

Science des données - Python

Germain Forestier, PhD

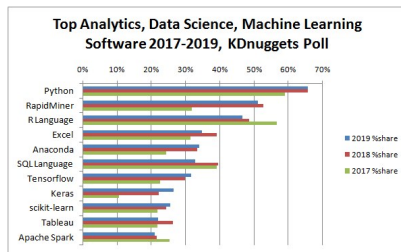
<https://germain-forestier.info>

Université de Haute-Alsace

Introduction à Python

Science des données

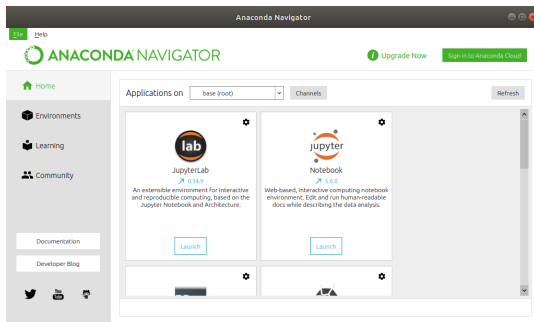
- ▷ Possibilité de faire de la science des données avec tous les langages : C, C++, Java, etc.
- ▷ Langages dédiés : R et Python



source : <https://www.kdnuggets.com/>

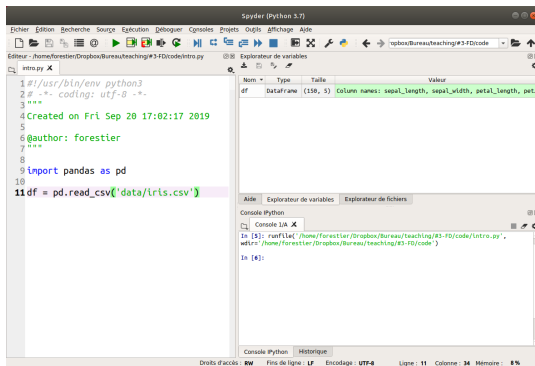
Anaconda

- ▷ Distribution Python (Python 2.x vs. Python 3.x)
- ▷ De nombreux paquets pré-installés
- ▷ Gestion automatique des dépendances entre paquets (conda)



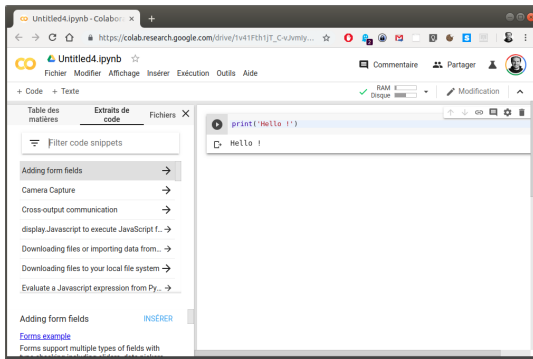
Spyder

- ▶ Éditeur simple pour Python (inclus dans Anaconda)
- ▶ Coloration syntaxique, exécution, etc.
- ▶ Console iPython, explorateur de variables, etc.



Google Colab

- ▷ Nécessite un compte Google
- ▷ Permet d'exécuter du code sur les serveurs Google
- ▷ Pratique si pas de possibilité d'installer des logiciels



Types de données en Python

- ▷ `float` : nombre réel
- ▷ `int` : nombre entier
- ▷ `str` : chaîne de caractère, texte
- ▷ `bool` : vrai ou faux
- ▷ `type()` : renvoie le type d'une variable

Exemple :

```
1 height = 1.73
2 tall = True
3
4 In [1]: height = 1.79
5 In [2]: weight = 68.7
6
7 In [6]: bmi = weight / height ** 2
8
9 In [7]: bmi
10 Out[7]: 21.4413
```

Exercice

- ▷ Créer une variable `factor` égale à 1.10
- ▷ Utiliser `savings` et `factor` pour calculer le montant obtenu après 7 ans
- ▷ Afficher le résultat

```
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable factor
5
6 # Calculate result
7
8 # Print out result
```


Exercice

- ▷ Exécuter le code ci-dessous et essayer de comprendre l'erreur
- ▷ Corriger le code à l'aide de la fonction `str()`
- ▷ Convertir la variable `pi_string` en float dans une nouvelle variable `pi_float`

```
1 # Definition of savings and result
2 savings = 100
3 result = 100 * 1.10 ** 7
4
5 # Fix the printout
6 print("I started with $" + savings + " and now have $" + result + ". Awesome!")
7
8 # Definition of pi_string
9 pi_string = "3.1415926"
10
11 # Convert pi_string into float: pi_float
```

Exercice

- ▷ Laquelle de ces instructions Python lancera une erreur ?

Réponses

1. "I can add integers, like " + str(5) + " to strings."
2. "I said " + ("Hey " * 2) + "Hey!"
3. "The correct answer to this multiple choice exercise is answer number " + 2
4. True + False

Les listes

Problème

- ▷ Science des données = de nombreuses données
- ▷ Tailles de toute une famille

```
1 In [3]: height1 = 1.73
2 In [4]: height2 = 1.68
3 In [5]: height3 = 1.71
4 In [6]: height4 = 1.89
```

- ▷ Pas très pratique...

