Exploration et analyse de données

S1 : Rappels généralités et outils

Plan du module

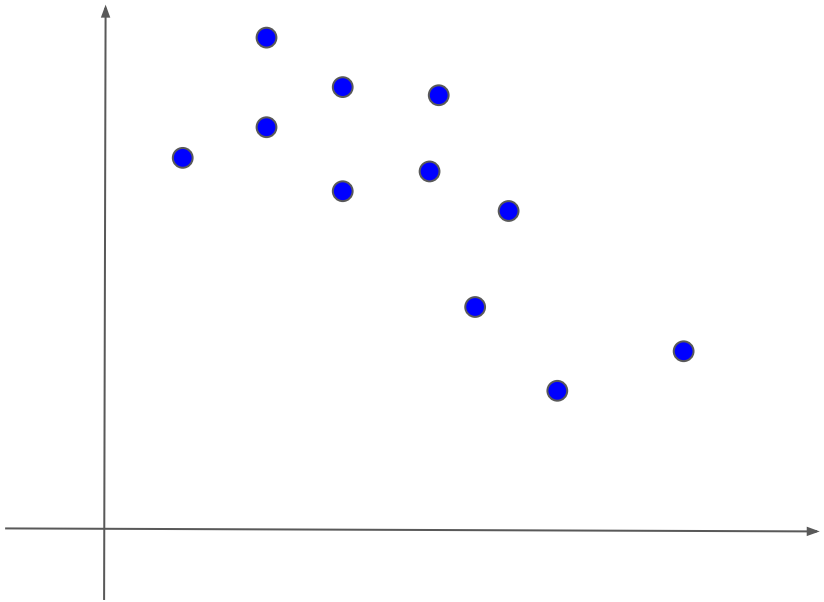
- S1 : généralité, outils et bibliothèques
- S2 : Data viz, exploration, projets
- S3 : Récap sur ensemble données projets
- S4 : Inférence statistique, scipy
- S5 : Régression linéaire, statmodels
- S6 : Régression logistique
- S7 : Recap decision science, données projets
- S8 : Introduction Machine Learning (modèles régression)
- S9 : Point avancée des projets, communication
- S10 : Présentations

Évaluations

- un projet (x2) : faire une analyse complète d'une problématique/question sur un jeu de données que vous choisirez (données ouvertes, données à votre travail, données Kaggle…), présentation à la dernière séance
- un QCM (Wooclap) à l'issue du module
- note de participation : implication durant le cours, possibilité de présenter volontairement (et rapidement) la mise en application de ce qui a été vu en cours sur un jeu de données de votre choix, etc.

