Day-24 Statistiques

Python pour l'analyse statistique

Statistiques

Les statistiques sont la discipline qui étudie
Le *collection*, *organization*, *displaying*, *analysing*, *interpretation* et *presentation* des don nées. Les statistiques sont une branche des mathématiques qui est recommandée comme une c ondition préalable à la science des données et à l'apprentissage automatique. Les statistiques s ont un domaine très large, mais nous nous concentrerons dans cette section uniquement sur la partie la plus pertinente. Après avoir terminé ce défi, vous pouvez accéder au développement Web, à l'analyse des données, à l'apprentissage automatique et à la science des données. En su ivant, vous pouvez suivre, à un moment donné de votre carrière, vous obtiendrez des données sur lesquelles vous pourriez travailler. Avoir des connaissances statistiques vous aidera à pren dre des décisions basées sur les données, *data tells as they say*.

Données

Qu'est-ce que les données? Les données sont tout ensemble de caractères recueillis et traduits dans une certaine fin, généralement une analyse. Il peut s'agir de n'importe quel personnage, y compris du texte et des numéros, des images, du son ou de la vidéo. Si les données ne sont pas placées dans un contexte, cela n'a aucun sens pour un humain ou un ordinateur. Pour donner u n sens aux données, nous devons travailler sur les données à l'aide de différents outils.

Le flux de travail de l'analyse des données, de la science des données ou de l'apprentissage automatique commence à partir des données. Les données peuvent être fournies à partir d'u ne source de données ou elles peuvent être créées. Il existe des données structurées et non s tructurées.

Les données peuvent être trouvées en petit ou grand format. La plupart des types de donné es que nous obtiendrons ont été couverts dans la section de gestion des fichiers.

Module statistique

Le module Python *statistics* fournit des fonctions pour calculer les statistiques mathématique s des données numériques. Le module n'est pas destiné à être un concurrent de bibliothèques de tiers telles que les packages de statistiques Numpy, Scipy ou propriétaires destinés à des st atisticiens professionnels tels que Minitab, SAS et Matlab. Il s'adresse au niveau de graphique et de calculatrices scientifiques.

Nombant

Dans la première section, nous avons défini Python comme un grand langage de programmat ion à usage général en soi, mais avec l'aide d'autres bibliothèques populaires comme (Numpy , Scipy, Matplotlib, Pandas, etc.), il devient un environnement puissant pour l'informatique s cientifique.

Numpy est la bibliothèque de base de l'informatique scientifique dans Python. Il fourni t un objet de tableau multidimensionnel à haute performance et des outils pour travaille r avec des tableaux.

Jusqu'à présent, nous utilisons VScode, mais à partir de maintenant, je recommanderais d'utilis er Jupyter Notebook. Pour accéder à Jupyter Notebook, installons Anaconda. Si vous utilisez Anaconda, la plupart des packages communs sont inclus et que vous n'avez pas de packages d'i nstallation si vous avez installé Anaconda.

```
asabeneh @ asabeneh: ~ / Desktop / 30daysofpython $ pip install numpy
```

Importation de Numpy

Jupyter Notebook est disponible si vous êtes en faveur du cahier Jupyter

```
    # Comment importer Numpy
    Importer Numpy comme NP
    # Comment vérifier la version du package Numpy
    Print ('Numpy:', NP .__ Version__)
    # Vérification des méthodes disponibles
    print (dir (np))
```

Création d'un tableau Numpy en utilisant

Création de tableaux int illustrés

```
# Création de la liste python python_list = [1,2,3,4,5] # Vérification des types de données prin t ('type:', type (python_list)) # < class 'list' > # print (python_list) # [1, 2, 3, 4, 5] Two_dimens ional_list = [[0,1,2], [3,4] [6,7,8]] print (Two_dimensional_list) # [[0, 1, 2], [3, 4, 5], [6, 7, 8]] # Création de Numpy (Numerical Python) Array à partir de la liste Python Numpy_Array_Fro m_List = np.array (python_list) print (Type (Numpy_Arnray_From_list)) < classe 'numpy.nd array' > print (numpy_array_from_list) # array ([1, 2, 3, 4, 5]))
```

Création de tableaux de flottaison flottants

Création d'un tableau Numpy Float à partir de la liste avec un paramètre de type de données flottants

```
# Liste Python
python_list = [1,2,3,4,5]

numy_array_from_list2 = np.array (python_list, dtype = float)
print (numy_array_from_list2) # array ([1., 2., 3., 4., 5.])
```

