

Python pour la Data Analyse

2ème partie

Angelo.Steffenel@univ-reims.fr

Rappel : trois façons de tester Python

1. **Installer** le langage sur son ordinateur

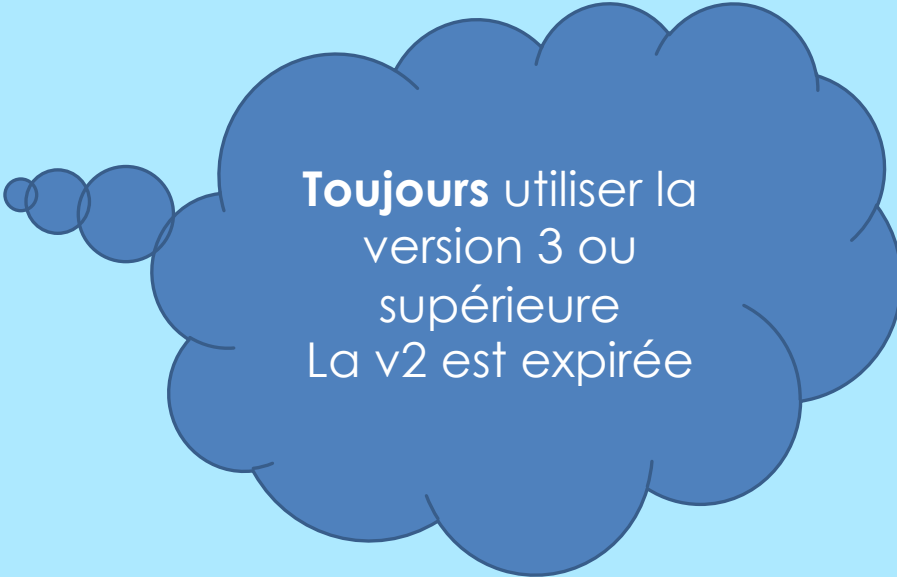
- <https://www.python.org/downloads/>
- Normalement cela inclut l'environnement de développement IDLE
- Certains OS (Mac, Linux) incluent déjà une version de Python

2. Utiliser un **environnement en ligne**

- Par exemple, <https://repl.it/languages/python3>

3. Utiliser un "**notebook**"

- Par exemple, <https://colab.research.google.com/>



Toujours utiliser la
version 3 ou
supérieure
La v2 est expirée

Rappel sur les Structures de Données en Python

- Python repose sur 3 types de structures de données
 - Tuples
 - Listes (dont les chaînes de caractères)
 - Dictionnaires

Rappel sur les Structures de Données en Python

- **Tuples** : séquence unidimensionnelle d'objets (données)
 - **Structure immuable**
 - On peut mélanger plusieurs types de données
 - Représentés par de parenthèses ()
 - Construction directe ou avec la fonction tuple()
 - Très peu de fonctions auxiliaires
- Intérêt ?
 - Utilise très peu de mémoire
 - Format de retour de fonctions à plusieurs valeurs

```
tup1=(1, True, 7.5, 9)
print(tup1[2])
>> 7.5
```

```
Tup2=tuple([1, True, 7.5, 9])
```

```
print(tup1.count(9))
>> 1
```

Rappel sur les Structures de Données en Python

- **Listes** : séquence d'objets (données)
 - **Structure modifiable**
 - On peut mélanger plusieurs types de données
 - Représentés par de crochets []
 - Construction directe ou avec la fonction list()
 - Beaucoup de fonctions auxiliaires
 - append(), insert(), pop(), reverse(), extend(), ...
 - Accepte certains raccourcis
 - liste1=liste1+liste2 équivaut à liste1.extend(liste2)

```
liste1=[1, True, 7.5, 9]  
print(liste1[2])  
>> 7.5
```

```
liste1[2]=200  
print(liste1[2])  
>> 200
```

```
liste1.pop(2)  
print(liste1[2])  
>> 9
```

Rappel sur les Structures de Données en Python

○ Listes et Tableaux

- Un tableau multidimensionnel n'est rien qu'une liste de listes
 - Chaque "objet" de la liste principale est une liste
 - Les tailles peuvent varier