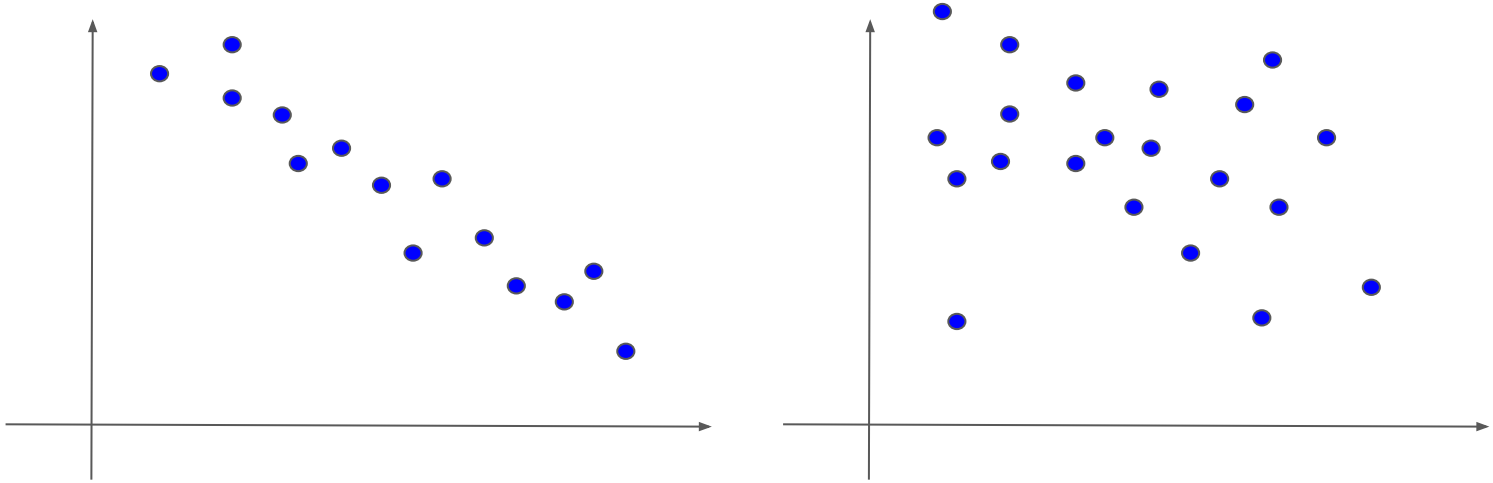
Analyse de donnée

- Objectif : utiliser les données pour répondre à des questions
- « Est-ce que la durée de livraison influence l'expérience utilisateur ? »



- Quelle est la relation exacte entre durée et satisfaction ?
- À quel point cette relation est généralisable ?

Le monde n’est pas parfait…



Expectation

Reality

- La question posée peut être vague : « comment améliorer la satisfaction client ? »
- Données de mauvaises qualités (labellisation, manquantes, collecte ???)
- Beaucoup de données, beaucoup de variables

Exploration vs. Analyse

- Exploration de données : « résumer les caractéristiques d'un ensemble de données, généralement en faisant usage de méthodes graphiques ou de visualisation de données » https://en.wikipedia.org/wiki/Exploratory_data_analysis
- Analyse de données : collecte, transformation, et organisation des données afin de formuler des hypothèses, en tirer des conclusions, faire des prévisions et favoriser la prise de décisions basée sur des faits statistiquement éprouvés

« Decision science » (data-driven decision making)

- On cherche à prendre des décisions concrètes (souvent départager plusieurs choix) pour résoudre des problèmes réels
- Connaissance métier
- Qualité des données
- Confiance/robustesse de nos conclusions sur les relations entre données
- Communiquer : pour comprendre le problème, le contexte métier, et pour apporter de la clarification (pédagogie, vulgarisation)

Data analysis vs. Data science/Machine Learning

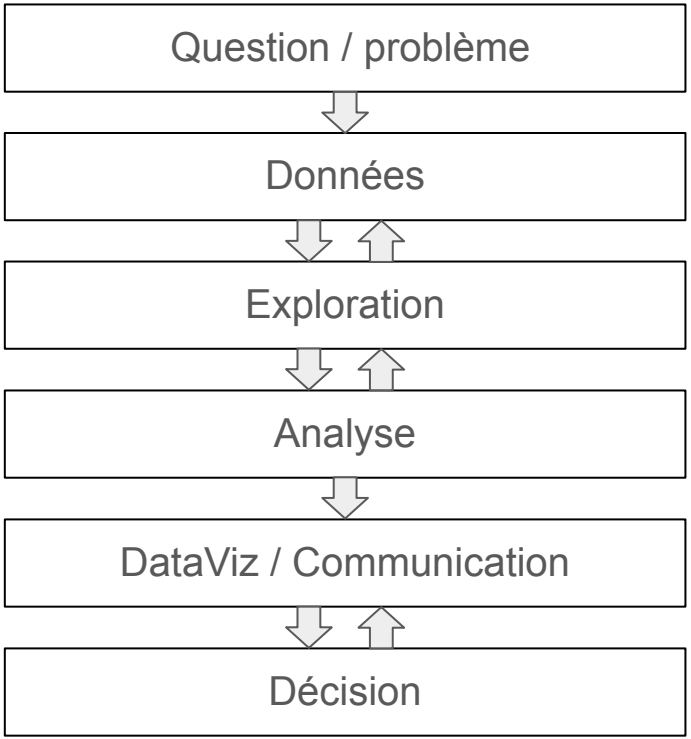
Depuis quelques années « data science » se rapporte plus au machine learning et l'IA

