

# **TPs — STATISTIQUE EXPLORATOIRE SPATIALE**

## **Synthèse des 6 TPs**

ISE1 CL  
Année académique 2025/2026

Cheikh Ahmadou Bamba FALL

# Table des matières

Introduction .....	1
TP1 — Analyse spatiale du paludisme au Tchad .....	3
TP2 — Réseau routier et services sociaux (Tchad) .....	15
TP3 — Population et accessibilité aux services sociaux .....	27
TP4 — Cartographie des terres arables (Burundi) .....	39
TP5 — Ratio consommation des terres / croissance démographique .....	55
TP6 — Analyse complète des indices spectraux (Niger) .....	67
Conclusion générale .....	85

# **Introduction Générale**

Dans le cadre du cours de Statistiques Spatiales, plusieurs travaux pratiques ont été réalisés afin de développer progressivement des compétences techniques et analytiques en traitement de données géospatiales. Ces travaux ont couvert à la fois la manipulation de données vectorielles et raster, l'analyse statistique spatiale, la modélisation d'indicateurs internationaux, ainsi que l'intégration de données environnementales et socio-économiques. Le présent rapport vise à présenter de manière détaillée les objectifs, les méthodes mises en œuvre, les résultats obtenus ainsi que les difficultés rencontrées pour chacun des travaux réalisés.

# TP1 — Inspection des données géographiques et analyse du paludisme (Tchad)

## 1. Objet du TP (objectif concret)

Ce travail pratique a pour but de mettre en place un pipeline complet et rigoureux d'analyse géospatiale en Python à partir de deux sources de données : (i) un shapefile décrivant les limites administratives de niveau 1 du Tchad (23 régions) et (ii) un fichier tabulaire contenant des indicateurs de paludisme par région et par année. L'objectif concret est double. D'abord, vérifier la qualité technique des données géographiques (structure, projection, étendue, géométrie, surfaces, valeurs manquantes). Ensuite, intégrer les données de paludisme au shapefile par jointure, produire des cartes thématiques et des graphiques descriptifs, et enfin exporter des résultats exploitables (Shapefile, GeoJSON, CSV et rapport HTML).

## 2. Données utilisées

La couche vectorielle correspond aux régions du Tchad (niveau administratif 1) issue de GADM. Elle contient notamment les champs `NAME_1` (nom de la région) et la géométrie (polygone/multipolygone). Les données de paludisme sont fournies sous forme de fichier CSV (`Subnational Unit-data.csv`) contenant les colonnes `Name` (région), `Year` (année) et `Value` (incidence, en cas pour 1000 habitants), ainsi que d'autres variables descriptives.

## 3. Environnement de travail (bibliothèques)

Le traitement repose principalement sur GeoPandas pour la manipulation vectorielle, Pandas pour le traitement des tables, Matplotlib pour les visualisations et NumPy pour certains calculs. Les bibliothèques Fiona et Shapely sont utilisées en arrière-plan pour les formats géographiques et les géométries.

Listing 1 – Installation et import des bibliothèques

```
1 !pip install geopandas matplotlib pandas fiona shapely
2
3 import geopandas as gpd
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 import numpy as np
7 from pathlib import Path
8 import warnings
9 warnings.filterwarnings('ignore')
10
11 plt.rcParams['figure.figsize'] = (12, 8)
12 plt.rcParams['font.size'] = 10
```

## 4. Chargement des données

La première étape consiste à charger le shapefile des régions (niveau 1) et le fichier CSV contenant l'historique du paludisme. L'accès au shapefile se fait via `gpd.read_file`. Une gestion d'erreur est prévue afin d'identifier immédiatement un problème de chemin d'accès.

Listing 2 – Chargement du shapefile et du CSV malaria

```
1 import geopandas as gpd
2 import pandas as pd
3
4 # Chemins des fichiers
5 shapefile_path = "data/gadm41_TCD_1.shp"
6 malaria_data_path = "data/Subnational_Unit_data.csv"
7
8 # Lecture des données
9 gdf_tchad = gpd.read_file(shapefile_path)
10 df_malaria = pd.read_csv(malaria_data_path)
11
12 # Vérification rapide
13 print(gdf_tchad.head())
14 print(df_malaria.head())
```

## 5. Inspection des propriétés géographiques

Une inspection détaillée du GeoDataFrame est réalisée : nombre d'entités, colonnes disponibles, système de projection, étendue géographique et type de géométrie. Cette étape est essentielle car elle conditionne la fiabilité des cartes et des calculs ultérieurs.

### 5.1 Structure générale

Dans notre cas, le shapefile contient 23 entités, correspondant aux 23 régions du Tchad, avec 12 colonnes (dont `geometry`). La présence de `NAME_1` est déterminante pour la jointure future.

### 5.2 Système de coordonnées (CRS)

Le shapefile est en **EPSG :4326** (WGS84). Ce système est adapté à la cartographie globale, mais n'est pas approprié pour calculer des surfaces précises, car il est exprimé en degrés. Pour obtenir des superficies en  $\text{km}^2$ , une reprojection en coordonnées métriques est donc nécessaire.

Listing 3 – CRS, étendue et surfaces

```
1 print(gdf_tchad.crs) # EPSG:4326
2 bounds = gdf_tchad.total_bounds
3
4 # Reprojection en UTM pour surfaces
5 gdf_projected = gdf_tchad.to_crs('EPSG:32633') # UTM zone 33N
6 gdf_tchad['superficie_km2'] = gdf_projected.geometry.area / 1_000_000
```

### 5.3 Étendue géographique

L'emprise obtenue (bounding box) permet de vérifier que le shapefile couvre bien le territoire tchadien, et fournit une approximation du centre géographique, utile pour le centrage des cartes.

## 6. Cartographie de base des régions

Deux cartes sont produites : (i) une carte simple des régions et (ii) une carte choroplèthe représentant la superficie des régions. Le script enregistre cette sortie sous le nom `carte_tchad_base.png`.

Listing 4 – Cartes de base (régions et superficies)

```
1 fig, axes = plt.subplots(1, 2, figsize=(15, 6))
```

```

2
3 gdf_tchad.plot(ax=axes[0], color='lightblue', edgecolor='black',
4                 linewidth=0.5)
5 axes[0].set_title('Carte du Tchad - Vue simple')
6
7 gdf_tchad.plot(column='superficie_km2', ax=axes[1], cmap='YlOrRd',
8                  legend=True, edgecolor='black', linewidth=0.5)
9 axes[1].set_title('Superficies des régions (km2)')
10 plt.tight_layout()
11 plt.savefig('carte_tchad_base.png', dpi=300, bbox_inches='tight')

```

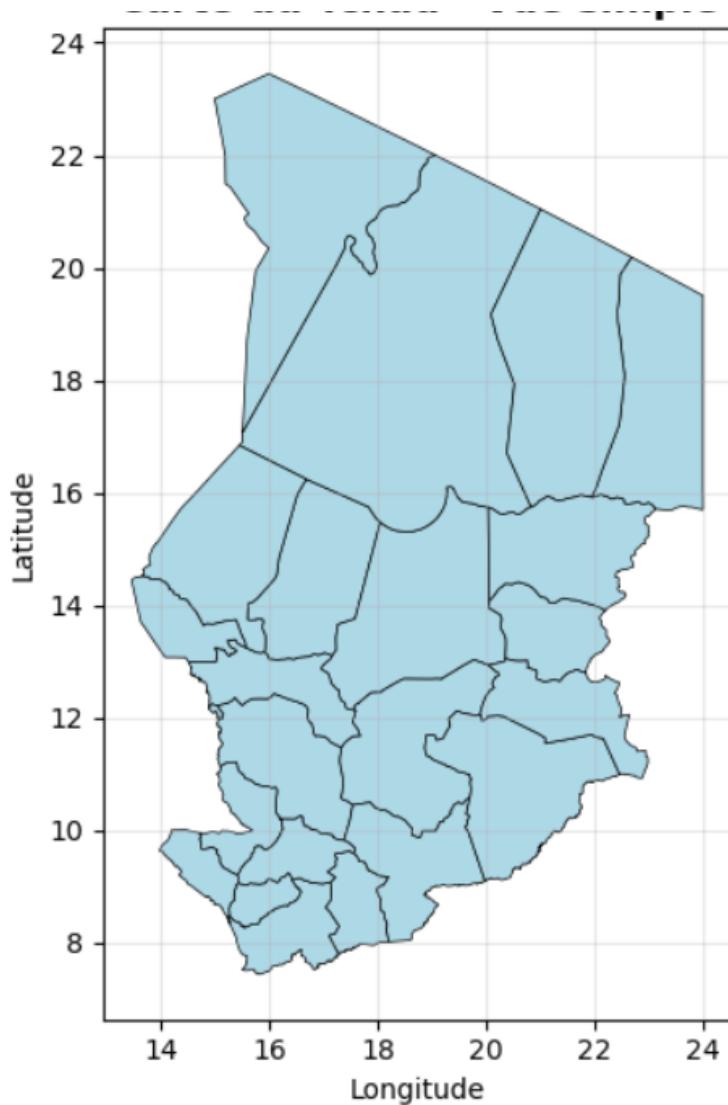


FIGURE 1 – Cartographie de base : régions du Tchad et superficies ( $\text{km}^2$ ).

## 7. Préparation et jointure des données de paludisme

Les données malaria couvrent plusieurs années. Pour produire une carte principale, l'approche retenue consiste à sélectionner l'année la plus récente disponible (dans l'exécution observée : 2024), puis à joindre la valeur d'incidence à la géométrie régionale.

Un point crucial est la cohérence des noms de régions. Dans les données malaria, certains noms peuvent être accentués différemment ou tronqués (*ex : Guéra vs Guera, Ouaddaï vs Ouaddai, Tandjilé vs Tandjile*). Le script prévoit donc une étape de nettoyage (**strip**) et la possibilité d'ajouter un dictionnaire de correspondance.

Listing 5 – Filtrage année récente + nettoyage noms + jointure

```
1 df_malaria.columns = df_malaria.columns.str.strip()
2 annee_recente = df_malaria['Year'].max()
3 df_malaria_recent = df_malaria[df_malaria['Year'] == annee_recente].copy()
4
5 df_malaria_recent['Name'] = df_malaria_recent['Name'].str.strip()
6 gdf_tchad['NAME_1'] = gdf_tchad['NAME_1'].str.strip()
7
8 correspondance_noms = {
9     'Guera': 'Gura',
10    'Ouaddai': 'Ouadda',
11    'Tandjile': 'Tandjil'
12 }
13
14 df_malaria_recent['Name_Clean'] =
15     df_malaria_recent['Name'].replace(correspondance_noms)
16
17 gdf_malaria = gdf_tchad.merge(df_malaria_recent,
18                                 left_on='NAME_1',
19                                 right_on='Name_Clean',
20                                 how='left')
21 col_cas_malaria = 'Value'
```

## 8. Visualisation : carte malaria, classes de risque et graphiques

La visualisation complète comprend quatre éléments : (i) une carte choroplèthe du taux d'incidence, (ii) une carte catégorielle en niveaux de risque, (iii) une série temporelle nationale (moyenne annuelle) et (iv) un classement des régions les plus affectées. La figure

est exportée sous analyse\_malaria\_tchad.png.

Listing 6 – Production de la figure globale (carte + risques + graphiques)

```
1 fig, axes = plt.subplots(2, 2, figsize=(16, 14))
2 fig.suptitle('Analyse de la Malaria au Tchad', fontsize=16,
3               fontweight='bold')
4
5 # Carte incidence
6 gdf_malaria.plot(column=col_cas_malaria, ax=axes[0, 0], cmap='Reds',
7                     legend=True, edgecolor='black', linewidth=0.5)
8
9 # Catgories de risque
10 gdf_malaria['categorie_risque'] = 'Pas de donne'
11 mask_valid = gdf_malaria[col_cas_malaria].notna()
12 gdf_malaria.loc[mask_valid, 'categorie_risque'] = pd.cut(
13     gdf_malaria.loc[mask_valid, col_cas_malaria],
14     bins=4, labels=['Faible', 'Moyen', 'lev', 'Trs lev']
15 ).astype(str)
16
17 # volution nationale
18 evolution = df_malaria.groupby('Year')['Value'].mean().reset_index()
19 axes[1, 0].plot(evolution['Year'], evolution['Value'], marker='o',
20                   linewidth=2)
21 plt.tight_layout()
22 plt.savefig('analyse_malaria_tchad.png', dpi=300, bbox_inches='tight')
```

## 9. Résultats numériques : statistiques et interprétation

À partir de l'année la plus récente, le script calcule la moyenne, la médiane, l'écart-type, le minimum et le maximum du taux d'incidence. L'exécution montre notamment que la région la plus affectée (année 2024) est **Mayo-Kebbi Ouest**, avec une incidence maximale d'environ **294,35 cas pour 1000 habitants**. Sur la période complète 2000–2024, la moyenne nationale passe d'environ 110,58 à 76,87 cas pour 1000 habitants, indiquant une baisse d'environ 30,5% sur la période, ce qui suggère une amélioration globale, malgré des disparités régionales persistantes.