○ Accès aux positions

- `s[i]` - accès à l'élément d'indice `i` (l'origine a l'indice 0)
- `s[i:j]` accès aux éléments entre les indices `i` (inclus) et `j` (exclus)
- `s[1,2:3]` – accès à la "ligne" 1, "colonnes" 2 à **3 (3 excluse)**
- Possibilité de d'accéder aux `n` derniers éléments avec `-n`
- Début et fin implicites si juste `[:]`

```
ligne1=[1, True, 7.5, 9]
ligne2=['toto',False,15, 10.5]
tab1=[ligne1,ligne2]
print(tab1[1][2:3])
>> [15]
```

```
print(ligne1[-1])
>> 9
```

```
print(tab1[0][:])
>> [1, True, 7.5, 9]
```

Les Chaînes de Caractères

- Ce sont des listes de caractères (stockés en format Unicode)
- On peut les déclarer de trois manières
 - Apostrophe = 'ma chaîne de caractères'
 - Guillemet = "ma chaîne de caractères"
 - TripleGuillemets = """ma chaîne de caractères"""
 - Permet de faire du "multiligne"
 - Attention aux caractères spéciaux (\' ou \" selon la déclaration)
- Plusieurs fonctions pour la gestion du texte
- On peut transformer un texte en une liste de mots
 - Liste = Apostrophe.split()
 - ['ma', 'chaîne', 'de', 'caractères']
- .capitalize(), .upper(), .lower()
- .count(val)
- .lstrip(), .rstrip(), .strip()
- .replace()
- .find()
- .split()

Dictionnaires

- Dernier type de base
 - Identifié par des crochets { }
- Permet un stockage "clé-valeur"
 - On n'accède plus par la position, mais par la clé
 - dict1["clé1"]
- Très utilisé car facile à chercher
 - En plus, chaque clé est **unique**
- Des fonctions pour obtenir la liste de clés, des valeurs...

```
dict1={"nom":"toto","rang":4}
print(dict1["nom"])
>> toto

print(dict1.keys())
>> dict_keys(['nom', 'rang'])

print(dict1.values())
>> dict_values(['toto', 4])

print(dict1.items())
>> dict_items([('nom', 'toto'), ('rang', 4)])
```


Rappel des bases de Pandas

- Bibliothèque **manipulation de données tabulaires** (tableaux, matrices, etc.)
- Pour démarrer, il suffit d'importer la bibliothèque
`import pandas as pd`
- Deux structures de données : Séries et **DataFrame**
- Pd.Series
 - Liste de valeurs stockées dans une colonne
 - Les individus de cette liste sont indexés
 - À mi-chemin entre un array et un dictionnaire

```
ma_serie = pd.Series([8, 70, 320, 1200],  
                     index = ["Suisse", "France", "USA", "Chine"])  
print(ma_serie)
```

```
Suisse      8  
France     70  
USA        320  
Chine     1200  
dtype: int64
```

```
ma-serie2 = pd.Series({"Suisse":8, "France":70,  
                      "USA":320, "Chine":1200})
```

DataFrames

- Les **DataFrame** (DF) sont des **structures de données tabulaires "riches"**
 - Nom et type des colonnes
 - Indexation ("clé primaire")
- On peut créer des DF manuellement ou à partir de fichiers (csv, excel, etc.)

Exemple : DataFrame avec les actions en bourse d'Apple

Date	Open	High	Low	Close	Volume	Adj Close
2014-09-16	99.80	101.26	98.89	100.86	66818200	100.86
2014-09-15	102.81	103.05	101.44	101.63	61216500	101.63
2014-09-12	101.21	102.19	101.08	101.66	62626100	101.66
...

Importation à partir d'un fichier CSV

- Pour créer un DataFrame à partir d'un **fichier csv**, il suffit de faire
- `df = pd.read_csv('nomfichier')`
- Des options pour gérer les entêtes, la transformation des données, etc.
 - `pd.read_csv('tmp.csv', index_col=[0], parse_dates=[0], header=None)`
 - `pd.read_csv('tmp.csv', index_col='Date', parse_dates=True, delimiter=';')`
 - `pd.read_csv('tmp.csv', decimal=',')`
- On peut aussi **exporter sur un fichier** avec `to_csv('nomfichier')`

Importation à partir d'autres sources

- Excel : utilisation de Pandas avec les bibliothèques xlrd/xlwt et openpyxl
 - Selon le système il faut les installer à part
- Deux possibilités
 - **pd.read_excel()**

```
credit = pd.read_excel("fichier.xls", sheetname="donnees", usecols="A:E")
```

- pd.ExcelFile()
- JSON : déjà inclus dans Pandas

```
jdata = pd.read_json("fichier.json")
```

Importation à partir de R

- Parfois on doit travailler avec des données stockés en format .Rdata
 - Utiliser la bibliothèque **rpy2**

```
import pandas as pd
import rpy2.robjects.pandas2ri as pandas2ri
from rpy2.robjects import r

def charger_fichier_rdata (nom_fichier):
    r_data = r['get'](r['load'](nom_fichier))
    df = pandas2ri.r2py(r_data)
    return df

frame = charger_fichier_rdata("mon_fichier.Rdata")
```

Et si mon dataset est trop grand ?