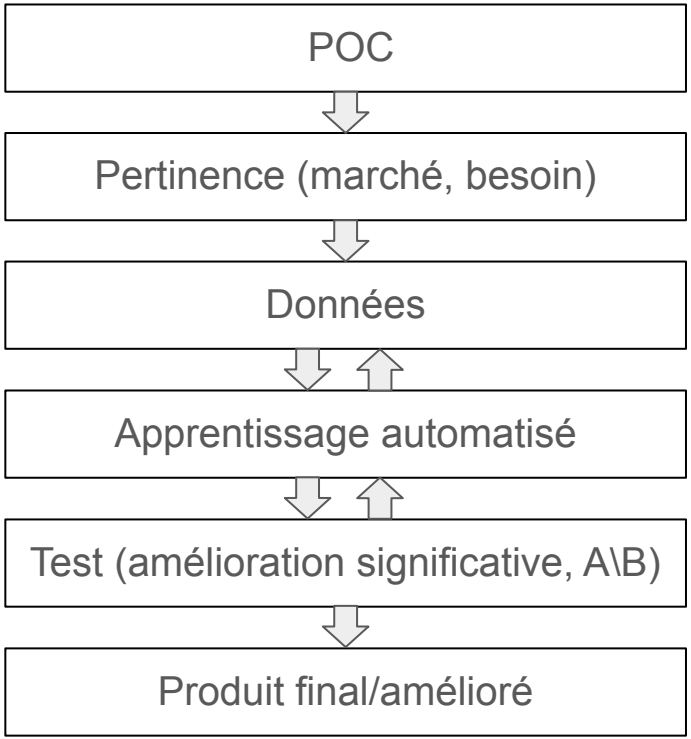
Il y a une différence de perspective - même au niveau de la conception du produit

- motivation : question/problème
- robustesse des conclusions
- « faire parler » les données
- établir de relations entre des variables
- « expliquer »
- utilisateur final : les décideurs
- communiquer, pédagogie

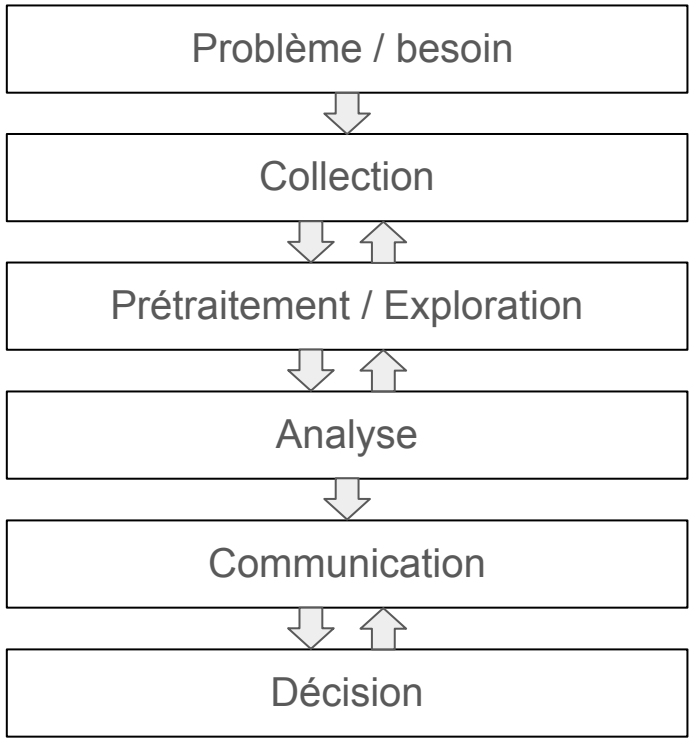
- motivation : produit/résultat
- qualité de la prédiction
- utiliser les données pour entraîner un modèle
- fiabilité, précision
- « généraliser »
- utilisateur final : usager

Phases projet data analysis vs machine learning





« Cycle de l’analyse »