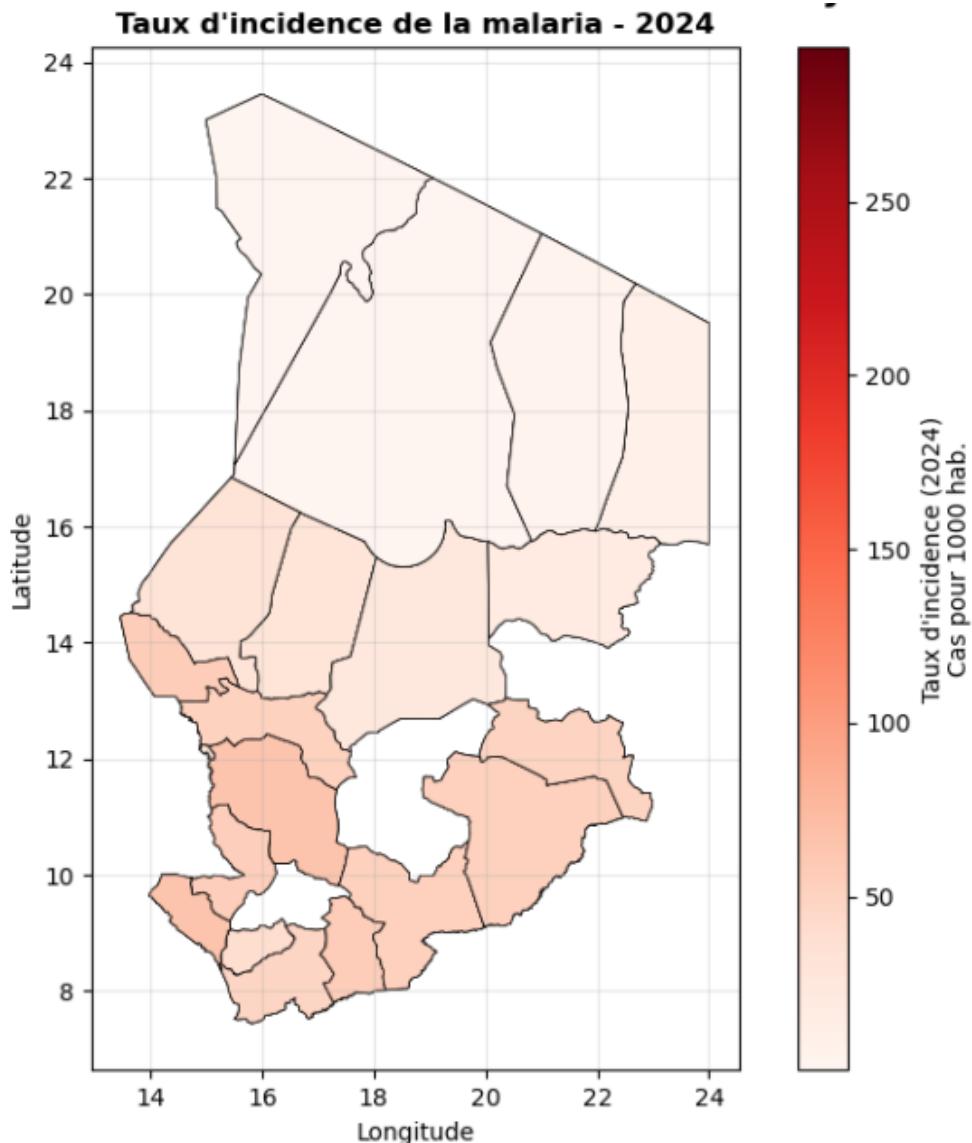


FIGURE 2 – Analyse malaria : incidence par région, catégories de risque et dynamique temporelle.

## 10. Export des résultats

Les résultats sont exportés sous plusieurs formats afin de faciliter leur réutilisation dans d'autres logiciels (QGIS, R, ArcGIS ou analyses statistiques). Les exports produits sont : (i) un shapefile enrichi, (ii) un GeoJSON, (iii) un fichier CSV des statistiques, et (iv) un rapport HTML simple.

Listing 7 – Exports Shapefile / GeoJSON / CSV / HTML

```

1 gdf_malaria.to_file("tchad_malaria_analyse.shp")
2 gdf_malaria.to_file("tchad_malaria_analyse.geojson", driver='GeoJSON')
3
4 stats_df = gdf_malaria[[c for c in gdf_malaria.columns if c != 'geometry']].copy()

```

```
5 stats_df.to_csv("statistiques_malaria.csv", index=False, encoding='utf-8')
```

## 11. Limites et difficultés rencontrées

La principale difficulté provient de la différence d'orthographe entre les noms de régions dans le shapefile et dans le CSV malaria, liée notamment aux accents et à certaines variantes d'écriture. Cela crée des régions sans données après jointure si aucune correction n'est effectuée. Une seconde limite est le choix de la projection UTM (EPSG :32633) : elle est adaptée au calcul de surfaces, mais doit être choisie avec prudence selon l'emprise géographique exacte. Enfin, la catégorisation en quatre classes (`pd.cut`) est utile pour une lecture rapide, mais reste une classification statistique automatique ; elle peut être remplacée par des seuils épidémiologiques si une interprétation sanitaire stricte est requise.

## 12. Conclusion

Ce TP1 a permis de réaliser un cycle complet d'analyse géospatiale : chargement et contrôle des données, calcul de surfaces en projection métrique, production de cartes et graphiques, jointure de données sanitaires, extraction de statistiques et exports multi-formats. Il constitue une base solide pour les TPs suivants, qui mobilisent les mêmes principes (cohérence CRS, agrégation spatiale, cartographie thématique), mais dans des contextes plus avancés (accessibilité, buffers, indicateurs ODD, indices spectraux).

# TP2 — Analyse spatiale des services sociaux de base au Tchad (GEE Python)

## 1. Objet du TP (objectif concret)

Ce travail pratique vise à analyser la répartition spatiale des services sociaux de base au Tchad, à partir de données vectorielles issues de sources ouvertes et structurées dans Google Earth Engine (GEE). L'objectif concret est de produire (i) des cartes interactives illustrant le réseau routier et certains espaces d'intérêt, (ii) des indicateurs par région sur la présence d'infrastructures sociales (santé, éducation, lieux de culte), et (iii) une analyse descriptive des aires protégées (WDPA) en termes de localisation et de nombre par unité administrative.

L'approche retenue repose sur l'exploitation d'assets GEE (GADM, couches OSM, WDPA) et sur des traitements de type filtrage spatial, filtrage attributaire et agrégation statistique par régions, avec export final des résultats en CSV/Excel et en cartes HTML interactives.

## 2. Données et sources mobilisées

Les limites administratives proviennent de GADM (niveaux 0 à 3), importées comme assets dans le projet Earth Engine. Les données thématiques proviennent d'OpenStreetMap (OSM) et incluent les routes (catégorisées en primaire/secondaire/tertiaire) ainsi que des points d'intérêt (*POI*) utilisés pour approximer les services sociaux (pharmacies, hôpitaux, écoles, etc.). Les aires protégées sont issues de la base WDPA, intégrée sous forme de `FeatureCollection` et utilisée pour cartographie et comptage par région.

### 3. Environnement de travail

Le traitement est réalisé dans un notebook Python en s'appuyant sur : Google Earth Engine (API Python) pour l'accès et le calcul serveur, et `geemap` pour la cartographie interactive et l'export en HTML.

Listing 8 – Connexion à Google Earth Engine et initialisation du projet

```
1 import ee  
2 import geemap  
3  
4 ee.Authenticate()  
5 ee.Initialize(project="project-2026-476714")  
6  
7 print("Earth Engine est connect :", ee.Number(1). getInfo())
```

### 4. Chargement des assets (GADM + OSM)

Un préfixe d'assets est défini afin de simplifier les appels à `ee.FeatureCollection`. Les couches GADM (Tchad) et les différentes couches OSM (bâtiments, routes, POI, lieux de culte, etc.) sont ensuite chargées.

Listing 9 – Préfixe d'assets et chargement des FeatureCollections

```
1 ASSET = "projects/project-2026-476714/assets/"  
2  
3 def fc(name):  
4     return ee.FeatureCollection(ASSET + name)  
5  
6 # Limites administratives GADM  
7 tcd0 = fc("gadm41_TCD_0")  
8 tcd1 = fc("gadm41_TCD_1")  
9 tcd2 = fc("gadm41_TCD_2")  
10 tcd3 = fc("gadm41_TCD_3")  
11  
12 # Couches principales OSM  
13 roads = fc("gis_osm_roads_free_1")  
14 pois_p = fc("gis_osm_pois_free_1")  
15 pofw_p = fc("gis_osm_pofw_free_1")
```

## 5. Cartographie du réseau routier (niveau national)

La première production cartographique consiste à représenter le réseau routier du Tchad à l'échelle nationale en distinguant les routes primaires, secondaires et tertiaires via le champ `fclass`. Une légende est ajoutée afin d'améliorer la lecture et un export HTML interactif est généré.

Listing 10 – Carte nationale des routes (primary/secondary/tertiary) + export HTML

```
1 import os
2
3 output_path = r"C:\Users\admin\Pictures\ISEP3\TP2\outputs"
4 os.makedirs(output_path, exist_ok=True)
5
6 m = geemap.Map()
7 m.add_basemap('OpenStreetMap')
8 m.center_object(tcd0, 5)
9
10 regions_style = {'color': 'blue', 'width': 2, 'fillColor': '00000000'}
11 m.addLayer(tcd1.style(**regions_style), {}, "Rgions (GADM 1)")
12
13 primary = roads.filter(ee.Filter.eq('fclass', 'primary'))
14 secondary = roads.filter(ee.Filter.eq('fclass', 'secondary'))
15 tertiary = roads.filter(ee.Filter.eq('fclass', 'tertiary'))
16
17 m.addLayer(primary.style(color='red', width=4), {}, "Routes primaires")
18 m.addLayer(secondary.style(color='orange', width=2.5), {}, "Routes
    secondaires")
19 m.addLayer(tertiary.style(color='yellow', width=1.5), {}, "Routes
    tertiaires")
20
21 html_path = os.path.join(output_path, "carte_reseau_routier_tchad.html")
22 m.save(html_path)
```

*Remarque :* le résultat est une carte HTML interactive. Pour insertion dans Overleaf, une capture PNG peut être ajoutée.

## 6. Cartographie ciblée : réseau routier d'une région (ex : Mayo-Kebbi Ouest)

Afin d'illustrer une analyse à échelle infra-nationale, une région est sélectionnée (ici *Mayo-Kebbi Ouest*) puis les routes sont filtrées sur l'emprise de cette région (`filterBounds`).

La carte est ensuite exportée au format HTML.

Listing 11 – Carte régionale des routes (Mayo-Kebbi Ouest) + export HTML

```
1 nom_region = "Mayo-Kebbi Ouest"
2 region_sel = tcd1.filter(ee.Filter.eq('NAME_1', nom_region))
3 roads_region = roads.filterBounds(region_sel.geometry())
4
5 m_reg = geemap.Map()
6 m_reg.add_basemap('OpenStreetMap')
7 m_reg.center_object(region_sel, 7)
8
9 m_reg.addLayer(region_sel.style(**regions_style), {}, f"Rgion :
{nom_region}")
10
11 primary_r = roads_region.filter(ee.Filter.eq('fclass', 'primary'))
12 secondary_r = roads_region.filter(ee.Filter.eq('fclass', 'secondary'))
13 tertiary_r = roads_region.filter(ee.Filter.eq('fclass', 'tertiary'))
14
15 m_reg.addLayer(primary_r.style(color='red', width=4), {}, "Primaires")
16 m_reg.addLayer(secondary_r.style(color='orange', width=2.5), {},
"Secondaires")
17 m_reg.addLayer(tertiary_r.style(color='yellow', width=1.5), {},
"Tertiaires")
18
19 html_path = os.path.join(output_path, f"carte_routes_{nom_region}.html")
20 m_reg.save(html_path)
```

## 7. Construction d'indicateurs régionaux : services sociaux de base

L'objectif central du TP est de produire des indicateurs par région sur la présence de services sociaux. Les services sont approximés à partir des catégories OSM disponibles dans les points d'intérêt (`pois_p`) et les lieux de culte (`pofw_p`). Une étape clé consiste à filtrer les POI par catégories grâce au champ `fclass`, puis à compter le nombre de points appartenant à chaque région.

Trois familles de services sont considérées : (i) santé (pharmacies, hôpitaux, cliniques), (ii) éducation (écoles, collèges, universités) et (iii) lieux de culte.

