Solar PV Site Suitability Analysis in Kenya

Business Understanding

Access to clean, affordable, and sustainable energy is a cornerstone of economic development. Kenya has significant solar potential due to its high solar irradiance, but identifying the most suitable sites for solar photovoltaic (PV) projects requires more than just measuring sunlight. Environmental, technical, and land use constraints (e.g., slope, water bodies, urban areas, protected areas) must be considered to avoid conflicts, minimize costs, and maximize power generation.

This project aims to provide a data-driven, GIS-based framework to locate optimal areas for solar PV installations.

Problem Statement

Although solar irradiance data is available, decision-makers lack an integrated, spatially explicit model that combines solar resources (DNI, GHI, GTI, PVOUT, DIF), climate factors (temperature), and land constraints (protected areas, water bodies, forests, urban areas) to identify the most suitable zones for large-scale PV deployment.

Without this, investments may target suboptimal or environmentally sensitive sites, leading to:

- Increased project costs
- Land-use conflicts
- Reduced energy efficiency

Objectives

The project will:

1. Integrate multi-source GIS data (irradiance, land use, constraints, climate).
2. Normalize and weight each factor based on its importance to solar PV suitability.
3. Develop a weighted overlay suitability model to produce a final suitability map.
4. Identify highly suitable regions for solar PV deployment in Kenya.

5. Provide insights to policymakers, investors, and planners to guide clean energy development.

Data Layers Used

1. Solar resource: DIF, DNI, GHI, GTI, PVOUT
2. Climate: Temperature (TEMP)
3. Constraints: Protected areas, water bodies, urban areas, forests, agricultural areas

Metrics of Success

✅ A suitability map that ranks all areas of Kenya (0–1 scale: unsuitable → highly suitable).

✅ At least 80% spatial agreement between identified suitable zones and known existing solar PV farms (validation).

✅ Usability of the model for decision-making: clear, interpretable maps for planners.

✅ Modular framework: ability to adjust weights and add/remove layers.

Methodology

Step 1: Preprocess rasters (resample, normalize irradiance layers).

Step 2: Rasterize shapefiles (protected areas, forests, urban, water).

Step 3: Apply constraints (mask unsuitable areas = 0).

Step 4: Weighted overlay analysis (AHP / MCA).

Step 5: Generate suitability map.

Step 6: Validation (compare with known solar farms or theoretical hotspots).

Data Understanding

The project integrates geospatial datasets from multiple sources to evaluate the suitability of locations for solar PV deployment in Kenya. These datasets represent both resource availability (how much solar energy is available) and constraints (where solar farms cannot or should not be built).

1. **Solar Radiation & Energy Potential Rasters**

- DNI (Direct Normal Irradiance) – measures solar radiation received per unit area by a surface always oriented toward the sun. Important for concentrating solar technologies but also a proxy for direct beam solar potential.
- DIF (Diffuse Horizontal Irradiance) – represents scattered sunlight received from the sky dome.
- GHI (Global Horizontal Irradiance) – the total solar radiation received on a horizontal surface; critical for PV system design.
- GTI (Global Tilted Irradiance) – the solar radiation on optimally tilted surfaces, more representative of actual PV installations.
- PVOUT (PV electricity output potential) – modeled electricity generation potential (kWh/kWp). This incorporates solar radiation and typical PV system efficiencies.
- TEMP (Ambient Temperature) – average surface air temperature, since higher temperatures reduce PV efficiency.

👉 Relevance: Together, these rasters describe both the solar energy resource (supply side) and the expected system performance (efficiency + yield).

2. **Environmental & Land-Use Constraints (Vector Data)**

- Protected Areas (shapefile) – National parks, reserves, or conservation areas where large-scale PV cannot be deployed.
- Water Bodies (shapefile) – Lakes, rivers, wetlands; unsuitable for PV development.
- Urban Areas (shapefile) – Cities and settlements; avoided for utility-scale PV but may be important for rooftop PV.
- Forests (shapefile) – Dense vegetation zones; often excluded due to environmental impacts.

👉 Relevance: These represent exclusion zones that reduce the technically usable land area.

3. **Auxiliary Data (Optional)**

- DEM (Digital Elevation Model) – used to derive slope. Steep terrain is less suitable for PV installation due to construction challenges.
- Slope & Elevation – if included, they add realism: flat/moderate slopes (<10°) are preferred.

👉 Note: DEM is not mandatory for a first version of the project, but improves accuracy.

4. **Spatial & Technical Considerations**

- Coordinate Reference System (CRS): All datasets must be projected to a common CRS (e.g., WGS84 EPSG:4326 or a UTM projection for Kenya).
- Resolution & Extent: Raster datasets vary in resolution (e.g., 1 km, 5 km). To compare them, resampling to a common grid is necessary.
- Data Quality Checks: Verifying missing values, nodata pixels, and valid attribute fields is essential before processing.

Data Preparation

This phase involves cleaning, transforming, and organizing your spatial and tabular data so it can be meaningfully analyzed. Since you're working with geographical raster and vector data, preparation is crucial.

Data cleaning

- **Check coordinate reference systems (CRS):**
  - All your layers (DNI, DIF, GHI, GTI, PVOUT, TEMP, DEM, protected areas, water bodies, forests, urban areas) must be in the same projection (commonly WGS84 - EPSG:4326, or a suitable local projection).
  - If not, reproject them.
- **Remove unnecessary data:**
  - Clip the data to your study area (e.g., your country, region, or specific boundary).
  - This reduces computation and makes results clearer.

1. Import necessary libraries

In [1]:
```
# importing the necessary libraries
import geopandas as gpd
import rasterio
from rasterio.plot import show
from rasterio.mask import mask
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
from shapely.geometry import mapping
from rasterio.features import rasterize
from rasterio.warp import calculate_default_transform, reproject, Resampling
from shapely.geometry import mapping
```

```
from rasterio.warp import reproject, Resampling
import os
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from rasterio.merge import merge
from rasterio.plot import show
from sklearn.model_selection import train_test_split
```

In [2]:
```
#pip install seaborn
```

2. Load the Kenya boundary

In [3]:
```
# Kenya boundary shapefile (this is your "study area")
kenya_boundary = gpd.read_file("kenya_Kenya_Country_Boundary.shp")
kenya_boundary = kenya_boundary.to_crs(epsg=4326)  # standard WGS84
```

3. Load raster layers

In [4]:
```
raster_files = {
    "DIF": "DIF.tif",
    "DNI": "DNI.tif",
    "GHI": "GHI.tif",
    "GTI": "GTI.tif",
    "PVOUT": "PVOUT.tif",
    "TEMP": "TEMP.tif"
}

rasters = {}
for key, path in raster_files.items():
    rasters[key] = rasterio.open(path)
    print(f"{key}: CRS={rasters[key].crs}, Bounds={rasters[key].bounds}")
```

```
DIF: CRS=GEOGCS["WGS 84",DATUM["World Geodetic System 1984",SPHEROID["WGS 84",637813
7,298.257223563]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433,AUTHORITY
["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS["Longitude",EAST]], Bounds=BoundingBox
(left=33.0, bottom=-5.0, right=43.0, top=5.0)
DNI: CRS=GEOGCS["WGS 84",DATUM["World Geodetic System 1984",SPHEROID["WGS 84",637813
7,298.257223563]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433,AUTHORITY
["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS["Longitude",EAST]], Bounds=BoundingBox
(left=33.0, bottom=-5.0, right=43.0, top=5.0)
GHI: CRS=GEOGCS["WGS 84",DATUM["World Geodetic System 1984",SPHEROID["WGS 84",637813
7,298.257223563]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433,AUTHORITY
["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS["Longitude",EAST]], Bounds=BoundingBox
(left=33.0, bottom=-5.0, right=43.0, top=5.0)
GTI: CRS=GEOGCS["WGS 84",DATUM["World Geodetic System 1984",SPHEROID["WGS 84",637813
7,298.257223563]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433,AUTHORITY
["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS["Longitude",EAST]], Bounds=BoundingBox
(left=33.0, bottom=-5.0, right=43.0, top=5.0)
PVOUT: CRS=GEOGCS["WGS 84",DATUM["World Geodetic System 1984",SPHEROID["WGS 84",6378
137,298.257223563]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433,AUTHORITY
["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS["Longitude",EAST]], Bounds=BoundingBox
(left=33.0, bottom=-5.0, right=43.0, top=5.0)
TEMP: CRS=GEOGCS["WGS 84",DATUM["World Geodetic System 1984",SPHEROID["WGS 84",63781
37,298.257223563]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433,AUTHORITY
["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS["Longitude",EAST]], Bounds=BoundingBox
(left=33.0, bottom=-5.0, right=43.0, top=5.0)
```