- Un dataset est stocké en mémoire
 - Avec toutes les autres structures de données
 - Ça peut rapidement remplir votre ordinateur
- Si le dataset est trop grand, on a quelques options
 - 1 – Choisir quels éléments charger en mémoire
 - 2 – Lire par morceaux
 - 3 – Faire appel à des bibliothèques "big data"

Et si mon dataset est trop grand ?

- Option 1 – choisir les données à charger
- Parfois on n'est pas intéressé par toutes les colonnes d'un dataset
 - On peut "regarder" ce que nous intéresse et charger juste ce qu'il faut
- Utilisation du paramètre "usecols"
- Ne pas oublier d'effacer les structures qui n'auront plus d'utilité
 - Commande "del"

```
frame2 = pd.read_csv('tmp.csv', nrows=2)

print(frame2)
   CountryName CountryCode Year TotalPopulation
Urban population (% of total)
0  ArabWorld    ARB  1960  9.249590e+07  31.285384
1  Caribbean    CSS  1960  4.190810e+06  31.597490

frame = pd.read_csv('tmp.csv',
usecols=["CountryCode", "Year", "TotalPopulation"])

del frame2
```

Et si mon dataset est trop grand ?

- Option 2 – lire par morceaux
- Souvent on veut effectuer des agrégations ou filtrages ligne par ligne
- Dans ce cas, on peut travailler par segments
 - Lecture du dataset avec l'option "chunksize"
 - Désactiver l'option 'low_memory'
 - Itérer sur chaque "chunk" et ne garder ce qui intéresse
- Attention au choix de la taille (trop petit = trop lent)

```
onlyFrance=pd.DataFrame()
for chunk in pd.read_csv('tmp.csv', chunksize=1000,
low_memory=False):
    onlyFrance = pd.concat([onlyFrance,
        chunk[chunk["CountryCode"]=="FRA"]])
```


Et si mon dataset est trop grand ?

- Option 3 – Faire appel à d'autres bibliothèques
- Certaines bibliothèques sont optimisées pour distribuer le calcul sur plusieurs machines
 - Ex : **Spark, Dask**
- Elles sont +- compatibles avec les DataSets Pandas et peuvent être intéressantes pour le traitement massif de données

```
import dask.dataframe as dd  
  
frame_dd = dd.read_csv("temp.csv")
```

Décrire et transformer des colonnes

- On peut obtenir des **informations sur les datasets** avec `info()` et `describe()`

```
import pandas as pd
```

```
aapl = pd.read_csv('aapl.csv',  
index_col='Date', parse_dates=True)
```

```
print(aapl.info())
```

```
print(aapl.describe())
```

- Afficher la matrice de corrélation avec `.corr()`
- Autre attribut utile : `shape`

```
print(aapl.shape)
```

```
(6081, 6)
```

```
<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 6081 entries, 2008-10-14 to 1984-09-07  
Data columns (total 6 columns):  
#   Column      Non-Null Count  Dtype    
---  ---  
0   Open        6081 non-null   float64  
1   High        6081 non-null   float64  
2   Low         6081 non-null   float64  
3   Close       6081 non-null   float64  
4   Volume      6081 non-null   int64  
5   Adj Close   6081 non-null   float64  
dtypes: float64(5), int64(1)  
memory usage: 332.6 KB  
None
```

info()

describe()

	Open	High	Low	Close	Volume	Adj Close
count	6081.000000	6081.000000	6081.000000	6081.000000	6.081000e+03	6081.000000
mean	46.823511	47.681506	45.913595	46.798619	1.363986e+07	23.529794
std	33.993517	34.578077	33.273106	33.947235	1.352107e+07	37.375601
min	12.880000	13.190000	12.720000	12.940000	8.880000e+04	1.650000
25%	24.730000	25.010000	24.200000	24.690000	5.530000e+06	7.380000
50%	38.250000	38.880000	37.460000	38.130000	8.976400e+06	9.910000
75%	53.500000	54.550000	52.500000	53.610000	1.631920e+07	14.360000
max	200.590000	202.960000	197.800000	199.830000	2.650690e+08	199.830000