Listing 12 – Filtrage des POI et comptage par région (santé, écoles, culte)

```
1 health_classes = ['pharmacy', 'hospital', 'clinic', 'doctors']
```

```

2 school_classes = ['school', 'college', 'university']
3
4 health_pois = pois_p.filter(ee.Filter.inList('fclass', health_classes))
5 school_pois = pois_p.filter(ee.Filter.inList('fclass', school_classes))
6 worship_pois = pofw_p
7
8 def add_point_count(regions_fc, points_fc, field_name):
9     def _count_points(region):
10         pts_in_region = points_fc.filterBounds(region.geometry())
11         return region.set(field_name, pts_in_region.size())
12     return regions_fc.map(_count_points)
13
14 regions_stats = tcd1
15 regions_stats = add_point_count(regions_stats, health_pois, 'nb_sante')
16 regions_stats = add_point_count(regions_stats, school_pois, 'nb_ecoles')
17 regions_stats = add_point_count(regions_stats, worship_pois, 'nb_culte')

```

## 8. Export des statistiques (CSV et Excel)

Les statistiques étant calculées côté serveur Earth Engine, elles sont converties en DataFrame afin de pouvoir être sauvegardées localement. Cette conversion est acceptable ici car le nombre de régions est limité (23), donc le volume reste faible.

Listing 13 – Conversion en DataFrame et exports CSV/Excel

```

1 import pandas as pd
2 import os
3
4 regions_info = regions_stats.getInfo()
5 rows = [f['properties'] for f in regions_info['features']]
6 df = pd.DataFrame(rows)
7
8 csv_path = os.path.join(output_path, "services_sociaux_par_region.csv")
9 xlsx_path = os.path.join(output_path, "services_sociaux_par_region.xlsx")
10
11 df.to_csv(csv_path, index=False)
12 df.to_excel(xlsx_path, index=False)

```

Les résultats permettent d'identifier les régions relativement mieux dotées (ex : Ville de N'Djamena) et celles où les dénombrements sont faibles ou nuls, ce qui peut correspondre à une absence réelle de services ou à une sous-déclaration OSM.

## 9. Analyse des aires protégées (WDPA) : cartographie et comptage

Une couche WDPA intégrée comme asset est cartographiée et superposée aux régions. Une seconde analyse consiste à compter, pour chaque région, le nombre d'aires protégées intersectant ou se trouvant à l'intérieur de son emprise (via `filterBounds`).

Listing 14 – Cartographie WDPA + export HTML

```
1 wdpa =
2     ee.FeatureCollection("projects/project-2026-476714/assets/WDPA_TCD_polygons_merged")
3
4 regions = ee.FeatureCollection("FAO/GAUL_SIMPLIFIED_500m/2015/level1") \
5     .filter(ee.Filter.eq("ADM0_NAME", "Chad"))
6
7 m = geemap.Map(center=[15, 19], zoom=5)
8
9 style_regions = {"color": "1f4e79", "fillColor": "00000000", "width": 1.5}
10 style_wdpa = {"color": "b30000", "fillColor": "ff000050", "width": 1}
11
12 m.addLayer(regions.style(**style_regions), {}, "Rgions")
13 m.addLayer(wdpa.style(**style_wdpa), {}, "Aires protgées (WDPA)")
14
15 html_path = os.path.join(output_path, "carte_aires_protegees_tchad.html")
16 m.save(html_path)
```

Listing 15 – Comptage WDPA par région + export CSV/Excel

```
1 def add_count(feat):
2     geom = feat.geometry()
3     count = wdpa.filterBounds(geom).size()
4     return feat.set("count", count)
5
6 wdpa_by_region = regions.map(add_count)
7 df_regions = geemap.ee_to_df(wdpa_by_region)
8 df_regions = df_regions[["ADM1_NAME", "count"]].sort_values("count",
9     ascending=False)
10
11 csv_path = os.path.join(output_path, "wdpa_par_region.csv")
12 xlsx_path = os.path.join(output_path, "wdpa_par_region.xlsx")
13
14 df_regions.to_csv(csv_path, index=False)
15 df_regions.to_excel(xlsx_path, index=False)
```

## 10. Résultats et interprétation

Ce TP produit deux familles de résultats. Premièrement, des productions cartographiques interactives qui permettent de visualiser la structure du réseau routier et sa distribution spatiale, notamment la hiérarchie des routes primaires/secondaires/tertiaries au niveau national et dans une région ciblée. Deuxièmement, des indicateurs quantitatifs par région sur le nombre de services sociaux (santé, éducation, culte). Ces indicateurs permettent de comparer l'intensité de l'offre de services selon l'espace, même si leur interprétation doit tenir compte de la qualité de complétude d'OpenStreetMap.

Concernant la WDPA, le comptage des polygones protégés par région fournit un premier diagnostic sur la répartition des espaces conservés et sur la concentration potentielle des aires protégées dans certaines zones. Cependant, le résultat dépend de la définition de l'intersection (ici, `filterBounds` : l'aire est comptée si elle tombe dans l'emprise géométrique régionale).

## 11. Limites et difficultés rencontrées

La principale limite est la dépendance aux attributs OSM, notamment le champ `fclass`. Les catégories sélectionnées (pharmacy, hospital, clinic, school, etc.) doivent être validées par inspection car certaines infrastructures peuvent être codées différemment selon les contributeurs. Une deuxième difficulté est le passage serveur→client : la récupération de `getInfo()` doit rester limitée à de petites collections. Enfin, la sortie en HTML interactive est très utile pour l'exploration, mais nécessite une conversion en image (capture PNG) pour un rapport LaTeX.

## 12. Conclusion

Ce TP2 met en œuvre une chaîne complète d'analyse spatiale sous Google Earth Engine via Python : chargement de données (GADM/OSM/WDPA), production de cartes interactives, filtrage spatial, calcul d'indicateurs par région et export de résultats tabulaires. Il constitue une base solide pour des analyses plus avancées d'accèsibilité (buffers, distances), car il fournit à la fois la localisation des services et un premier diagnostic quantitatif de leur distribution.

# TP3 — Analyse spatiale de la population et accessibilité aux services sociaux (Tchad)

## 1. Objet du TP (objectif concret)

Ce troisième travail pratique constitue une extension directe du TP2. Alors que le TP2 établissait une première cartographie et des indicateurs de présence des services sociaux (OSM) par région, le TP3 introduit explicitement la dimension **démographique** à partir des données raster **WorldPop**. L'objectif concret est d'évaluer l'accessibilité potentielle de la population aux services sociaux de base (santé, éducation, lieux de culte) en estimant la **population desservie** dans des zones tampons (*buffers*) autour de ces services.

Plus précisément, le TP vise (i) à cartographier la population à deux résolutions (100 m et 1 km), (ii) à construire des buffers de 1 km, 5 km et 10 km autour de différents services, (iii) à calculer la population totale contenue dans chaque buffer, et (iv) à analyser la population présente à l'intérieur des aires protégées (WDPA), afin d'illustrer des enjeux d'aménagement et de conservation.

## 2. Données et sources mobilisées

Trois familles de données sont mobilisées :

- **Limites administratives** : GADM (Tchad) chargées comme assets (`tcd0`, `tcd1`).
- **Population** : rasters WorldPop 2025 à deux résolutions (100 m et 1 km), chargés comme assets Earth Engine.
- **Services sociaux et points d'intérêt** : données OpenStreetMap (POI et lieux de culte) déjà utilisées au TP2, filtrées par catégories (`fclass`).
- **Aires protégées** : WDPA (polygones) en asset Earth Engine.

### 3. Environnement de travail et connexion à GEE

Le traitement est réalisé via l'API Python de Google Earth Engine, avec `geemap` pour la cartographie interactive et la conversion des sorties en tables.

Listing 16 – Connexion à Earth Engine (API Python)

```
1 import ee
2 import geemap
3
4 ee.Authenticate()
5 ee.Initialize(project="project-2026-476714")
```

### 4. Chargement des assets (administratif, OSM, WorldPop)

Les limites administratives et les couches OSM proviennent du projet GEE (assets), tandis que la population WorldPop est chargée depuis des rasters importés au préalable.

Listing 17 – Chargement GADM (Tchad) + POI/culte (OSM) + WorldPop

```
1 ASSET = "projects/project-2026-476714/assets/"
2 def fc(name):
3     return ee.FeatureCollection(ASSET + name)
4
5 # GADM
6 tcd0 = fc("gadm41_TCD_0")
7 tcd1 = fc("gadm41_TCD_1")
8
9 # OSM (points)
10 pois_p = fc("gis_osm_pois_free_1")
11 pofw_p = fc("gis_osm_pofw_free_1")
12
13 # WorldPop (rasters imports en assets)
14 pop100 =
15     ee.Image("projects/project-2026-476714/assets/tcd_pop_2025_CN_100m_R2025A_v1")
15 pop1km =
16     ee.Image("projects/project-2026-476714/assets/tcd_pop_2025_CN_1km_R2025A_UA_v1")
```

## 5. Cartographie de la population (100 m et 1 km)

La population est d'abord **clippée** à l'emprise du Tchad afin de limiter les calculs et éviter des traitements inutiles hors zone d'étude. Deux cartes interactives sont ensuite produites (une pour le raster 100 m, une pour le raster 1 km), avec superposition des limites régionales.

Listing 18 – Clip du raster population + visualisation (exemple 100 m)

```
1 pop100_tchad = pop100.clip(tcd0)
2
3 # Nom de bande (rcupration automatique)
4 band_pop = pop100_tchad.bandNames().get(0)
5
6 m = geemap.Map()
7 m.add_basemap("OpenStreetMap")
8 m.center_object(tcd0, 5)
9
10 vis_pop = {"min": 0, "max": 500, "palette":
11     ["blue", "green", "yellow", "orange", "red"]}
12 m.addLayer(pop100_tchad, vis_pop, "Population 100 m")
13 regions_style = {"color": "blue", "width": 2, "fillColor": "00000000"}
14 m.addLayer(tcd1.style(**regions_style), {}, "Rgions (GADM 1)")
15
16 # Export HTML
17 m.to_html("map_population100.html")
```

*Insertion figure (optionnelle, si capture PNG disponible).*

## 6. Sélection des services sociaux (OSM) : santé, écoles, culte

Les services sociaux sont définis à partir des catégories du champ **fclass**. Les POI sont filtrés pour construire des collections de points santé et écoles, tandis que les lieux de culte proviennent directement de **pofw\_p**.

Listing 19 – Filtrage OSM : santé, écoles, culte

```
1 health_classes = ["pharmacy", "hospital", "clinic", "doctors"]
2 school_classes = ["school", "college", "university"]
3
4 sante = pois_p.filter(ee.Filter.inList("fclass", health_classes))
```

```

5  ecoles = pois_p.filter(ee.Filter.inList("fclass", school_classes))
6  culte = pofw_p

```

## 7. Buffers et indicateur d'accessibilité : population des-servie

L'idée centrale du TP est de mesurer, autour de chaque service, la population contenue dans des buffers de 1 km, 5 km et 10 km. Concrètement, pour chaque point service, on construit trois zones tampons, puis on applique un `reduceRegion` avec un réducteur `sum` pour agréger la population (raster 100 m) à l'intérieur de chaque buffer.

Ce calcul fournit un indicateur de **population potentiellement desservie** à différentes distances (proximité immédiate, accessibilité moyenne, accessibilité plus large). L'échelle (`scale=100`) est cohérente avec la résolution du raster 100 m.

Listing 20 – Fonction générique : population dans buffers (1/5/10 km) autour d'un point

```

1  pop100_tchad = pop100.clip(tcd0)
2  band_pop = pop100_tchad.bandNames().get(0)
3
4  def pop_in_buffer(feature):
5      geom = feature.geometry()
6
7      buf1 = geom.buffer(1000)
8      buf5 = geom.buffer(5000)
9      buf10 = geom.buffer(10000)
10
11     pop1 = pop100_tchad.reduceRegion(
12         reducer=ee.Reducer.sum(),
13         geometry=buf1,
14         scale=100,
15         maxPixels=1e13
16     ).get(band_pop)
17
18     pop5 = pop100_tchad.reduceRegion(
19         reducer=ee.Reducer.sum(),
20         geometry=buf5,
21         scale=100,
22         maxPixels=1e13
23     ).get(band_pop)
24
25     pop10 = pop100_tchad.reduceRegion(

```

```

26     reducer=ee.Reducer.sum(),
27     geometry=buf10,
28     scale=100,
29     maxPixels=1e13
30   ).get(band_pop)
31
32   return feature.set({
33     "pop_1km": pop1,
34     "pop_5km": pop5,
35     "pop_10km": pop10
36   })

```

## 7.1 Application aux services santé et export

Listing 21 – Application aux services santé + export CSV/Excel

```

1 import pandas as pd
2 import os
3
4 sante_pop = sante.map(pop_in_buffer)
5
6 info = sante_pop.getInfo()
7 rows = [f["properties"] for f in info["features"]]
8 df_sante = pd.DataFrame(rows)
9
10 output_path = "outputs"
11 os.makedirs(output_path, exist_ok=True)
12
13 df_sante.to_csv(os.path.join(output_path, "pop_buffer_sante.csv"),
14                 index=False)
14 df_sante.to_excel(os.path.join(output_path, "pop_buffer_sante.xlsx"),
15                   index=False)

```

## 7.2 Application aux écoles et lieux de culte

Le même principe est appliqué aux écoles et aux lieux de culte, ce qui conduit à produire des fichiers similaires : `pop_buffer_ecoles.csv/xlsx` et `pop_buffer_culte.csv/xlsx`. Cette homogénéité est importante : elle garantit que les indicateurs sont comparables entre services.