4. Clip rasters to the Kenyan boundaries

In [5]:
```python
def clip_raster(raster, boundary):
    geo = [mapping(boundary.unary_union)]
    out_img, out_transform = mask(raster, geo, crop=True)
    out_meta = raster.meta.copy()
    out_meta.update({
        "driver": "GTiff",
        "height": out_img.shape[1],
        "width": out_img.shape[2],
        "transform": out_transform
    })
    return out_img, out_meta

clipped_rasters = {}
for key, raster in rasters.items():
    clipped_rasters[key] = clip_raster(raster, kenya_boundary)
    print(f"{key} clipped successfully.")
```

```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\3870657882.py:2: DeprecationWarning: T
he 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  geo = [mapping(boundary.unary_union)]
DIF clipped successfully.
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\3870657882.py:2: DeprecationWarning: T
he 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  geo = [mapping(boundary.unary_union)]
DNI clipped successfully.
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\3870657882.py:2: DeprecationWarning: T
he 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  geo = [mapping(boundary.unary_union)]
```

```
GHI clipped successfully.
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\3870657882.py:2: DeprecationWarning: T
he 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  geo = [mapping(boundary.unary_union)]
GTI clipped successfully.
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\3870657882.py:2: DeprecationWarning: T
he 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  geo = [mapping(boundary.unary_union)]
PVOUT clipped successfully.
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\3870657882.py:2: DeprecationWarning: T
he 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  geo = [mapping(boundary.unary_union)]
TEMP clipped successfully.
```

5. Load vector layers

In [6]:

```
# Kenya boundary shapefile (already loaded earlier in your workflow)
kenya_boundary = gpd.read_file("kenya_Kenya_Country_Boundary.shp")

# Dictionary of vector layers
vector_files = {
    "Agriculture": "ke_agriculture.shp",
    "Protected": "ke_protected-areas.shp",
    "Water": "ke_waterbodies.shp",
    "Urban": "ke_urban.shp",
    "Forests": "ke_forests.shp",
    "Counties": "County.shp"
}

vectors = {}
for key, path in vector_files.items():
    gdf = gpd.read_file(path)

    # If CRS is missing, assign WGS84 (EPSG:4326)
    if gdf.crs is None:
        print(f"{key} has no CRS, assigning EPSG:4326 (WGS84).")
        gdf = gdf.set_crs(epsg=4326)

    # Reproject to Kenya CRS
    gdf = gdf.to_crs(kenya_boundary.crs)

    # Clip to Kenya boundary
    gdf = gpd.overlay(gdf, kenya_boundary, how="intersection")

    vectors[key] = gdf
    print(f"{key}: {len(gdf)} features after clipping.")
```

```
Agriculture: 2254 features after clipping.
Protected: 189 features after clipping.
Water has no CRS, assigning EPSG:4326 (WGS84).
Water: 201 features after clipping.
Urban: 259 features after clipping.
Forests: 4715 features after clipping.
Counties: 47 features after clipping.
```

6. Rasterize Vector Layers

In [7]:
```
def rasterize_layer(gdf, reference_meta):
    transform = reference_meta["transform"]
    out_shape = (reference_meta["height"], reference_meta["width"])
    shapes = ((geom, 1) for geom in gdf.geometry)
    raster = rasterize(shapes=shapes, out_shape=out_shape, transform=transform, fil
    return raster

# Use DNI raster as reference
reference_meta = clipped_rasters["DNI"][1]

rasterized_layers = {}
for key, gdf in vectors.items():
    rasterized_layers[key] = rasterize_layer(gdf, reference_meta)
    print(f"{key} rasterized with values: {np.unique(rasterized_layers[key])}")
```

```
Agriculture rasterized with values: [0 1]
Protected rasterized with values: [0 1]
Water rasterized with values: [0 1]
Urban rasterized with values: [0 1]
Forests rasterized with values: [0 1]
Counties rasterized with values: [0 1]
```

Exploratory Data Analysis

Spatial Analysis

Univariate Spatial Analysis

In [8]:
```
# --- Load Kenya boundary ---
kenya_boundary = gpd.read_file("kenya_Kenya_Country_Boundary.shp")

def plot_raster(raster_path, title, cmap="viridis"):
    """Plot a raster (.tif) file clipped to Kenya boundary"""
    with rasterio.open(raster_path) as src:
        fig, ax = plt.subplots(figsize=(8, 8))
        show(src, ax=ax, cmap=cmap)
        kenya_boundary.boundary.plot(ax=ax, color="black", linewidth=1)
        ax.set_title(title)
        plt.show()

def plot_vector(vector_path, title, facecolor="lightgrey", edgecolor="black"):
    """Plot a shapefile clipped to Kenya boundary"""
    gdf = gpd.read_file(vector_path)

    # Ensure CRS is set
    if gdf.crs is None:
        gdf = gdf.set_crs(epsg=4326)
```

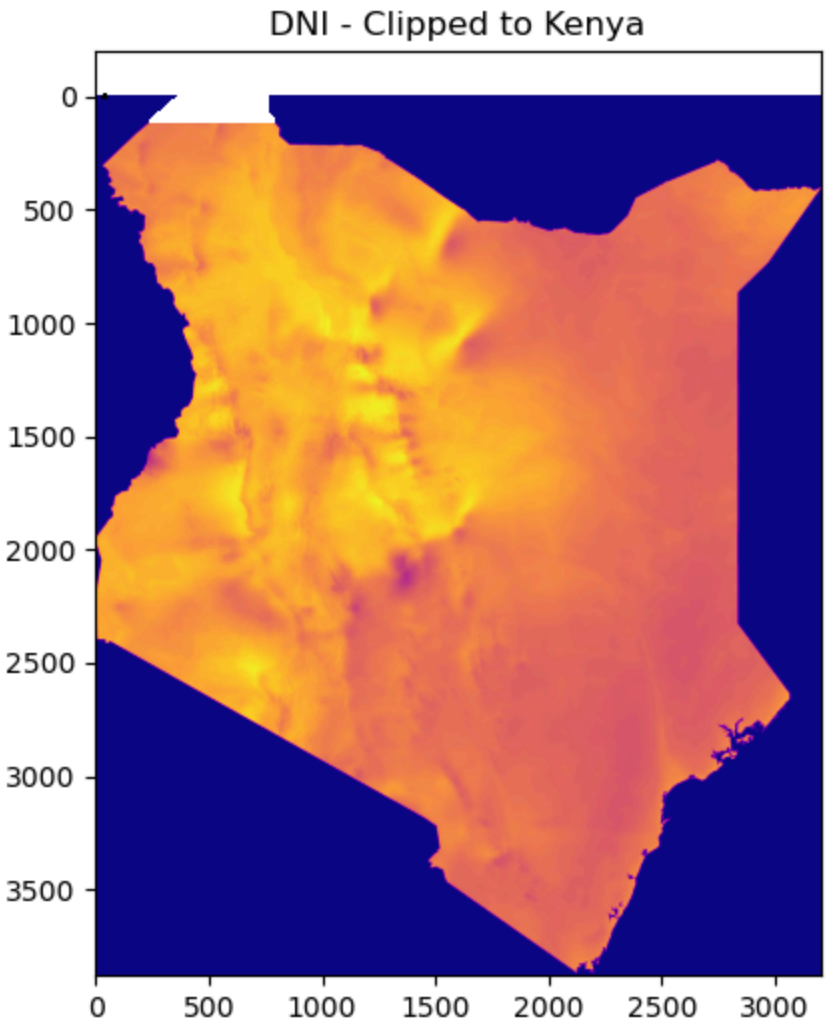
```
    # Reproject to Kenya CRS
    gdf = gdf.to_crs(kenya_boundary.crs)

    # Clip to Kenya boundary
    gdf = gpd.overlay(gdf, kenya_boundary, how="intersection")

    # Plot
    fig, ax = plt.subplots(figsize=(8, 8))
    kenya_boundary.boundary.plot(ax=ax, color="black", linewidth=1)
    gdf.plot(ax=ax, facecolor=facecolor, edgecolor=edgecolor)
    ax.set_title(title)
    plt.show()
```

In [9]:

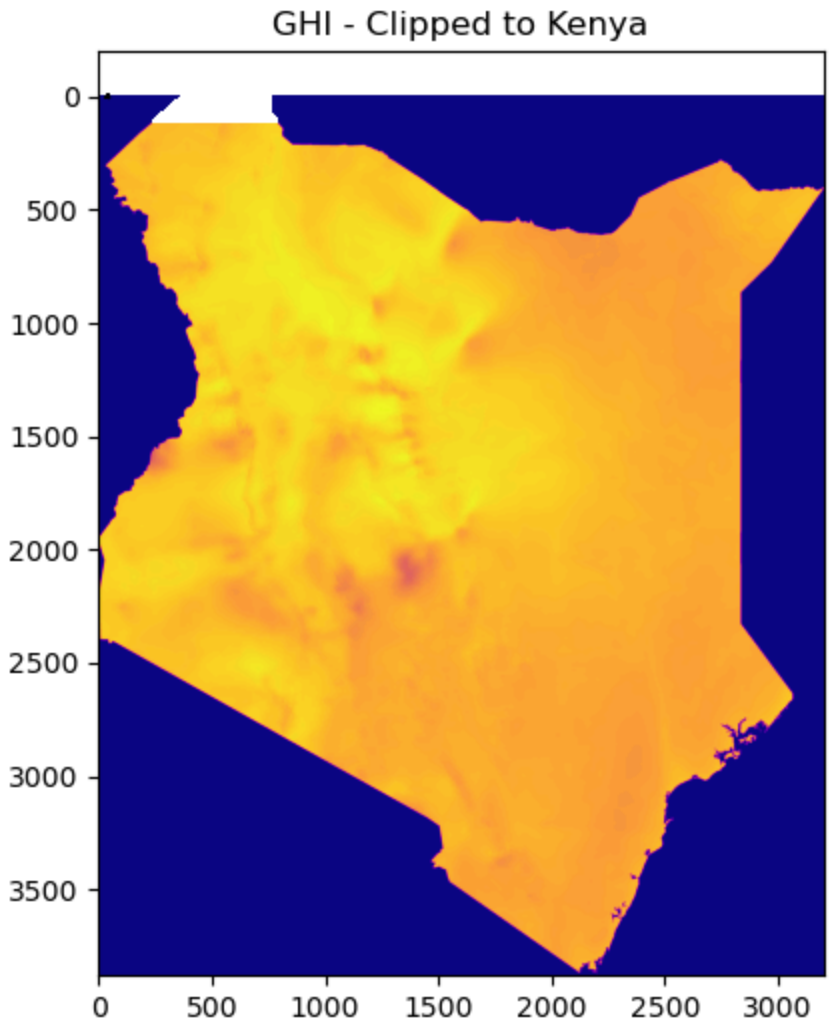
```
# --- Load Kenya boundary ---
kenya_boundary = gpd.read_file("kenya_Kenya_Country_Boundary.shp")
kenya_boundary = kenya_boundary.to_crs("EPSG:4326")   # base CRS

# --- List of rasters you want to clip ---
raster_files = {
    "DNI": "DNI.tif",
    "GHI": "GHI.tif",
    "GTI": "GTI.tif",
    "DIF": "DIF.tif",
    "PVOUT": "PVOUT.tif",
    "TEMP": "TEMP.tif"
}

clipped_rasters = {}

# --- Loop through rasters ---
for name, path in raster_files.items():
    with rasterio.open(path) as src:
        # match CRS
        kenya_boundary = kenya_boundary.to_crs(src.crs)

        # mask & crop
        out_image, out_transform = mask(src, kenya_boundary.geometry, crop=True)
        out_meta = src.meta.copy()

        # update metadata
        out_meta.update({
            "driver": "GTiff",
            "height": out_image.shape[1],
            "width": out_image.shape[2],
            "transform": out_transform
        })

        # save clipped raster
        out_path = f"{name}_KE.tif"
        with rasterio.open(out_path, "w", **out_meta) as dest:
            dest.write(out_image)

        clipped_rasters[name] = out_image
        print(f"{name} clipped and saved as {out_path}")

        # quick plot
```

```
        plt.figure(figsize=(8,6))
        plt.imshow(out_image[0], cmap="plasma")
        plt.grid(False)
        kenya_boundary.boundary.plot(ax=plt.gca(), color="black", linewidth=1)
        plt.title(f"{name} - Clipped to Kenya")
        plt.grid(False)     # removes gridlines
        plt.show()
```

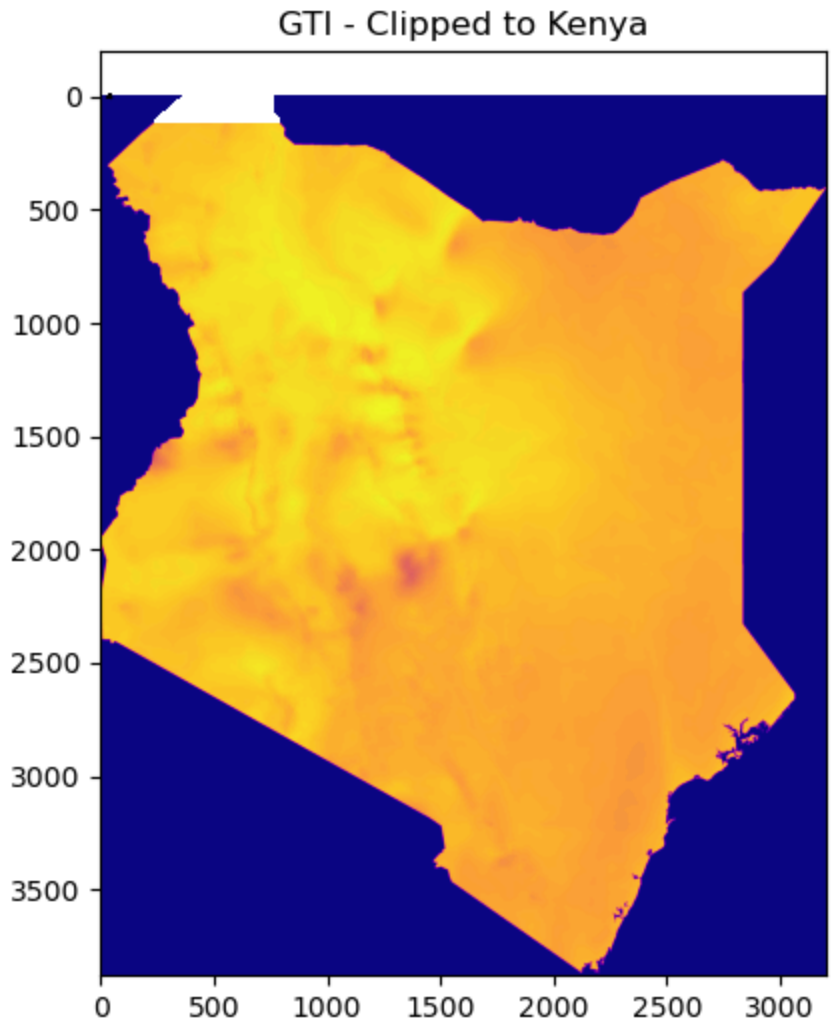
```
DNI clipped and saved as DNI_KE.tif
```



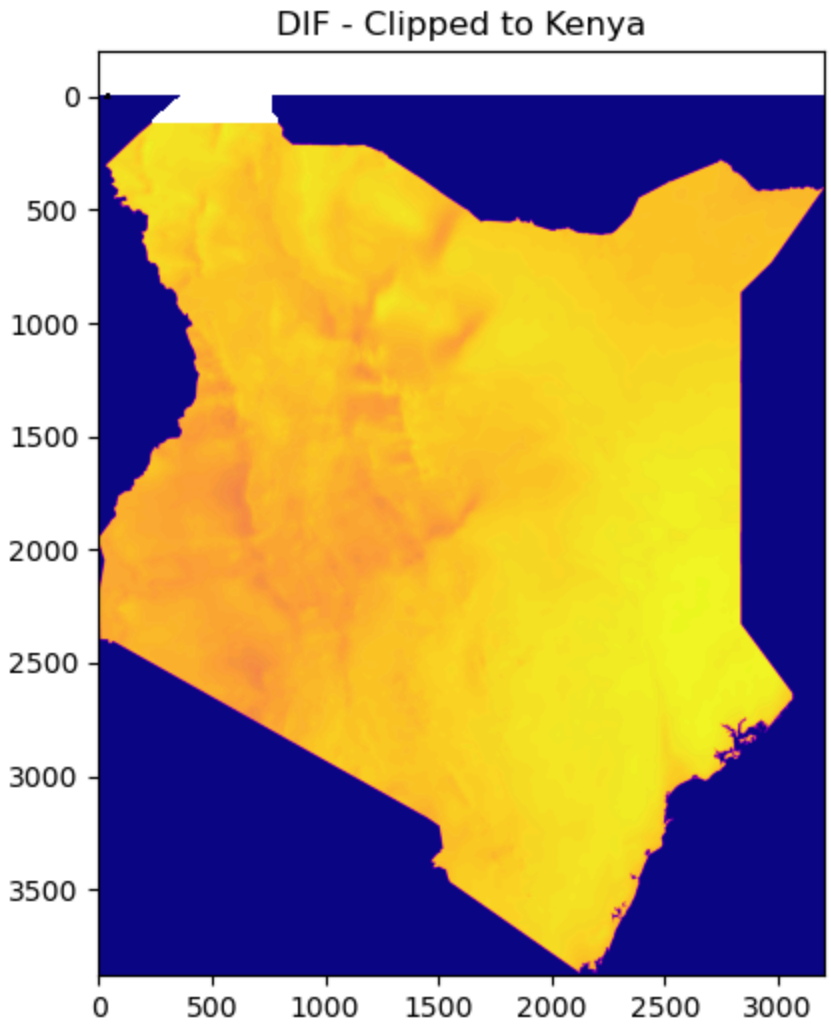

```
GHI clipped and saved as GHI_KE.tif
```