Question métier, observation intuition

Collecte des données, stockage, gestion (ETL, pipeline, infra)

nettoyage, vérification de l’intégrité, data viz, agrégation, labellisation, segmentation, métriques

sélection d’un modèle, tests d’hypothèse, interpréter, comparer

data viz, partager les interprétations, impact des décisions

mise en œuvre opérationnelle des décisions

Actions et points de vigilance

- Connaissance métier
- Poser des questions (fermées, structurées)
- (re)Définir le problème
- Comprendre le contexte
- Communiquez !
- Premiers éléments de pédagogie

- Maintenir l'intégrité de la donnée
- Vérifier le nettoyage
- Tester les données
- Choisir des métriques
- Construire des features
- Agréger (par pays, produit…)
- DÉFINIR QUI LABELLISE LES DONNÉES

- Mettre en œuvre les outils d'analyse
- Sélectionner des modèles
- Tests statistiques
- Comparer les modèles
- Interpréter les paramètres, les indices
- Évaluer la significativité, les tailles d'effet
- Tirer des conclusions

Problème → Prétraitement → Analyse → Agir

Collection

- D'où vient la donnée (collecte, méthode, origine) ?
- Idenfier les formats, les protocoles, les différents types de sources
- Identifier les biais potentiels
- S'entendre sur les accès, les droits, l'anonymisation (RGPD, éthique)
- COPIER LES DONNÉES BRUTES

Exploration

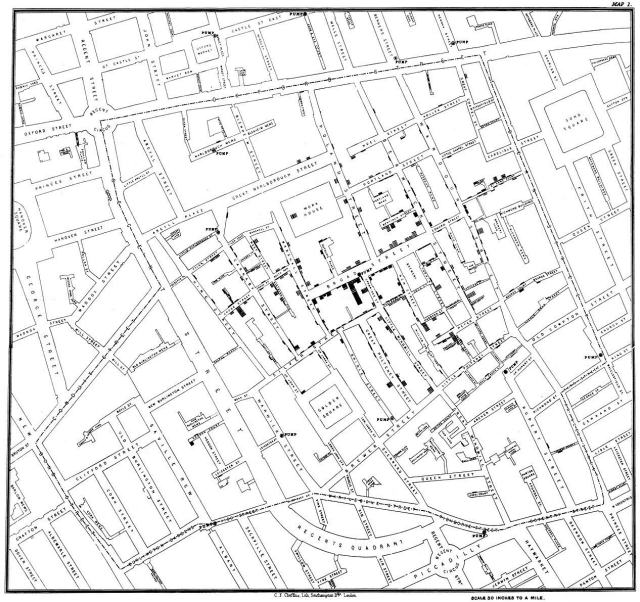
- Trier les données
- Filtrer les données
- Visualiser les données
- Rechercher des patterns
- Formuler des hypothèses

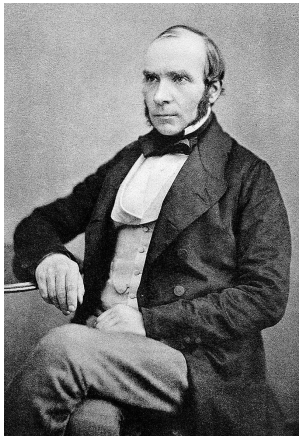
Make it scalable ! (package, pipeline…)

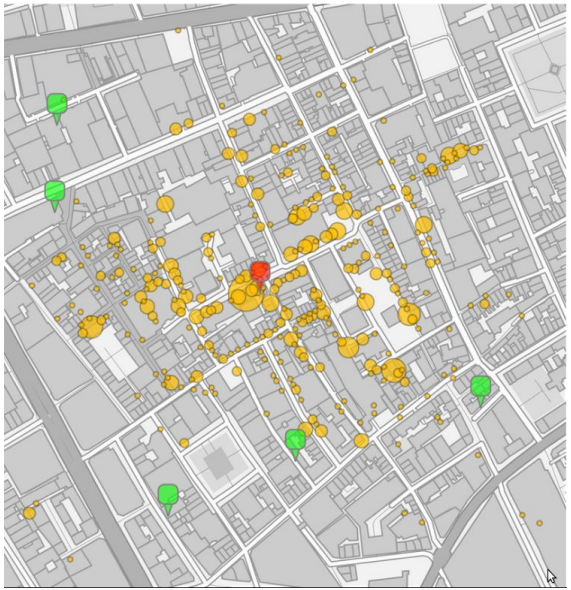
Communiquer

- Créer des dataviz parlantes
- Vulgariser, clarifier, pédagogie
- Aider à prendre des décisions éclairées par les faits
- Créer une « narration »
- Traduire les conclusions en éléments opérationnels
- Mettre en perspective (« si vous faites ça il se passera ça, si au contraire… »)