## 8. Cartes interactives des buffers (visualisation)

Les buffers sont visualisés sous forme de cartes HTML interactives (ex. `buffer_sante.html`, `buffer_ecoles.html`, `buffer_culte.html`). Ces cartes permettent de vérifier visuellement que les zones tampons sont correctement construites et qu'elles enveloppent bien les points de service.

## 9. Population à l'intérieur des aires protégées (WDPA)

La dernière partie du TP consiste à estimer la population vivant à l'intérieur des aires protégées. L'idée est simple : pour chaque polygone WDPA, on calcule la somme des valeurs de population (raster 100 m) à l'intérieur de la géométrie de l'aire protégée. Le résultat est stocké comme un nouvel attribut (`pop_in`) dans la `FeatureCollection` WDPA.

Listing 22 – Population dans les polygones WDPA (attribut `pop_in`)

```
1 wdpa =
2     ee.FeatureCollection("projects/project-2026-476714/assets/WDPA_TCD_polygons_merged")
3
4     def add_pop_in_wdpa(feat):
5         geom = feat.geometry()
6         pop_total = pop100_tchad.reduceRegion(
7             reducer=ee.Reducer.sum(),
8             geometry=geom,
9             scale=100,
10            maxPixels=1e13
11        ).get('band_pop')
12
13        return feat.set("pop_in", pop_total)
14
15 wdpa_pop = wdpa.map(add_pop_in_wdpa)
```

### 9.1 Export des résultats WDPA

Les résultats sont exportés sous forme de table (CSV/Excel) afin de disposer, pour chaque aire protégée, d'informations de base telles que le nom, le statut, la catégorie IUCN, et la population estimée à l'intérieur.

Listing 23 – Export `pop_protected_areas` (CSV/Excel)

```
1 import pandas as pd
2 import os
3
```

```

4 info_wdpa = wdpa_pop.getInfo()
5 rows_wdpa = [f["properties"] for f in info_wdpa["features"]]
6 df_wdpa = pd.DataFrame(rows_wdpa)
7
8 output_path = "outputs"
9 os.makedirs(output_path, exist_ok=True)
10
11 df_wdpa.to_csv(os.path.join(output_path, "pop_protected_areas.csv"),
12                 index=False)
12 df_wdpa.to_excel(os.path.join(output_path, "pop_protected_areas.xlsx"),
13                 index=False)

```

## 10. Résultats et lecture analytique

Les résultats obtenus répondent à une logique d'**accessibilité potentielle**. Pour chaque type de service, le TP fournit un tableau contenant, pour chaque point, trois mesures : la population dans le buffer 1 km, 5 km et 10 km. Une valeur élevée à 1 km suggère une bonne couverture de proximité, tandis qu'une valeur faible même à 10 km indique une desserte faible ou une zone faiblement peuplée.

La cartographie WorldPop à deux résolutions permet en parallèle d'interpréter les résultats à différentes échelles : le raster 100 m met en évidence des contrastes fins (taches de forte densité), alors que le raster 1 km facilite la lecture macro-spatiale. Enfin, l'analyse WDPA enrichit l'étude en introduisant une dimension environnementale : elle montre qu'une part non nulle de population peut se trouver à l'intérieur (ou en bordure immédiate) de zones protégées, ce qui peut poser des enjeux de gestion, de conservation et de services de base.

## 11. Limites et difficultés

Trois limites principales doivent être mentionnées. Premièrement, les services OSM ne représentent pas nécessairement une exhaustivité totale : une région peut apparaître sous-dotée simplement par manque de saisie OSM. Deuxièmement, la méthode `reduceRegion(sum)` suppose que les valeurs WorldPop sont correctement calibrées, et que l'agrégation à l'intérieur de buffers fournit une approximation raisonnable de la population desservie. Troisièmement, la construction de buffers en mètres suppose implicitement un repère métrique ; Earth Engine gère cela via géodésie, mais l'interprétation doit rester prudente, notamment sur de grandes distances ou dans des zones aux géométries complexes.

## 12. Conclusion

Ce TP3 introduit une composante essentielle de l'analyse spatiale appliquée : la confrontation d'une distribution démographique (WorldPop) à l'implantation spatiale des services sociaux (OSM), via des buffers multi-distances et des statistiques zonales. Il fournit des indicateurs concrets de couverture potentielle et prépare naturellement les analyses plus avancées (accessibilité par réseau, distances routières, optimisation d'implantation, etc.), tout en ouvrant une perspective environnementale grâce à l'évaluation de la population présente dans les aires protégées.

# TP4 — Cartographie des terres arables et statistiques spatiales (Burundi)

## 1. Objet du TP (objectif concret)

Ce TP4 a pour objectif de produire une **carte opérationnelle des terres arables** au Burundi à partir d'une approche par masques binaires combinant plusieurs sources d'information. Concrètement, l'idée est de définir une **base de terres potentiellement arables** (terres cultivées + forêts récemment déboisées), puis de retirer systématiquement les zones où l'agriculture est jugée non pertinente ou incompatible (zones d'eau permanentes, pentes trop élevées, zones protégées WDPA, surfaces fortement imperméabilisées).

Une fois la carte finale obtenue, le TP calcule des **statistiques spatiales** par niveaux administratifs (provinces et communes) : superficie totale, superficie arable et ratio en pourcentage. Le script propose aussi une composante **interactive** (panneaux d'information et clic sur la carte) afin d'explorer localement les résultats, ainsi que des exports CSV vers Google Drive pour exploitation hors GEE.

## 2. Données utilisées et logique des couches

Le TP mobilise des données raster et vectorielles, toutes traitées dans Google Earth Engine :

- **Limites administratives (assets)** : Burundi (niveau 0, 1 et 2) importés depuis GADM.
- **Terres cultivées (GFSAD)** : masque binaire des zones cultivées.
- **Déboisement (Hansen Global Forest Change)** : pertes forestières entre 2000 et 2015 (`lossyear` 1 à 15).
- **Zones protégées (WDPA)** : polygones WDPA convertis en masque binaire.
- **Eaux permanentes (JRC Global Surface Water)** : occurrence > 75%.
- **Pente (SRTM)** : pentes > 15° considérées comme contrainte majeure.

- **Surfaces imperméables (GMIS)** : masque à partir d'un seuil d'imperméabilisation ( $> 10\%$ ).

L'approche repose sur une logique simple et explicite : **définir** une base puis **exclure** ce qui ne doit pas être compté.

### 3. Définition de la zone d'étude et couches administratives

La zone d'intérêt (*AOI*) est le Burundi (niveau 0). Les provinces et communes sont ensuite chargées, **clippées** à l'emprise nationale et stylisées (contours) afin de servir de référence de lecture cartographique.

Listing 24 – Définition AOI (Burundi) et affichage des limites administratives

```

1 var EE_GAUL_LEVEL0 =
  'projects/travaux-pratique-478314/assets/gadm41_BDI_0';
2 var EE_GAUL_LEVEL1 =
  'projects/travaux-pratique-478314/assets/gadm41_BDI_1';
3 var EE_GAUL_LEVEL2 =
  'projects/travaux-pratique-478314/assets/gadm41_BDI_2';
4
5 var AOI_FC = ee.FeatureCollection(EE_GAUL_LEVEL0);
6 Map.centerObject(AOI_FC, 8);
7 var AOI_GEOM = AOI_FC.geometry();
8
9 var clip_and_style = function(feature_collection, color, width, name,
  visibility) {
10   var clipped_fc = feature_collection
11     .filterBounds(AOI_FC)
12     .map(function(f) {
13       return ee.Feature(
14         f.geometry().intersection(AOI_GEOM, ee.ErrorMargin(100)),
15         f.toDictionary()
16       );
17     });
18
19   Map.addLayer(
20     clipped_fc.style({color: color, width: width, fillColor: '00000000'}),
21     {},
22     name,
23     visibility
24   );

```

```

25     return clipped_fc;
26 }
27
28 Map.addLayer(AOI_FC.style({color: '000000', width: 3, fillColor:
  '00000000'}), {}, 'Limites Nationales');
29 var provinces = clip_and_style(ee.FeatureCollection(EE_GAUL_LEVEL1),
  '555555', 2, 'Limites Provinces', true);
30 var communes = clip_and_style(ee.FeatureCollection(EE_GAUL_LEVEL2),
  'AAAAAA', 1, 'Limites Communes', false);

```

## 4. Construction des masques thématiques

Chaque couche d'analyse est transformée en **masque binaire** (0/1). Cette standardisation est cruciale : elle permet ensuite de combiner les informations de manière logique (**or**, **and**, **not**) et de maîtriser exactement les pixels conservés ou exclus.

### 4.1 Terres cultivées (GFSAD)

La couche GFSAD est utilisée pour extraire les pixels dont la classe correspond aux terres cultivées. Le résultat est un masque booléen.

Listing 25 – Masque des terres cultivées (GFSAD)

```

1 var GFSAD_ASSET_ID =
  'projects/travaux-pratique-478314/assets/GFSAD_Burundi_2015';
2 var gfsad_image = ee.Image(GFSAD_ASSET_ID);
3 var cropland_mask = gfsad_image.select(0).eq(2).clip(AOI_FC);
4 Map.addLayer(cropland_mask.updateMask(cropland_mask), {palette:
  ['FF00FF']}, 'Terres Cultivees (GFSAD)', false);

```

### 4.2 Forêts déboisées (Hansen 2000–2015)

Le déboisement est mesuré via `lossyear` dans Hansen Global Forest Change. Les pertes entre 2000 et 2015 (valeurs 1 à 15) sont retenues.

Listing 26 – Masque des forêts déboisées (Hansen LossYear 1–15)

```

1 var gfc = ee.Image('UMD/hansen/global_forest_change_2015_v1_3');
2 var lossYear = gfc.select(['lossyear']);
3 var cleared_forest_mask =
  lossYear.gte(1).and(lossYear.lte(15)).gt(0).clip(AOI_FC);
4 Map.addLayer(cleared_forest_mask.updateMask(cleared_forest_mask),
  {palette: ['FF8C00']}, 'Forets Deboisees (Hansen)', false);

```

## 4.3 Zones protégées (WDPA)

Les zones WDPA sont converties en masque raster via `reduceToImage` : un pixel vaut 1 s'il appartient à un polygone protégé.

Listing 27 – Masque des zones protégées (WDPA)

```
1 var wdpa_polygons =  
    ee.FeatureCollection('WCMC/WDPA/current/polygons').filterBounds(AOI_FC);  
2 var reserved_mask = wdpa_polygons.reduceToImage({  
3   properties: ['WDPAID'],  
4   reducer: ee.Reducer.count()  
5 }).unmask(0).gt(0).clip(AOI_FC);  
6  
7 Map.addLayer(reserved_mask.updateMask(reserved_mask), {palette:  
  ['0000FF']}, 'Zones Protegees (WDPA)', false);
```

## 4.4 Eaux permanentes (JRC Global Surface Water)

Un pixel est considéré comme eau permanente si l'occurrence dépasse 75%.

Listing 28 – Masque des eaux permanentes (JRC GSW, occurrence > 75%)

```
1 var gsw = ee.Image('JRC/GSW1_4/GlobalSurfaceWater');  
2 var water_mask = gsw.select('occurrence').gt(75).clip(AOI_FC);  
3 Map.addLayer(water_mask.updateMask(water_mask), {palette: ['00FFFF']},  
  'Eaux Permanentes (JRC)', false);
```

## 4.5 Pentes fortes (SRTM > 15°)

On calcule la pente à partir du MNT SRTM puis on retient les pentes supérieures à 15°.

Listing 29 – Masque des pentes > 15° (SRTM)

```
1 var srtm = ee.Image('USGS/SRTMGL1_003');  
2 var slope = ee.Terrain.slope(srtm);  
3 var slope_mask = slope.gt(15).clip(AOI_FC);  
4 Map.addLayer(slope_mask.updateMask(slope_mask), {palette: ['FF0000']},  
  'Pentes > 15 deg', false);
```

## 4.6 Surfaces imperméables (GMIS > 10%) et gestion des NoData

La couche GMIS contient des valeurs particulières : 255 correspond à *NoData* et 200 à *Non-HBASE*. Ces valeurs sont traitées avant application du seuil afin d'éviter des erreurs

d'interprétation.

Listing 30 – Masque surfaces imperméables (GMIS) avec traitement NoData

```
1 var GMIS_ASSET_ID = 'projects/travaux-pratique-478314/assets/GMIS_Burundi';
2 var gmis_image = ee.Image(GMIS_ASSET_ID);
3 var percent_impervious = gmis_image.select(0);
4
5 // 1) Masquer NoData (=255)
6 var gmis_valid =
    percent_impervious.updateMask(percent_impervious.neq(255));
7
8 // 2) Remplacer 200 (Non-HBASE) par 0
9 var gmis_clean = gmis_valid.where(gmis_valid.eq(200), 0);
10
11 // 3) Seuil > 10%
12 var impervious_mask = gmis_clean.gt(10).clip(AOI_FC);
13
14 Map.addLayer(impervious_mask.updateMask(impervious_mask), {palette:
    ['808080']}, 'Surfaces Impermeables (GMIS)', false);
```

## 5. Calcul final des terres arables (base + exclusion)

La construction finale se fait en 4 étapes logiques :

1. **Base arable** : cultures **OU** forêts déboisées.
2. **Exclusion totale** : eau **OU** pente **OU** WDPA **OU** imperméable.
3. **Masque d'application** : inverse de l'exclusion (zones à conserver).
4. **Terres arables finales** : base  $\cap$  zones conservées (via updateMask).