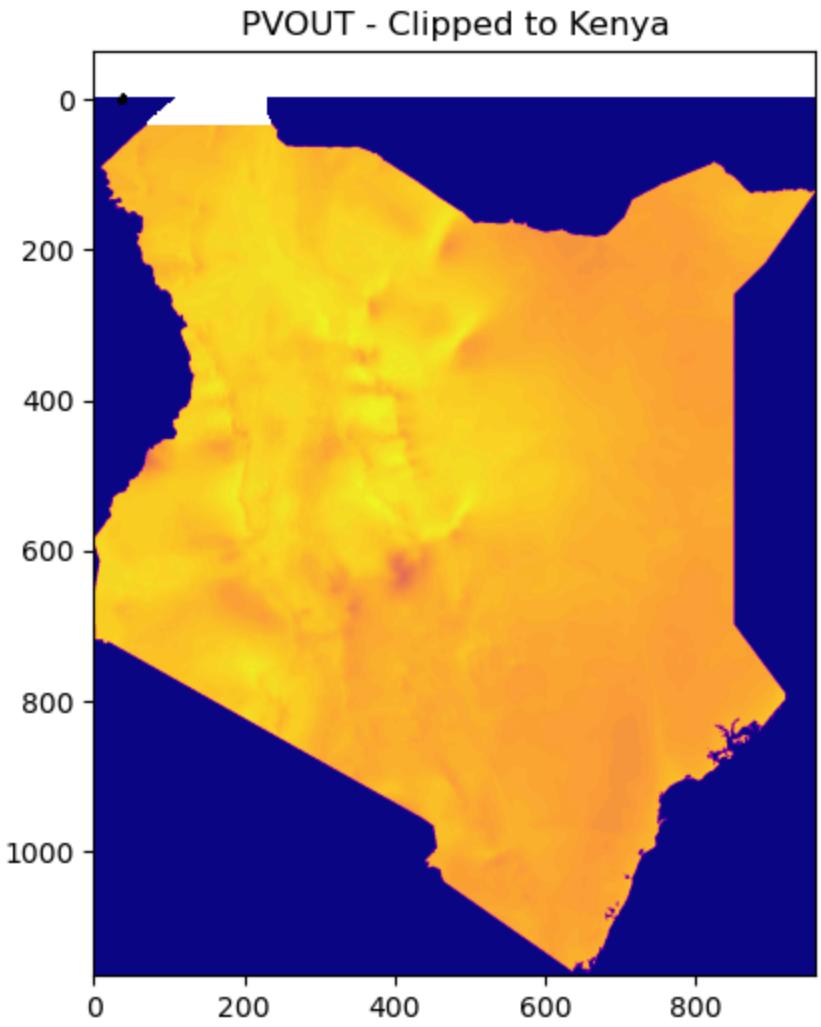

```
GTI clipped and saved as GTI_KE.tif
```



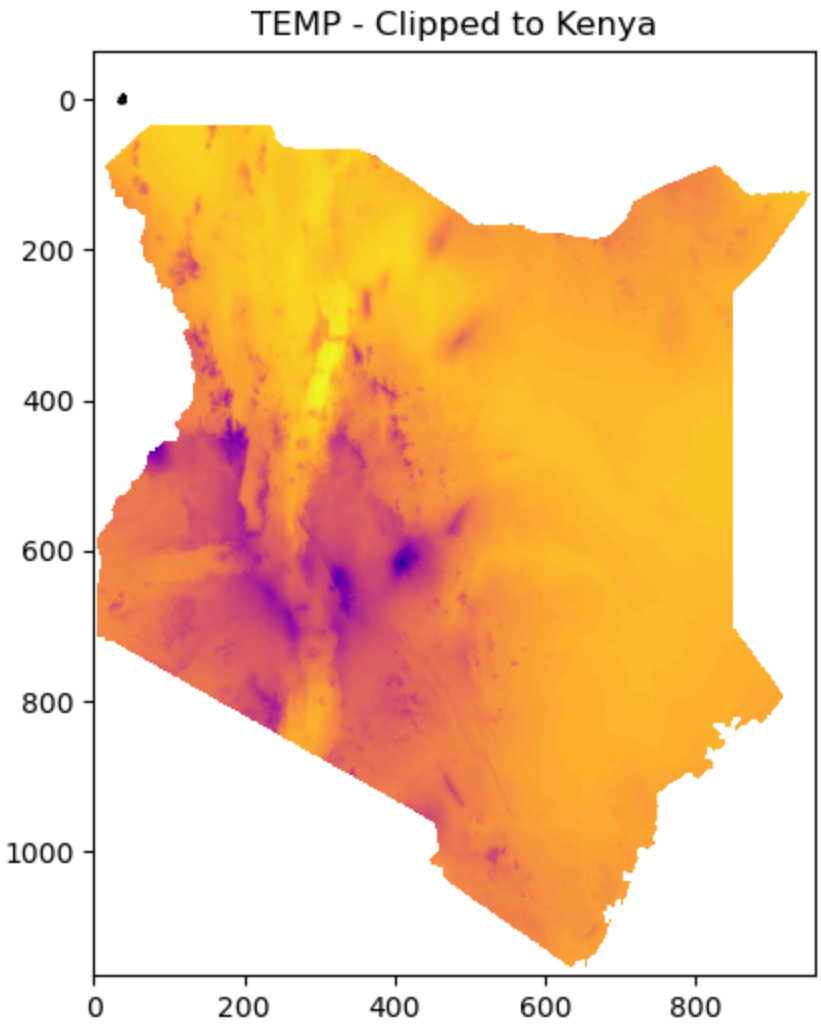

```
DIF clipped and saved as DIF_KE.tif
```



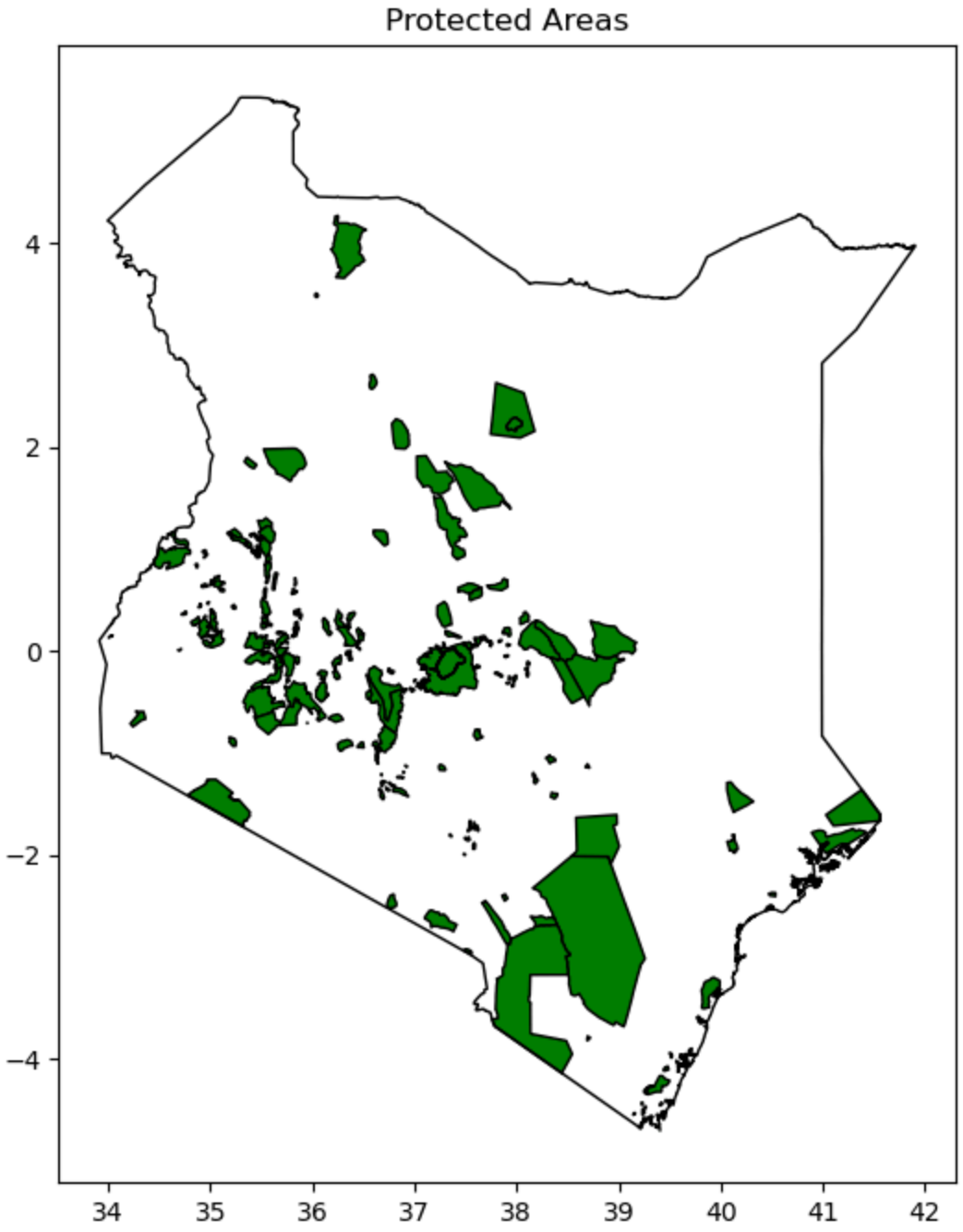

```
PVOUT clipped and saved as PVOUT_KE.tif
```