Création de tableaux numpy booléens

Création d'un booléen un tableau Numpy à partir de la liste

```
numpy_bool_array = np.array ([0, 1, -1, 0, 0], dtype = bool) print (numpy_bool_array) # arra
y ([false, true, true, false, false]))
```

Création d'un tableau multidimensionnel à l'aide de Numpy

Un numpyArray peut avoir une ou plusieurs lignes et COLU MNS

```
two_dimensional_list = [[0,1,2], [3,4,5], [6,7,8]] numpy_two_dimensional_list = n
p.array (deux_dimensional_list) print (type (numpy_two_dimensional_list)) print (
numpy_two_didimatedal_dimension
```

```
< classe 'numpy.ndarray' > [[0 1 2] [3
4 5] [6 7 8]]
```

Conversion du tableau Numpy en liste

```
# Nous pouvons toujours convertir un tableau en une liste de python à l'aide de Tolist ().

np_to_list = numpy_array_from_list.tolist () print (type (np_to_lis
t)) imprimer ('Array unidimensionnel:', np_to_list) print ('Twendi
mensional Array:', numpy_two_dimensional_list.tolist ())
```

```
< classe 'List' > Array unidimensionnel: [1, 2, 3, 4, 5] Array en deux dimensions: [[0, 1, 2], [3
, 4, 5], [6, 7, 8]]
```

```
Création du tableau numpy à partir de tuple
# Array Numpy à partir de tuple # créati
on de tuple en python
```

```
python_tuple = (1,2,3,4,5)                    print (type (python_tuple)) # < classe 'Tuple' > print ('
python_tuple:', python_tuple) # python_tuple: (1, 2, 3, 4, 5)



numpy_array_from_tuple = np.array (python_tuple) print (type (numpy_array_from_tuple)) #
< classe 'numpy.ndarray' > print ('numpy_array_from_tuple:', numpy_array_from_Tuple) # nu
mpy_array_from_tuple: [1 2 3 4 5]
```

Forme du tableau nu

La méthode de forme fournit la forme du réseau sous forme de tuple. Le premier est la ligne et la seconde est la colonne. Si le tableau n'est qu'une dimension, il renvoie la taille du tableau.

```
nums = np.array ([1, 2, 3, 4, 5]) print (nums) print ('forme de nums:', nums.s
hape) print (numpy_two_dimensional_list) print ('forme de numpy_two_dim
ensional_list:', numpy_two_dimenteal_list.list.shape) troisième) troisième_or
y_ary athy_ np.array ([[0, 1, 2, 3], [4,5,6,7], [8,9,10, 11]]) imprimer (trois_by
_four_array.shape)
```

```
[1 2 3 4 5] Forme de Nums: (5,) [[0 1 2] [3 4 5] [6 7 8]] Forme de Numpy
_two_dimensional_list: (3, 3) (3, 4)
```

Type de données du tableau Numpy

Type de types de données: str, int, float, complexe, bool, liste, aucun

```
int_lists = [-3, -2, -1, 0, 1, 2,3] int_array = np.array (int_lists) float_ar
ray = np.array (int_lists, dtype = float)



print (int_array) imprimer (int_array.
dtype) imprimer (float_array) imprim
er (float_array.dtype)
```

```
    [-3 -2 -1  0  1  2  3]
    int64
    [-3. -2. -1.  0.  1.  2.  3.]
    float64
```

Taille d'un tableau nu

En Numpy pour connaître le nombre d'articles dans une liste de tableaux Numpy, nous utilisons la taille

```
numpy_array_from_list = np.array ([1, 2, 3, 4, 5])
Two_dimensional_list = np.array ([[0, 1, 2], [3, 4, 5], [6, 7, 8]])



print ('la taille:', numpy_array_from_list.size) # 5 print ('la taille:', deux_dime
nsional_list.size) # 3
```

```
La taille: 5
La taille: 9
```

Fonctionnement mathématique en utilisant Numpy

Numpy Array n'est pas exactement comme la liste Python. Pour effectuer un fonctionne ment mathématique dans la liste Python, nous devons traverser les éléments, mais Nump y peut permettre de faire n'importe quelle opération mathématique sans boucle. Fonction nement mathématique:

- Addition (+)
- soustra ction (-)
- multiplicatio n (*)
- division (/)
- m odules (%)
- Division de plancher (//)
- Expo nential (**)

Ajout

Fonctionnement mathématique

```
# Addition
```

```
numpy_array_from_list = np.array ([1, 2, 3, 4, 5])
print ('Array d'origine:', numpy_array_from_list) ten_plus_original = num
py_array_from_list + 10 print (ten_plus_original)
```

```
Tableau d'origine: [1 2 3 4 5] [11 12 13 14 15]
```