Listing 31 – Terres arables : base, exclusion, masque d'application et résultat final

```
1 var terres_arables_base =
    cropland_mask.unmask(0).or(cleared_forest_mask.unmask(0));
2
3 var zone_exclusion_totale = water_mask.unmask(0)
4     .or(slope_mask.unmask(0))
5     .or(reserved_mask.unmask(0))
6     .or(impervious_mask.unmask(0));
7
8 var masque_application = zone_exclusion_totale.not();
9
```

```

10 var terres_arables_finales =
    terres_arables_base.updateMask(masque_application);
11
12 Map.addLayer(terres_arables_base.updateMask(terres_arables_base),
    {palette: ['00FF00']},
13 'Terres Arables de Base (Cultures + Déboisement)', false);
14
15 Map.addLayer(zone_exclusion_totale.updateMask(zone_exclusion_totale),
    {palette: ['000000']},
16 'Zone Exclusion Totale (NOIR)', false);
17
18 Map.addLayer(terres_arables_finales.updateMask(terres_arables_finales),
    {palette: ['006400']},
19 'Terres Arables Finales', true);

```

## 6. Statistiques spatiales par commune et province

L'étape suivante consiste à quantifier l'information : pour chaque commune/province, on calcule :

- la superficie totale ( $\text{km}^2$ ),
- la superficie arable ( $\text{km}^2$ ),
- le ratio arable (en %).

Le calcul repose sur une multiplication du masque final par l'aire des pixels (`pixelArea`), puis agrégation par somme via `reduceRegion`.

Listing 32 – Fonction générique de calcul des statistiques arables

```

1 var calculate_stats = function(feature, area_name_prop) {
2     var geom = feature.geometry();
3     var area_total = geom.area().divide(1e6); // km2
4
5     var arable_area = terres_arables_finales.multiply(ee.Image.pixelArea())
6         .reduceRegion({
7             reducer: ee.Reducer.sum(),
8             geometry: geom,
9             scale: 30,
10            maxPixels: 1e10,
11            bestEffort: true
12        }).values().get(0);
13
14     var arable_km2 = ee.Number(arable_area).divide(1e6);

```

```

15     var ratio = arable_km2.divide(area_total).multiply(100);
16
17     return feature.set({
18       'NOM': feature.get(area_name_prop),
19       'area_total_km2': area_total,
20       'arable_km2': arable_km2,
21       'ratio_percent': ratio
22     });
23   };

```

Les statistiques sont appliquées aux communes (niveau 2) et aux provinces (niveau 1). Le script ajoute aussi des champs descriptifs (Type, NOM\_PROVINCE) pour faciliter l'exploitation.

Listing 33 – Application : communes et provinces

```

1  var communes_with_stats = communes.map(function(f) {
2    return calculate_stats(f, 'NAME_2').set({
3      'Type': 'Commune',
4      'NOM_PROVINCE': f.get('NAME_1')
5    });
6  });
7
8  var provinces_with_stats = provinces.map(function(f) {
9    return calculate_stats(f, 'NAME_1').set({
10      'Type': 'Province',
11      'NOM_PROVINCE': f.get('NAME_1')
12    });
13  });

```

## 7. Classements (Top 5) via graphiques (console)

Le script produit des graphiques (bar chart) directement dans la console GEE pour classer les unités administratives selon : (i) superficie arable, (ii) ratio arable, (iii) superficie totale. Ces graphiques sont utiles pour identifier rapidement les zones dominantes en disponibilité de terres arables.

*Insertion figure (optionnelle, capture des charts console).*

## 8. Exports vers Google Drive

Pour permettre une exploitation hors Earth Engine, les tables statistiques (communes et provinces) sont exportées au format CSV dans un dossier Drive dédié. Le script contrôle

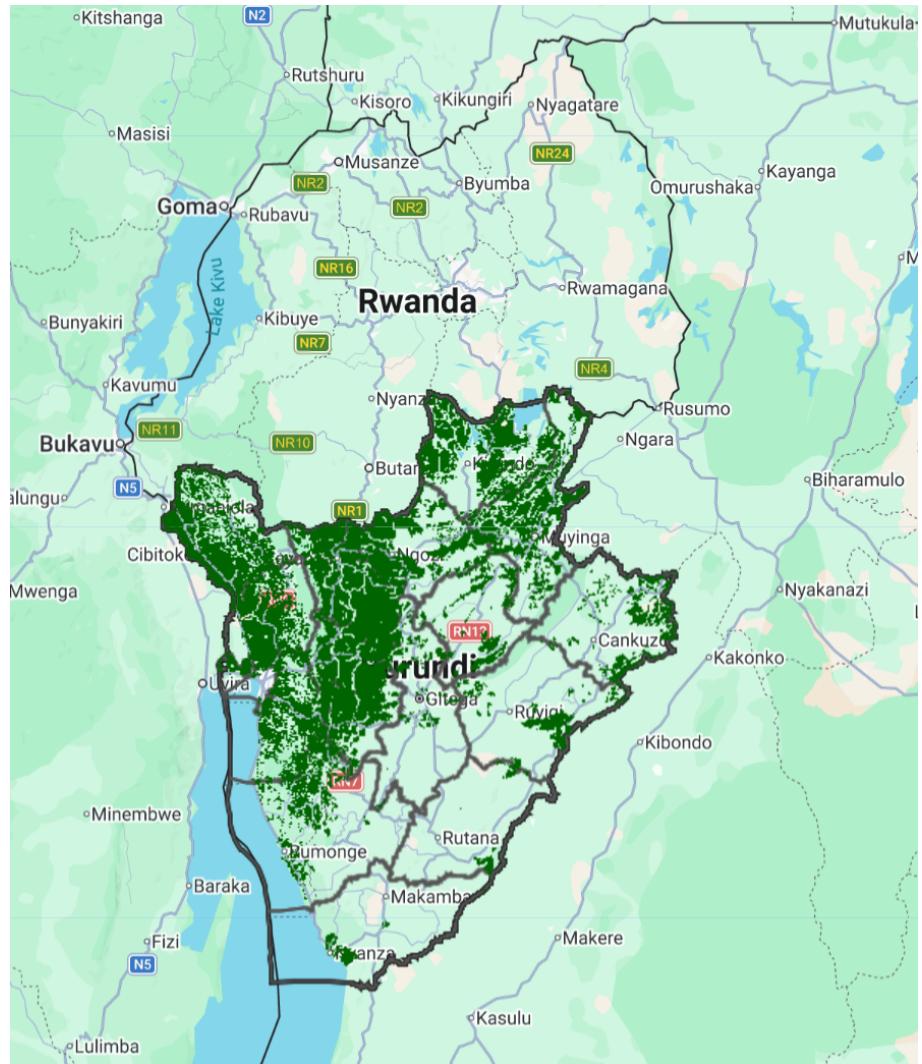


FIGURE 3 – Exemple de classement (Top 5) des provinces selon la superficie arable.

explicitement les colonnes exportées.

#### Listing 34 – Exports CSV Drive : communes et provinces

```

1
2 var EXPORT_FOLDER = 'TP4_SES_Groupe_1_outputs';
3 var stats_columns_global = [
4   'NOM',
5   'NOM_PROVINCE',
6   'Type',
7   'area_total_km2',
8   'arable_km2',
9   'ratio_percent'
10 ];
11
12 Export.table.toDrive({
13   collection: communes_with_stats.select(stats_columns_global),

```

```

14     description: 'Stats_Communes',
15     folder: EXPORT_FOLDER,
16     fileNamePrefix: 'Stats_Communes',
17     fileFormat: 'CSV'
18   });
19
20 Export.table.toDrive({
21   collection: provinces_with_stats.select(stats_columns_global),
22   description: 'Stats_Provinces',
23   folder: EXPORT_FOLDER,
24   fileNamePrefix: 'Stats_Provinces',
25   fileFormat: 'CSV'
26 });

```

## 9. Interface utilisateur (UI) : statistiques nationales et clic interactif

Une originalité importante du TP est la mise en place d'une interface interactive :

- un panneau affichant des **statistiques nationales** (superficie du pays, superficie arable totale, ratio),
- un panneau d'**informations locales** qui se met à jour lorsqu'on clique sur la carte, en affichant les statistiques de la commune et de la province correspondant au point sélectionné.

Cette partie améliore fortement l'exploration des résultats, car elle permet de passer de la carte à des valeurs numériques précises sans quitter l'environnement GEE.

## 10. Conclusion

Ce TP4 met en œuvre une méthode transparente et reproductible de délimitation des terres arables au Burundi, fondée sur la combinaison d'informations multi-sources et l'application de contraintes d'exclusion. Le résultat final est une carte des zones arables, enrichie par des statistiques par communes et provinces, des classements analytiques et une interface interactive facilitant l'interprétation. Les exports CSV permettent enfin de prolonger l'analyse dans des outils externes (Excel, R, Python) pour produire des indicateurs complémentaires ou des modèles plus avancés.

# TP5 — ODD 11.3.1 : Ratio consommation des terres / croissance démographique (RDC, 2017–2020)

## 1. Objectif du TP (ce qu'on veut calculer)

L'objectif de ce TP est de calculer l'indicateur des Nations Unies **ODD 11.3.1**, appelé **LCRPGR** (*Land Consumption Rate to Population Growth Rate*). Il mesure si l'expansion urbaine consomme des terres plus vite (ou moins vite) que la population urbaine ne croît.

Dans ce TP, la RDC est analysée sur la période **2017–2020** (soit  $y = 3$  ans), en appliquant une classification urbaine **DEGURBA** dite « stricte », fondée sur la densité de population, la continuité spatiale des cellules, et la population totale des grappes.

## 2. Rappel méthodologique (DEGURBA stricte)

### 2.1 Classification des cellules (principe)

La méthodologie DEGURBA repose sur 3 classes conceptuelles :

- **Centre urbain** : cellules à très forte densité ( $\geq 1500 \text{ hab/km}^2$ ) formant des grappes dont la population totale est  $\geq 50\,000$ .
- **Groupe urbaine** : cellules d'une densité plus modérée (souvent  $\geq 300 \text{ hab/km}^2$ ) formant des grappes dont la population totale est  $\geq 5\,000$ .
- **Rural** : tout le reste.

Dans ce TP, l'indicateur ODD 11.3.1 est calculé à partir des **centres urbains uniquement** (grappes  $\geq 50\,000$  hab), conformément à l'orientation de « DEGURBA stricte » annoncée dans le script.

### 2.2 Formule ODD 11.3.1

On note :

- $Urb_{t_0}$  : surface urbaine à l'année initiale (2017)

- $Urb_{t1}$  : surface urbaine à l'année finale (2020)
- $Pop_{t0}$  : population urbaine à 2017
- $Pop_{t1}$  : population urbaine à 2020
- $y = 2020 - 2017 = 3$  ans

Les taux sont :

$$LCR = \left( \frac{\ln(Urb_{t1}/Urb_{t0})}{y} \right) \times 100 \quad \text{et} \quad PGR = \left( \frac{\ln(Pop_{t1}/Pop_{t0})}{y} \right) \times 100$$

et l'indicateur final :

$$LCRPGR = \frac{LCR}{PGR}$$

### 2.3 Interprétation

- $LCRPGR < 1$  : densification (croissance plus efficace)
- $LCRPGR \approx 1$  : croissance proportionnelle (densité stable)
- $LCRPGR > 1$  : étalement urbain (consommation foncière plus rapide que la croissance de population)

# Chapitre 1

## Préparation des données

### 1.1 Packages et structure de dossiers

Le script commence par charger les packages nécessaires aux traitements raster/vecteur (`terra`, `sf`), à la manipulation (`tidyverse`), aux graphiques (`ggplot2`) et aux exports. Ensuite, il crée une arborescence claire : `data/` pour les entrées, `output/` pour les résultats et `figures/` pour les cartes.

Listing 1.1 – Chargement des packages et création des dossiers

```
1 packages_requis <-  
2   c("sf", "terra", "tidyverse", "exactextractr", "ggplot2", "gridExtra", "viridis", "scales")  
3   for (pkg in packages_requis) {  
4     if (!require(pkg, character.only = TRUE, quietly = TRUE)) {  
5       install.packages(pkg)  
6       library(pkg, character.only = TRUE)  
7     }  
8   }  
9  
9 dossiers <- c("data/population", "data/lulc", "data/boundaries", "data/temp",  
10      "output", "output/rasters", "output/degurba",  
11      "figures", "figures/cartes", "figures/graphiques")  
12 for (dossier in dossiers) dir.create(dossier, showWarnings = FALSE,  
  recursive = TRUE)
```

### 1.2 Limites administratives RDC

Les limites nationales et provinciales sont chargées depuis des shapefiles. Le script inspecte ensuite l'emprise (bbox), le CRS et la surface totale afin de valider la cohérence géographique.