Puissance des Dataviz : John Snow et l'épidémie de 1854







Voir aussi :
https://archive.cdc.gov/#/details?url=https://www.cdc.gov/csels/dsepd/ss1978/lesson1/section2.html
https://storymaps.arcgis.com/stories/9fc482aa122849f8af4008c38991000e

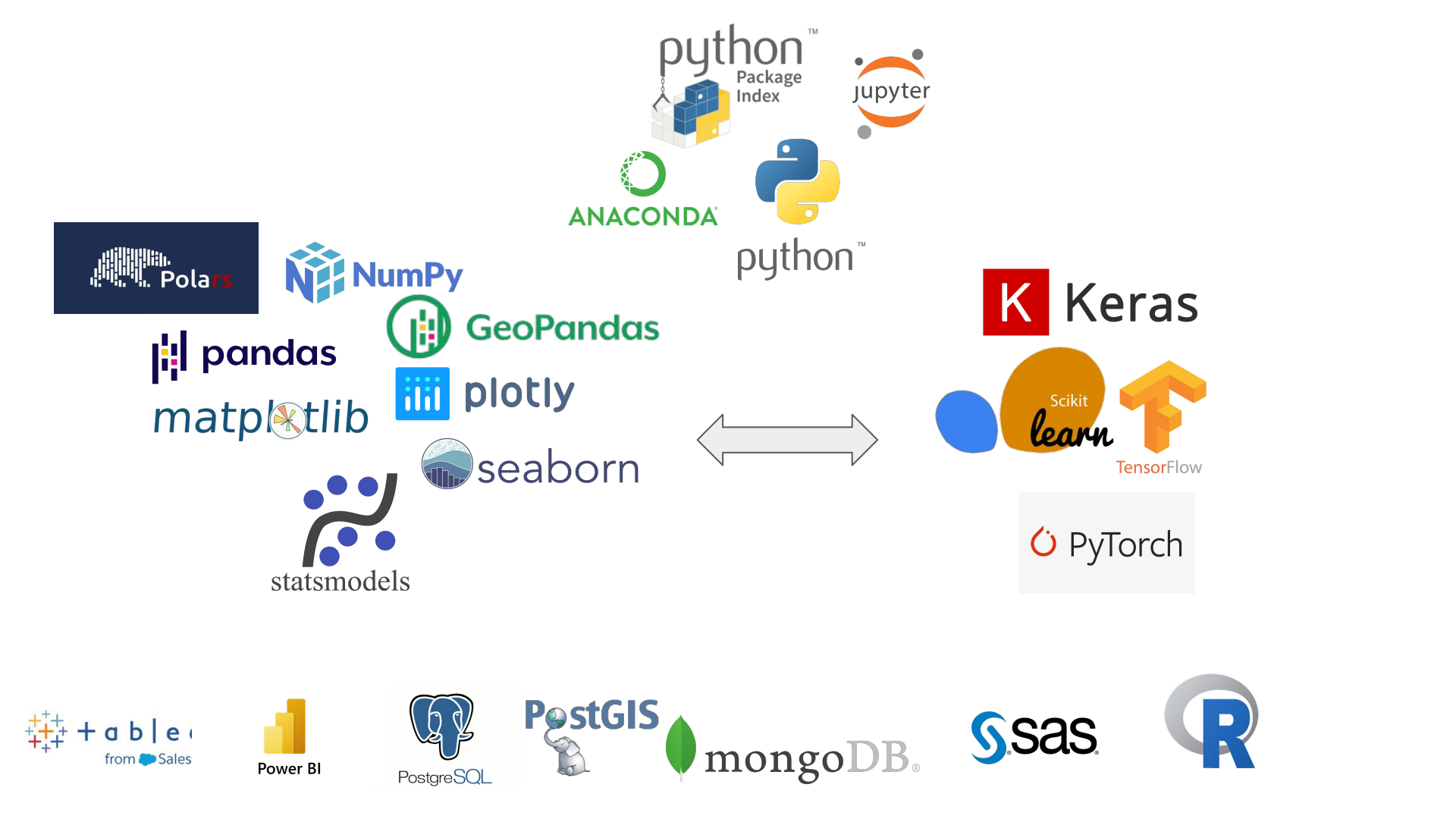
Sources de données ouverte

- Kaggle
- Sites institutionnels :
  - data.gouv.fr ou IGN par exemple
  - https://inpn.mnhn.fr/accueil/donnees-referentiels
  - https://data.giss.nasa.gov/
  - https://infoterre.brgm.fr/
  - https://data.worldbank.org/ ou https://data.un.org/
- Initiatives « ONG » ou indépendantes :
  - OpenFoodFact, voir la doc pour récupérer les données.
  - https://www.opentopography.org/
  - https://ourworldindata.org/

Questions Éthiques

- Données / propriété
- Données / personnes
- RGPD :
  - étendre les droits des personnes
  - responsabilité jusqu'aux sous-traitants
  - étendre la régulation (même référent pour coopération entre États applicant RGPD)

Outils


python
Package
Index
jupyter
ANACONDA
python
Polars
NumPy
GeoPandas
pandas
plotly
matplotlib
seaborn
statsmodels
Keras
Scikit
learn
TensorFlow
PyTorch
+ a b l e a u
from Sales
Power BI
PostgreSQL
PostGIS
mongoDB.
sas
R

Environnements virtuels

- éviter d'installer les différentes bibliothèques « à la base » de son système
- module **`venv`** (officiel) :

  `$ python3 -m venv <nom_projet>`
- `virtualenv` : **`venv`** ne dispose que d'un sous-ensemble de ses fonctionnalités, `pyenv-virtualenv` permet de gérer en outre plusieurs versions de python

  `$ pyenv virtualenv <version> <nom_projet>`
- `conda / miniconda` : facilite l'installation de nombreux packages de data science, mais plus lourd et lent