Changement de types de données

- Par défaut Pandas utilise 3 types principaux
 - Entiers - int (32 ou 64 bits)
 - Réels – float (32 ou 64 bits)
 - Objets – tout autre type (String, booléen, etc.)
- La détection automatique peut être incorrecte
 - Ex : traiter un booléen comme String
- Changer le type peut économiser de la mémoire
 - Passer de int64 à int32, par exemple
- Approche "directe" : utiliser la fonction `astype()`
 - `df = df.astype({"Column 1": float, "Column 2": int})`
- "Approche indirecte"
 - Parfois on doit "nettoyer" les données
 - Ex : `prix = '$40.5'`
 - Dans ce cas, on peut modifier le contenu de la colonne puis demander la re-détection du type
- Ex :

```
tab["prix"] = pd.to_numeric(tab["prix"].str.strip("$"))
```
- Categorical
 - Transformer des "Object" en catégories
 - Comme dans "factor" en R

```
df['CP'] = df['CP'].astype("category")
```

Jointures et Concaténations

- Il est possible de construire des dataframes à partir de la jonction d'autres dataframes
- On parle de **jointure** quand on utilise une **clé de jointure** présente sur les deux dataframes
 - Fonction **pd.merge**
 - `pd.merge(tabgauche, tabdroite, left_on="id", right_on="prod_number", how="inner")`
 - **how** indique la méthode de jointure : inner, outer, left, right
- La concaténation est la **juxtaposition** de deux dataframes
 - On doit juste indiquer sur quelle dimension on fera le collage
 - `List_concat = pd.concatenat([tab1, tab2], axis=1)` → axis=1 par colonnes, axis=0 par lignes
 - Les champs absents sur l'un des côtés auront les valeurs NaN

Données Manquantes et Dupliqués

- Python utilise un code spécifique pour les données manquantes → NaN
 - Intérêt : n'interfère pas avec les opérations (sum, mean, median, max, min, etc.)
- Supprimer ou remplacer la valeur ?
 - Pour supprimer, il suffit de faire appel à `df.dropna()`
 - Supprime les lignes avec des valeurs NaN. Pour supprimer les colonnes, utiliser `dropna(axis=1)`
 - Pour remplacer on utilise `df.fillna(valeur)`
 - Ex : `df[col] = df[col].fillna(df[col].mean())`
- Pour les données dupliquées, on peut les afficher avec `.duplicated()` ou les supprimer
 - Ex : `df_nodup = df.drop_duplicates(['Name', 'First'], keep="first")`

Discrétisation

- Transformer une variable quantitative (âge) en variable qualitative (classe d'âge)
- Deux fonctions dans Panda : **cut()** et **qcut()**
- Intervalles constants – on indique le nombre de classes, réparties entre min et max
 - `pd.cut(produits["prix"], bins=5)`
- Intervalles définis par l'utilisateur – on donne les bornes des intervalles
 - `pd.cut(produits["prix"], bins=[produits["prix"].min(), 50, 100, 500, produits["prix"].max()])`
- Intervalles de fréquence constante – nombre constant d'individus dans chaque classe
 - `pd.qcut(produits["prix"], q=5)`

Transformation de Données Qualitatives

- La plupart des algorithmes de ML n'acceptent que des entrées numériques
 - Il faut transformer nos données qualitatives (catégoriques)
- Possible avec Pandas mais souvent on fait appel à la bibliothèque Scikit-Learn
- Exemple : Données sur les logements AirBnB
 - <http://insideairbnb.com/get-the-data.html>

```
import pandas as pd
airdf = pd.read_csv("listings.csv")
print(airdf['room_type'].value_counts())
```

```
Entire home/apt    57880
Private room       7179
Hotel room         1409
Shared room        432
Name: room_type, dtype: int64
```