Soustraction

```
# Soustraction
numpy_array_from_list = np.array ([1, 2, 3, 4, 5])
print ('Array original:', numpy_array_from_list) ten_minus_original = nu
mpy_array_from_list - 10 print (ten_minus_original)
```

```
Tableau d'origine: [1 2 3 4 5]
[-9 -8 -7 -6 -5]
```

Multiplication

```
# Multiplication
numpy_array_from_list = np.array ([1, 2, 3, 4, 5])
print ('Array original:', numpy_array_from_list) ten_times_original = num
py_array_from_list * 10 print (ten_times_original)
```

```
Tableau d'origine: [1 2 3 4 5]
[10 20 30 40 50]
```

Division

```
# Division
numpy_array_from_list = np.array ([1, 2, 3, 4, 5])
print ('Array original:', numpy_array_from_list) Ten_times_original = nu
mpy_array_from_list / 10 print (ten_times_original)
```

```
Tableau d'origine: [1 2 3 4 5]
[0,1 0,2 0,3 0,4 0,5]
```

Module

```
# Module; Trouver le reste Numpy_Array_From_List = np.array ([1, 2, 3,
4, 5])
```

```
print ('Array d'origine:', numpy_array_from_list) Ten_times_original = nu
mpy_array_from_list% 3 imprimer (ten_times_original)
```

```
Tableau d'origine: [1 2 3 4 5] [1 2 0 1 2]
```

Division des étages

```
# Division du sol: le résultat de la division sans le reste numpy_array_from_list = np.array
([1, 2, 3, 4, 5]) imprimer ('Array original:', numpy_array_from_list) ten_times_original =
```

Exponentiel

```
# Exponentielle trouve un certain nombre de puissance d'un autre: numpy_array_from_li
st = np.array ([1, 2, 3, 4, 5]) print ('Array original:', numpy_array_from_list) ten_times_o
riginal =
```

```
Tableau d'origine: [1 2 3 4 5] [1 4 9 16 25]
```

Vérification des types de données

```
#Int, nombres de flottants numpy_int_arr = np.array ([1,2,3,4]) numpy_float_arr = np.arra
y ([1.1, 2.0,3.2]) Numpy_bool_arr = np.array ([- 3, -2, 0, 1,2,3], dtype = ')
```

```
print (numpy_int_arr.dtype) print (numpy
_float_arr.dtype)
print (numpy_bool_arr.dtype)
```

```
    int64
    float64
    bool
```

Types de conversion

Nous pouvons convertir les types de données de tableau Numpy

1. Int pour flotter

```
numpy_int_arr = np.array ([1,2,3,4], dtype = 'float') numpy_int_arr
```

```
Array ([1., 2., 3., 4.])
```

2. Flotter à int

```
numpy_int_arr = np.array ([1., 2., 3., 4.], dtype = 'int') numpy_int_arr
```

```
Array ([1, 2, 3, 4])
```

3. Int ot booléen

```
np.array ([- 3, -2, 0, 1,2,3], dtype = 'bool')
```

```
Array ([vrai, vrai, faux, vrai, vrai, vrai])
```

4. int à str

```
numpy_float_list.astype ('int'). Astype ('str')
```

```
array(['1', '2', '3'], dtype='<U21')
```

Tableaux multidimensionnels

```
# 2 Dimension Array Two_dimension_Array = np.array ([(1,2,3), (4,5,6), (7,8,9)]) impri
mer (type (deux_dimension_array)) imprimer (deux_dimension_array) print ('Type de do
nnées:', deux_dimension_array.dtype)
```

```
< classe 'numpy.ndarray' > [[1 2 3] [4 5 6]
[7 8 9]] Forme: (3, 3) Taille: 9 Type de do
nnées: int64
```

Obtenir des articles d'un tableau Numpy

```
# 2 Dimension Array Two_dimension_Array = np.array ([[1,2,3], [4,5,6], [7,8,9]])) Prem
ier_Row = Two_dimension_Array [0]
```

```
second_row = deux_dimension_array [1]
tiers_row = Two_dimension_Array [2] print ('premiè
re ligne:', first_row) print ('deuxième ligne:', second_
row) print ('troisième ligne:', tiers_row)
```

```
    Première rangée: [1 2 3]
    Deuxième rangée: [4 5 6]
    Troisième rangée: [7 8 9]
```

```
first_column = deux_dimension_array [:, 0] second_column
= deux_dimension_array [:, 1]
tiers_column = deux_dimension_array [:, 2] print ('première
colonne:', first_column) print ('deuxième colonne:', second_
column) print ('troisième colonne:', troisième_column) print
(two_dimension_array)
```

```
  Première colonne: [1 4 7]
  Deuxième colonne: [2 5 8]
  Troisième colonne: [3 6 9]
  [[1 2 3]
  [4 5 6]
  [7 8 9]]
```