  `$ conda install <package>`

Cours sur les environnements virtuels
Un fil Stack Overflow qui récapitule les différents outils de gestion d'environnement sur python

Notebooks

- https://jupyter.org/
  - Jupyter notebook : interface classique (historique)
  - Jupyter lab : interface plus intégrée, interactive et flexible
  - `$ pip install jupyterlab`
- Tour du propriétaire (-> démo)
- À savoir :
  - Comme dans `vi`, **<esc>** permet de passer en mode « commande » (**<entrée>** édition):
    - **<esc>-M** -> markdown
    - **<esc>-Y** -> revient à l'écriture de code
    - **dd** -> supprime une cellule
  - **<shift>-m** -> regroupe (merge) deux cellules
  - **<tab>** pour autocomplétion
  - Jupyterlab dispose d'un terminal, mais on peut écrire des commandes précédées d'un `'!'`
  - on peut installer des extensions
  - Cheatsheet MD. On peut faire de la mise en forme avec HTML/CSS. Intégrer du LateX.

Notebook to slides (Reveal.js)

- **`$ jupyter nbconvert -to slides -no-input -post serve <notebook.ipynb>`** pour convertir en slides .js ou alors Files / Save and export… / .html après avoir changé le type des cellules
- Pour changer le type des cellules, afficher le panneau latéral droit (au besoin `view -> appearance -> show right sidebar`) et dérouler « common tools » puis sélectionner le type de cellule :
  - slide -> chapitres
  - sub-slide -> diapo en dessous
  - fragment -> sous-partie de la diapo précédente, apparaîtra quand appui sur flèche du bas
  - skip -> cellule à sauter (généralement cellule avec du code - parfois il vaut mieux recourir à `-no-input`)
  - notes -> cellule à ne pas afficher