Les listes en Python

- ▷ Exemple de liste :

```
1 fam = [1.73, 1.68, 1.71, 1.89]
```

- ▷ Permet de nommer une collection de valeurs
- ▷ Peut contenir n'importe quel type de données
- ▷ Peut contenir plusieurs types de données :

```
1 fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
```

- ▷ Possibilité de créer une liste de listes :

```
1 fam2 = ["liz", 1.73],  
2       ["emma", 1.68],  
3       ["mom", 1.71],  
4       ["dad", 1.89]]
```

Exercice

- ▷ Créer une liste, `areas`, qui contiendra la taille des différentes pièces
- ▷ Afficher la liste à l'aide de la fonction `print()`

```
1 # area variables (in square meters)
2 hall = 11.25
3 kit = 18.0
4 liv = 20.0
5 bed = 10.75
6 bath = 9.50
7
8 # Create list areas
9
10 # Print areas
```

Exercice

▷ Finir le code pour que chaque pièce possède son propre nom

```
1 # area variables (in square meters)
2 hall = 11.25
3 kit = 18.0
4 liv = 20.0
5 bed = 10.75
6 bath = 9.50
7
8 # Adapt list areas
9 areas = [hall, kit, "living room", liv, bed, "bathroom", bath]
10
11 # Print areas
```

Exercice

▷ Parmi les instructions suivantes, lesquelles sont valides en Python ?

```
1 A. [1, 3, 4, 2]
2 B. [[1, 2, 3], [4, 5, 7]]
3 C. [1 + 2, "a" * 5, 3]
```

Réponses

1. A, B et C
2. B
3. B et C
4. C

Exercice

- ▶ Terminer la liste de listes pour qu'elle contienne toutes les pièces puis afficher la variable `house` et son type (avec la fonction `type()`)

```
1 # area variables (in square meters)
2 hall = 11.25
3 kit = 18.0
4 liv = 20.0
5 bed = 10.75
6 bath = 9.50
7
8 # house information as list of lists
9 house = ["hallway", hall],
10         ["kitchen", kit],
11         ["living room", liv]]
12
13 # Print out house
14
15 # Print out the type of house
```

Récupération d'un élément d'une liste

```
1 In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
2 In [3]: fam[3]
3 Out[3]: 1.68
4 In [5]: fam[-1]
5 Out[5]: 1.89
```

Découpage d'une liste (slicing)

```
1 In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
2
3 In [8]: fam[3:5]
4 Out[8]: [1.68, 'mom']
```

[début, fin] ([inclus, exclus])

```
1 In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
2
3 In [10]: fam[:4]
4 Out[10]: ['liz', 1.73, 'emma', 1.68]
5
6 In [11]: fam[5:]
7 Out[11]: [1.71, 'dad', 1.89]
```

Exercice

- ▷ Afficher le deuxième élément de la liste `areas` (11.25)
- ▷ Extraire et afficher le dernier élément de `areas` (9.5), utiliser un index négatif
- ▷ Sélectionner et afficher l'aire de "living room"

```
1 # Create the areas list
2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom",
3         10.75, "bathroom", 9.50]
4
5 # Print out second element from areas
6
7 # Print out last element from areas
8
9 # Print out the area of the living room
```

Exercice

- ▷ Créer une variable `eat_sleep_area` qui contiendra la somme des aires de "kitchen" et "bedroom"
- ▷ Afficher cette nouvelle variable

```
1 # Create the areas list
2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom",
3         10.75, "bathroom", 9.50]
4 # Sum of kitchen and bedroom area: eat_sleep_area
5
6 # Print the variable eat_sleep_area
```

Exercice

- ▷ Utiliser la sélection pour créer une liste ,downstairs, qui contient les 6 premiers éléments de areas
- ▷ Faire la même chose pour créer upstairs qui contiendra les 4 derniers éléments de areas

```
1 # Create the areas list
2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom",
3         10.75, "bathroom", 9.50]
4 # Use slicing to create downstairs
5
6 # Use slicing to create upstairs
7
8 # Print out downstairs and upstairs
```

Exercice

- Utiliser la sélection pour créer les listes downstairs et upstairs en utilisant uniquement les indexes nécessaires

```
1 # Create the areas list
2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom",
          10.75, "bathroom", 9.50]
3
4 # Alternative slicing to create downstairs
5
6 # Alternative slicing to create upstairs
```

Exercice

▷ Que retournerait `house[-1][1]` ?

1. Un réel : la taille de "kitchen"
2. Une chaîne: "kitchen"
3. Un réel : a taille de "bathroom"
4. Une chaîne : "bathroom"

Changer une valeur dans une liste

```
1 In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
2
3 In [2]: fam
4 Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
5
6 In [3]: fam[7] = 1.86
7
8 In [4]: fam
9 Out[4]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
10
11 In [5]: fam[0:2] = ["lisa", 1.74]
12
13 In [6]: fam
14 Out[6]: ['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```


Ajouter et retirer un élément d'une liste

```
1 In [7]: fam + ["me", 1.79]
2 Out[7]: ['lisa', 1.74, 'emma', 1.68,
3         'mom', 1.71, 'dad', 1.86, 'me', 1.79]
4
5 In [8]: fam_ext = fam + ["me", 1.79]
6 In [9]: del(fam[2])
7
8 In [10]: fam
9 Out[10]: ['lisa', 1.74, 1.68, 'mom', 1.71, 'dad', 1.86]
10
11 In [11]: del(fam[2])
12
13 In [12]: fam
14 Out[12]: ['lisa', 1.74, 'mom', 1.71, 'dad', 1.86]
```

Exercise

- ▷ Changer la taille de la piscine en 10.50 à la place de 9.50
- ▷ Changer la pièce "living room" en "chill zone"

```
1 # Create the areas list
2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom",
          10.75, "bathroom", 9.50]
3
4 # Correct the bathroom area
5
6 # Change "living room" to "chill zone"
```

Exercice

- ▷ Utiliser l'opérateur `+` pour rajouter `["poolhouse", 24.5]` à la fin de la liste `areas`. Stocker le résultat dans `areas_1`
- ▷ Ajouter un garage à `areas_1` de taille 15.45 et stocker le résultat dans `areas_2`