Tranchant le tableau numpy

Trancher dans Numpy est similaire au tranchage dans la liste Python

```
Two_dimension_Array = np.array ([[1,2,3], [4,5,6], [7,8,9]]) premier_two_rows_and_col
umns = Two_dimension_Array [0: 2, 0: 2] print (first_two_rows_and_colums)
```

```
    [[1 2]
      [4 5]]
```

Comment inverser les lignes et tout le tableau?

```
deux_dimension_array [::]
```

```
Array ([[1, 2, 3], [4, 5, 6], [7, 8, 9
]])
```

Inverser les positions de ligne et de colonne

```
    deux_dimension_array = np.array ([[1,2,3], [4,5,6], [7,8,9]])
    deux_dimension_array [:: - 1, :: - 1]
```

```
Array ([[9, 8, 7], [6, 5, 4], [3, 2, 1
]])
```

Comment représenter les valeurs manquantes?

```
imprimer (deux_dimension_array)
deux_dimension_array [1,1] = 55
deux_dimension_array [1,2] = 44
imprimer (deux_dimension_array)
```

```
[[1 2 3] [4 5 6] [7 8 9]] [[1 2 3] [4 55 44] [7 8 9]] # Numpy Zeros # Numpy.zeros (
Forme, Dtype = Float, Ordre = 'C') Numpy_ZEROES = np.zeros ((3,3), dtype = int
, ordre = 'c') numpy_zeroes
```

```
Array ([[0, 0, 0], [0, 0, 0], [0, 0, 0
]])
```

```
# Zéros numpy
numpy_ones = np.ones ((3,3), dtype = int, ordre = 'c')
imprimer (Numpy_ones) [[
1 1 1] [1 1 1] [1 1 1]]
```

```
Twoes = numpy_ones * 2
```

```
# Reshape # Numpy.Reshape (), Numpy.flatten () first_shape = n
p.array ([(1,2,3), (4,5,6)]))

print (first_shape) remodelé = first_shape.Reshape (3,
2)
imprimer (remodelé)
[[1 2 3] [4 5 6]] [[1 2
]
```

```
     [3 4]
     [5 6]]
```

```
flattened = reshaped.flatten()
flattened
```

```
    array([1, 2, 3, 4, 5, 6])
    ## Horitzontal Stack
    np_list_one = np.array([1,2,3])
    np_list_two = np.array([4,5,6])

    print(np_list_one + np_list_two)

    print('Horizontal Append:', np.hstack((np_list_one,
np_list_two)))
```

```
    [5 7 9]
    Horizontal Append: [1 2 3 4 5 6]
```

```
    ## Vertical Stack
    print('Vertical Append:', np.vstack((np_list_one,
np_list_two)))
```

```
   Vertical Append: [[1 2 3]
     [4 5 6]]
```

Generating Random Numbers

```
    # Generate a random float  number
    random_float = np.random.random()
    random_float
```

```
    0.018929887384753874
```

```
    # Generate a random float  number
    random_floats = np.random.random(5)
    random_floats
```

```
  array([0.26392192, 0.35842215, 0.87908478, 0.41902195,
0.78926418])
```

```
  # Generating a random integers between 0 and 10

    random_int = np.random.randint(0, 11)
    random_int
```

```
    4
```

```
   # Generating a random integers between 2 and 11, and
creating a one row array
    random_int = np.random.randint(2,10, size=4)
    random_int
```

```
    array([8, 8, 8, 2])
```

```
    # Generating a random integers between 0 and 10
    random_int = np.random.randint(2,10, size=(3,3))
    random_int
```

```
  array([[3, 5, 3],
         [7, 3, 6],
         [2, 3, 3]])
```

