OpenCenter

Datacenters européens — PeeringDB

Suivi et synthèse de projet

Matière	NSI Terminale – Traitement de données & Bases de données
Auteur	Arthur Corcessin
Date	Octobre 2025 - Février 2026
Source	https://www.peeringdb.com/api/fac?depth=0

1. Présentation du jeu de données

Source

PeeringDB (https://www.peeringdb.com) est une base de données collaborative recensant les infrastructures réseau mondiales : datacenters, points d'échange internet (IXP), opérateurs réseau. L'API publique est accessible sans authentification.

```
https://www.peeringdb.com/api/fac?depth=0
```

Pourquoi ce jeu de données ?

- Latitude et longitude présentes → cartographie directe
- Plus de 2020 enregistrements après filtrage Europe
- Données réelles, mises à jour régulièrement, librement exploitables
- Données riches : adresse, pays, réseaux connectés, points d'échange
- Thématique concrète : internet, infrastructure, géographie mondiale

Statistiques globales du jeu final

Indicateur	Valeur
Total enregistrements (Europe)	2020
Nombre d'attributs (colonnes)	26
Pays représentés	38
Datacenters avec coordonnées GPS	2018
Datacenters sans GPS (résiduels)	2
Moyenne réseaux connectés / DC	11.8
Maximum réseaux connectés (1 DC)	604
Minimum réseaux connectés	0
Moyenne points d'échange (IX)	0.91
Maximum points d'échange (IX)	34

Top 5 des pays par nombre de datacenters

Rang	Code	Nb datacenters
1	DE	331
2	GB	247
3	FR	221
4	NL	161
5	RU	105

Top 5 des datacenters les plus connectés

Nom	Ville	Pays	Réseaux
Digital Realty Frankfurt FRA1-16	Frankfurt	DE	604
Equinix FR5 - Frankfurt, KleyerStrasse	Frankfurt	DE	509
Telehouse - London	London	GB	492

(Docklands North)			
NIKHEF Amsterdam	Amsterdam	NL	451
Telehouse - Paris 2 (Voltaire - Léon Frot)	Paris	FR	343

Évolution des créations par année

Année	Nouveaux datacenters
2010	386
2011	62
2012	58
2013	79
2014	96
2015	75
2016	122
2017	146
2018	127
2019	144
2020	138
2021	136
2022	148
2023	113
2024	108
2025	76
2026	6

2. Arborescence du projet

```
OpenCenter/
├── data/
│   ├── datacenter.csv          ← Jeu de données nettoyé (Europe)
│   ├── datacenter.sqlite3      ← Base SQLite3 (tables : datacenter, pays)
│   └── pays_europe.csv         ← 46 pays européens (référence JOIN)
│
├── scripts/                    ← À lancer une seule fois (pipeline de données)
│   ├── json_to_csv.py          ← fac-0.json → datacenter.csv
│   ├── clean_csv.py            ← Nettoyage et filtrage Europe
│   ├── csv_to_sqlite.py        ← datacenter.csv → datacenter.sqlite3
│   └── import_pays.py          ← pays_europe.csv → table pays
│
├── output/                     ← Cartes HTML (générées automatiquement)
│   ├── carte_datacenters.html  ← Carte globale (cluster)
│   ├── carte_par_pays.html     ← Bulles proportionnelles par pays
│   ├── carte_fr.html           ← Datacenters France
│   └── carte_de.html           ← Datacenters Allemagne
│
├── queries.sql                 ← Activité 4 : 15 requêtes SQL commentées
├── db_query.py                 ← Activité 5 : accès Python → BDD
├── carte.py                    ← Activité 6 : cartes Folium
├── jointure.py                 ← Activité 7 : JOIN + nouvelles cartes
├── geocode.py                  ← Bonus  : géocodage adresses manquantes
├── interface.py                ← Activité 8 : interface graphique Tkinter
```

3. Recréer le jeu de données depuis zéro

Cette section explique le pipeline complet pour passer du fichier brut de l'API PeeringDB à une base de données SQLite3 exploitable.