```
1 # Create the areas list and make some changes
2 areas = ["hallway", 11.25, "kitchen", 18.0, "chill zone", 20.0,
3         "bedroom", 10.75, "bathroom", 10.50]
4
5 # Add poolhouse data to areas, new list is areas_1
6
7 # Add garage data to areas_1, new list is areas_2
```

Exercice

```
1 areas = ["hallway", 11.25, "kitchen", 18.0,  
2         "chill zone", 20.0, "bedroom", 10.75,  
3         "bathroom", 10.50, "poolhouse", 24.5,  
4         "garage", 15.45]
```

Quelle commande exécuter pour retirer la piscine ?

1. `del(areas[10]); del(areas[11])`
2. `del(areas[10:11])`
3. `del(areas[-4:-2])`
4. `del(areas[-3]); del(areas[-4])`

Fonctions et packages

Les fonctions

```
1 In [1]: fam = [1.73, 1.68, 1.71, 1.89]
2
3 In [2]: fam
4 Out[2]: [1.73, 1.68, 1.71, 1.89]
5
6 In [3]: max(fam)
7 Out[3]: 1.89
8
9 In [4]: tallest = max(fam)
10
11 In [5]: tallest
12 Out[5]: 1.89
13
14 In [6]: round(1.68, 1)
15 Out[6]: 1.7
16
17 In [8]: help(round)
```

Exercice

- ▷ Utiliser `print()` avec `type()` pour afficher le type de `var1`
- ▷ Utiliser `len()` pour obtenir la longueur de `var1`
- ▷ Utiliser `int()` pour convertir `var2` en entier. Stocker le résultat dans `out2`

```
1 # Create variables var1 and var2
2 var1 = [1, 2, 3, 4]
3 var2 = True
4
5 # Print out type of var1
6
7 # Print out length of var1
8
9 # Convert var2 to an integer: out2
```

Exercice

- ▷ Utiliser `+` pour fusionner les contenus de `first` et `second` et stocker le résultat dans une liste `full`
- ▷ Appeler `sorted()` sur `full` et spécifier l'argument `reverse` à `True`. Enregistrer le résultat dans `full_sorted`.
- ▷ Terminer par afficher `full_sorted`.

```
1 # Create lists first and second
2 first = [11.25, 18.0, 20.0]
3 second = [10.75, 9.50]
4
5 # Paste together first and second: full
6
7 # Sort full in descending order: full_sorted
8
9 # Print out full_sorted
```


Méthodes list

```
1 In [4]: fam
2 Out[4]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
3
4 In [5]: fam.index("mom")
5 Out[5]: 4
6
7 In [6]: fam.count(1.73)
8 Out[6]: 1
```

Méthodes str

```
1 In [7]: sister
2 Out[7]: 'liz'
3
4 In [8]: sister.capitalize()
5 Out[8]: 'Liz'
6
7 In [9]: sister.replace("z", "sa")
8 Out[9]: 'lisa'
```

Exercice

- ▷ Utiliser la méthode `upper()` sur "room" et stocker le résultat dans `room_up`
- ▷ Afficher `room` et `room_up`
- ▷ Afficher le nombre de " o " dans la variable `room` en appelant la méthode `count()` sur `room`

```
1 # string to experiment with: room
2 room = "poolhouse"
3
4 # Use upper() on room: room_up
5
6 # Print out room and room_up
7
8 # Print out the number of o's in room
```

Exercice

- ▷ Utiliser la méthode `index()` pour obtenir l'index de l'élément égal à 20.0. Afficher cet index.
- ▷ Appeler la méthode `count()` sur `areas` afin de trouver combien de fois 14.5 apparaît dans la liste. Afficher le résultat.

```
1 # Create list areas
2 areas = ['hallway', 'kitchen', 'living room', '
3         '
4 # Print out the index of the element living room
5
6 # Print out how often hallways appears in areas
```

Exercice

- ▷ Utiliser `append()` deux fois afin d'ajouter la taille de poolhouse et garage : 24.5 and 15.45
- ▷ Afficher `areas`
- ▷ Utiliser la méthode `reverse()` afin de d'inverser l'ordre des éléments de `areas`
- ▷ Afficher `areas`

```
1 # Create list areas
2 areas = ['hallway', 'kitchen', 'living room', '
3         poolhouse', 'garage', '
4         bedroom', '
5         bathroom']
6 # Use append twice to add poolhouse and garage size
7 # Print out areas
8 # Reverse the orders of the elements in areas
9
10 # Print out areas
```

Packages

- ▷ Ensemble de scripts Python
- ▷ Spécifie des fonctions, des méthodes et des types
- ▷ Des milliers de packages sont disponibles (Numpy, Matplotlib, Scikit-learn)

```
1 import numpy
2 numpy.array([1, 2, 3])
3
4 import numpy as np
5 np.array([1, 2, 3])
6
7 from numpy import array
8 array([1, 2, 3])
```

Exercice

- ▷ Importer le package `math`. Vous pouvez maintenant utiliser `math.pi`
- ▷ Calculer la circonférence du cercle et stocker le dans `C`
- ▷ Calculer l'aire du cercle et stocker le dans `A`

```
1 # Definition of radius
2 r = 0.43
3
4 # Import the math package
5
6 # Calculate C
7 C = 0
8
9 # Calculate A
10 A = 0
11
12 # Build printout
13 print("Circumference: " + str(C))
14 print("Area: " + str(A))
```

Exercice

- ▷ Effectuer un import sélectif du package `math` pour importer uniquement la fonction `radians`
- ▷ Calculer la distance parcourue par la lune sur 12 degrés de son orbite. Assigner le résultat à `dist`. Vous pouvez la calculer par `r * phi`, où `r` est le rayon et `phi` est l'angle en radians. Pour convertir l'angle de degré à radians, utiliser la fonction `radians()` que vous venez d'importer
- ▷ Afficher `dist`