Listing 1.2 – Lecture des shapefiles RDC (pays et provinces)

```
1 rdc_pays <- st_read("data/boundaries/rdc_country.shp", quiet = TRUE)
2 rdc_provinces <- st_read("data/boundaries/rdc_provinces.shp", quiet = TRUE)
3 bbox_rdc <- st_bbox(rdc_pays)
```

## 1.3 Population WorldPop (2017 et 2020)

Les rasters de population (résolution  $\sim 1$  km) sont chargés. Si les fichiers n'existent pas, le script génère des données simulées (démonstration) : base Poisson + croissance globale + pics urbains (Kinshasa, Lubumbashi, Mbuji-Mayi, Kisangani), puis masque par la frontière.

Listing 1.3 – Chargement (ou simulation) des populations 2017 et 2020

```
1 pop_2017 <- rast("data/population/rdc_pop_2017_1km.tif")
2 pop_2020 <- rast("data/population/rdc_pop_2020_1km.tif")
3
4 stats_2017 <- global(pop_2017, fun=c("sum", "mean", "min", "max", "sd"),
  na.rm=TRUE)
5 stats_2020 <- global(pop_2020, fun=c("sum", "mean", "min", "max", "sd"),
  na.rm=TRUE)
```

## 1.4 LULC : mosaïque des 9 tuiles puis ré-échantillonnage 1 km

La RDC étant couverte par plusieurs tuiles LULC, une fonction charge toutes les tuiles d'une année, vérifie le CRS, mosaïque, découpe sur la RDC, puis masque.

Ensuite, point méthodologique clé : le LULC haute résolution est **ré-échantillonné** sur la grille 1 km de la population (`resample(..., method="near")`). Ceci garantit que toutes les étapes DEGURBA opèrent au même niveau spatial.

Listing 1.4 – Ré-échantillonnage du LULC à 1 km (même grille que WorldPop)

```
1 lulc_2017_resample <- resample(lulc_2017, pop_2017, method = "near")
2 lulc_2020_resample <- resample(lulc_2020, pop_2020, method = "near")
3
4 lulc_2017 <- lulc_2017_resample
5 lulc_2020 <- lulc_2020_resample
6
7 writeRaster(lulc_2017, "output/rasters/lulc_2017_1km.tif", overwrite =
  TRUE)
```

```
8 writeRaster(lulc_2020, "output/rasters/lulc_2020_1km.tif", overwrite =  
TRUE)
```

# Chapitre 2

## Classification DEGURBA (zones urbaines)

### 2.1 Calcul de densité (hab/km<sup>2</sup>)

La densité doit être calculée correctement car la surface d'une cellule n'est pas strictement constante en coordonnées géographiques. Le script calcule donc la surface des cellules (`cellSize`) puis :

$$\text{Densité} = \frac{\text{Population}}{\text{Surface cellule}}$$

Listing 2.1 – Densité de population 2017 et 2020 (hab/km<sup>2</sup>)

```
1 surface_cellule <- cellSize(pop_2017, unit = "km")
2 densite_2017 <- pop_2017 / surface_cellule
3 densite_2020 <- pop_2020 / surface_cellule
4
5 writeRaster(densite_2017, "output/rasters/densite_population_2017.tif",
             overwrite = TRUE)
6 writeRaster(densite_2020, "output/rasters/densite_population_2020.tif",
             overwrite = TRUE)
```

### 2.2 Binarisation haute densité (seuil 1500 hab/km<sup>2</sup>)

Pour identifier les **cellules candidates aux centres urbains**, le script applique un seuil strict :

Cellule HD = 1 si densité  $\geq$  1500, sinon NA

Listing 2.2 – Cellules de haute densité : seuil 1500 hab/km<sup>2</sup>

```
1 seuil_centre_urbain <- 1500
```

```

2 cellules_haute_densite_2017 <- ifel(densite_2017 >= seuil_centre_urbain,
 1, NA)
3 cellules_haute_densite_2020 <- ifel(densite_2020 >= seuil_centre_urbain,
 1, NA)
4
5 writeRaster(cellules_haute_densite_2017,
  "output/rasters/cellules_haute_densite_2017.tif", overwrite = TRUE)
6 writeRaster(cellules_haute_densite_2020,
  "output/rasters/cellules_haute_densite_2020.tif", overwrite = TRUE)

```

## 2.3 Clustering spatial : construction des grappes

Les cellules HD adjacentes (8 directions) sont regroupées en **grappes** via `patches(..., directions=8)`. Chaque grappe reçoit un identifiant unique.

Listing 2.3 – Identification des grappes (patches, 8-connectivité)

```

1 grappes_2017 <- patches(cellules_haute_densite_2017, directions = 8,
  zeroAsNA = TRUE)
2 grappes_2020 <- patches(cellules_haute_densite_2020, directions = 8,
  zeroAsNA = TRUE)
3
4 writeRaster(grappes_2017, "output/rasters/grappes_urbaines_2017.tif",
  overwrite = TRUE)
5 writeRaster(grappes_2020, "output/rasters/grappes_urbaines_2020.tif",
  overwrite = TRUE)

```

## 2.4 Population par grappe et classification

Le script agrège la population par identifiant de grappe, puis classe selon les seuils :

- Centre urbain :  $\geq 50\,000$
- Grappe urbaine :  $\geq 5\,000$
- Rural :  $< 5\,000$

Listing 2.4 – Agrégation population par grappe et classification DEGURBA

```

1 pop_grappes_2017 <- calculer_pop_grappes(grappes_2017, pop_2017, 2017) %>%
2   mutate(classe_degurba = case_when(
3     population_totale >= 50000 ~ "Centre urbain",
4     population_totale >= 5000 ~ "Grappe urbaine",
5     TRUE ~ "Zone rurale"
6   ))

```

```

7
8 pop_grappes_2020 <- calculer_pop_grappes(grappes_2020, pop_2020, 2020) %>%
9   mutate(classe_degurba = case_when(
10     population_totale >= 50000 ~ "Centre urbain",
11     population_totale >= 5000 ~ "Grappe urbaine",
12     TRUE ~ "Zone rurale"
13   ))

```

## 2.5 Raster binaire des centres urbains

Pour le calcul ODD 11.3.1, on extrait uniquement les **centres urbains** et on crée un raster binaire (1 = centre urbain, NA = reste).

Listing 2.5 – Rasters des centres urbains (DEGURBA)

```

1 urbain_2017 <- creer_raster_urbain(grappes_2017, pop_grappes_2017, 2017)
2 urbain_2020 <- creer_raster_urbain(grappes_2020, pop_grappes_2020, 2020)
3
4 writeRaster(urbain_2017, "output/rasters/zones_urbaines_degurba_2017.tif",
  overwrite = TRUE)
5 writeRaster(urbain_2020, "output/rasters/zones_urbaines_degurba_2020.tif",
  overwrite = TRUE)

```

# Chapitre 3

## Calcul de l'indicateur ODD 11.3.1

### 3.1 LCR : taux de consommation des terres

La surface urbaine totale est calculée par :

$$Urb = (\text{nb cellules urbaines}) \times (\text{surface moyenne cellule})$$

Puis le LCR est obtenu par la formule logarithmique (taux annuel en %).

Le script sécurise le calcul (zéro, NA, égalité des surfaces).

Listing 3.1 – Calcul de LCR (Land Consumption Rate)

```
1  annees <- 2020 - 2017
2
3  centres_urbains_2017 <- pop_grappes_2017 %>% filter(classe_degurba ==
   "Centre urbain")
4  centres_urbains_2020 <- pop_grappes_2020 %>% filter(classe_degurba ==
   "Centre urbain")
5
6  centres_urbains_2017$surface_km2 <- centres_urbains_2017$nb_cellules *
   stats_surface$mean
7  centres_urbains_2020$surface_km2 <- centres_urbains_2020$nb_cellules *
   stats_surface$mean
8
9  surface_totale_2017 <- sum(centres_urbains_2017$surface_km2, na.rm = TRUE)
10 surface_totale_2020 <- sum(centres_urbains_2020$surface_km2, na.rm = TRUE)
11
12 LCR <- (log(surface_totale_2020 / surface_totale_2017) / annees) * 100
```

## 3.2 PGR : taux de croissance démographique urbaine

La population urbaine totale est la somme des populations des centres urbains (grappes  $\geq 50\,000$ ). On applique ensuite la même formule logarithmique annualisée.

Listing 3.2 – Calcul de PGR (Population Growth Rate)

```
1 pop_urbaine_2017 <- sum(centres_urbains_2017$population_totale, na.rm =  
    TRUE)  
2 pop_urbaine_2020 <- sum(centres_urbains_2020$population_totale, na.rm =  
    TRUE)  
3  
4 PGR <- (log(pop_urbaine_2020 / pop_urbaine_2017) / annees) * 100
```

## 3.3 LCRPGR : indicateur final et interprétation

L'indicateur final est le ratio :

$$LCRPGR = \frac{LCR}{PGR}$$

Le script gère explicitement les cas non calculables ( $PGR = 0$ , valeurs NA).

Listing 3.3 – Calcul de LCRPGR (ODD 11.3.1)

```
1 LCRPGR <- LCR / PGR
```

## Interprétation

- Si  $LCRPGR > 1$ , la surface urbaine augmente plus vite que la population : **étalement urbain**.
- Si  $LCRPGR < 1$ , la population augmente plus vite que la surface : **densification**.
- Si  $LCRPGR \approx 1$ , croissance proportionnelle.

# Chapitre 4

## Analyse par centre urbain

En plus du résultat global, le TP calcule un LCRPGR **par centre urbain** (centre par centre). L'appariement est fait par rang de population (approximation : les plus grandes grappes correspondent entre 2017 et 2020). Pour chaque centre :

- $LCR_i$  à partir de la surface ( $\text{km}^2$ ),
- $PGR_i$  à partir de la population,
- $LCRPGR_i = LCR_i / PGR_i$  si  $PGR_i \neq 0$ .

Le fichier produit est : `output/degurba/odd_11_3_1_par_centre.csv`.

# Chapitre 5

## Cartographie et graphiques

### 5.1 Cartes densité 2017 et 2020

Le TP convertit les rasters en dataframes (`as.data.frame(..., xy=TRUE)`) et cartographie la densité en échelle log ( $\log_{10}(densité + 1)$ ) avec limites du pays et des provinces.

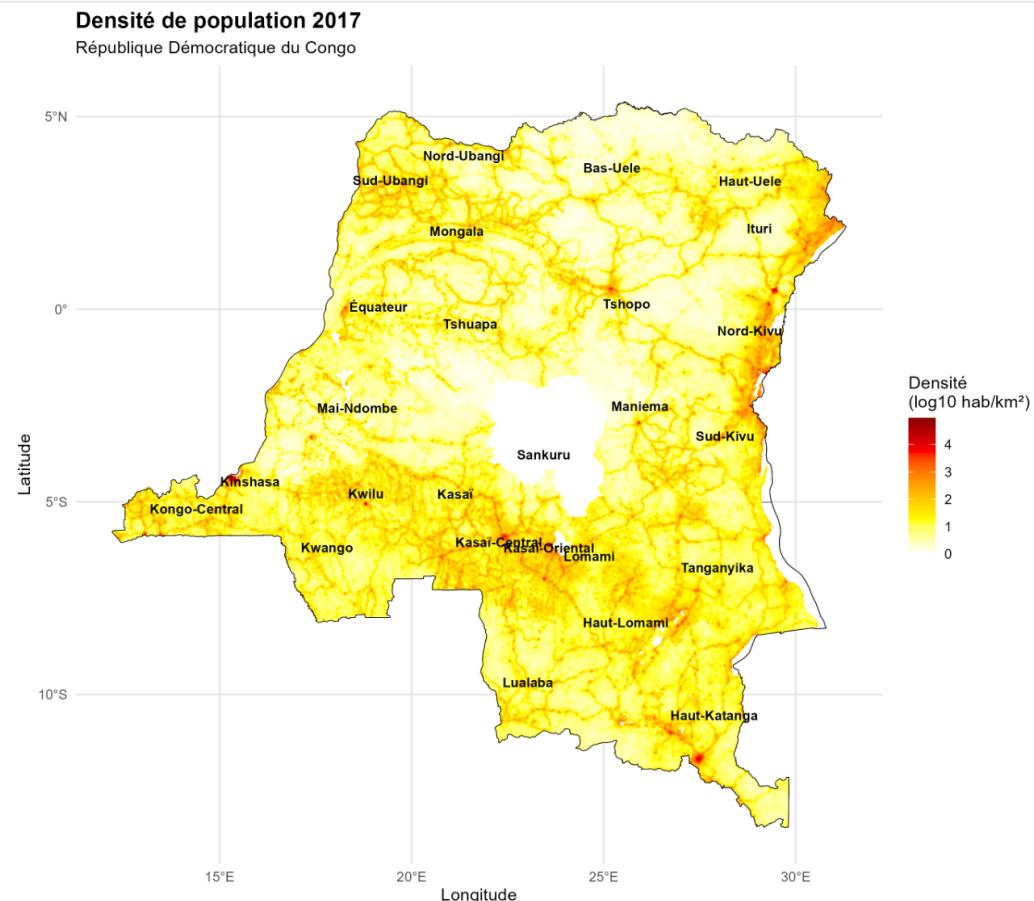


FIGURE 5.1 – Densité de population (2017) — RDC.

