vaut 2,
— créer un dataframe dont la colonne mpgbis['origin'] vaut 3.

Ensuite, exporter ces 3 dataframes dans fichier excel export_car.xlsx ayant 3 feuillets : origine1, origine2 et origine3. Vous pouvez consulter la page internet suivante : https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_excel.html

7) Donner la moyenne et la médianne mpgbis['horsepower'] par mpgbis['model year']. Le résultat sera stocker dans un dataframe agg_horse. Changer l'index de ce dataframe. Cet index sera de la forme suivante : '1970-01-01','1971-02-01',...,'1980-03-01'. Vous pouvez utiliser la fonction date du package datetime. Vous pouvez également utiliser la fonction du package pandas pd.date_range. https://pandas.pydata.org/pandas-docs/stable/generated/pandas.date_range.html

Chapitre 3

Date et datetime

Les notebooks du cours sont accessibles dans le zip accessible sur le lien suivant : https://1drv.ms/u/s!Am09h0q2OIX0bJxMkXN1DDmMIY4?e=P0n2zz.

Dans ce chapitre, on présente les objets date et datetime. On peut traiter des objets datetime en utilisant notamment les 3 packages suivants : datetime, pandas, numpy

Les liens suivants sont des références :

1. objet datetime :https://docs.python.org/fr/3/library/datetime.html
2. fonctionalité de date pandas :https://pandas.pydata.org/pandasdocs/stable/user_guide/timeseries.html
3. pd.date_range :https://pandas.pydata.org/pandasdocs/stable/reference/api/pandas.date_range.html

3.1 Objet datetime et date avec le package datetime

En python les objets datetime sont crées avec le package datetime.

```
In [3]: from datetime import datetime, date

        dt1 = datetime(2015,5,21,3,2,0)
        dt1
```

```
Out[3]: datetime.datetime(2015, 5, 21, 3, 2)
```

Ci-dessous la création d'une date :

```
In [10]: from datetime import datetime, date
         date1 = date(2015,6,21)
         date1
```

```
Out[10]: datetime.date(2015, 6, 21)
```

On convertie l'objet dt1 datetime en date :

```
In [9]: dt1.date()
```

```
Out[9]: datetime.date(2015, 5, 21)
```

Soit dt un objet datetime :

— dt.year, dt.month, dt.day permettent d'extraire l'année, le mois, le jour du mois
— dt.weekday(), dt.isoweekday() permettent d'extraire le jour de la semaine
— dt.isocalendar() renvoie un tuple de 3 éléments, (année ISO, numéro de semaine ISO, jour de la semaine ISO).
— dt.toordinal() renvoie l'ordinal grégorien proleptique de la date de l'objet datetime dt

On va extraire l'année, le mois, le jour, le jour de la semaine de datetime(2019,4,25,2,3,5) :

```
In [1]: from datetime import datetime, date
        dt2 = datetime(2019,4,25,2,3,5)

        annee = dt2.year
        mois = dt2.month
        jour = dt2.day
        jour_semaine = dt2.weekday()
        jour_semaine_iso = dt2.isoweekday()
        print("annee:"+str(annee)+str(" ,")+"mois:"+str(mois)+" ,"+"jour_semaine:
 ↪"+str(jour_semaine)+" ,"+"jour_semaine_iso:"+str(jour_semaine_iso))
```

```
annee:2019 ,mois:4 ,jour_semaine:3 ,jour_semaine_iso:4
```

weekday() renvoie le jour de la semaine sous forme de nombre, où lundi vaut 0 et dimanche vaut 6. La méthode isoweekday() renvoie le jour de la semaine sous forme de nombre, où lundi vaut 1 et dimanche vaut 7.

On applique la méthode isocalendar pour déterminer la semaine et le jour du 29-05-1990.

```
In [3]: from datetime import datetime, date
        dt3 = datetime(1990,5,29)
        dt3.isocalendar()
```

```
Out[3]: (1990, 22, 2)
```

Le 29-05-1990 était un mardi et c'était la semaine 22 de l'année 1990. fromordinal(ordinal) Renvoie la date correspondant à l'ordinal grégorien proleptique. Nous allons illustrer son application. On ajoute 3 à l'ordinal de dt3 = 29-05-1990 puis on va déterminer le jour que cela représente :

```
In [4]: from datetime import datetime, date
        dt3 =  datetime(1990,5,29)
        dt3_ordinal = dt3.toordinal()
        dt3_ordinal
```

```
Out[4]: 726616
```

```
In [7]: dt4 = datetime.fromordinal(dt3_ordinal+3)
        dt4.isocalendar()
```

```
Out[7]: (1990, 22, 5)
```

Lorsque l'on ajoute 3 jour à la date dt3 on est toujours dans la semaine 22 de l'année 1990. C'est le jour 5 de la semaine, c'est à dire un vendredi. **fromordinal(ordinal)** Renvoie la date correspondant à l'ordinal grégorien proleptique.