Étape 0 — Prérequis

Créer et activer l'environnement virtuel puis installer les dépendances.

```
pip install pandas folium geopy certifi
```

Étape 1 — Télécharger le JSON brut

Télécharger cette URL et renommer le fichier en fac-0.json à la racine du projet. PeeringDB ne fournit pas d'export CSV natif : tout passe par l'API JSON.

```
https://www.peeringdb.com/api/fac?depth=0
```

Étape 2 — Convertir JSON → CSV

Lit fac-0.json, extrait le tableau data[], écrit les en-têtes puis les valeurs. Produit data/datacenter.csv avec environ 2 600+ lignes et 37 colonnes.

```
python scripts/json_to_csv.py
```

Étape 3 — Nettoyer le CSV

Supprime 13 colonnes inutiles, filtre uniquement les datacenters européens (région Europe). Résultat : 2020 lignes × 26 colonnes.

```
python scripts/clean_csv.py
```

Étape 4 — Importer dans SQLite3

Crée automatiquement la table datacenter avec les bons types (INTEGER / TEXT), insère les 2020 lignes. Option --reset pour réimporter après modification du CSV.

```
python scripts/csv_to_sqlite.py
```

Étape 5 — Importer les pays

Crée la table pays (46 pays européens avec capitales, population) dans la BDD. Nécessaire pour les jointures de l'activité 7.

```
python scripts/import_pays.py
```

Étape 6 — Géocoder les adresses manquantes

Résout les coordonnées GPS manquantes via Nominatim (OpenStreetMap). Résultat : 2018/2020 datacenters localisés (2 résiduels).

```
python geocode.py
```

Étape 7 — Lancer les scripts

Exécute les analyses (activités 5, 6, 7) et génère les cartes dans output/.

```
python db_query.py && python carte.py && python jointure.py
```

4. Activité 1 — Choix et analyse du jeu

Le jeu de données a été exploré après la première conversion JSON → CSV.

Observation	Valeur
Lignes (monde entier)	~2 655
Colonnes avant nettoyage	37
Colonnes après nettoyage	26
Lignes après filtrage Europe	2020
Latitude / longitude présentes	✔ Oui (sur la grande majorité)
Champs multi-lignes (notes)	✔ Gérés par Pandas

- **Attributs retenus pour l'exploitation :** name, city, country, latitude, longitude, net_count, ix_count, website, org_name

5. Activité 2 — Nettoyage Python

Trois opérations de nettoyage successives : conversion, suppression de colonnes, filtrage géographique.

▸ Suppression des colonnes inutiles (scripts/clean_csv.py)

```
df.drop(columns=['campus_id','name_long','tech_email','tech_phone',
  'available_voltage_services','diverse_serving_substations',
  'property','status_dashboard','rencode','npanxx','logo','floor','suite'])
```

▸ Filtrage Europe

```
df = df[df['region_continent'] == 'Europe']
```

Difficultés rencontrées

- Colonnes JSON imbriquées dans le CSV (social_media) → gérées nativement par Pandas
- eval() utilisé pour le filtre de colonne → risque de sécurité en production, acceptable ici
- Résultat : 2020 lignes × 26 colonnes

6. Activité 3 — Import dans SQLite3

L'import se fait entièrement via le script csv_to_sqlite.py, sans manipulation manuelle.

▸ Import automatique (remplace DB Browser)

```
python scripts/csv_to_sqlite.py
python scripts/csv_to_sqlite.py --reset   # recréer la table
```

▸ Schéma généré automatiquement (extrait)

```
CREATE TABLE "datacenter" (
    "id"        INTEGER,
```

```
    "net_count" INTEGER,
    "ix_count"  INTEGER,
    "latitude"  TEXT,     -- PeeringDB livre les coords en TEXT
    "longitude" TEXT,
    ...                   -- 21 autres colonnes TEXT
);
```

Difficultés rencontrées

- latitude/longitude stockées en TEXT → CAST(latitude AS REAL) dans toutes les requêtes
- DB Browser reste utile pour explorer visuellement la BDD (activité 4)

7. Activité 4 — Requêtes SQL

---

15 requêtes dans queries.sql couvrant comptages, agrégats, filtres et analyses temporelles.

