# TD 3 : GeoPandas

Partie 1 : Fondamentaux & Chargement des Données

**M2 GER**
Université de Nantes -- IGARUN

# 1. L'Environnement GeoPandas

# Qu'est-ce que GeoPandas ?



**Pandas** gère des tableaux (Excel-like).
**GeoPandas** étend Pandas pour gérer des données géographiques.

- Il permet de faire en Python ce que vous faites dans QGIS.
- Il repose sur des géants : **GDAL** (lecture), **Shapely** (géométrie), **Fiona** (fichiers).

**GeoDataFrame**

=

DataFrame + Geometry

# Le Problème de l'Installation



GeoPandas n'est **pas** une simple librairie Python pure. Elle dépend de bibliothèques C++ compilées (GDAL, GEOS, PROJ).

> **Attention Danger**
>
> Si vous tentez un simple `pip install geopandas` sous Windows, cela échouera souvent car il manquera les fichiers binaires (.dll) de GDAL. C'est la cause n°1 des problèmes en TD.

Nous devons utiliser **Conda**, qui est conçu pour gérer ces dépendances complexes.

# Installation Sécurisée



Fermez Jupyter Notebook. Ouvrez **Anaconda Prompt** (l'écran noir) et exécutez ces commandes :

```
# Option 1 : Installation directe dans l'environnement de base
conda install -c conda-forge geopandas fiona pyogrio matplotlib mapclassify

# Option 2 (Si Option 1 échoue) : Création d'un environnement dédié
conda create -n geo_env python=3.9
conda activate geo_env
conda install -c conda-forge geopandas
```

*(Le téléchargement peut prendre quelques minutes. Validez par 'y' si demandé).*

# Vérification de l'Installation



Rouvrez Jupyter Notebook. Dans une nouvelle cellule, testez les imports critiques :

```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt

print("Installation réussie ! Version :", gpd.__version__)
```

Si vous n'avez pas de message d'erreur rouge, nous pouvons commencer.

# 2. Données & Structure

# Notre Terrain de Jeu : Nantes



Nous allons travailler avec des données réelles issues de l'Open Data de Nantes Métropole.

### JEU DE DONNÉES 1 : LES QUARTIERS

- **Contenu :** Limites administratives des 11 quartiers.
- **Type géométrique :** Polygones.
- **Source :** `data.nantesmetropole.fr`

### JEU DE DONNÉES 2 : LES PARCS

- **Contenu :** Parcs et Jardins de la ville.
- **Type géométrique :** Points (Attention !).

# Récupération des URLs



GeoPandas peut lire directement des fichiers depuis Internet (GeoJSON). Copiez ces URLs dans votre notebook :

```
# URL des Quartiers (Polygones)
url_quartiers =
"https://data.nantesmetropole.fr/api/explore/v2.1/catalog/datasets/244400404_quartiers-
admin-nantes-2015/exports/geojson"

# URL des Parcs (Points)
url_parcs =
"https://data.nantesmetropole.fr/api/explore/v2.1/catalog/datasets/244400404_parcs-
jardins-nantes/exports/geojson"
```

# Lecture des Fichiers



La fonction magique est `read_file()`.

```
# Chargement des données
quartiers = gpd.read_file(url_quartiers)
parcs = gpd.read_file(url_parcs)

# Vérification des dimensions (Lignes, Colonnes)
print("Quartiers :", quartiers.shape)
print("Parcs :", parcs.shape)
```

Si cela échoue (erreur SSL ou réseau), téléchargez les fichiers manuellement et mettez le chemin local (ex: `"data/quartiers.geojson"`).

# Anatomie d'un GeoDataFrame



```
quartiers.head(3)
```

Observez la dernière colonne : **geometry**.

- C'est elle qui contient l'information spatiale.
- Le format est du **WKT (Well Known Text)** : `POLYGON ((-1.55 47.21, ...))`.
- Les autres colonnes sont des attributs classiques (Nom, ID, etc.).

# Inspection des Parcs



```
parcs.head(3)
```

Regardez bien la colonne `geometry` des parcs.

> **Observation :** Vous devriez voir `POINT (-1.56 47.22)`.
>
> C'est crucial pour la suite : bien que ce soient des "zones" dans la réalité, ce fichier les représente comme des **points** (probablement le centroïde ou l'entrée).

# 3. Systèmes de Coordonnées (CRS)

# Qu'est-ce que le CRS ?



**CRS = Coordinate Reference System.**
Un jeu de données spatiales n'a de sens que si l'on sait comment projeter ses coordonnées sur la Terre.
Chaque système a un code unique **EPSG**.

**EPSG:4326 (WGS 84)**
Système mondial (GPS).
Unité : **Degrés**.
*Par défaut dans les GeoJSON.*

**EPSG:2154 (Lambert-93)**
Système officiel France.
Unité : **Mètres**.
*Indispensable pour calculer des surfaces.*

# Vérifier le CRS Actuel

Code

GeoPandas stocke l'info du CRS dans l'attribut `.crs`.

```
print(quartiers.crs)
```

**Résultat attendu :** `EPSG:4326`.
Cela signifie que nos coordonnées (ex: -1.55, 47.21) sont des latitudes/longitudes en degrés.

# Pourquoi Projeter ?



**Pourquoi ne pas rester en WGS84 ?**

Si vous calculez une distance ou une surface en degrés :

- Le résultat sera en "degrés carrés" (inintelligible).
- Le calcul sera mathématiquement faux car un degré de longitude n'a pas la même taille à l'équateur qu'à Nantes.

Pour faire de l'analyse, il faut passer en **Mètres**.

# La Reprojection (to_crs)



La méthode `to_crs()` permet de convertir les coordonnées.

```
# Convertir en Lambert-93 (France Métropolitaine)
quartiers_L93 = quartiers.to_crs(epsg=2154)
parcs_L93 = parcs.to_crs(epsg=2154)

# Vérifier le résultat
print(quartiers_L93.crs)
quartiers_L93.head()
```

Regardez la géométrie : les chiffres sont devenus énormes (ex: 350000, 6700000). Ce sont des mètres !

# Visualisation Rapide



Utilisons `matplotlib` intégré pour vérifier que tout s'aligne.

```
fig, ax = plt.subplots(figsize=(10, 10))

# Afficher les quartiers en fond (blanc, bord noir)
quartiers_L93.plot(ax=ax, color="none", edgecolor="black")

# Afficher les parcs par dessus (points verts)
parcs_L93.plot(ax=ax, color="green", markersize=5)

plt.title("Carte de contrôle : Nantes")
plt.show()
```