Exercice

Quel jour de la semaine correspondes aux dates suivantes : 21-05-1981, 30-06-1999, 25-03-1986, 26-04-1986 ?

```
In [21]: from datetime import datetime
         listdate =␣
 ↪[datetime(1981,5,21),datetime(1999,6,30),datetime(1986,3,25),datetime(1986,4,26)]
         [(o,o.isoweekday()) for o in listdate]
```

```
Out[21]: [(datetime.datetime(1981, 5, 21, 0, 0), 4),
          (datetime.datetime(1999, 6, 30, 0, 0), 3),
          (datetime.datetime(1986, 3, 25, 0, 0), 2),
          (datetime.datetime(1986, 4, 26, 0, 0), 6)]
```

le 21-05-1981 est un jeudi, le 30-06-1999 est un mercredi

Exercice

Nous somme le dimanche 18 août 2019. Quel jour seront-nous 1000 jours plus tard ?

```
In [4]: from datetime import datetime, date

        #On créer la date
        dim18_2019 = datetime(2019,8,18)

        #on prend l'ordinale de la date
        date_nouv_ordinal = dim18_2019.toordinal()+1000
        #On crée la nouvelle date
        date_nouv = datetime.fromordinal(date_nouv_ordinal)
        print(date_nouv)
        #On cherche le jour
        date_nouv.isoweekday()
```

```
2022-05-14 00:00:00


Out[4]: 6
```

1000 jours plus tard nous seront le 14/05/2022 et se sera un samedi (isoweekday=6).

3.1.1 Les timedelta : ajout de quantité de temps

On a va montrer sur quelques exemples l'utilisation des timedelta

Exemple utilisation timedelta

Nous somme le 25/06/2009 à 22h30.

1/ Quelle date seront nous 399 jours plus ? On utilise timedelta avec le paramêtre days.

```
In [8]: from datetime import datetime, date, timedelta
        jour = datetime(2009,6,25,22,30)
        jour_ajout = jour+timedelta(days=399)
        jour_ajout.date()
```

```
Out[8]: datetime.date(2010, 7, 29)
```

Nous seront le 29 juillet 2010 399 jours plus tard.

2/ Quelle heure et quelle date seront nous 300 000 secondes plus tard ? On utilise timedelta avec le paramêtre seconds.

```
In [12]: from datetime import datetime, date, timedelta
         jour = datetime(2009,6,25,22,30)
         jour_ajout = jour+timedelta(seconds=300000)
         jour_ajout_heure= jour_ajout.hour
         print("heure:" +str(jour_ajout_heure))
         jour_ajout.date()
```

```
heure:9
```

```
Out[12]: datetime.date(2009, 6, 29)
```

Nous serons le 29 juin 2009 300 000 secondes plus tard, il sera 9 heure.

3/ Quelle date seront nous 10 000 heures plus tard ?

```
In [13]: from datetime import datetime, date, timedelta
         jour = datetime(2009,6,25,22,30)
         jour_ajout = jour+timedelta(hours=10000)
         jour_ajout.date()
```

```
Out[13]: datetime.date(2010, 8, 16)
```

Nous seront le 16/08/2010 10000 heures plus tard.

3.2 Les méthodes strptime et strftime : convertion datetime en string et inversement

- datetime.strptime(date_string, fmt) : convertie un string date_string représantant une date en datetime selon le format fmt
- d.strftime(fmt) : convertie un le datetime d en un string selon le format fmt

Un format d'un string représentant une date peut être '%b %d,%Y' ce qui veut dire Nom du mois jour du mois en 2 chiffres et année en 4 chiffres. Par exemple, jan 22 2018.