```
1 # Definition of radius
2 r = 192500
3
4 # Import radians function of math package
5
6 # Travel distance of Moon over 12 degrees. Store in dist.
7
8 # Print out dist
```

Numpy

Numpy

- ▷ Numerical Python
- ▷ Alternative aux listes Python : Numpy Array
- ▷ Facilite les calculs sur des tableaux
- ▷ Simple et rapide

Numpy

```
1 In [6]: import numpy as np
2
3 In [7]: np_height = np.array(height)
4
5 In [8]: np_height
6 Out[8]: array([ 1.73, 1.68, 1.71, 1.89, 1.79])
7
8 In [9]: np_weight = np.array(weight)
9
10 In [10]: np_weight
11 Out[10]: array([ 65.4, 59.2, 63.6, 88.4, 68.7])
12
13 In [11]: bmi = np_weight / np_height ** 2
14
15 In [12]: bmi
16 Out[12]: array([ 21.852, 20.975, 21.75, 24.747, 21.441])
```

Numpy

- ▷ Numpy Array ne contient qu'un seul type

```
1 In [19]: np.array([1.0, "is", True])
2 Out[19]: array(['1.0', 'is', 'True'], dtype='<U32')
3 NumPy arrays: contain only one type
4
5 In [20]: python_list = [1, 2, 3]
6
7 In [21]: numpy_array = np.array([1, 2, 3])
8
9 In [22]: python_list + python_list
10 Out[22]: [1, 2, 3, 1, 2, 3]
11
12 In [23]: numpy_array + numpy_array
13 Out[23]: array([2, 4, 6])
```

Numpy

▷ Numpy Array sélection

```
1 In [24]: bmi
2 Out[24]: array([ 21.852, 20.975, 21.75, 24.747, 21.441])
3
4 In [25]: bmi[1]
5 Out[25]: 20.975
6
7 In [26]: bmi > 23
8 Out[26]: array([False, False, False,  True, False], dtype=bool)
9
10 In [27]: bmi[bmi > 23]
11 Out[27]: array([ 24.747])
```

Exercice

- ▷ Importer le package numpy avec l'alias np
- ▷ Utiliser `np.array()` pour créer un tableau numpy à partir de `baseball`. Appeler le `np_baseball`
- ▷ Afficher le type de `np_baseball` pour vérifier son type.

```
1 # Create list baseball
2 baseball = [180, 215, 210, 210, 188, 176, 209, 200]
3
4 # Import the numpy package as np
5
6 # Create a numpy array from baseball: np_baseball
7
8 # Print out type of np_baseball
```

Exercise

- ▷ Créer un tableau numpy height appelé np_height
- ▷ Afficher np_height
- ▷ Multiplier np_height par 0.0254 afin de convertir le tableau des pouces aux mètres. Stocker les valeurs dans np_height_m
- ▷ Afficher np_height_m et vérifier le résultat

```
1 # height is available as a regular list
2 from baseballdata import *
3
4 # Import numpy
5 import numpy as np
6
7 # Create a numpy array from height: np_height
8
9 # Print out np_height
10
11 # Convert np_height to m: np_height_m
12
13 # Print np_height_m
```

<https://germain-forestier.info/teaching/files/DS/baseballdata.py>

Exercice

- ▶ Créer un tableau numpy pour la liste de poids avec la bonne unité. Multiplier par 0.453592 pour passer des livres aux kilos. Stocker le résultat dans `np_weight_kg`
- ▶ Utiliser `np_height_m` and `np_weight_kg` pour calculer le BMI (IMC) de chaque joueur. Utiliser l'équation suivante : $BMI = \frac{weight(kg)}{height(m)^2}$
- ▶ Stocker le résultat dans la variable `bmi` et l'afficher

```
1 # height and weight are available as a regular lists
2 from baseballdata import *
3
4 # Import numpy
5 import numpy as np
6
7 # Create array from height with correct units: np_height_m
8 np_height_m = np.array(height) * 0.0254
9
10 # Create array from weight with correct units: np_weight_kg
11
12 # Calculate the BMI: bmi
13
14 # Print out bmi
```

Exercice

- ▶ Créer un tableau numpy de boolean: les éléments du tableau doivent être True si le BMI du joueur correspondant est inférieur à 21 (vous pouvez utiliser l'opérateur <). Nommer le tableau `light` et l'afficher.
- ▶ Afficher un tableau numpy avec les BMI de tous les joueurs qui ont un BMI inférieur à 21. Utiliser le tableau `light` pour faire la sélection sur `bmi`.

```
1 # height and weight are available as a regular lists
2 from baseballdata import *
3
4 # Import numpy
5 import numpy as np
6
7 # Calculate the BMI: bmi
8 np_height_m = np.array(height) * 0.0254
9 np_weight_kg = np.array(weight) * 0.453592
10 bmi = np_weight_kg / np_height_m ** 2
11
12 # Create the light array
13
14 # Print out light
15
16 # Print out BMIs of all baseball players whose BMI is below 21
```


Exercice

Quelle proposition donnerait le même résultat que :

```
1 np.array([True, 1, 2]) + np.array([3, 4, False])
```

1. `np.array([True, 1, 2, 3, 4, False])`
2. `np.array([4, 3, 0]) + np.array([0, 2, 2])`
3. `np.array([1, 1, 2]) + np.array([3, 4, -1])`
4. `np.array([0, 1, 2, 3, 4, 5])`

Exercice

- ▷ Afficher l'élément à l'index 50 de `np_weight`
- ▷ Afficher le sous tableau de `np_weight` qui contient les éléments de 100 à 110 inclus

```
1 # height and weight are available as a regular lists
2 from baseballdata import *
3
4 # Import numpy
5 import numpy as np
6
7 # Store weight and height lists as numpy arrays
8 np_weight = np.array(weight)
9 np_height = np.array(height)
10
11 # Print out the weight at index 50
12
13 # Print out sub-array of np_height: index 100 up to and including index 110
```

2D Numpy Arrays

▷ Tableau à deux dimensions