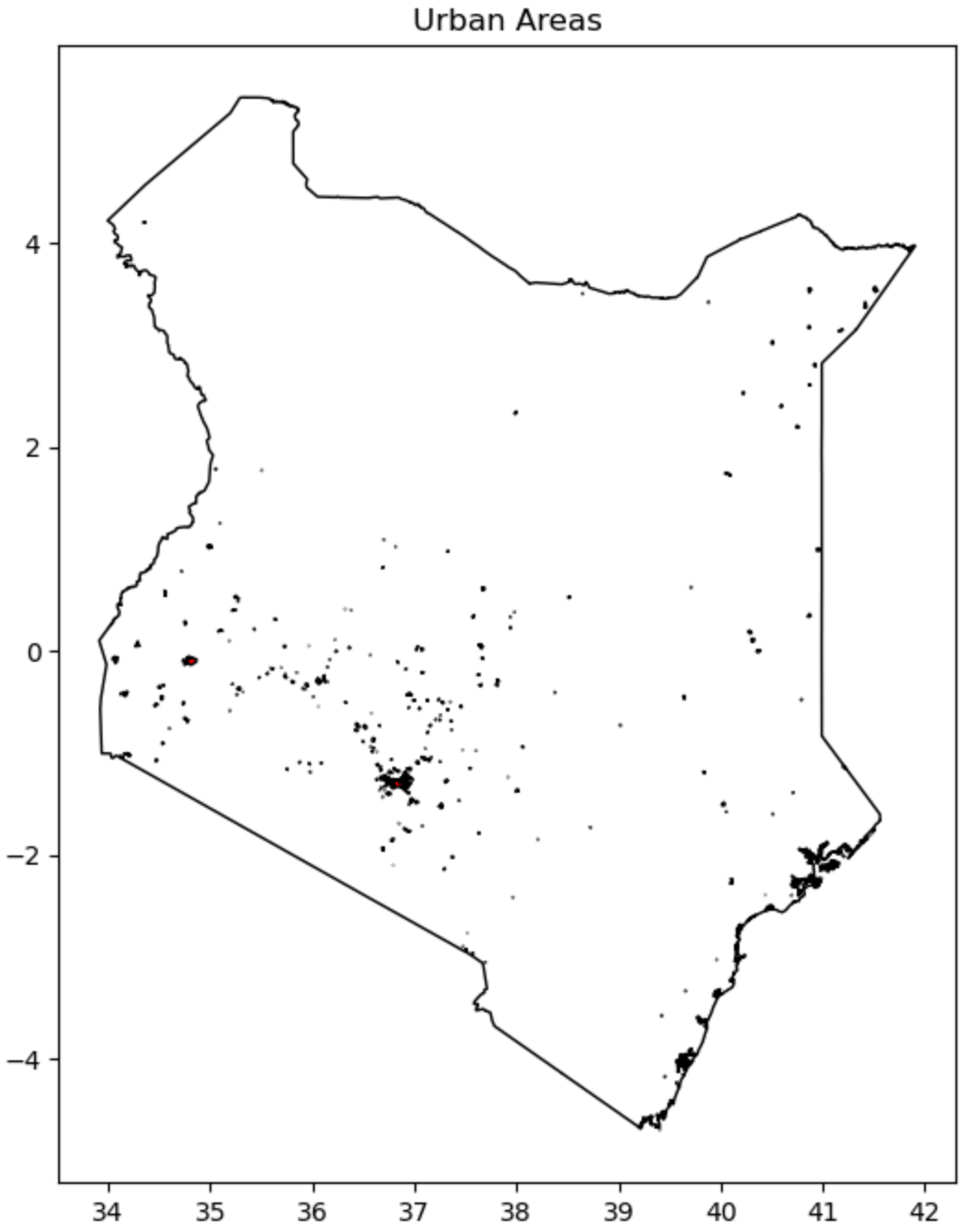


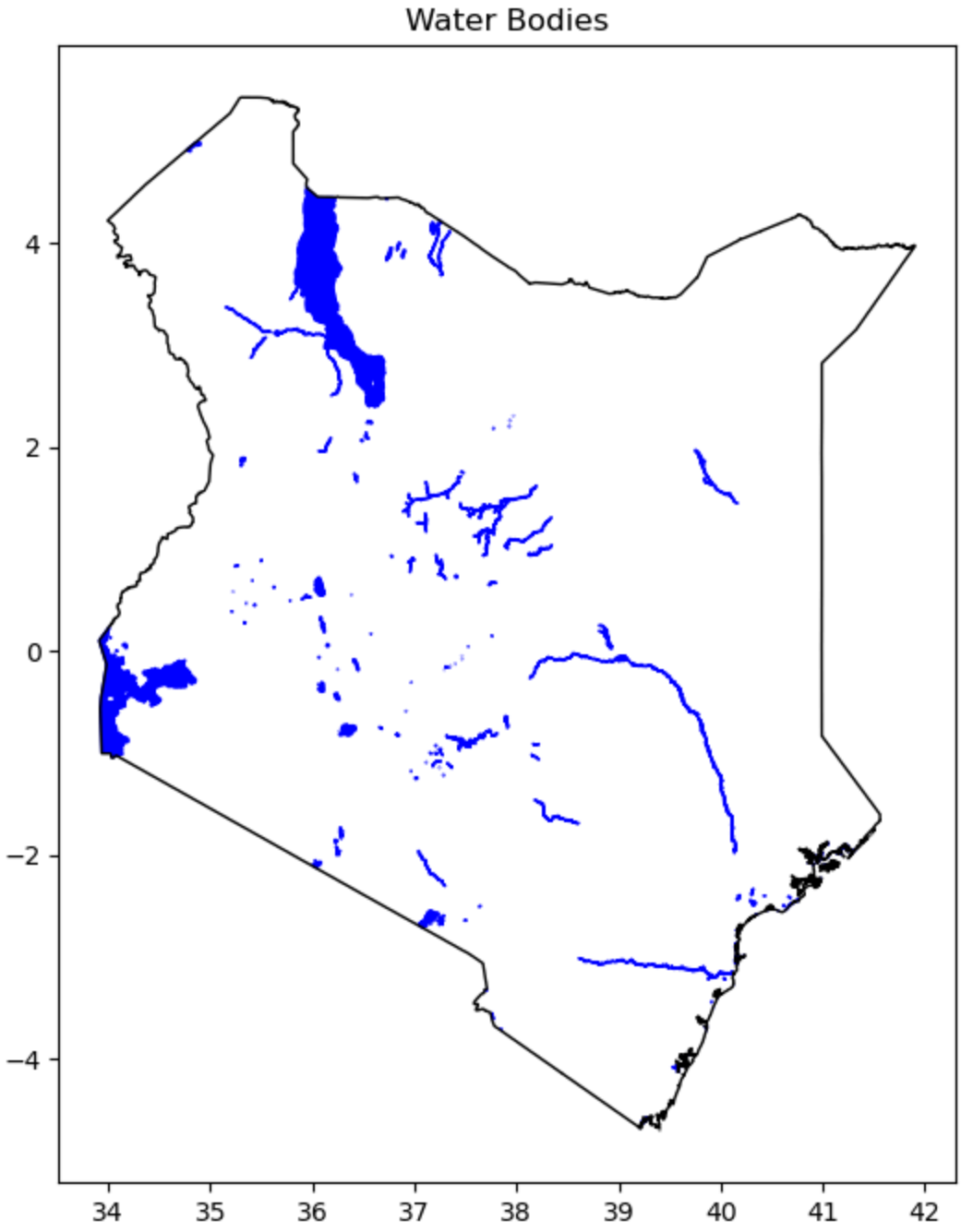

```
TEMP clipped and saved as TEMP_KE.tif
```