%a	Jour de la semaine abrégé dans la langue locale.	Mon, . . . , Sat (en_US) ;Lu,. . . , Di (fr_FR)
%A	Jour de la semaine complet dans la langue locale.	Monday, . . . , Saturday (en_US) ;Lundi,Dimanche(fr_FR)
%w	Jour de la semaine en chiffre, avec 0 pour le dimanche	0, 1, 6
%d	Jour du mois sur deux chiffres.	01, 02, . . . , 31
%b	Nom du mois abrégé dans la langue locale	Jan, Feb, . . . , Dec (en_US) ;janv., févr, déc. (fr_FR)
%B	Nom complet du mois dans la langue locale.	January, December (en_US) ;janvier, . . . , décembre (fr_FR)
%m	Numéro du mois sur deux chiffres.	01, 02, . . . , 12
%y	Année sur deux chiffres (sans le siècle).	00, 01, . . . , 99
%Y	Année complète sur quatre chiffres.	2013, 2014

Exemple utilisation strftime

On va transformer le datetime(2019,5,26) en 3 strings différents :

- string1 résultat du format '%A',
- string2 résultat du format '%A , le %d %B %Y',
- string3 résultat du format '%a le %d/%m/%y'

```
In [11]: from datetime import datetime, date
         date = datetime(2019,5,26)
         string1 = date.strftime('%A')
         string2 = date.strftime('%A , le %d %B %Y')
         string3 = date.strftime('%a le %d/%m/%y')
         print('string1: '+str(string1)+" ,"+'string2: '+str(string2)+" ,"+'string3:␣
 ↪'+str(string3))
```

```
string1: Sunday ,string2: Sunday , le 26 May 2019 ,string3: Sun le 26/05/19
```

Exemple utilisation strftime

On va transformer le datetime(2015,10,23) en string du type 'yyyy-mm-dd', 'dd-mm-yyyy' et 'le dd mois-lettre yyyy'.

```
In [5]: from datetime import datetime, date
        jour = datetime(2015,10,23)
        string1 = jour.strftime('%Y-%m-%d')
        string2 = jour.strftime('%d-%m-%Y')
        string3 = jour.strftime('le %d %B %Y')
        print('string1: '+str(string1)+" ,"+'string2: '+str(string2)+" ,"+'string3:␣
 ↪'+str(string3))
```

```
string1: 2015-10-23 ,string2: 23-10-2015 ,string3: le 23 October 2015
```

Exemple utilisation datetime.strptime(date_string, fmt)

Ici on va convertir les chaînes de caractères suivantes en datetime :le 23 October 2015, Sun le 26/05/19 et 2015-10-23.

```
In [16]: from datetime import datetime
         date1 = datetime.strptime('le 23 October 2015','le %d %B %Y')
         date2 = datetime.strptime('Sun le 26/05/19','%a le %d/%m/%y')
         date3 = datetime.strptime('2015-10-23','%Y-%m-%d')

         print('date1: '+str(date1)+" ,"+'date2: '+str(date2)+" ,"+'date3: '+str(date3))
```

```
date1: 2015-10-23 00:00:00 ,date2: 2019-05-26 00:00:00 ,date3: 2015-10-23 00:00:00
```

3.3 Datetime dans le contexte pandas

Les fonction du package datetime sont utilisables avec la logique apply dans la cadre des dataframe. Le package pandas fournit également des fonctionnalitées direct sur les date/datetimes. On va montrer comment utiliser les 2 approches.

Exemple

Dans le dataframe exemple_date suivant, il y a une colonne string exemple['datestring'] représentant des dates. La colonne exemple['volume'] représente des volumes.

```
In [2]: #Création du dataframe
        import numpy as np
        from datetime import datetime
        import pandas as pd

        exemple_date = pd.DataFrame({'datestring':[datetime(2018,np.random.
 ↪randint(1,12),np.random.randint(1,25)).strftime('Annee:%Y mois:%B jour:%d') for o in␣
 ↪range(0,15)],'volume':np.random.randint(20,100,size=15)})
        exemple_date.head(3)
```

Out[2]:

	datestring	volume
0	Annee:2018 mois:May jour:15	28
1	Annee:2018 mois:January jour:13	68
2	Annee:2018 mois:June jour:07	20