▸ Nombre total d'enregistrements

```
SELECT COUNT(*) AS nb FROM datacenter;
-- Résultat : 2020
```

▸ Moyenne / max réseaux connectés

```
SELECT ROUND(AVG(net_count),2) AS moy, MAX(net_count) AS max
FROM datacenter;
-- moy : 11.8  |  max : 604
```

▸ Top 5 pays

```
SELECT country, COUNT(*) AS nb FROM datacenter
GROUP BY country ORDER BY nb DESC LIMIT 5;
```

▸ Datacenters avec GPS valide

```
SELECT COUNT(*) FROM datacenter
WHERE latitude IS NOT NULL AND latitude != ''
  AND CAST(latitude AS REAL) BETWEEN -90 AND 90;
-- Résultat : 2018
```

8. Activité 5 — Accès Python à la BDD

Le module sqlite3 (bibliothèque standard) permet d'interroger la base depuis Python.

▸ Connexion et requête minimale (db_query.py)

```
import sqlite3, os
FICHIER_BDD = os.path.join('data', 'datacenter.sqlite3')
conn = sqlite3.connect(FICHIER_BDD)
conn.row_factory = sqlite3.Row   # accès par nom de colonne
c = conn.cursor()
c.execute('SELECT * FROM datacenter')
liste = c.fetchall()
print('J\'ai', len(liste), 'enregistrements.')  # → 2020
```

Difficultés rencontrées

- Sans row_factory, les résultats sont des tuples indexés (row[0], row[1]...)
- Avec row_factory = sqlite3.Row, on accède par nom : row['name'], row['city']

9. Activité 6 — Carte Folium

Folium génère des cartes HTML interactives basées sur OpenStreetMap, affichables dans n'importe quel navigateur.

▸ Création de base (carte.py)

```
import folium
from folium.plugins import MarkerCluster

carte = folium.Map(location=(48.8, 10.0), zoom_start=5,
                   attr='© Contributeurs OpenStreetMap')
cluster = MarkerCluster().add_to(carte)
folium.Marker(
    location=(48.85, 2.35),
    popup='Paris',
    icon=folium.Icon(color='blue', icon='server', prefix='fa')
).add_to(cluster)
carte.save('output/carte_datacenters.html')
```

Difficultés rencontrées

- 170 DC sans GPS au départ → résolus par geocode.py (2018/2020 localisés)
- Coordonnées en TEXT → CAST(...) dans la requête SQL + filtre BETWEEN -90 AND 90
- Icône 'server' invisible → ajout de prefix='fa' (Font Awesome)
- Cartes générées à la racine → déplacées dans output/ via os.path.join

10. Activité 7 — Jointure de tables

Une seconde table pays (46 pays européens) a été ajoutée à la BDD. La jointure se fait sur datacenter.country = pays.code_pays.

▸ Requête JOIN : nom complet du pays + nb de DC

```
SELECT p.nom_pays, d.country, COUNT(*) AS nb
FROM datacenter d
JOIN pays p ON d.country = p.code_pays
GROUP BY d.country ORDER BY nb DESC;
```

▸ Requête JOIN : moyenne réseaux vs population

```
SELECT p.nom_pays, COUNT(*) AS nb_dc,
       ROUND(AVG(d.net_count),1) AS moy_reseaux, p.population
FROM datacenter d
```

```
JOIN pays p ON d.country = p.code_pays
GROUP BY d.country ORDER BY moy_reseaux DESC;
```

Difficultés rencontrées

- La table datacenter ne contient que des codes ISO-2 (FR, DE…) → nom complet via JOIN
- Les pays sans datacenter apparaissent avec nb=0 grâce au LEFT JOIN dans carte_par_pays
- Trois nouvelles cartes générées : carte_par_pays.html, carte_fr.html, carte_de.html

11. Activité 8 — Interface Tkinter

Interface graphique sombre avec statistiques en temps réel, sélecteur de pays, tableaux et génération de cartes.