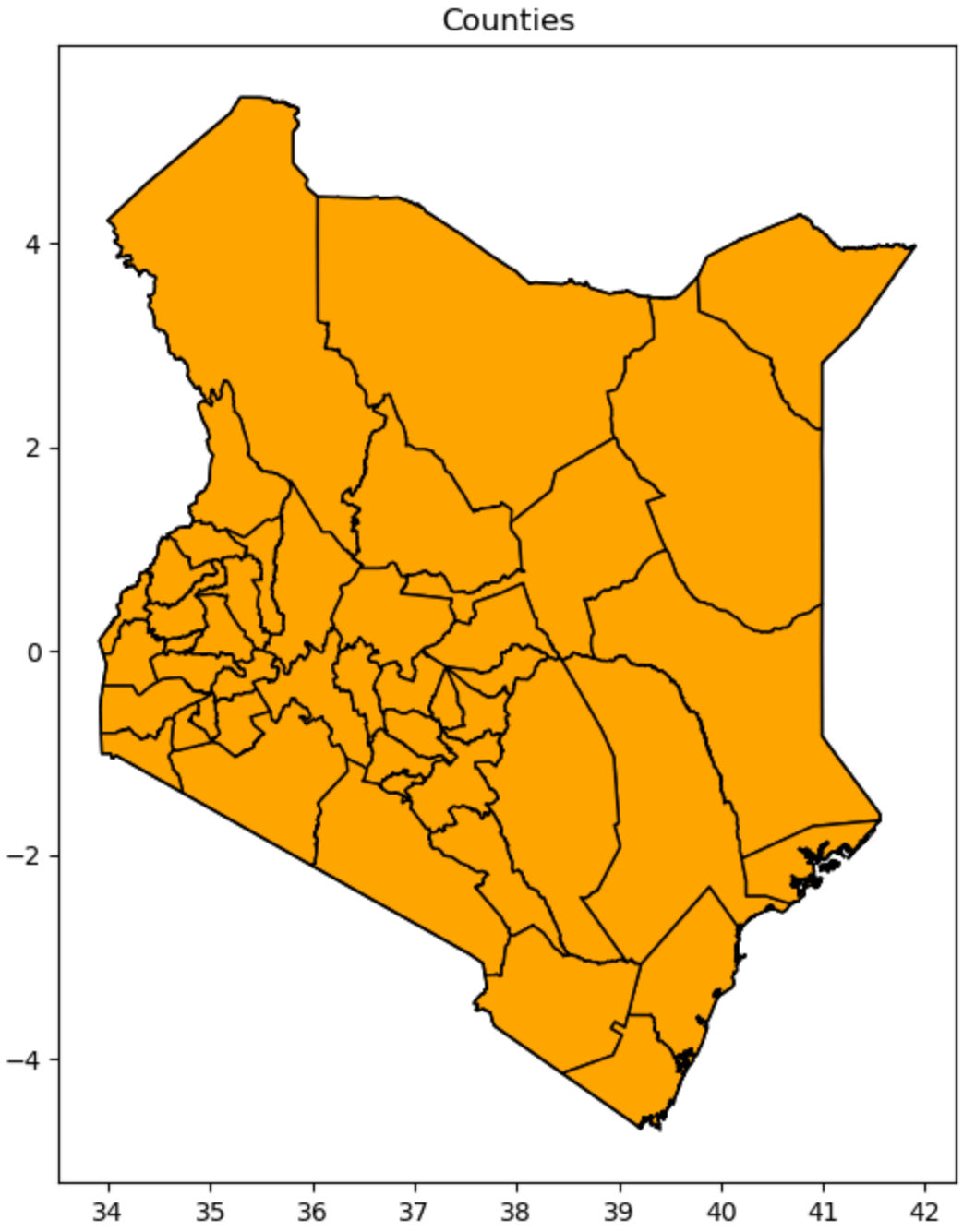


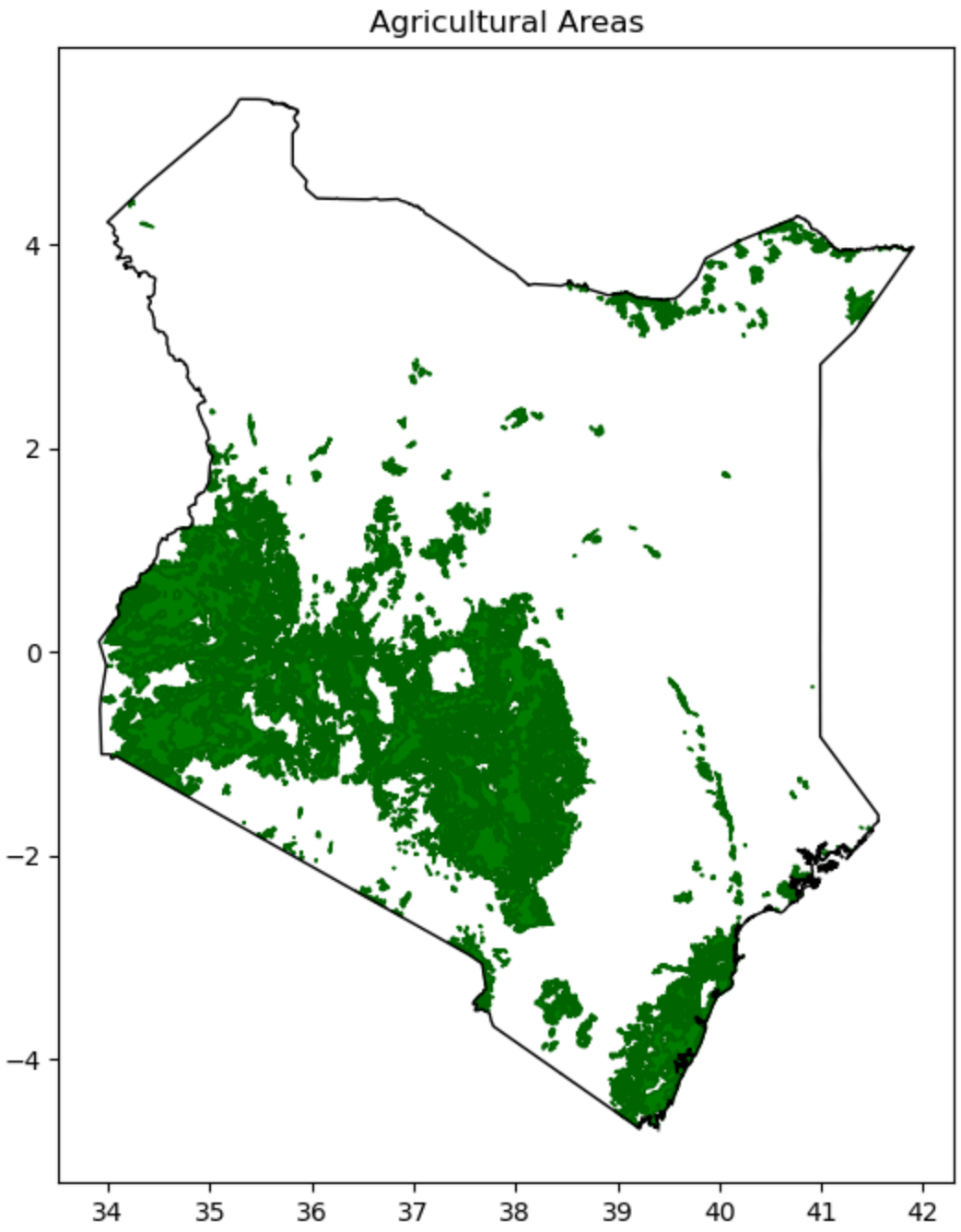
In [10]:
```
# For shapefile (Protected areas)
plot_vector("ke_protected-areas.shp", "Protected Areas", facecolor="green", edgecol

# For shapefile (Urban areas)
plot_vector("ke_urban.shp", "Urban Areas", facecolor="red", edgecolor="black")

#water bodies
plot_vector("ke_waterbodies.shp", "Water Bodies", facecolor="blue", edgecolor="blue

#counties
plot_vector("County.shp", "Counties", facecolor="orange", edgecolor="black")

#Agriculture
plot_vector("ke_agriculture.shp", "Agricultural Areas", facecolor="green", edgecolo
```


Protected Areas
4
2
0
−2
−4
34
35
36
37
38
39
40
41
42


Urban Areas
4
2
0
−2
−4
34
35
36
37
38
39
40
41
42


Water Bodies
4
2
0
−2
−4
34
35
36
37
38
39
40
41
42


Counties
4
2
0
−2
−4
34
35
36
37
38
39
40
41
42



Observation: DNI - Clipped to Kenya

- The Direct Normal Irradiance (DNI) map shows spatial variation in direct solar radiation across Kenya.
- Higher DNI values are visible in the northeastern and some interior regions, indicating strong direct beam solar potential — beneficial for concentrating solar technologies and high-tilt PV systems.
- Coastal and lake-adjacent areas show relatively lower DNI due to increased atmospheric moisture and cloudiness.
- Use this map to prioritize locations where DNI is consistently high, but cross-reference with land-use constraints (protected areas, water bodies, urban zones) to ensure feasible

site selection.

Observation: GHI - Clipped to Kenya

- The Global Horizontal Irradiance (GHI) map represents total solar radiation on a horizontal surface; values are generally high across most of Kenya, highlighting broad PV potential.
- Regions in central and northern Kenya typically exhibit higher GHI — good candidates for utility-scale PV.
- Lower GHI along the western lake region and coastal strip suggests local cloudiness and atmospheric effects.
- Follow-up: compare GHI with PVOUT to account for temperature and other system losses.

Observation: GTI - Clipped to Kenya

- Global Tilted Irradiance (GTI) estimates irradiance on an optimally tilted surface; it often shows slightly higher values than GHI in regions where tilt improves capture.
- GTI highlights areas where panel orientation would significantly increase yield — useful for site-level design.
- Use GTI together with slope/aspect from DEM to identify locations where natural terrain supports optimal tilt without excessive grading.

Observation: DIF - Clipped to Kenya

- Diffuse Horizontal Irradiance (DIF) measures scattered light; higher DIF indicates cloudier or more overcast regions where diffuse radiation contributes more to total irradiance.
- Coastal and lake-adjacent zones often show elevated DIF relative to interior arid regions.
- While DNI favors direct-beam concentrating systems, higher DIF areas still support PV but may favor flat-plate modules optimized for diffuse light.

Observation: PVOUT - Clipped to Kenya

- PVOUT maps modeled electricity output (kWh/kWp) and integrates irradiance and system performance factors; high PVOUT regions align with high GHI/GTI and moderate temperatures.
- Use PVOUT to prioritize sites for maximum energy yield per installed capacity.
- Cross-reference high PVOUT zones with land constraints (protected, urban, water) before site selection.