1/ On va convertir cette colonne en utilisant la fonctionnalité to_datetime du package pandas. Le résultat sera stocké dans date1 :

```
In [3]: exemple_date['date1'] = pd.to_datetime(exemple_date['datestring'],format='Annee:
 ↪%Y mois:%B jour:%d')
        exemple_date['date1'].head(2)

Out[3]: 0   2018-05-15
        1   2018-01-13
        Name: date1, dtype: datetime64[ns]
```

2/ On va convertir cette colonne en utilisant la fonction strptime du package. Le résultat sera stocké dans date2

```
In [4]: from datetime import datetime
        exemple_date['date2'] = exemple_date['datestring'].apply(lambda x:datetime.
 ↪strptime(x,'Annee:%Y mois:%B jour:%d'))
        exemple_date['date2'].head(2)

Out[4]: 0   2018-05-15
        1   2018-01-13
        Name: date2, dtype: datetime64[ns]
```

3/ On va ajouter un timedelta aléatoire compris entre 3 et 20 jours en utilisant le package datetime. Ce timedelta est ajouté a exemple_date[‘date2’]. On stocke le résultat dans exemple_date[‘date3’].

```
In [14]: from datetime import datetime, date, timedelta
         import numpy as np
         np.random.seed(1998)
         exemple_date['date3'] = exemple_date['date2'].apply(lambda x:
 ↪x+timedelta(days=np.random.randint(3,20)))
         exemple_date[['date2','date3']].head(2)

Out[14]:        date2      date3
         0 2018-05-15 2018-05-26
         1 2018-01-13 2018-01-17
```

https://www.tutorialspoint.com/python_pandas/python_pandas_timedelta.htm

4/ On va ajouter un timedelta aléatoire compris entre 3 et 20 jours en utilisant le package pandas pd.Timedelta(days=i). Le résultat est stocké dans exemple_date[‘date4’].

```
In [19]: import numpy as np
         np.random.seed(1998)
         import pandas as pd
         pd.Timedelta(days=np.random.randint(3,20))
         #exemple_date['date4'] = exemple_date['date4']
         exemple_date['date4'] = exemple_date['date2'].apply(lambda x:x+pd.
 ↪Timedelta(days=np.random.randint(3,20)))
         exemple_date['date4'].head(2)

Out[19]: 0   2018-05-19
         1   2018-01-17
         Name: date4, dtype: datetime64[ns]
```

5/ On extrait le mois de exemple_date[‘date2’] de 2 façons différentes :

```
In [21]: exemple_date['month1'] = exemple_date['date2'].dt.month
         exemple_date['month1'].head(2)
```

```
Out[21]: 0    5
         1    1
         Name: month1, dtype: int64
```

```
In [23]: exemple_date['month2'] = exemple_date['date2'].apply(lambda x:x.month)
         exemple_date['month2'].head(2)
```

```
Out[23]: 0    5
         1    1
         Name: month2, dtype: int64
```

3.4 Exercice à rendre (datetime)

Le but de cet exercice est de manipuler des datetimes/date. On utilisera à la fois des fonctionnalité provenant du package datetime et des fonctionnalités provenant du package pandas. Les fonctionnalités du package datetime sont utilisées avec la logique apply /map. Les colonnes du dataframe DATADATE :

- annee :variable numérique
- mois : varaible numérique
- jour : variable numérique

Ces trois colonnes n'ont aucun rapport avec les autres.

- datestring : variable de type string représantant une date
- date_debut :variable de type datetime
- date_fin : variable de type datetime

(date_fin>date_debut) datestring n'a aucun lien avec les variables datestring et datestring. Exécuter la cellule ci-dessous permettant la création du dataframe.

```
In [4]: import numpy as np
        from datetime import datetime, timedelta
        import pandas as pd
        np.random.seed(2018)
        mois = np.random.randint(1,12,size=200,dtype=int)
        annee = np.random.randint(2015,2019,size=200,dtype=int)
        jour = np.random.randint(1,25,size=200,dtype=int)
        datestring = [datetime(np.random.randint(2015,2019,dtype=int),np.random.
 ↪randint(1,12,dtype=int),np.random.randint(1,25,dtype=int)).strftime('%b %d,%Y') for o␣
 ↪in range(0,200)]
        date_debut = [datetime(np.random.randint(2015,2019,dtype=int),np.random.
 ↪randint(1,12,dtype=int),np.random.randint(1,25,dtype=int)) for o in range(0,200)]
        date_fin = [o+timedelta(days=np.random.randint(12,90,dtype=int)) for o in␣
 ↪date_debut]
        VOLUM1 = np.random.randint(10,15,size=200)
        VOLUM2 = np.random.randint(6,10,size=200)

        DATADATE = pd.DataFrame({'annee':annee,'mois':mois,'jour':jour,'datestring':
 ↪datestring,'date_debut':date_debut,'date_fin':date_fin,'VOLUM1':VOLUM1,'VOLUM2':
 ↪VOLUM2})
        DATADATE.head(3)
```

```
Out[4]:    annee  mois  jour   datestring date_debut   date_fin  VOLUM1  VOLUM2
        0   2018    11     3  Aug 23,2018 2018-08-11 2018-09-09      10       9
        1   2015     7     5  Apr 13,2016 2015-02-05 2015-05-04      14       8
        2   2016     3     6  Sep 07,2016 2017-07-22 2017-10-09      11       7
```