```
1 In [6]: np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
2                             [65.4, 59.2, 63.6, 88.4, 68.7]])
3
4 In [7]: np_2d
5 Out[7]:
6 array([[ 1.73,  1.68,  1.71,  1.89,  1.79],
7         [ 65.4,  59.2,  63.6,  88.4,  68.7 ]])
8
9 In [8]: np_2d.shape
10 Out[8]: (2, 5)
11
12 In [9]: np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
13                   [65.4, 59.2, 63.6, 88.4, "68.7"]])
14
15 out[9]:
16 array([[ '1.73', '1.68', '1.71', '1.89', '1.79'],
17         [ '65.4', '59.2', '63.6', '88.4', '68.7']],
18        dtype='<U32')
```

2D Numpy Arrays

▷ Sélection dans un tableau 2D

```
1 In [10]: np_2d[0]
2 Out[10]: array([ 1.73, 1.68, 1.71, 1.89, 1.79])
3
4 In [11]: np_2d[0][2]
5 Out[11]: 1.71
6
7 In [12]: np_2d[0,2]
8 Out[12]: 1.71
9
10 In [13]: np_2d[:,1:3]
11 Out[13]:
12 array([[ 1.68, 1.71],
13        [ 59.2, 63.6 ]])
14
15 In [14]: np_2d[1,: ]
16 Out[14]: array([ 65.4, 59.2, 63.6, 88.4, 68.7])
```

Exercice

- ▷ Utiliser `np.array()` pour créer un tableau numpy 2D pour baseball, appeler le `np_baseball` et afficher le
- ▷ Afficher la forme du tableau, utiliser `np_baseball.shape`

```
1 # Create baseball, a list of lists
2 baseball = [[180, 78.4],
3             [215, 102.7],
4             [210, 98.5],
5             [188, 75.2]]
6
7 # Import numpy
8 import numpy as np
9
10 # Create a 2D numpy array from baseball: np_baseball
11
12 # Print out the type of np_baseball
13
14 # Print out the shape of np_baseball
```

Exercice

- ▷ Utiliser `np.array()` pour créer un tableau numpy 2D à partir de `baseball` et l'appeler `np_baseball`
- ▷ Afficher la forme de `np_baseball`

```
1 # baseball is available as a regular list of lists
2 from baseballdata import *
3
4 # Import numpy package
5 import numpy as np
6
7 # Create a 2D numpy array from baseball: np_baseball
8
9 # Print out the shape of np_baseball
```

Exercice

- ▷ Afficher la 50ème ligne de `np_baseball`
- ▷ Créer une nouvelle variable, `np_weight`, contenant la deuxième colonne de `np_baseball`
- ▷ Sélectionner la taille (première colonne) du joueur 124 dans `np_baseball` et l'afficher

```
1 # baseball is available as a regular list of lists
2 from baseballdata import *
3
4 # Import numpy package
5 import numpy as np
6
7 # Create np_baseball (2 cols)
8 np_baseball = np.array(baseball)
9
10 # Print out the 50th row of np_baseball
11
12 # Select the entire second column of np_baseball: np_weight
13
14 # Print out height of 124th player
```

Exercice

- Vous avez récupéré les changements de poids taille et age des joueurs de baseball. Ils sont stockés dans le tableau numpy 2D array. Ajouter `np_baseball` et `updated` et afficher le résultat.
- Vous souhaitez convertir les unités de taille et poids. Créer un tableau numpy avec les valeurs 0.0254, 0.453592 et 1. Nommer ce tableau `conversion`. Multiplier `np_baseball` et `conversion` et afficher le résultat.

```
1 # updated is available as 2D numpy array
2 from baseballdata2 import *
3
4 # Import numpy package
5 import numpy as np
6
7 # Create np_baseball (3 cols)
8 np_baseball = np.array(baseball)
9
10 # Print out addition of np_baseball and updated
11
12 # Create numpy array: conversion
13
14 # Print out product of np_baseball and conversion
```

<https://germain-forestier.info/teaching/files/DS/baseballdata2.py>

Numpy statistiques

- ▷ Comprendre vos données
- ▷ Peu de données : simplement les observer
- ▷ Beaucoup de données : ?

```
1 In [1]: import numpy as np
2 In [2]: np_city = ... # Implementation left out
3 In [3]: np_city
4 Out[3]:
5 array([[ 1.64, 71.78],
6        [ 1.37, 63.35],
7        [ 1.6 , 55.09],
8        ...,
9        [ 2.04, 74.85],
10       [ 2.04, 68.72],
11       [ 2.01, 73.57]])
```

Numpy statistiques

```
1 In [4]: np.mean(np_city[:,0])
2 Out[4]: 1.7472
3
4 In [5]: np.median(np_city[:,0])
5 Out[5]: 1.75
6
7 In [6]: np.corrcoef(np_city[:,0], np_city[:,1])
8 Out[6]:
9 array([[ 1., -0.01802],
10        [-0.01803, 1.]])
11
12 In [7]: np.std(np_city[:,0])
13 Out[7]: 0.1992
```

▷ `sum()`, `sort()`, ...

Exercice

- ▷ Créer un tableau numpy `np_height` qui contient la première colonne de `np_baseball`
- ▷ Afficher la moyenne de `np_height`
- ▷ Afficher le médiane de `np_height`
- ▷ Constatez vous une erreur ? Quelle statistique est la plus adaptée pour détecter cette erreur ?