Nombres aléatoires de Generationg

```
# np.random.normal (mu, sigma, taille) normal_array = np.random.orma
l (79, 15, 80)
normal_array      array([ 89.49990595,  82.06056961, 107.21445842,   38.693070
86,          47.85259157,  93.07381061,  76.40724259,   78.55675184,          7
2.17358173, 47.9888899, 65.10370622, 76.29696568, 95.58234254, 68.1489721
3, 38.75862686, 122.5587927, 67.0762565, 95.739908644, 81.9745563, 95.7399
0864, 81.9745563,   92.54264805,          59.37035153,  77.76828101,  52.3075
2166,   64.43109931,          62.63695351,  90.04616138,  75.70009094,   49.87
586877,          80.22002414,  68.56708848,  76.27791052,   67.24343975,
81.86363935,  78.22703433, 102.85737041,   65.15700341,          84.87033426
,  76.7569997 ,  64.61321853,   67.37244562, 74.4068773, 58.65119655, 71.6648
8727, 53.42458179, 70.26872028, 60.96588544, 83.56129414, 72.14255326, 83,
56129414, 72.14255326
```

```
            81.00787609,  71.81264853,  72.64168853,
86.56608717,
            94.94667321,  82.32676973,  70.5165446 ,
85.43061003,
            72.45526212,  87.34681775,  87.69911217,
103.02831489,
            75.28598596,  67.17806893,  92.41274447,
101.06662611,
            87.70013935,  70.73980645,  46.40368207,
50.17947092,
            61.75618542,  90.26191397,  78.63968639,
70.84550744,
            88.91826581, 103.91474733,  66.3064638 ,
79.49726264,
            70.81087439,  83.90130623,  87.58555972,
59.95462521])
```

Numpy et statistiques

Importer Matplotlib.pyplot as PLT Import SeaBorn comme sns sns.
set () plt.hist (normal_array, couleur = "gris", bacs = 50)

```
    (array([2., 0., 0., 0., 1., 2., 2., 0., 2., 0., 0., 1.,
2., 2., 1., 4., 3.,
            4., 2., 7., 2., 2., 5., 4., 2., 4., 3., 2., 1.,
5., 3., 0., 3., 2.,
            1., 0., 0., 1., 3., 0., 1., 0., 0., 0., 0., 0.,
0., 0., 0., 1.]),
     array([ 38.69307086,  40.37038529,  42.04769973,
43.72501417,
             45.4023286 ,  47.07964304,  48.75695748,
50.43427191,
             52.11158635,  53.78890079,  55.46621523,
57.14352966,
             58.8208441 ,  60.49815854,  62.17547297,
63.85278741,
             65.53010185,  67.20741628,  68.88473072,
70.56204516,
             72.23935959,  73.91667403,  75.59398847,
77.27130291,
             78.94861734,  80.62593178,  82.30324622,
83.98056065,
             85.65787509,  87.33518953,  89.01250396,
90.6898184 ,
             92.36713284,  94.04444727,  95.72176171,
97.39907615,
```

```
99.07639058, 100.75370502, 102.43101946,  104.1083339 ,            105.78564833,
107.46296277, 109.14027721,  110.81759164,            112.49490608, 114.1722205
2, 115.84953495, 117.52684939, 119.20416383, 120.88147826, 122.5587927]), < U
ne liste de 50 objets de patch >)
```

Matrice en numpy

```
quatre_by_four_matrix = np.matrix (np.ones ((4,4), dtype = float)))
```

```
quatre_by_four_matrix
```

```
matrix([[1., 1., 1., 1.],
            [1., 1., 1., 1.],
            [1., 1., 1., 1.],
            [1., 1., 1., 1.]])
```

```
np.asarray (quatre_by_four_matrix) [2] = 2 quatre_by_fou
r_matrix
```

```
matrix([[1., 1., 1., 1.],
            [1., 1., 1., 1.],
            [2., 2., 2., 2.],
            [1., 1., 1., 1.]])
```

Numpy numpy.arange ()

Qu'est-ce que l'organisation?

Parfois, vous voulez créer des valeurs qui sont également espacées dans un intervalle d
éfini. Par exemple, vous souhaitez créer des valeurs de 1 à 10; vous pouvez utiliser la f
onction numpy.arange ()

```
# Création de liste à l'aide de la plage (démarrage, arrêt, étape) LST = plage
(0, 11, 2) LST
```

```
gamme (0, 11, 2)

pour l dans lst: impri
mer (l)
```

```
2
```

```
    4
    6
    8
    10
```

```
# Similaire à la gamme Arange Numpy.arange (start, stop, étape) Whole_Numbers = np
.arange (0, 20, 1) Whole_Numbers
```

```
Array ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
Natural_numbers = np.arange (1, 20, 1) Natural_numbers
```

```
Odd_Numbers = np.arange (1, 20, 2)
Odd_numbers
```

```
    array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])
```

```
Même_Numbers = np.arange (2, 20, 2) pair_number
s
```

```
    array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

Création d'une séquence de nombres à l'aide de lispace

```
# numpy.linspace () # numpy.logspace () en python avec exem
ple
# Par exemple, il peut être utilisé pour créer 10 valeurs de 1 à 5 également espacées.

np.linspace (1.0, 5.0, num = 10)
```

```
Array ([1., 1.44444444, 1.88888889, 2.33333333, 2.7777778, 3.22222222, 3.66666667, 4.11
11111, 4.55555556, 5.