1. Construire une colonne DATADATE[‘datecreat’] à partir des colonnes DATADATE[‘annee’], DATADATE[‘mois’], DATADATE[‘jour’]. On utilisera la fonction datetime du package datetime et la logique apply.
2. Créer la colonne DATADATE[‘dateconv’] de type datetime à partire de la colonne DATADATE[‘datestring’]. Utiliser 2 méthodes
3. Convertir la colonne DATADATE[‘date_debut’] en string. Le datetime ‘2018-08-23’ doit être transformé en ‘Thursday 23 August 2018’. Les résultats sont stockés dans DATADATE[‘date_debut_string’].
4. Calculer le nombre de jour entre DATADATE[‘date_debut’] et DATADATE[‘date_fin’]. Les résultats sont stoké dans DATADATE[‘date_diff’]. Vous pouvez vous inspirer de ce site : http://www.datasciencemadesimple.com/difference-two-dates-days-weeks-months-years-pandas-python-2/
5. Calculer le nombre de jour entre DATADATE[‘date_debut’] et DATADATE[‘date_fin’]. Les résultats sont stoké dans DATADATE[‘date_diff_bis’]. Vous utiliserez la logique apply avec les méthode date() et toordinal().
6. Calculer le nombre de samedi et dimanche entre DATADATE[‘date_fin’] et DATADATE[‘date_debut’]. * Faire une fonction qui prend 2 datetime en entrée. Cette sort le nombre de samedi et dimanche * On utilisera toordinal, isocalendar(), fromordinal
7. Calculer le nombre de semaine différente entre DATADATE[‘date_fin’] et DATADATE[‘date_debut’]. Le résultat sera stocké dans DATADATE[‘nb_semaine’].
8. 8/ A l’aide d’un tableau croisé dynamique faire le tableau donnant la somme de VOLUM1 par année et jour de la date de début. Les année sont en ligne et les jours sont en colonne :

annee	lundi	mardie
2015	60	60
2017	50	40

On utilisera la fonctionnalité pd.pivot_table : https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot_table.html

Chapitre 4

Projet Python

Les données des projets sont dans un un zip accessible sur le lien suivant : https://1drv.ms/u/s!Am09h0q2OIX0a3BCjIUmAJbvvYE?e=1eyeyQ.

4.1 Projet 1 à 5 : Sample Insurance Portfolio, Real estate transactions, Sales transactions...

Dans le site internet https://support.spatialkey.com/spatialkey-sample-csv-data/ vous avez un ensemble de 5 jeux de données cvs.

- Sample insurance portfolio
- Realestate transactions
- Sales transactions
- Company Funding Records
- Crime Records

Chaque jeux de donnée correspond à un projet. Vous devez :

- choisir l'un des jeux de données
- décrire de manière sommaire de quoi parle le jeux de données choisi
- décrire les variables du jeux de données (leurs types et leurs sens)
- Créer 2 tableaux agrégés : faire un commentaire de ces 2 tableaux
- Créer 2 datavisualisations (diagramme en bâton ou bien histogramme, camembert, nuage de point) : faire un commentaire de ces 2 graphes

Dans le cadre des ces projets vous devez rendre un notebook avec les codes qui vous ont permis de faire les tableaux agrégés et les datavisualisations. Vous devez également faire une présentation power point ou bien latex de 11 slides maximums.