```
1 # np_baseball is available
2 from baseballdata3 import *
3
4 # Import numpy
5 import numpy as np
6
7 # Create np_height from np_baseball
8
9 # Print out the mean of np_height
10
11 # Print out the median of np_height
```

<https://germain-forestier.info/teaching/files/DS/baseballdata3.py>

Exercise

- ▷ Compléter le code pour calculer la médiane.
- ▷ Utiliser `np.std()` sur la première colonne de `np_baseball`.
- ▷ Est-ce que les grands joueurs sont plus lourd ? Utiliser `np.corrcoef()` pour stocker la corrélation entre la première et la deuxième colonne de `np_baseball` dans `corr`.

```
1 # np_baseball is available
2 from baseballdata3 import *
3
4 # Print mean height (first column)
5 avg = np.mean(np_baseball[:,0])
6 print("Average: " + str(avg))
7
8 # Print median height. Replace 'None'
9 med = None
10 print("Median: " + str(med))
11
12 # Print out the standard deviation on height. Replace 'None'
13 stddev = None
14 print("Standard Deviation: " + str(stddev))
15
16 # Print out correlation between first and second column. Replace 'None'
17 corr = None
18 print("Correlation: " + str(corr))
```

Exercise

- ▷ Convertir `heights` et `positions` en tableaux numpy
- ▷ Extraire les tailles des goalkeepers dans `gk_heights` (utiliser `np_positions == 'GK'`)
- ▷ Extraire les tailles des autres joueurs dans `other_heights` (utiliser `np_positions != 'GK'`)
- ▷ Afficher la taille médiane des goalkeepers avec `np.median()`
- ▷ Afficher la taille médiane des autres joueurs avec `np.median()`

```
1 # heights and positions are available as lists
2 from foot import *
3
4 # Convert positions and heights to numpy arrays: np_positions, np_heights
5
6 # Heights of the goalkeepers: gk_heights
7
8 # Heights of the other players: other_heights
9
10 # Print out the median height of goalkeepers. Replace 'None'
11 print("Median height of goalkeepers: " + str(None))
12
13 # Print out the median height of other players. Replace 'None'
14 print("Median height of other players: " + str(None))
```

Pandas

Pandas

- ▷ Package haut-niveau pour la manipulation de données
- ▷ Basé sur Numpy
- ▷ Permet de gérer des données de différents types
- ▷ L'objet DataFrame permet de manipuler les données

```
1 import numpy as np
2 import pandas as pd
3
4 df = pd.DataFrame({'A' : 1.,
5 'B' : pd.Timestamp('20130102'),
6 'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
7 'D' : np.array([3] * 4,dtype='int32'),
8 'E' : pd.Categorical(["test","train","test","train"]),
9 'F' : 'foo' })
10
11 print(df)
12 print(df.dtypes)
```

Pandas

- ▷ Package haut-niveau pour la manipulation de données
- ▷ Basé sur Numpy
- ▷ Permet de gérer des données de différents types
- ▷ L'objet DataFrame permet de manipuler les données

```
1      A      B      C      D      E      F
2  0  1.0  2013-01-02  1.0  3   test   foo
3  1  1.0  2013-01-02  1.0  3  train   foo
4  2  1.0  2013-01-02  1.0  3   test   foo
5  3  1.0  2013-01-02  1.0  3  train   foo
6
7  print(df.dtypes)
8  A      float64
9  B  datetime64[ns]
10 C      float32
11 D      int32
12 E      category
13 F      object
14 dtype: object
```


Pandas

- ▷ L'objet DataFrame permet de faire de la sélection de données
- ▷ La fonction `loc` permet de recherche par nom
- ▷ La fonction `iloc` permet de recherche par index

```
1 df['A']  
2 df[['A','B']]  
3 df[0:3]  
4  
5 # recherche par nom  
6 df.loc[:,['A','B']]  
7  
8 # recherche par index  
9 df.iloc[3]  
10 df.iloc[1:3]
```

Python Pandas Selections and Indexing

.iloc selections - position based selection

`data.iloc[<row selection>, <column selection>]`

Integer list of rows: [0,1,2]

Slice of rows: [4:7]

Single values: 1

Integer list of columns: [0,1,2]

Slice of columns: [4:7]

Single column selections: 1

loc selections - position based selection

`data.loc[<row selection>, <column selection>]`

Index/Label value: 'john'

List of labels: ['john', 'sarah']

Logical/Boolean index: data['age'] == 10

Named column: 'first_name'

List of column names: ['first_name', 'age']

Slice of columns: 'first_name':'address'

<https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/>

Lire des données

Lire de données

- ▷ Il est souvent nécessaire de lire des données depuis des fichiers afin de les traiter et de pouvoir construire des modèles prédictifs
- ▷ La fonction `genfromtxt()` de Numpy permet de lire des fichiers, mais uniquement avec des attributs de même type
- ▷ La fonction `read_csv()` de Pandas permet de lire des fichiers avec des types d'attributs différents

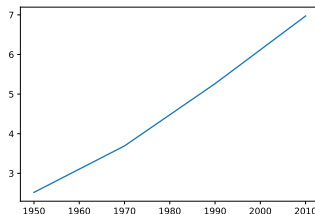
```
1 # lecture de données avec Numpy
2 import numpy as np
3 data = np.genfromtxt('data/iris.csv', delimiter=',', skip_header=True)
4
5 # lecture de données avec Pandas
6 import pandas as pd
7 df = pd.read_csv('data/iris.csv', header=0)
```

Visualisation de données

Visualisation de données

- ▷ Très important en analyse de données
- ▷ Permet d'explorer les données
- ▷ Permet de rapporter des résultats
- ▷ Matplotlib permet de facilement visualiser des données

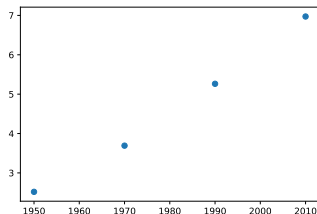
```
1 import matplotlib.pyplot as plt
2
3 year = [1950, 1970, 1990, 2010]
4 pop = [2.519, 3.692, 5.263, 6.972]
5
6 plt.plot(year, pop)
7 plt.show()
```



Visualisation de données

- ▷ Très important en analyse de données
- ▷ Permet d'explorer les données
- ▷ Permet de rapporter des résultats
- ▷ Matplotlib permet de facilement visualiser des données

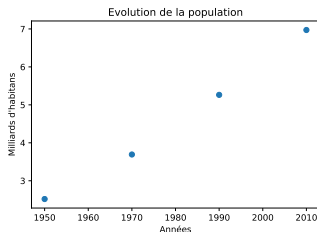
```
1 import matplotlib.pyplot as plt
2
3 year = [1950, 1970, 1990, 2010]
4 pop = [2.519, 3.692, 5.263, 6.972]
5
6 plt.scatter(year, pop)
7 plt.show()
```



Visualisation de données

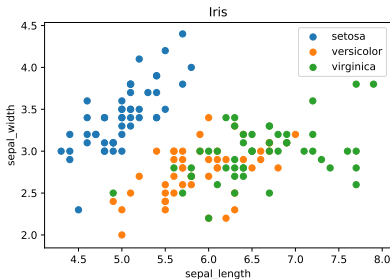
- ▷ Très important en analyse de données
- ▷ Permet d'explorer les données
- ▷ Permet de rapporter des résultats
- ▷ Matplotlib permet de facilement visualiser des données