Observation: TEMP - Clipped to Kenya

- Ambient temperature affects PV efficiency negatively — higher temperatures reduce module performance.
- Temperature maps show warmer lowland and interior regions; while these areas may have strong irradiance, slightly reduced module efficiency should be considered when estimating yield.
- Consider cooling strategies, module selection (temperature coefficient), or slightly increased capacity in hotter zones to compensate for efficiency losses.

Observation: Protected Areas

- The protected areas map identifies conservation zones and legally protected lands where utility-scale solar development should be avoided.
- These zones must be excluded from suitability calculations or assigned zero suitability.
- For planning, buffer protected areas to avoid edge impacts and account for ecological corridors.

Observation: Urban Areas

- Urban maps show settlements and built-up areas; these are generally unsuitable for large utility-scale PV (but suitable for distributed/rooftop PV).
- Exclude dense urban zones from candidate areas for ground-mounted farms, or treat them separately for rooftop potential assessments.

Observation: Water Bodies

- Water bodies (lakes, rivers, wetlands) are unsuitable for PV installations and should be masked out of the suitability map.
- Note that nearshore wetlands and seasonal ponds may require additional local validation to avoid accidental inclusion.

Observation: Counties

- The county boundaries map helps link spatial suitability outputs to administrative units for policy and planning.
- Use county-level aggregation of suitability (e.g., area above a suitability threshold) to prioritize regions for investment and permitting.

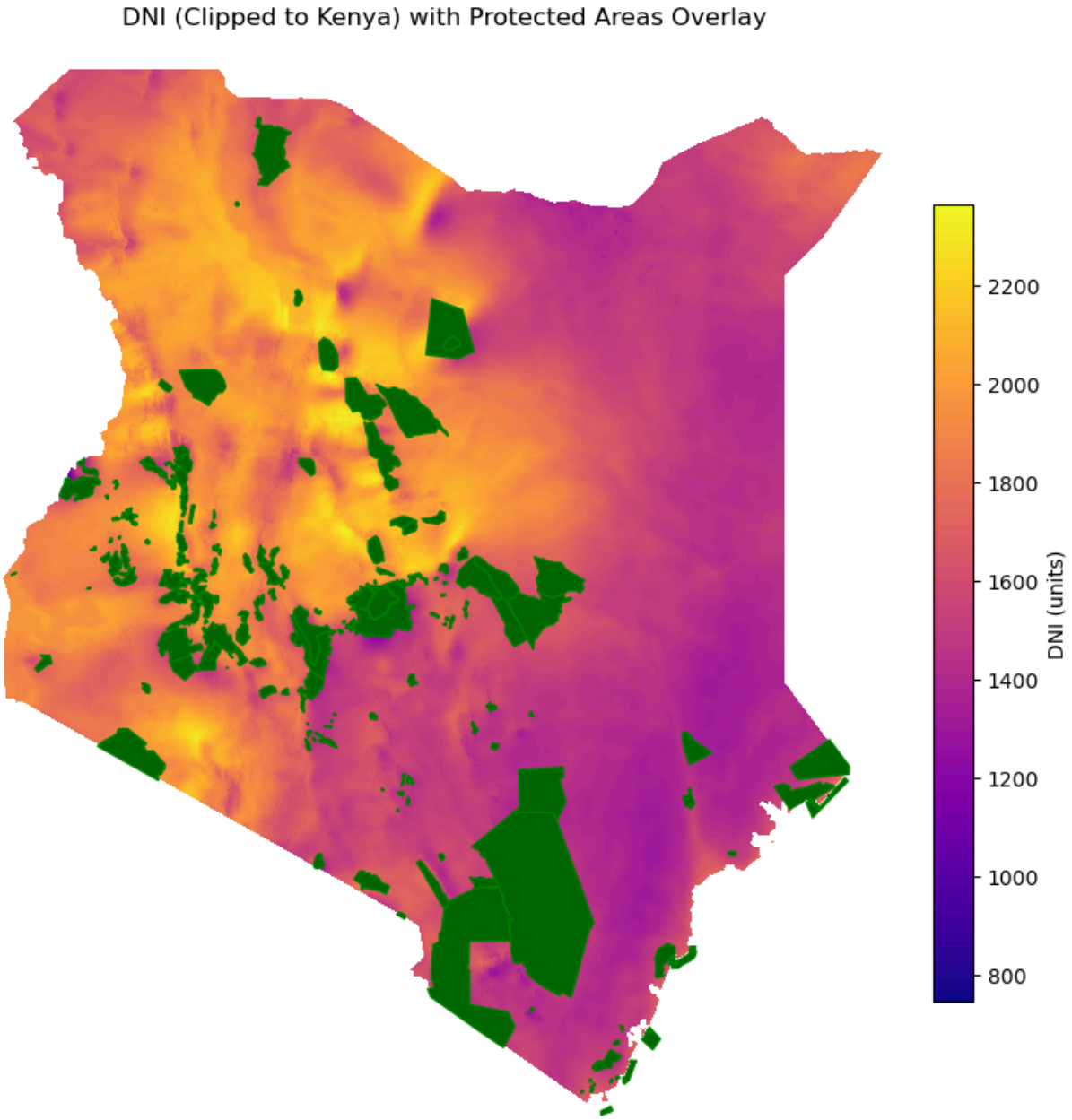
Observation: Agricultural Areas

- Agricultural land layers identify croplands that might be in competition with solar farms; these areas may be lower priority or require stakeholder engagement.
- Consider dual-use (agrivoltaics) where appropriate, but validate site-level suitability and local land tenure.

Spatial Relationship Analysis

In [11]:
```
# Plot DNI raster with Protected Areas overlaid (with colorbar)


# Paths
dni_path = 'DNI_KE.tif'
protected_path = 'ke_protected-areas.shp'
out_png = 'DNI_with_Protected_Areas.png'

# Read protected areas
protected = gpd.read_file(protected_path)

# Open DNI raster and plot with overlay using imshow for colorbar control
with rasterio.open(dni_path) as src:
    arr = src.read(1)
    # mask nodata values for plotting
    arr = np.where(arr == src.nodata, np.nan, arr)
    extent = (src.bounds.left, src.bounds.right, src.bounds.bottom, src.bounds.top)

    fig, ax = plt.subplots(1, 1, figsize=(10, 10))
    im = ax.imshow(arr, cmap='plasma', extent=extent, origin='upper')
    cbar = fig.colorbar(im, ax=ax, fraction=0.035, pad=0.04)
    cbar.set_label('DNI (units)')

    # ensure protected areas are in same CRS as raster
    try:
        if protected.crs != src.crs:
            protected = protected.to_crs(src.crs)
    except Exception:
        protected = protected.set_crs(epsg=4326, allow_override=True).to_crs(src.cr

    # plot protected areas as outlines so the raster remains visible
    protected.plot(ax=ax, facecolor='darkgreen', edgecolor='green', linewidth=1, la

    # cosmetics
    ax.set_title('DNI (Clipped to Kenya) with Protected Areas Overlay')
    ax.axis('off')
    handles, labels = ax.get_legend_handles_labels()
    if handles:
        ax.legend()

    # save figure
    fig.savefig(out_png, dpi=300, bbox_inches='tight')
    print(f'Saved map to {out_png}')
    plt.show()
```

```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\690459459.py:37: UserWarning: Legend d
oes not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
Saved map to DNI_with_Protected_Areas.png
```