4.2 Projet 6 : Consumer Complaint Database

Les données de ce projet sont accessibles ici :

- https://catalog.data.gov/dataset/consumer-complaint-database
- ou bien sur kaggle https://www.kaggle.com/cfpb/us-consumer-finance-complaints

Vous avez une illustration de l'utilisation des donnée ici : https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17. Bien sur, cette illustration dépasse le cadre de ce cours. En revanche vous pouvez en déduire le sens de certaine variable et comprendre l'utilisation de ces dernières dans le cadre du deep learning.

4.2.1 Travail à faire

Vous devez dans un premier temps :

— décrire de manière sommaire de quoi parle le jeux de données choisi

— décrire les variables du jeux de données (leurs types et leur sens)

Ensuite vous devez garder les lignes de cette base de donnée ayant la colonne consumer_complaint_narrative non vide. Ensuite vous devez :

— Créer 2 tableaux agrégés : faire un commentaire de ces 2 tableaux

— Créer 2 datavisualisations (diagramme en bâton ou bien histogramme, camembert, nuage de point) : faire un commentaire de ces 2 graphes

Dans le cadre des ce projet vous devez rendre un notebook avec les codes qui vous ont permis de faire les tableaux agrégés et les datavisualisations. Vous devez également faire une présentation power point ou bien latex de 11 slides maximums.

4.3 Projet 6 bis : Nuage de mots sur les données Consumer Complaint Database

Le but de ce projet est de faire des nuages de mots sur les textes liés aux plaintes de clients d'assurance.

Ce projet est techniquement plus dure que les précédents. Les données de ce projet sont accessibles ici :

— https ://catalog.data.gov/dataset/consumer-complaint-database

— ou bien sur kaggle https ://www.kaggle.com/cfpb/us-consumer-finance-complaints

Vous avez une illustration de l'utilisation des données ici : https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17. Bien sur, cette illustration dépasse le cadre de ce cours. En revanche vous pouvez en déduire le sens de certaine variable et comprendre l'utilisation de ces dernières dans le cadre du deep learning.

4.3.1 Travail à faire

Vous devez dans un premier temps :

— décrire de manière sommaire de quoi parle le jeux de données choisi

— décrire les variables du jeux de données (leurs types et leur sens)

Ensuite vous devez garder les lignes de cette base de donnée ayant la colonne consumer_complaint_narrative non vide. Vous devez transformer la variable Product en utilisant le code ci-dessous.

```
In [0]: df.loc[df['Product'] == 'Credit reporting', 'Product'] = 'Credit reporting,␣
 ↪credit repair services, or other personal consumer reports'
        df.loc[df['Product'] == 'Credit card', 'Product'] = 'Credit card or prepaid card'
        df.loc[df['Product'] == 'Payday loan', 'Product'] = 'Payday loan, title loan, or␣
 ↪personal loan'
        df.loc[df['Product'] == 'Virtual currency', 'Product'] = 'Money transfer,␣
 ↪virtual currency, or money service'
        df = df[df.Product != 'Other financial service']
```

Vous devez ensuite préparer les données relatif aux textes des plaintes des clients. ces textes sont dans la variable Consumer complaint narrative.

— On doit convertir en minuscule la colonne Consumer complaint narrative

— Enlever les chiffres de la colonne Consumer complaint narrative

— Enlever la ponctuation de la colonne Consumer complaint narrative

Vous devez faire des nuages de mots sur les données consumer_complaint_narrative. Il y a un nuage de mots par product. La construction d'un nuage de mot sous python est expliqué sur le lien suivant : https://www.datacamp.com/community/tutorials/wordcloud-python.

Annexe A

Structure et instruction de base

Les notebooks du cours sont accessibles dans le zip accessible sur le lien suivant : https://1drv.ms/u/s!Am09h0q2OIX0bJxMkXN1DDmMIY4?e=P0n2zz.

A.1 Type simple

[3]:
```
var1 = 7.66
var2 = "Python"
var3 = 9
```

On regarde les types var1, var2 et var3 :

[2]:
```
type(var1)
```

[2]:
```
float
```

[4]:
```
type(var2)
```

[4]:
```
str
```

[5]:
```
type(var3)
```

[5]:
```
int
```

A.2 Liste Python

Un liste Python est composé d'élément simple.