```
1 import matplotlib.pyplot as plt
2
3 year = [1950, 1970, 1990, 2010]
4 pop = [2.519, 3.692, 5.263, 6.972]
5
6 plt.title('Evolution de la population')
7 plt.xlabel('Années')
8 plt.ylabel('Milliards d\'habitants')
9 plt.scatter(year, pop)
10 plt.show()
```



Exercice

- ▷ Récupérer le fichier "iris.csv"¹
- ▷ Importer les données à l'aide de `read_csv()` de pandas
- ▷ Générer le graphique suivant à l'aide de matplotlib (choisir deux caractéristiques) :



¹<https://germain-forestier.info/teaching/files/FD/dataset/iris.csv>

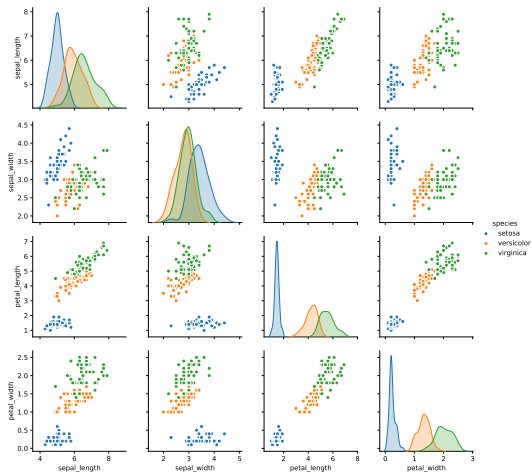
Visualisation de données

- ▷ Le package Seaborn est une alternative à Matplotlib
- ▷ Rajoute des fonctionnalités
- ▷ Permet de faire des visualisations avancées

```
1 import seaborn as sns
2 import pandas as pd
3
4 df = pd.read_csv('data/iris.csv', header=0)
5
6 ax = sns.pairplot(df, hue='species', height=1)
```

<https://seaborn.pydata.org/>

Visualisation de données



Validation

Validation (train/test)

- ▷ Afin de pouvoir valider un modèle, il est important de pouvoir tester sa capacité à faire de bonnes prédictions
- ▷ On évite de tester un modèle sur les mêmes données utilisées pour construire le modèle
- ▷ Si on utilise les mêmes données, les performances auront tendance à être surévaluées et on ne teste pas la capacité de généralisation
- ▷ La méthode la plus classique consiste à découper le jeu de données en deux ensembles, un pour l'apprentissage (*train*) et un pour l'évaluation (*test*)

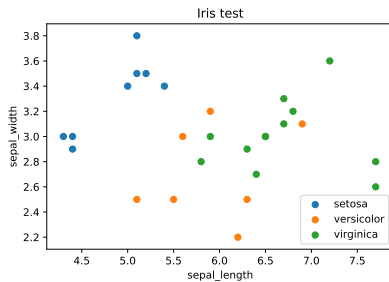
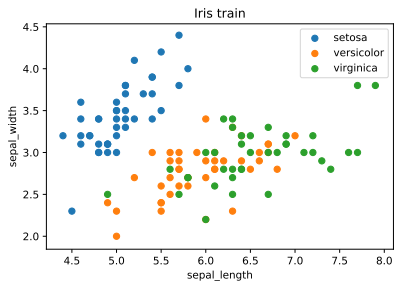
Validation (train/test)

- ▷ On garde 80% des données pour l'apprentissage et 20% pour le test :

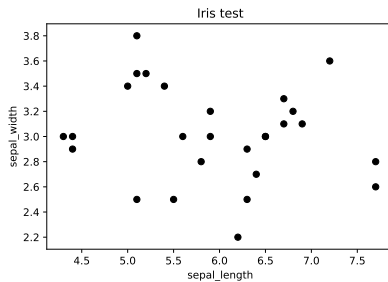
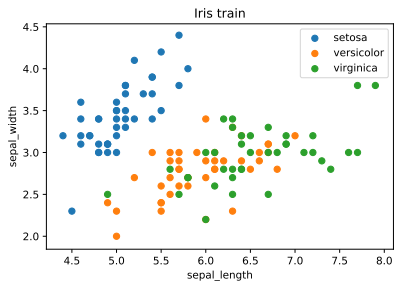
```
1 import pandas as pd
2 import numpy as np
3
4 df = pd.read_csv('data/iris.csv', header=0)
5 mask = np.random.rand(len(df)) < 0.8
6 df_train = df[mask]
7 df_test = df[~mask]
```

- ▷ Il faut faire attention à la distribution des classes (nombre d'éléments de chaque classe qui peut être différent)
- ▷ Afin de conserver la même distribution, on parle de stratification

Validation (train/test)



Validation (train/test)



Validation (train/test)

sepal_length	sepal_width	species
5.1	3.5	setosa
4.9	3	setosa
6.7	3.1	versicolor
6.3	2.3	versicolor
6.3	2.8	virginica
6.1	2.6	virginica

Table 1: Train set

sepal_length	sepal_width	species	species
4.7	3.2	setosa	setosa
5.6	3	versicolor	versicolor
6.7	3.3	virginica	virginica

Table 2: Test set

Objectif : prédire la classe des objets du "test set" à l'aide des données du "train set" (sans utiliser la classe des objets du "test set")

Matrice de confusion

- ▷ Matrice de comparaison entre les prédictions et la vraie classe des objets
- ▷ Permet d'évaluer les performances d'une méthode de classification
- ▷ Permet de mettre en évidence les confusions (erreurs) entre les classes