])
```

```
# pour ne pas inclure la dernière valeur dans l'interval NP.Linspace (1.0,
5.0, num = 5, endpoint = false)
```

```
Array ([1., 1,8, 2,6, 3,4, 4.2])
```

```
# LogSpace
# LogSpace returns even spaced numbers on a log scale.
Logspace has the same parameters as np.linspace.

# Syntax:

# numpy.logspace(start, stop, num, endpoint)

np.logspace(2, 4.0, num=4)
```

```
array([  100.        ,   464.15888336,  2154.43469003, 10000.
])
```

```
# to check the size of an array
x = np.array([1,2,3], dtype=np.complex128)
```

```
x
```

```
    array([1.+0.j, 2.+0.j, 3.+0.j])
```

```
x.itemsize
```

```
16
```

```
# indexing and Slicing NumPy Arrays in Python
np_list = np.array([(1,2,3), (4,5,6)])
np_list
```

```
    array([[1, 2, 3],
           [4, 5, 6]])
```

```
print('First row: ', np_list[0])
print('Second row: ', np_list[1])
```

```
    First row:  [1 2 3]
    Second row:  [4 5 6]
```

```
print('First column: ', np_list[:,0])
print('Second column: ', np_list[:,1])
print('Third column: ', np_list[:,2])
```

```
    First column:  [1 4]
    Second column:  [2 5]
    Third column:  [3 6]
```

Fonctions statistiques Numpy avec exemple

Numpy a des fonctions statistiques très utiles pour trouver un minimum, maximum, moyen, médian, centile, écart-type et variance, etc. par rapport aux éléments donnés dans le tableau. Les fonctions sont expliquées comme suit − La fonction statistique Numpy est équipée de la fonction statistique robuste comme indiqué ci-dessous

- Fonctions Numpy de min np.min () o ma x np.max () o moyen np.mean () o média n np.median () o variété o centile o écart-type np.std ()

```
np_normal_dis = np.random.normal (5, 0,5, 100) np_normal_dis ## Min, Max, Mean, Median, SD Print ('min:', deux_dimension_array.min ()) print ( 'MAX:', Two_dimension_Array.Max ()) print ('MEX:', Two_dimension_Ar ray. ', deux_dimension_array.mean ()) # print (' median: ', two_dimension_ array.median ()) print (' sd: ', deux_dimension_array.std ())
```

```
Min: 1 Max: 55 Moyenne: 14.7777777
7777779 SD: 18.913709183069525
```


```
Min: 1 Max: 55 Moyenne: 14.7777777
7777779 SD: 18.913709183069525
```


```
print(two_dimension_array)  print('Column with minim um: ',  np.amin(two_dimension_array,axis=0))  print('C olumn with maximum: ',  np.amax(two_dimension_arra y,axis=0))  print('=== Row ==')  print('Row with mini mum: ', np.amin (deux_dimension_array, axe = 1)) prin t ('ligne avec maximum:', np.amax (deux_dimension_ar ray, axe = 1))
```

```
[[1 2 3] [4 55 44] [7 8 9]] Colonne avec minimum
: [1 2 3] colonne avec maximum: [7 55 44] === li
gne == avec minimum: [1 4 7] ligne avec maxim
um: [3 55 9]
```

Comment créer des séquences répétitives?

```
a = [1,2,3]

# Répéter tout de 'a' deux fois imprimé ('tuile:', n
p.tile (a, 2))

# Répéter chaque élément de 'a' deux fois imprimer ('répéte
r:', np.repeat (a, 2))
```

```
Carreau: [1 2 3 1 2 3] répéter: [1 1
2 2 3 3]
```

Comment générer des nombres aléatoires?

```
# Un nombre aléatoire entre [0,1) One_Random_Nu
m = np.random.random ()
one_random_in = np.random print (one
_random_num)
```

```
0,6149403282678213  0,476
3968133790438  0,4763968
133790438
```

```
# Nombres aléatoires entre [0,1) de forme 2,3 r = np.random.rando
m (taille = [2,3]) print (r)
```

```
[[0,13031737 0,4429537 0,1129527] [0,76811539 0,88256594 0,6754075]] imprimer (np.r
andom.choice (['a', 'e', 'i', 'o', 'u'], size = 10))))
```

```
['u' 'o' 'o' 'i' 'e' 'e' 'u' 'o' 'u' 'a']
```

```
['i' 'u' 'e' 'o' 'a' 'i' 'e' 'u' 'o' 'i']
```

```
['iueoaieuoi']
```

```
## Random numbers between [0, 1] of shape 2, 2
rand = np.random.rand(2,2)
rand
```