[12]:
```
list1 = [1,2.5,9,"Python","R","SAS"]
```

On regarde la longeur de cette liste :

[7]:
```
len(list1)
```

[7]:
```
6
```

L'élément 0 de la list est :

[8]:
```
list1[0]
```

[8]:
```
1
```

L'élément 5 de la list est :

[9]:
```
list1[5]
```

[9]:
```
'SAS'
```

On va concatener deux listes :

[10]:
```
list3 = ["rr","dd",8.9,5] + ["gg","Julia",8,"Java"]
list3
```

[10]:
```
['rr', 'dd', 8.9, 5, 'gg', 'Julia', 8, 'Java']
```

On va ajouter l'élément "scala" à la list list1 en utilisant append() :

[13]:
```
list1.append("scala")
list1
```

[13]:
```
[1, 2.5, 9, 'Python', 'R', 'SAS', 'scala']
```

On peut faire une liste de liste :

[15]:
```
list_de_list = [[2.6,8,9.4],["java","latex","c++"],["machine␣
↪learning","pandas","stochastic"]]
```

On extrait l'élément 2 de l'élément 1 de list_de_list :

[16]:
```
list_de_list[1][2]
```

[16]:
```
'c++'
```

A.3 Tuple

Un tuple est composé d'élément simple et ne peut pas être modifié.

[18]:
```
tuple1 = (5,6,9.5,8,9)
```

On affiche la longeur de tuple1 :

[20]:
```
len(tuple1)
```

[20]:
```
5
```

On affiche l'élément 0 de tuple1 :

[21]:
```
tuple1[0]
```

[21]:
```
5
```

On affiche l'élement 4 de tuple1 :

[22]:
```
tuple1[4]
```

[22]:
```
9
```

On concaténe 2 tuples :

[24]:
```
tuple3 = (5,2.9,"oo") + (8,"Python","8","99")
tuple3
```

[24]:
```
(5, 2.9, 'oo', 8, 'Python', '8', '99')
```

Ici nous faisont une list de tuple :

[25]:
```
list_de_tuple = [("2",9),(9,8,7),("scala","java","python","R")]
```

On extrait l'élément 3 de l'élément 2 de list_de_tuple :

[26]:
```
list_de_tuple[2][3]
```

[26]:
```
'R'
```

Ici on convertit un tuple en list :

[27]:
```
tuple4 = ("Paris 6","Dauphine","Saclay")
list(tuple4)
```

[27]:
```
['Paris 6', 'Dauphine', 'Saclay']
```

Ici on convertit une liste en tuple :

[28]:
```
list5 = ["nager","courire","pédaler"]
tuple(list5)
```

[28]:
```
('nager', 'courire', 'pédaler')
```

Ici on montre un exemple de tuple de tuple :

[29]:
```
tuple_de_tuple = ((5,8,9),("OM","PSG","OL","ASSE"),("AJA","GF38","DFCO"))
```

A.4 Dictionnaire

Un dictionnaire est ensemble clefs valeurs :

[30]:
```
dict1 = {'annee':2018,'mois':5,'jour':12,'nom_moi':"mai"}
```

On affciche les clefs de dict1 (méthode .keys()) :

[31]:
```
dict1.keys()
```

[31]:
```
dict_keys(['annee', 'mois', 'jour', 'nom_moi'])
```

On affciche les valeurs de dict1 (méthode .values()) :

[33]:
```
dict1.values()
```

[33]:
```
dict_values([2018, 5, 12, 'mai'])
```

On affiche la valeur de la clé 'mois' :

[34]:
```
dict1['mois']
```

[34]:
```
5
```

[0]:
```
On ajoute la couple clé = 'date' et valeur='12/05/2018'
```

[37]:
```
dict1['date'] = '12/05/2018'
```

[38]:
```
dict1['date']
```

[38]:
```
'12/05/2018'
```

ici on construit une list de dictionnaire :

[3]:
```
list_dict = [{'annee':2018,'mois':5,'jour':12,'nom_moi':"mai"},{'annee':2019,'mois':
↪6,'jour':13,'nom_moi':"juin"},\
            {'annee':2020,'mois':7,'jour':14,'nom_moi':"juillet"}]
```

On va afficher la valeur de la clé 'mois' de l'élément 1 de la list _dict_list :

[4]:
```
list_dict[1]['mois']
```

[4]:
```
6
```

On construit un dictionaire dont la valeur de chaque clé est une list :

[5]:
```
dict_de_list = {'annee':[2018,209,2020],'mois':[10,5,6],'jours':[12,25,20]}
```

A.5 Structure conditionnelle

On montre un exemple de if python. Si les seconde de la date actuelle est divisible par 3 * on affiche les secondes sont divisibles par 3 * sinon on affiche les secondes ne sont pas divisibles par 3

[48]:
```
from datetime import datetime
date = datetime.now()
print(str(date))
```

```
2019-09-17 14:41:09.601255
```

[49]:
```
if date.second%3==0:
    print("les secondes sont divisibles par 3")
else:
    print("les seconde ne sont pas divisibles par 3")
```

```
les secondes sont divisibles par 3
```

On remarque l'indentation en python.