- Panneau gauche : stats globales + sélecteur de pays + 3 boutons de carte
- Onglet 'Datacenters (pays)' : liste des DC du pays sélectionné triés par nb réseaux
- Onglet 'Top 20 (réseaux)' : classement mondial des DC les plus connectés
- Les cartes générées s'ouvrent automatiquement dans le navigateur

Difficultés rencontrées

- Thème Tkinter gris par défaut → surcharge complète via ttk.Style(theme='clam')
- UI bloquée pendant la génération → self.update() appelé avant le calcul
- Import de jointure.py depuis interface.py → les deux doivent être à la racine

12. Bonus — Géocodage des adresses manquantes

Au départ, 170 datacenters n'avaient pas de coordonnées GPS. Le script geocode.py les résout via Nominatim (OpenStreetMap).

Indicateur	Avant	Après
DC sans GPS	170	2
DC avec GPS	1 830	2018
Taux de localisation	91,5 %	99.9 %

▸ Dry-run (test sans modifier la BDD)

```
python geocode.py --dry-run --limite 10
```

▸ Géocodage complet (~4 min pour 170 entrées)

```
python geocode.py
```

▸ Fix SSL macOS (certifi)

```
import ssl, certifi, geopy.geocoders
_ssl_ctx = ssl.create_default_context(cafile=certifi.where())
geopy.geocoders.options.default_ssl_context = _ssl_ctx
```

Difficultés rencontrées

- Erreur SSL sur macOS → certificats Python ≠ certificats système → fix avec certifi
- Limite Nominatim : 1 requête/seconde max → délai 1,1 s entre chaque appel
- Stratégie en cascade : adresse complète → code postal + ville → ville + pays

13. Tableau de bord des difficultés

#	Activité	Problème	Solution
1	Acquisition	Pas de CSV natif sur PeeringDB	Script json_to_csv.py
2	Nettoyage	eval() sur nom de colonne (fragile)	Conservé ; noté dette technique
3	BDD	latitude/longitude en TEXT	CAST(... AS REAL) dans les requêtes
4	Carte	170 DC sans GPS	Script geocode.py (Nominatim)
5	Géocodage	Erreur SSL macOS	Fix certifi + contexte SSL geopy
6	Carte	Icône 'server' invisible	prefix='fa' dans folium.Icon()
7	Carte	Coords aberrantes (0,0)	Filtre BETWEEN -90 AND 90
8	Structure	HTML générés à la racine	Dossier output/ + os.makedirs
9	Structure	Chemins relatifs cassés	os.path.join(__file__, ...) partout
10	Interface	UI bloquée pendant la carte	self.update() avant le calcul

14. Bilan et perspectives

Ce qui fonctionne

- Pipeline complet : JSON brut → CSV propre → SQLite3 (2020 enregistrements)
- 15 requêtes SQL documentées (comptages, agrégats, filtres, temps)
- Cartes interactives avec 2018 marqueurs clusterisés
- Taux de localisation GPS : 99.9 % (2 non résolus)
- Interface graphique Tkinter avec génération de cartes à la volée
- Jointure fonctionnelle entre les tables datacenter et pays

Perspectives d'amélioration

Amélioration	Description
Heatmap	Carte de chaleur proportionnelle au net_count (folium.plugins.HeatMap)
Filtre net_count	Curseur dans l'interface pour filtrer par nb de réseaux connectés
Mise à jour auto	Bouton 'Synchroniser' qui re-télécharge l'API et met à jour la BDD
Export PDF	Génération d'un rapport PDF depuis l'interface
Recherche	Barre de recherche par nom de datacenter dans l'interface

Dépendances Python

Package	Usage
pandas	Nettoyage du CSV (activité 2)
folium	Génération des cartes HTML (activités 6, 7, 8)
geopy	Géocodage Nominatim (bonus)
certifi	Fix SSL macOS pour geopy
sqlite3	Accès BDD (stdlib Python, toutes activités)
tkinter	Interface graphique (stdlib Python, activité 8)

Document rédiger le 28 février 2026