```
array([[0.97992598, 0.79642484],
       [0.65263629, 0.55763145]])
```

```
rand2 = np.random.randn(2,2)
rand2
```

```
array([[ 1.65593322, -0.52326621],
       [ 0.39071179, -2.03649407]])
```

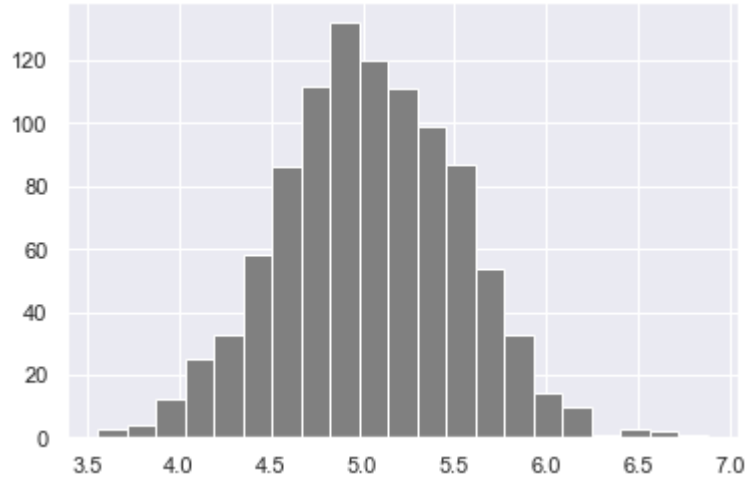
```
# Random integers between [0, 10) of shape 2,5
rand_int = np.random.randint(0, 10, size=[5,3])
rand_int
```

```
array([[0, 7, 5],
       [4, 1, 4],
       [3, 5, 3],
       [4, 3, 8],
       [4, 6, 7]])
```

```
from scipy import stats
np_normal_dis = np.random.normal(5, 0.5, 1000) # mean,
standard deviation, number of samples
np_normal_dis
## min, max, mean, median, sd
print('min: ', np.min(np_normal_dis))
print('max: ', np.max(np_normal_dis))
print('mean: ', np.mean(np_normal_dis))
print('median: ', np.median(np_normal_dis))
print('mode: ', stats.mode(np_normal_dis))
print('sd: ', np.std(np_normal_dis))
```

```
    min:  3.557811005458804
    max:  6.876317743643499
    mean:  5.035832048106663
    median:  5.020161980441937
    mode:  ModeResult(mode=array([3.55781101]),
count=array([1]))
    sd:  0.489682424165213
```

```
plt.hist(np_normal_dis, color="grey", bins=21)
plt.show()
```




```
# Numpy.dot (): produit DOT en Python en utilisant Numpy # Dot Produit

# Numpy est une bibliothèque puissante pour le calcul des matrices. Par exemple, vous pouvez calculer le produit DOT avec NP.DOT

# Syntaxe

# numpy.dot (x, y, out = Aucun)
```

Algèbre linéaire

1. Produit DOT

```
## Algèbre linéaire ### Dot Produit: produit de deux table
aux f = np.array ([1,2,3]) g = np.array ([4,5,3]) ### 1 * 4 +
2 * 5 + 3 * 6 np.dot (f, g) # 23
```

Multiplication de la matrice Numpy avec np.matmul ()

```
### Matmul: Matruc Produit de deux tableaux h = [[1,2], [3,4
]] i = [[5,6], [7,8]] ### 1 * 5 + 2 * 7 = 19 np.matmul (h, i) Ab
at ([[19, 22], [43, 50])
```

```
## Déterminant 2 * 2 Matrix ### 5 * 8-7 *
6NP.Linalg.det (i)
```

```
np.linalg.det (i)
```

```
-1.999999999999999
```

```
Z = np.zeros ((8,8))
Z [1 :: 2, :: 2] = 1 z [:: 2
,1 :: 2] = 1
```

```
Z
```

```
array([[0., 1., 0., 1., 0., 1., 0., 1.],
       [1., 0., 1., 0., 1., 0., 1., 0.],
       [0., 1., 0., 1., 0., 1., 0., 1.],
       [1., 0., 1., 0., 1., 0., 1., 0.],
       [0., 1., 0., 1., 0., 1., 0., 1.],
       [1., 0., 1., 0., 1., 0., 1., 0.],
       [0., 1., 0., 1., 0., 1., 0., 1.],
       [1., 0., 1., 0., 1., 0., 1., 0.]])
```