Transformation de Données Qualitatives

- Approche Pandas
 - Utilisation de `get_dummies()`
 - `roomdf = pd.get_dummies(airdf['room_type'])`
- Approche Scikit-Learn
 - Utilisation de `OneHotEncoder`
 - Plus complexe mais plus de possibilités
 - Sklearn est la bibliothèque principale pour du machine learning (hors DeepLearning)

```
roomdf = pd.get_dummies(airdf['room_type'])
```

ou

```
from sklearn.preprocessing import  
OneHotEncoder  
encoder=OneHotEncoder(sparse=False)  
arrayout=encoder.fit_transform(airdf['room_type']  
.to_numpy().reshape(-1, 1))  
ohe_df = pd.DataFrame(arrayout,  
columns=airdf['room_type'].unique())
```


Transformation de Données Qualitatives

- Approche Pandas
 - Utilisation de `get_dummies()`
 - `roomdf = pd.get_dummies(airdf['room_type'])`
- Approche Scikit-Learn
 - Utilisation de `OneHotEncoder`
 - Plus complexe mais plus de possibilités
 - Sklearn est la bibliothèque principale pour du machine learning (hors DeepLearning)

```
print (roomdf)
```

	Entire home/apt	Hotel room	Private room	Shared room
0	1	0	0	0
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0
4	1	0	0	0

```
print(ohe_df)
```

	Entire home/apt	Private room	Hotel room	Shared room
0	1.0	0.0	0.0	0.0
1	1.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0
4	1.0	0.0	0.0	0.0

Transformation de Données Numériques

- Même les entrées numériques doivent souvent être "normalisées" pour les algorithmes ML
 - Souvent, des valeurs entre 0 et 1
- Possible avec Pandas avec des transformations "à la main"
- Ou avec Scikit-Learn
- Exemple : Changer l'échelle

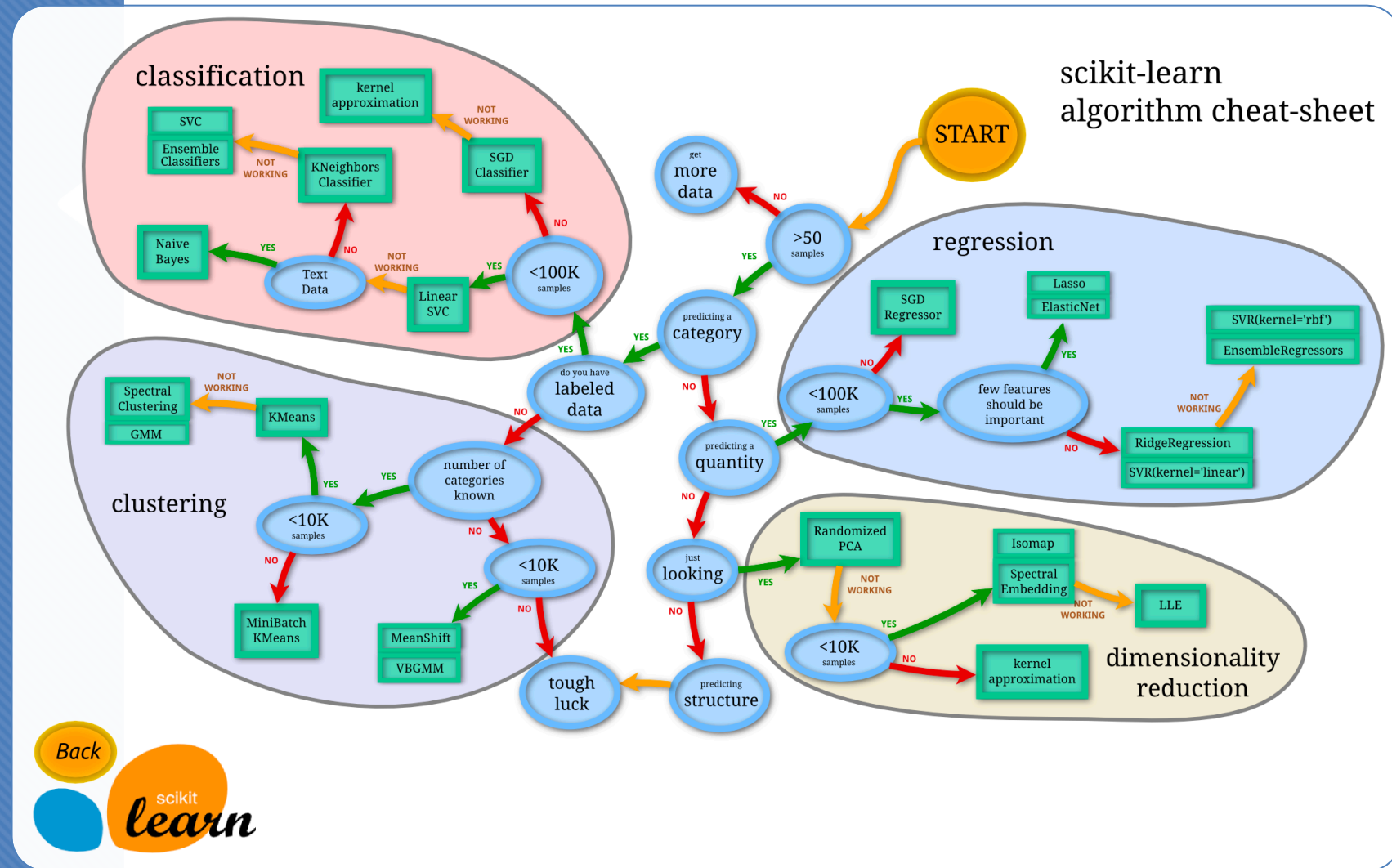
```
#Pandas
avail_0_100=pd.DataFrame(airdf['availability_365'])
avail_0_100=avail_0_100.apply(lambda x: (x-x.min())/(x.max()-x.min())*100)
```

```
from sklearn.preprocessing import MinMaxScaler
myscaler=MinMaxScaler((0,100))
avail_365=pd.DataFrame(airdf['availability_365'])
avail_0_100=myscaler.fit_transform(avail_365)
availdf=pd.DataFrame(avail_0_100,index=avail_365.index,
columns=avail_365.columns)
```