Packages

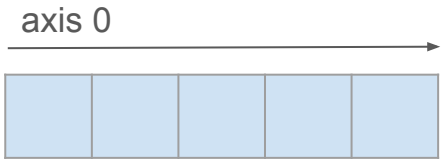
- Les notebooks ne sont pas des IDE, encore moins des environnements pour la mise en production !
- Une fois que l'on a fait son exploration, ses visualisations, comparé des modèles on « fixe » son code éprouvé dans des packages qu'on importera dans les nb suivants
- Nettoyer régulièrements ses notebooks, fusionner les cellules, mettre en forme (sections)

Structure d'un projet :

```
Mon_projet
|- data
|    |- csv    on fera référence aux données par des chemins relatifs
|    |- description_data
|- notebooks
|- projet
     |- getdata.py      ATTENTION une classe par fichier/script !
     |- preprocessing.py
     |- __init__.py fichier vide qui indique que c'est un package importable
```

NumPy

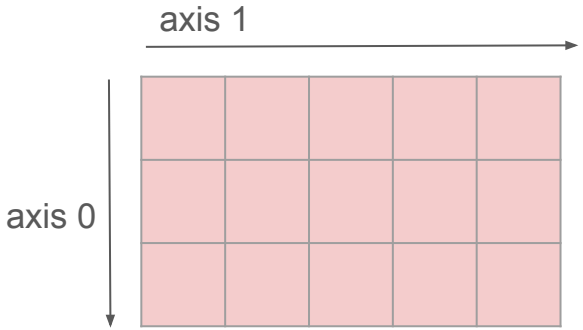
- bibliothèque avec des bouts écrits en C/C++ et Fortran pour optimiser le traitement d'objets *arrays* à n dimension
- ce sont des objets assez compliqué en natif (notamment pour parcourir les colonnes)
  `np.array[2, 1:4]`



shape (5,)

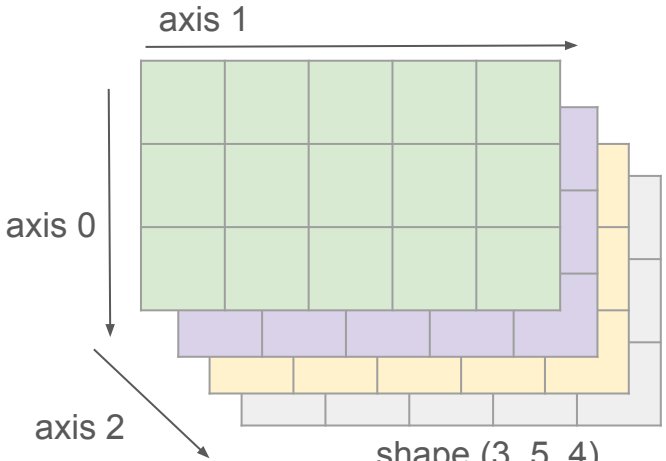
1 dimension

`np.sum(array, axis=1)`



shape (3, 5)

2 dimensions



shape (3, 5, 4)

3 dimensions

Toujours dans cet ordre :

LIGNES / COLONNES

Numpy (2)

- « Vectorisation » : les méthode numpy (np.sum(), np.mean()…) sont ~100x plus rapide que de boucler sur un ensemble de listes ou des arrays
- on dispose d'autres facilités comme le boolean indexing :

```
a = np.array([[1, 3, 5],
              [2, 4, 6]])
```

1	3	5
2	4	6

`a [a >= 4] = 0`

1	3	0
2	0	0

Boolean mask

false	false	true
false	true	true

- les ndarrays doivent être homogènes (un seul datatype dans toute l'array)
- pas d'index de colonnes et de ligne
- il faut construire à la main de nombreuses méthodes pour des analyses sophistiquées
- utilisé surtout pour prétraiter de manière optimisée des formats de données spécifiques (geometry, images, sons, nuages de points…)

Pandas

Panda Serie :

1D-array

index \ columns	value
idx 1	1
index 2	2.0
3	3
un autre	‘four’

Pandas serie

Panda Dataframe :

index \ columns	col 1	col 2	col 3	col 4
ligne 1	**D**			
ligne 2		**A**		
ligne 3			**T**	
ligne 4				**A**
	Serie 1	Serie 2	Serie 3	Serie 4