```
1 [[6 0 0]
2  [0 9 0]
3  [0 1 4]]
4
5         precision    recall  f1-score   support
6
7  setosa         1.00        1.00        1.00         6
8  versicolor    0.90        1.00        0.95         9
9  virginica      1.00        0.80        0.89         5
10
11 avg / total    0.96        0.95        0.95        20
```

sepal_length	sepal_width	species	prédictions
5.1	3.5	setosa	setosa
4.7	3.2	setosa	setosa
6.7	3.1	versicolor	versicolor
5.6	3	versicolor	virginica
6.3	2.8	virginica	virginica
6.7	3.3	virginica	virginica

Table 3: Ensemble de test

Table 4: Prédictions

	setosa	versicolor	virginica
setosa	2	0	0
versicolor	0	1	1
virginica	0	0	2

Table 5: Matrice de confusion

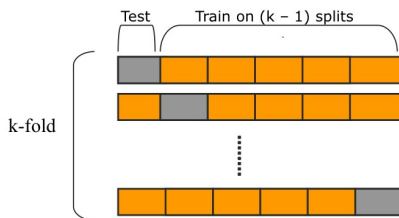
Erreur de classification

Bonnes prédictions (diagonale) = 2 + 1 + 2

Mauvaise prédiction (hors diagonale) = 1

Validation croisée (cross validation)

- ▷ Alternative au découpage train / test
- ▷ Le jeu de données est coupé en K sous ensembles
- ▷ K-1 ensembles utilisés pour le train, 1 ensemble utilisé pour le test
- ▷ Chaque ensemble est utilisé une fois pour le test



source : <https://raw.githubusercontent.com/qingkaikong/>

Scikit-learn

Scikit-learn

- ▷ Scikit-learn contient la majorité des algorithmes de fouille de données
- ▷ Il offre également de nombreux outils pour l'évaluation des modèles
- ▷ Scikit-learn décompose les données et la variable à prédire (*target*)

```
1 from sklearn.model_selection import train_test_split
2
3 df = pd.read_csv('data/iris.csv', header=0)
4 X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
5 y = df[['species']]
6
7 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Scikit-learn

- ▷ Scikit-learn décompose les données et la variable à prédire (*target*)

sepal_length	sepal_width	species
5.1	3.5	setosa
4.9	3	setosa
6.7	3.1	versicolor
6.3	2.3	versicolor
6.3	2.8	virginica
6.1	2.6	virginica

Table 6: Train set

sepal_length	sepal_width	species
4.7	3.2	setosa
5.6	3	versicolor
6.7	3.3	virginica

Table 7: Test set

Scikit-learn

- ▷ Scikit-learn décompose les données et la variable à prédire (*target*)

X_train		y_train
sepal_length	sepal_width	species
5.1	3.5	setosa
4.9	3	setosa
6.7	3.1	versicolor
6.3	2.3	versicolor
6.3	2.8	virginica
6.1	2.6	virginica

Table 8: Train set

X_test		y_test
sepal_length	sepal_width	species
4.7	3.2	setosa
5.6	3	versicolor
6.7	3.3	virginica

Table 9: Test set

Scikit-learn

- Scikit-learn permet également de calculer la matrice de confusion et d'évaluer les performances des modèles

```

1 from sklearn.metrics import confusion_matrix, classification_report,
   precision_score
2
3 real = ['setosa', 'setosa', 'versicolor', 'versicolor', 'virginica', 'virginica']
4 pred = ['setosa', 'setosa', 'versicolor', 'virginica', 'virginica', 'virginica']
5
6 print(confusion_matrix(real, pred))
7 print(classification_report(real, pred))
8 print(precision_score(real, pred, average='micro'))

```

		precision	recall	f1-score	support
[[2 0 0]	setosa	1.00	1.00	1.00	2
[0 1 1]	versicolor	1.00	0.50	0.67	2
[0 0 2]]	virginica	0.67	1.00	0.80	2
0.8333333333333334	accuracy			0.83	6
	macro avg	0.89	0.83	0.82	6
	weighted avg	0.89	0.83	0.82	6

Un premier classifieur

Exercice

- ▷ A l'aider de la visualisation de données du jeu de données Iris, créer un classifieur simple à base de seuils afin de classer les différents types d'iris
- ▷ Évaluer les performances de votre classifieur sur toutes les données
- ▷ Faire un graphique pour illustrer la frontière des classes

```
1 import pandas as pd
2 from sklearn.metrics import confusion_matrix
3 from sklearn.metrics import classification_report
4
5 # lecture des données
6 df = pd.read_csv('data/iris.csv', header=0)
7
8 # écrire un classifieur à base de seuils
```