Ici on rentre un nombre x. Si $x^3 - 1$ est divisble par 4 : * on affiche "x au cube -1 est divisible par 4" * sinon on affiche le reste da la division par 4"

[54]:
```
x = input ("Enter number")
```

```
Enter number 7
```

[55]:
```
if (int(x)**3%4-1)==0:
    print("x au cube -1 est divisible par 4")
else:
    res = int(x)%4
    print(res)
```

```
3
```

A.6 Boucle for

On va afficher tous les éléments d'une list avec une boucle for.

[56]:
```
list_a_parcourir = [0,7.4,"R","Julia","c++","Java","Python","pandas","SAS"]

for o in list_a_parcourir:
    print(o)
```

```
0
7.4
R
Julia
c++
Java
Python
pandas
SAS
```

Ici on calcule tous les carrés d'une liste de nombre que l'on stock dans une list.

[57]:
```
nombre = [6.5,3,9.7,5,4]

nb_carre = []

for nb in nombre:
    nb_carre.append(nb**2)
```

[58]:
```
nb_carre
```

[58]: [42.25, 9, 94.08999999999999, 25, 16]

Ici on va mettre en majuscule une liste de mot :

[60]:
```
list_mot = ["r","c","sas","python"]

mot_maj = []

for mot in list_mot:
    mot_maj.append(mot.upper())
```

[61]:
```
mot_maj
```

[61]: ['R', 'C', 'SAS', 'PYTHON']

A.7 LA COMPRÉHENSION DE LISTE

Certaine boucle python peuvent être réalisée dans une liste. On va partire d'une liste de mot que l'on va mettre en majuscule.

[1]:
```
list_mot = ['formation','bac','ects','limonade']
```

A partir de list_mot on construit list_mot_maj comme suit :

[4]:
```
list_mot_maj = [mot.upper() for mot in list_mot]
list_mot_maj
```

[4]: ['FORMATION', 'BAC', 'ECTS', 'LIMONADE']

On va retenir les nombre d'une liste dont le carré est inférieur à 100 :

[6]:
```
list_nb = [7,8,5.5,4.9,7.5,11,99,10,4]
list_nb_retenu = [nb for nb in list_nb if nb**2<100]
list_nb_retenu
```

[6]: [7, 8, 5.5, 4.9, 7.5, 4]

On va retenir les mots commençant par un a ou un A :

[10]:
```
list_mot_bis = ['actuaire','Blame','rire','alibaba','Ada','trouver']
list_mot_ret = [mot for mot in list_mot_bis if mot[0] in ['a','A']]
list_mot_ret
```

[10]: ['actuaire', 'alibaba', 'Ada']

A.8 Fonction Python

On va programmer la fonction $f(x) = x^2 - 1$.

[11]:
```
def f(x):
    nb = x**2-1
    return(nb)
```

[12]:
```
print("f(2)="+str(f(2))+" f(3)="+str(f(3)))
```

```
f(2)=3 f(3)=8
```

On va faire une fonction qui affiche tous les éléments d'une liste :

[13]:
```
def aff(x):
    for el in x:
        print(el)
```

[14]:
```
list_a_afficher = ['Python','R','Scala','Julia']
aff(list_a_afficher)
```

```
Python
R
Scala
Julia
```

On peut utiliser une fonction dans une list de compréhension. On programme la foncion $g(x) = x^2 + 3x + 4$. Ensuite on applique cette fonction à une list de nombre.

[15]:
```
def g(x):
    nb = x**2+3*x+4
    return(nb)
```

[17]:
```
list_nb_bis = [1,2,-1,6]
list_nb_tier = [g(nb) for nb in list_nb_bis]
list_nb_tier
```

[17]:
```
[8, 14, 2, 58]
```