Les objets pd.dataframe() sont des assemblages de colonnes (cf. efficacité .apply() selon ligne ou colonne)

.iloc() et loc()

- `df['nom_col']` -> sélectionne une colonne
- `df[['nom_col1', 'nom_col2']` -> plusieurs colonnes

Pour sélectionner des lignes :

- `df.loc[0:5, ['nom_col1', 'nom_col2']]`
- `df.setindex(inplace=True)` peut-être utile avant d'utiliser `.loc()`
- `df.iloc[0]` -> sélectionne la première ligne (retourne une série)
- `df.iloc[[0]]` -> idem (retourne un dataframe !)
- `df.iloc[0, 1]` -> première ligne, 2e colonne (une cellule)
- `df.iloc[1:3, 0:3]` -> ligne 2 à 3 et colonnes 1 à 3 (un bloc de cellules)
- `df.iloc[lambda l: l.index %2 == 0]` -> sélectionne lignes paires

Boolean indexing

Le boolean indexing permet de sélectionner certaines lignes :

`selection_100 = df[df['col1']>100]` -> les lignes où valeur col1 > 100

ATTENTION : cela crée une référence vers une partie (sélection) d'un objet en mémoire vers lequel pointe df. C'est ce que l'on appelle une **vue**. Si on fait ensuite des opérations sur **`selection_100`** ou **`df`**, cela peut avoir des effets de bords importants.

Le mieux est de faire une copie explicite :

```
selection_100 = df[df['col1']>100].copy()
```

Méthodes utiles en boolean indexing : **`.isin()`**, **`.str.contains()`**, **`.str.strip()`**, l'opérateur **`~`** (bitwise not) également.

Dataframes

```
df = pd.dataframe([[val1-1, val1-2, val1-3,…], [val2-1, val2-2,…], …],
                  columns = [col1, col2, …],
                  index = [idx1, idx2, …])
```

```
data = {'ligne1': [val1-1, val1-2,…], 'ligne2': [val2-1,…]}
```

```
df = pd.DataFrame.from_dict(data,
                  orient='index',
                  columns=['col1', 'col2', …])
```

```
df = pd.read_csv('<nom_fichier>',
                  decimal=',',
                  delimiter=';',
                  header=None)
```

Bien sûr il y a aussi les méthode `.to_csv()` et `.to_dict()` !

Attributs et méthodes

Pour avoir un premier « sens » des données dans un dataframe :

Attributs

- `df.columns`
- `df.index`
- `df.values = df.to_numpy()`
- `df.dtypes`
- `df.shape`

Méthodes

- `df.info()`
- `df.describe()`
- `df.head()`
- `df.tail()`
- `df.isnull.sum()`
- `df.<col>.unique()`
- `df.<col>.plot(kind='…')`

Manipuler les données dans un dataframe

Les df sont dotés de méthodes pour réaliser des opérations comparables à du SQL :

- `.merge()` avec `join='…'` pour préciser la jointure
- `.groupby()` que l'on peut chaîner avec `.sum()` ou `.count()`, etc.
- `.sort_index(ascending = <True/False>`
- `.sort_values(by='column', ascending = <True/False>, na_position = 'first')`

Il faudra par ailleurs souvent convertir les formats de date… (attentions aux versions de pandas si vous récupérer des scripts !

- `.to_datetime()` du type `objet` (string en général) vers `timestamp`
- page sur les formats de date (p. ex. : `%a %d %b %Y`)

Mise en pratique

Clôner le dépot https://github.com/virgilus/science-des-donnees créé par Virgile Pesce (un collègue d'Adalab), et se rafraîchir la mémoire

Les cheatsheets de datacamp.org sont excellentes :

- Numpy (ancienne version)
- Pandas (ancienne version)

La cheatsheet sur le site de pandas est très claire et de grande qualité.

Geopandas

cf. notebook dans https://github.com/Jehadel/Geopandas-basics

Cheatsheet

CRS (code EPSG) : voir cette page sur les projections

Ou les pages Wikipédia sur :

- projection de Mercator
- projection conique conforme de Lambert
- systèmes de coordonnées cartographiques