Ce TP5 met en œuvre une chaîne complète et reproductible pour calculer l'indicateur ODD 11.3.1 en RDC (2017–2020). La classification DEGURBA stricte repose sur la den-

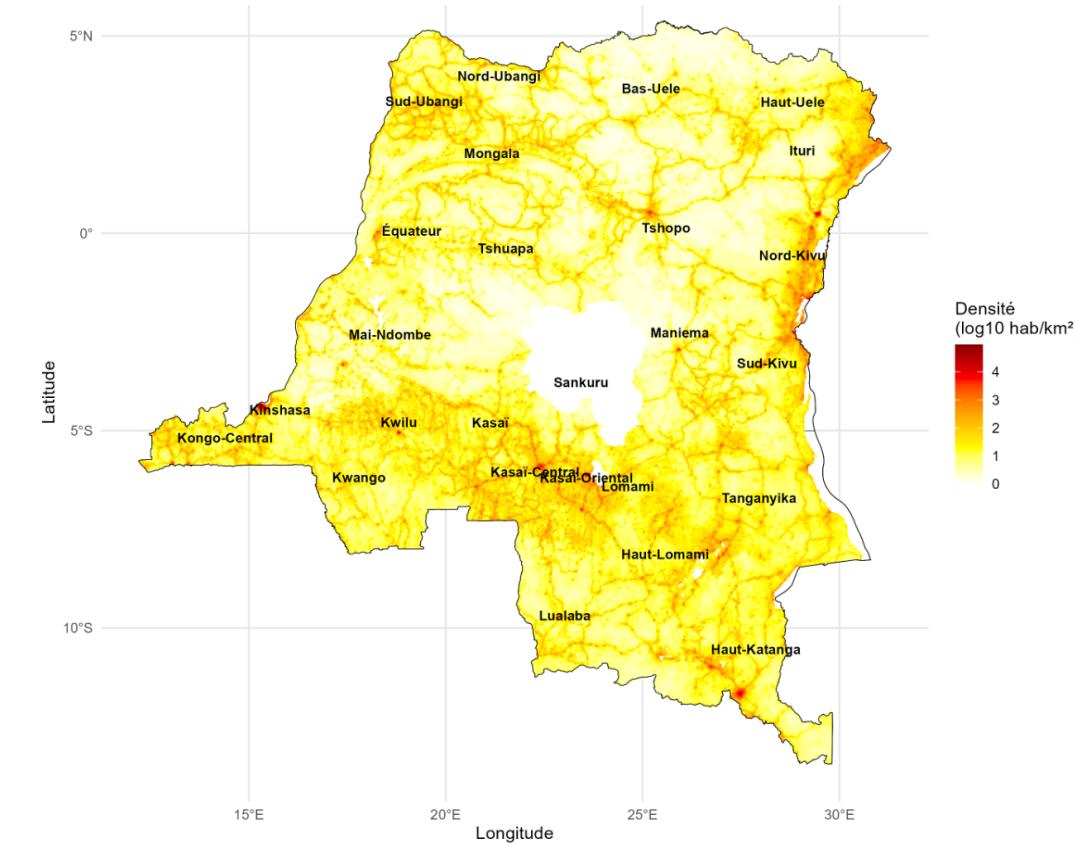


FIGURE 5.2 – Densité de population (2020) — RDC.

sité, la continuité spatiale (grappes) et la population totale des grappes. Le calcul final de LCRPGR permet d'évaluer si l'urbanisation est associée à une densification ( $< 1$ ) ou à un étalement ( $> 1$ ). Les résultats sont produits sous forme de rasters, de tableaux CSV et de cartes/graphes pour interprétation.

# TP6 — Indices spectraux & température de surface : Niger (Juin–Septembre 2022)

## 1. Objectif du TP

Ce TP vise à produire une analyse environnementale complète du Niger sur la période **01 juin 2022 – 30 septembre 2022** en combinant :

- un composite **Sentinel-2 SR** filtré par nuages ;
- plusieurs **indices spectraux** (végétation, eau, humidité) calculés à partir des bandes Sentinel-2 ;
- la **température de surface (LST)** issue de **MODIS** ;
- des **statistiques zonales** (moyenne, écart-type, min, max) par **département** (ADM2) et **région** (ADM1) ;
- des **cartes choroplèthes** (départements) ;
- des **exports** (tables CSV, rasters indices, bandes brutes Sentinel-2) et un **template EHCVM** pour faciliter la fusion socio-économique.

## 2. Données et paramètres

### 2.1 Paramètres de configuration

Le script s'appuie sur une configuration centrale :

- dates : 2022-06-01 à 2022-09-30
- seuil nuages : 20%
- résolution statistiques : 100 m (compromis vitesse/précision)
- résolution export indices : 10 m (Sentinel-2)
- dossier Drive : **GEE\_Niger\_Analyse**

Listing 5.1 – Configuration globale du TP6

```
1 var CONFIG = {
```

```
2     dateDebut: '2022-06-01',
3     dateFin: '2022-09-30',
4     cloudThreshold: 20,
5     exportFolder: 'GEE_Niger_Analyse',
6     scale: 100,
7     scaleExport: 10
8   };
```

## 2.2 Jeux de données mobilisés

- **Limites admin** : FAO/GAUL/2015 niveaux 0,1,2 (Niger).
- **Sentinel-2 SR** : COPERNICUS/S2\_SR.
- **MODIS LST** : MODIS/061/MOD11A2 bande LST\_Day\_1km.

# Chapitre 6

## Partie 1 — Limites administratives et zone d'étude

### 6.1 Chargement des limites GAUL

Le Niger est extrait de GAUL 2015 niveau 0, puis on charge les régions (ADM1) et départements (ADM2). Ces couches servent :

- à définir l’AOI (zone d’étude) ;
- à réaliser les statistiques zonales ;
- à construire les cartes choropléthes.

Listing 6.1 – Chargement des limites GAUL : Niger, régions, départements

```
1 var niger = ee.FeatureCollection('FAO/GAUL/2015/level0')
2   .filter(ee.Filter.eq('ADM0_NAME', 'Niger'));
3
4 var regions = ee.FeatureCollection('FAO/GAUL/2015/level1')
5   .filter(ee.Filter.eq('ADM0_NAME', 'Niger'));
6
7 var departements = ee.FeatureCollection('FAO/GAUL/2015/level2')
8   .filter(ee.Filter.eq('ADM0_NAME', 'Niger'));
9
10 Map.centerObject(niger, 6);
```

### 6.2 Visualisation des contours

Les couches limites sont ajoutées à la carte (désactivées par défaut) pour faciliter l’inspection visuelle et le contrôle du découpage.

Listing 6.2 – Ajout des limites sur la carte

```
1 Map.addLayer(niger, {color: 'black'}, 'Niger', false);
2 Map.addLayer(regions, {color: 'blue'}, 'Regions', false);
3 Map.addLayer(departements, {color: 'red'}, 'Departements', false);
```

# Chapitre 7

## Partie 2 — Sentinel-2 : composite et bandes

### 7.1 Pré-traitement : sélection de bandes

On restreint Sentinel-2 aux bandes nécessaires aux indices :

$B2$ (Blue),  $B3$ (Green),  $B4$ (Red),  $B8$ (NIR),  $B11$ (SWIR1)

On conserve aussi la propriété temporelle `system:time_start`.

Listing 7.1 – Fonction de sélection des bandes Sentinel-2

```
1 function selectBands(image) {  
2     return image.select(['B2', 'B3', 'B4', 'B8', 'B11'])  
3     .copyProperties(image, ['system:time_start']);  
4 }
```

### 7.2 Filtrage : période + nuages + AOI

On filtre spatialement sur le Niger, temporellement sur Juin–Septembre 2022, puis par pourcentage de nuages (<20%).

Listing 7.2 – Chargement Sentinel-2 SR et filtrage nuages

```
1 var s2 = ee.ImageCollection('COPERNICUS/S2_SR')  
2     .filterBounds(niger)  
3     .filterDate(CONFIG.dateDebut, CONFIG.dateFin)  
4     .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', CONFIG.cloudThreshold))  
5     .map(selectBands);
```

## 7.3 Composite médian et extraction des bandes

Le composite médian réduit l'influence d'images anormales (nuages résiduels, bruit), puis on découpe au Niger.

Listing 7.3 – Composite médian + extraction des bandes

```
1 var composite = s2.median().clip(niger);
2
3 var B2 = composite.select('B2');
4 var B3 = composite.select('B3');
5 var B4 = composite.select('B4');
6 var B8 = composite.select('B8');
7 var B11 = composite.select('B11');
```

## 7.4 Visualisation RGB

Une visualisation RGB (B4,B3,B2) est proposée pour inspection.

Listing 7.4 – Affichage du composite Sentinel-2 en RGB

```
1 var rgbVis = {
2   bands: ['B4', 'B3', 'B2'],
3   min: 0,
4   max: 3000,
5   gamma: 1.4
6 };
7 Map.addLayer(composite, rgbVis, 'Sentinel-2 RGB', false);
```

# Chapitre 8

## Partie 3 — Calcul des indices spectraux

### 8.1 Rappels (formules)

Les indices calculés sont :

#### NDVI (végétation)

$$NDVI = \frac{NIR - RED}{NIR + RED}$$

#### EVI (végétation améliorée)

$$EVI = 2.5 \times \frac{NIR - RED}{NIR + 6RED - 7.5BLUE + 1}$$

#### NDWI (eau)

$$NDWI = \frac{GREEN - NIR}{GREEN + NIR}$$

#### NDMI (humidité végétation/sol)

$$NDMI = \frac{NIR - SWIR}{NIR + SWIR}$$

## MDVI (indice de végétation modifié)

$$MDVI = \frac{2NIR - RED}{2NIR + RED}$$

## 8.2 Implémentation GEE

On ajoute 0.0001 pour éviter division par zéro.

Listing 8.1 – Calcul des indices NDVI, EVI, NDWI, NDMI, MDVI

```
1 var NDVI = B8.subtract(B4)
2   .divide(B8.add(B4).add(0.0001))
3   .rename('NDVI')
4   .clip(niger);
5
6 var EVI = B8.subtract(B4)
7   .divide(B8.add(B4.multiply(6)).subtract(B2.multiply(7.5)).add(1).add(0.0001))
8   .multiply(2.5)
9   .rename('EVI')
10  .clip(niger);
11
12 var NDWI = B3.subtract(B8)
13   .divide(B3.add(B8).add(0.0001))
14   .rename('NDWI')
15   .clip(niger);
16
17 var NDMI = B8.subtract(B11)
18   .divide(B8.add(B11).add(0.0001))
19   .rename('NDMI')
20   .clip(niger);
21
22 var MDVI = B8.multiply(2).subtract(B4)
23   .divide(B8.multiply(2).add(B4).add(0.0001))
24   .rename('MDVI')
25   .clip(niger);
26
27 var indices =
  NDVI.addBands(EVI).addBands(NDWI).addBands(NDMI).addBands(MDVI);
```

# Chapitre 9

## Partie 4 — MODIS LST : température de surface

### 9.1 Chargement et conversion en degrés Celsius

MODIS MOD11A2 fournit une température échelonnée. La conversion est :

$$LST(^{\circ}C) = LST\_raw \times 0.02 - 273.15$$

Listing 9.1 – MODIS LST (moyenne période) et conversion Celsius

```
1 var modis = ee.ImageCollection('MODIS/061/MOD11A2')
2   .filterBounds(niger)
3   .filterDate(CONFIG.dateDebut, CONFIG.dateFin)
4   .select('LST_Day_1km');
5
6 var LST = modis.mean()
7   .multiply(0.02)
8   .subtract(273.15)
9   .rename('LST')
10  .clip(niger);
```

### 9.2 Image complète indices + LST

On agrège tout en une image multi-bandes.

Listing 9.2 – Image finale multi-bandes : indices + LST

```
1 var indicesComplet = indices.addBands(LST);
```

# Chapitre 10

## Partie 5 — Visualisation cartographique

### 10.1 Palettes et couches

Six couches raster sont ajoutées : NDVI, EVI, NDWI, NDMI, MDVI, LST. Elles sont désactivées par défaut pour éviter de surcharger l'affichage.