Au Delà de la Préparation de Données

- "Préparer les données c'est 80% du boulot. Extraire de la connaissance ce sont les autres 80%"
- Différentes bibliothèques permettent la valorisation des données
 - Machine Learning : Scikit-Learn
 - Deep Learning : Tensorflow (+Keras)
 - Visualisation : Matplotlib, Seaborn, Cartopy, Bokeh
- Big Data
 - Si le volume (ou la vitesse) sont des contraintes, des solutions BigData s'imposent
 - Dask ou Spark
 - Spark.ml

ML Simple avec Scikit- Learn



Classification

- Plusieurs algorithmes de pour la prédiction d'une variable qualitative (classification)
 - SVM - `sklearn.svm.SVC()`
 - Régression Logistique – `sklearn.linear_model.LogisticRegression()`
 - K plus proches voisins – `sklearn.neighbors.KNeighborsClassifier()`
 - Arbre de décision – `sklearn.tree.DecisionTreeClassifier()`
 - Forêts aléatoires – `sklearn.ensemble.RandomForestRegressor()`
 - Gradient Boost Machine – `sklearn.ensemble.GradientBoostingClassifier()`
 - Bayésien naïf – `sklearn.naive_bayes.GaussianNB()`
 - Réseaux de neurones – `sklearn.neural_network.MPLClassifier()`

Régression

- De même, plusieurs algorithmes de pour la prédiction d'une variable quantitative
 - Régression linéaire - `sklearn.linear_model.LinearRegression()`
 - ElasticNet – `sklearn.linear_model.ElasticNet()`
 - Ridge/Lasso - `sklearn.linear_model.Ridge()` / `Lasso()`
 - Gradient Boost – `sklearn.ensemble.GradientBoostRegressor()`
 - SVM – `sklearn.svm.SVR()`
 - K plus proches voisins – `sklearn.neighbors.KNeighborsRegressor()`

Méthodes non-supervisées

- En plus des méthodes précédentes, Scikit-learn a aussi droit à des méthodes d'apprentissage non-supervisées :
 - clustering (k-means)
 - réduction de dimensionnalité (PCA, etc.)
- Par contre, quand l'objectif est le deep learning, la tendance est d'utiliser d'autres bibliothèques plus performantes
 - TensorFlow
 - Keras (en vérité, un frontend simplifié compatible avec TensorFlow et autres frameworks)

Exercices

- L'énoncé se trouve à l'adresse
 - <https://github.com/lsteffene1/IntroPythonData/blob/master/Exercices2.pdf>

Bibliographie suggérée

- Emmanuel Jakobowicz – Python pour le Data Scientist, Dunod
- P Lemberger, M Batty, M Morel, J-L Raffaëli – Big Data et Machine Learning, Dunod