```
new_list = [x + 2 pour x dans la plage (0, 11)]
```

```
new_list
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
np_arr = np.array (plage (0, 11))
np_arr + 2
```

Array ([2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

Nous utilisons l'équation linéaire pour les quantités qui ont une relation linéaire. Voyons l' exemple ci-dessous:

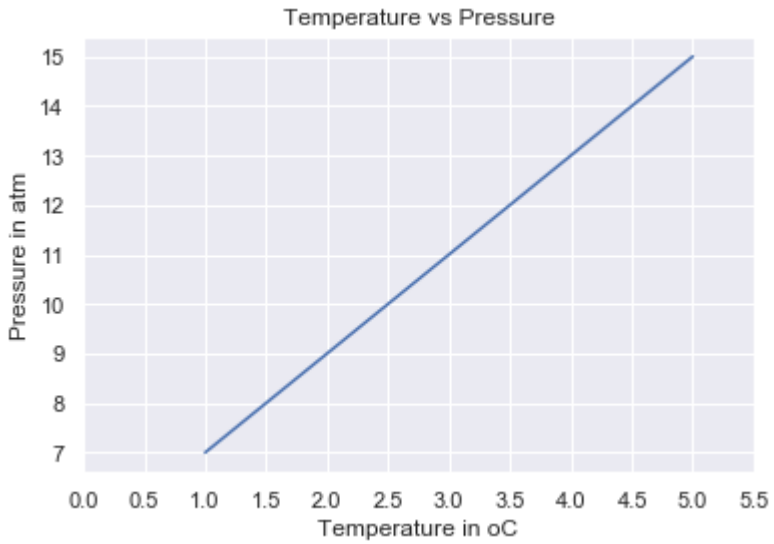
```
Temp = np.array ([1,2,3,4,5]) Pression = te
mp * 2 + 5 pression
```

Array ([7, 9, 11, 13, 15])

```
plt.plot (temp, pression) plt.xlabel ('température en oc')
plt.ylabel ('pression dans atm') plt.title ('température vs
pression') plt.xticks (np.arange (0, 6, étape = 0,5)))



plt.show ()
```

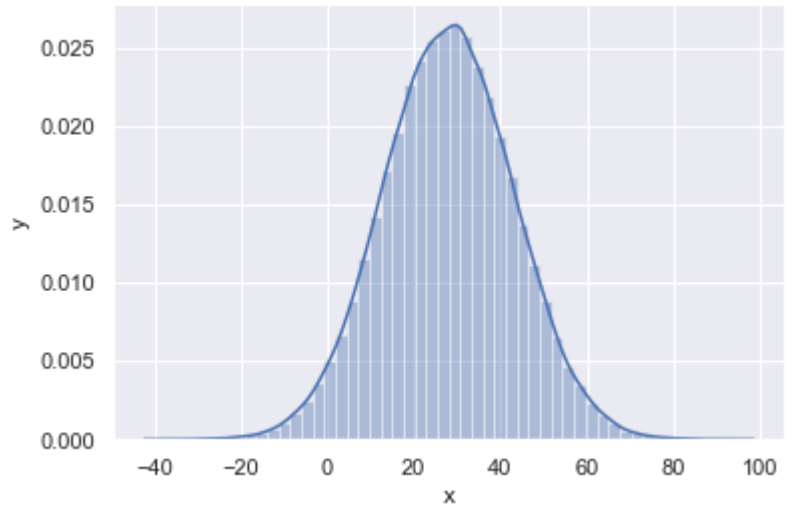


Pour dessiner la distribution normale gaussienne en utilisant Numpy. Comme vous pouvez le voir ci-dessous, le Numpy peut générer des nombres aléatoires. Pour créer un échantillon aléatoire, nous avons besoin de la moyenne (MU), Sigma (écart-type), Mumber of Data Poi nts.

```
mu = 28 Sigma = 15 écha
ntillons = 100000



x = np.random.normal (mu, sigma, échantillons) ax = sns.dist
plot (x);  ax.set (xlabel = "x", ylabel = 'y') plt.show ()
```



Résumé

Pour résumer, les principales différences avec les listes de python sont:

1. Les tableaux prennent en charge les opérations vectorielles, tandis que les listes ne le font pas.

2. Une fois un tableau créé, vous ne pouvez pas modifier sa taille. Vous devrez créer un no uveau tableau ou écraser celui existant. 3. Chaque tableau a un et un seul dtype. Tous les articles dedans doivent être de ce DTYPE.

4. Un tableau Numpy équivalent occupe beaucoup moins d'espace qu'une liste de listes python. 5. Les tableaux Numpy prennent en charge l'indexation booléenne.

Exercices: Jour 24

1. Répétez tous les exemples

Félicitations!