Listing 10.1 – Ajout des couches indices et LST sur la carte

```
1 var ndviPalette = ['brown', 'yellow', 'lightgreen', 'green', 'darkgreen'];
2 var eviPalette = ['white', 'lightgreen', 'green', 'darkgreen'];
3 var ndwiPalette = ['brown', 'tan', 'white', 'lightblue', 'blue'];
4 var ndmiPalette = ['red', 'orange', 'yellow', 'lightgreen', 'green'];
5 var mdviPalette = ['brown', 'yellow', 'green', 'darkgreen'];
6 var lstPalette = ['blue', 'cyan', 'yellow', 'orange', 'red'];
7
8 Map.addLayer(NDVI, {min: 0, max: 0.8, palette: ndviPalette}, 'NDVI',
    false);
9 Map.addLayer(EVI, {min: 0, max: 1, palette: eviPalette}, 'EVI', false);
10 Map.addLayer(NDWI, {min: -0.5, max: 0.5, palette: ndwiPalette}, 'NDWI',
    false);
11 Map.addLayer(NDMI, {min: -0.5, max: 0.5, palette: ndmiPalette}, 'NDMI',
    false);
12 Map.addLayer(MDVI, {min: 0, max: 0.8, palette: mdviPalette}, 'MDVI',
    false);
13 Map.addLayer(LST, {min: 20, max: 45, palette: lstPalette}, 'LST', false);
```

# Chapitre 11

## Partie 6 — Statistiques zonales

### 11.1 Principe

On calcule, pour chaque zone administrative (département ou région), des statistiques sur chaque bande de `indicesComplet` :

mean, stdDev, min, max

Ces statistiques sont extraites via `reduceRegion` à une résolution **100 m.**

### 11.2 Fonction générique de calcul

La fonction `calculerStats(feature)` :

- prend une entité administrative (`feature`) ;
- calcule les statistiques sur sa géométrie ;
- renvoie la même feature enrichie avec ces résultats.

Listing 11.1 – Fonction de calcul des statistiques (mean/std/min/max)

```
1 var calculerStats = function(feature) {  
2     var stats = indicesComplet.reduceRegion({  
3         reducer: ee.Reducer.mean()  
4             .combine({reducer2: ee.Reducer.stdDev(), sharedInputs: true})  
5             .combine({reducer2: ee.Reducer.min(), sharedInputs: true})  
6             .combine({reducer2: ee.Reducer.max(), sharedInputs: true}),  
7         geometry: feature.geometry(),  
8         scale: CONFIG.scale,  
9         maxPixels: 1e13  
10    });  
11    return feature.set(stats);  
12};
```

## 11.3 Extraction par département (ADM2)

Listing 11.2 – Statistiques par départements

```
1 var statsDept = departements.map(calculerStats);
```

## 11.4 Extraction par région (ADM1)

Listing 11.3 – Statistiques par régions

```
1 var statsRegion = regions.map(calculerStats);
```

## 11.5 Statistiques globales Niger

On calcule les mêmes statistiques à l'échelle nationale. On en extrait notamment les moyennes par indice.

Listing 11.4 – Statistiques globales Niger

```
1 var statsNiger = indicesComplet.reduceRegion({  
2   reducer: ee.Reducer.mean()  
3   .combine({reducer2: ee.Reducer.stdDev(), sharedInputs: true})  
4   .combine({reducer2: ee.Reducer.min(), sharedInputs: true})  
5   .combine({reducer2: ee.Reducer.max(), sharedInputs: true}),  
6   geometry: niger.geometry(),  
7   scale: CONFIG.scale,  
8   maxPixels: 1e13  
9 });
```

# Chapitre 12

## Partie 7 — Cartes choroplèthes (départements)

### 12.1 Principe

Le script produit des cartes « par département » en peignant une image à partir de la table `statsDept`. Trois cartes sont réalisées :

- `NDVI_mean`,
- `EVI_mean`,
- `LST_mean`.

Puis on ajoute les limites des départements en noir.

Listing 12.1 – Cartes choroplèthes départementales (NDVI, EVI, LST)

```
1 var carteDeptNDVI = ee.Image().byte().paint({
2   featureCollection: statsDept,
3   color: 'NDVI_mean'
4 }).visualize({min: 0, max: 0.8, palette: ndviPalette});
5
6 var carteDeptEVI = ee.Image().byte().paint({
7   featureCollection: statsDept,
8   color: 'EVI_mean'
9 }).visualize({min: 0, max: 1, palette: eviPalette});
10
11 var carteDeptLST = ee.Image().byte().paint({
12   featureCollection: statsDept,
13   color: 'LST_mean'
14 }).visualize({min: 20, max: 45, palette: lstPalette});
15
16 var limitesDept = ee.Image().byte().paint({
17   featureCollection: departements,
```

```
18     color: 0,
19     width: 1
20   });
21
22 Map.addLayer(carteDeptNDVI, {}, 'Carte NDVI Departements', false);
23 Map.addLayer(carteDeptEVI, {}, 'Carte EVI Departements', false);
24 Map.addLayer(carteDeptLST, {}, 'Carte LST Departements', false);
25 Map.addLayer(limitesDept, {palette: 'black'}, 'Limites Departements',
  false);
```

*Remarque :* Ici, la cartographie est une **visualisation** (rendu coloré), utile pour l'inspection dans GEE. Pour une cartographie avancée hors GEE, on utilisera plutôt les CSV exportés et/ou les rasters indices.

# Chapitre 13

## Partie 8 — Exports (Drive) et livrables

### 13.1 Exports CSV : départements et régions

Deux tables principales sont exportées, avec sélection explicite des colonnes utiles (identifiants admin + stats des indices).

Listing 13.1 – Export CSV départements

```
1 Export.table.toDrive({  
2   collection: statsDept,  
3   description: 'Niger_Stats_Departements_2022',  
4   folder: CONFIG.exportFolder,  
5   fileNamePrefix: 'Niger_Stats_Departements_2022',  
6   fileFormat: 'CSV',  
7   selectors: ['ADMO_NAME', 'ADM1_NAME', 'ADM2_NAME', 'ADM2_CODE',  
8             'NDVI_mean', 'NDVI_stdDev', 'NDVI_min', 'NDVI_max',  
9             'EVI_mean', 'EVI_stdDev', 'EVI_min', 'EVI_max',  
10            'NDWI_mean', 'NDWI_stdDev', 'NDWI_min', 'NDWI_max',  
11            'NDMI_mean', 'NDMI_stdDev', 'NDMI_min', 'NDMI_max',  
12            'MDVI_mean', 'MDVI_stdDev', 'MDVI_min', 'MDVI_max',  
13            'LST_mean', 'LST_stdDev', 'LST_min', 'LST_max']  
14});
```

Listing 13.2 – Export CSV régions

```
1 Export.table.toDrive({  
2   collection: statsRegion,  
3   description: 'Niger_Stats_Regions_2022',  
4   folder: CONFIG.exportFolder,  
5   fileNamePrefix: 'Niger_Stats_Regions_2022',
```

```

6   fileFormat: 'CSV',
7   selectors: ['ADM0_NAME', 'ADM1_NAME', 'ADM1_CODE',
8               'NDVI_mean', 'NDVI_stdDev', 'NDVI_min', 'NDVI_max',
9               'EVI_mean', 'EVI_stdDev', 'EVI_min', 'EVI_max',
10              'NDWI_mean', 'NDWI_stdDev', 'NDWI_min', 'NDWI_max',
11              'NDMI_mean', 'NDMI_stdDev', 'NDMI_min', 'NDMI_max',
12              'MDVI_mean', 'MDVI_stdDev', 'MDVI_min', 'MDVI_max',
13              'LST_mean', 'LST_stdDev', 'LST_min', 'LST_max']
14 });

```

## 13.2 Exports rasters : indices (6 fichiers)

Chaque indice est exporté en GeoTIFF (Sentinel-2 à 10 m ; MODIS LST à 1 km).

Listing 13.3 – Paramètres généraux d'export raster

```

1 var exportParams = {
2   folder: CONFIG.exportFolder,
3   region: niger.geometry(),
4   scale: CONFIG.scaleExport,
5   crs: 'EPSG:4326',
6   maxPixels: 1e13
7 };

```

### Indices Sentinel-2 (10 m)

Listing 13.4 – Exemple : export NDVI (les autres indices suivent le même schéma)

```

1 Export.image.toDrive({
2   image: NDVI,
3   description: 'Niger_NDVI_2022',
4   fileNamePrefix: 'Niger_NDVI_2022',
5   folder: exportParams.folder,
6   region: exportParams.region,
7   scale: exportParams.scale,
8   crs: exportParams.crs,
9   maxPixels: exportParams.maxPixels
10 });

```

### LST MODIS (1 km)

Listing 13.5 – Export LST (résolution MODIS native)

```
1 Export.image.toDrive({  
2     image: LST,  
3     description: 'Niger_LST_2022',  
4     fileNamePrefix: 'Niger_LST_2022',  
5     folder: exportParams.folder,  
6     region: exportParams.region,  
7     scale: 1000,  
8     crs: exportParams.crs,  
9     maxPixels: exportParams.maxPixels  
10});
```

### 13.3 Exports rasters : bandes Sentinel-2 (5 fichiers)

On exporte les bandes utiles (B2,B3,B4,B8,B11) pour d'éventuels traitements externes (QGIS/R/Python). La bande B11 est à 20 m (résolution native).

Listing 13.6 – Exemple : export bande B11 (SWIR1, 20 m)

```
1 Export.image.toDrive({  
2     image: B11,  
3     description: 'Niger_S2_B11_2022',  
4     fileNamePrefix: 'Niger_S2_B11_2022',  
5     folder: exportParams.folder,  
6     region: exportParams.region,  
7     scale: 20,  
8     crs: exportParams.crs,  
9     maxPixels: exportParams.maxPixels  
10});
```

# Chapitre 14

## Partie 9 — Template de fusion EHCVM (livrable clé)

### 14.1 Objectif

L'objectif est de faciliter la fusion entre les données environnementales (indices) et les données socio-économiques EHCVM. Le script génère une table simplifiée, centrée sur les identifiants administratifs et les **moyennes** des indices, prête à être fusionnée.

### 14.2 Construction du template

Le template contient : Pays, Region, Departement, Code\_Dept, NDVI, EVI, NDWI, NDMI, MDVI, LST, Année, Saison.

Listing 14.1 – Création du template EHCVM (par département)

```
1 var templateEHCVM = statsDept.map(function(feature) {  
2   return ee.Feature(null, {  
3     'Pays': feature.get('ADMO_NAME'),  
4     'Region': feature.get('ADM1_NAME'),  
5     'Departement': feature.get('ADM2_NAME'),  
6     'Code_Dept': feature.get('ADM2_CODE'),  
7     'NDVI': feature.get('NDVI_mean'),  
8     'EVI': feature.get('EVI_mean'),  
9     'NDWI': feature.get('NDWI_mean'),  
10    'NDMI': feature.get('NDMI_mean'),  
11    'MDVI': feature.get('MDVI_mean'),  
12    'LST': feature.get('LST_mean'),  
13    'Année': 2022,  
14    'Saison': 'Juin-Septembre'  
15  });
```

```
16});
```

### 14.3 Export du template

Listing 14.2 – Export Drive : Niger\_Template\_EHCVM\_2022.csv

```
1 Export.table.toDrive({  
2   collection: templateEHCVM,  
3   description: 'Niger_Template_EHCVM_2022',  
4   folder: CONFIG.exportFolder,  
5   fileNamePrefix: 'Niger_Template_EHCVM_2022',  
6   fileFormat: 'CSV'  
7});
```

### Instructions de fusion (hors GEE)

- Télécharger `Niger_Template_EHCVM_2022.csv` depuis Google Drive.
- Charger l'EHCVM dans R/Stata/Python.
- Fusionner sur **Département** (nom ou code ADM2) : idéalement **Code\_Dept** si disponible côté EHCVM.
- Variables ajoutées à l'EHCVM : NDVI, EVI, NDWI, NDMI, MDVI, LST (moyennes).

# Chapitre 15

## Partie 10 — Résumé final

### 15.1 Résultats produits

Le TP prépare au total **14 livrables** (selon le récapitulatif du script) :

- **Tables (3)** : Stats départements, Stats régions, Template EHCVM.
- **Rasters indices (6)** : NDVI, EVI, NDWI, NDMI, MDVI, LST.
- **Rasters bandes S2 (5)** : B2, B3, B4, B8, B11.

### 15.2 Étapes pratiques pour exécuter

1. Dans GEE Code Editor : onglet **Tasks**.
2. Cliquer **RUN** pour chaque export (tables + rasters).
3. Attendre la fin des exports (durée variable selon quota et taille).
4. Télécharger depuis Google Drive → dossier **GEE\_Niger\_Analyse**.

### 15.3 Conclusion

Ce TP6 fournit une chaîne complète d'analyse environnementale multi-source (Sentinel-2 + MODIS), des statistiques à plusieurs niveaux administratifs (départements/régions), une visualisation cartographique rapide dans GEE et des exports structurés pour un usage externe. Le livrable stratégique est le **template EHCVM**, conçu pour relier les indices environnementaux aux variables socio-économiques, afin d'étudier par exemple les liens entre végétation/humidité/température et les indicateurs de conditions de vie.