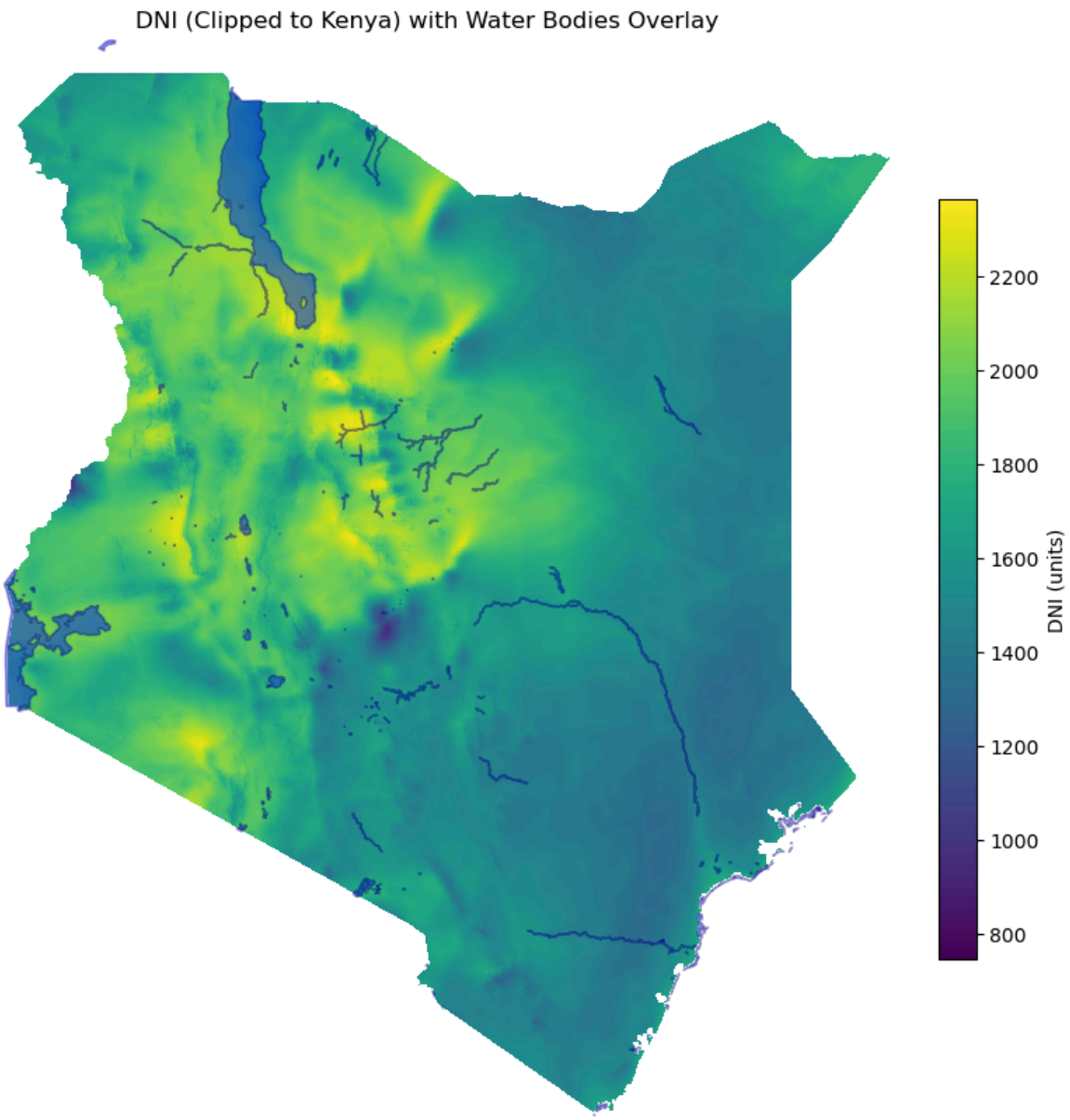
Observations: DNI with Protected Areas

- Protected areas overlap some regions with moderate-to-high DNI; these areas should be excluded from large-scale PV development to avoid ecological impacts.
- Note: Important conservation zones (e.g., national parks and reserves) may remove significant contiguous land from availability, fragmenting otherwise promising high-DNI regions.

- Recommendation: Use this overlay to mask protected zones from further suitability scoring and consider buffer zones to reduce edge impacts.

In [12]:
```
# Plot DNI raster with Water Bodies overlaid (with colorbar and save)

dni_path = 'DNI_KE.tif'
water_path = 'ke_waterbodies.shp'
out_png = 'DNI_with_Water_Bodies.png'

water = gpd.read_file(water_path)
with rasterio.open(dni_path) as src:
    arr = src.read(1)
    arr = np.where(arr == src.nodata, np.nan, arr)
    extent = (src.bounds.left, src.bounds.right, src.bounds.bottom, src.bounds.top)
    fig, ax = plt.subplots(1,1,figsize=(10,10))
    im = ax.imshow(arr, cmap='viridis', extent=extent, origin='upper')
    cbar = fig.colorbar(im, ax=ax, fraction=0.035, pad=0.04)
    cbar.set_label('DNI (units)')
    try:
        if water.crs != src.crs:
            water = water.to_crs(src.crs)
    except Exception:
        water = water.set_crs(epsg=4326, allow_override=True).to_crs(src.crs)
    water.plot(ax=ax, facecolor='blue', edgecolor='navy', alpha=0.4, label='Water B
    ax.set_title('DNI (Clipped to Kenya) with Water Bodies Overlay')
    ax.axis('off')
    handles, labels = ax.get_legend_handles_labels()
    if handles:
        ax.legend()
    fig.savefig(out_png, dpi=300, bbox_inches='tight')
    print(f'Saved map to {out_png}')
    plt.show()
```

```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\3698423876.py:24: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
Saved map to DNI_with_Water_Bodies.png
```



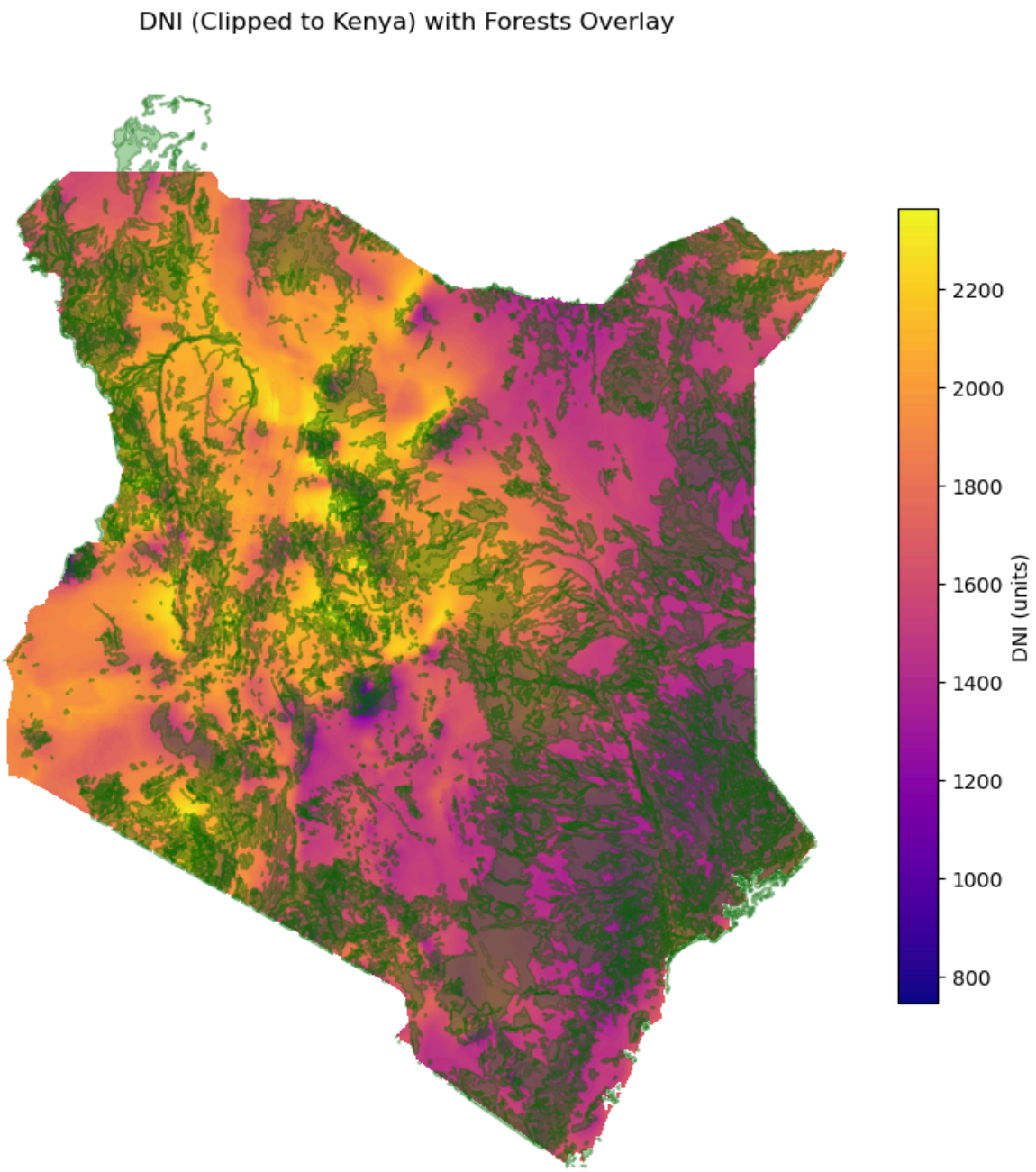
Observations: DNI with Water Bodies

- Water bodies (lakes, rivers, wetlands) appear in areas with lower DNI along shorelines; these are unsuitable for ground-mounted PV and should be masked out.
- Large lakes and river corridors can create local microclimates (increased humidity and cloudiness) that reduce direct irradiance near shorelines.
- Recommendation: Exclude water bodies from candidate areas and add a small buffer to account for wetland transitions and seasonal fluctuations.

In [13]:
```
# Plot DNI raster with Forests overlaid (with colorbar and save)


dni_path = 'DNI_KE.tif'
forest_path = 'ke_forests.shp'
out_png = 'DNI_with_Forests.png'
```

```
forest = gpd.read_file(forest_path)
with rasterio.open(dni_path) as src:
    arr = src.read(1)
    arr = np.where(arr == src.nodata, np.nan, arr)
    extent = (src.bounds.left, src.bounds.right, src.bounds.bottom, src.bounds.top)
    fig, ax = plt.subplots(1,1,figsize=(10,10))
    im = ax.imshow(arr, cmap='plasma', extent=extent, origin='upper')
    cbar = fig.colorbar(im, ax=ax, fraction=0.035, pad=0.04)
    cbar.set_label('DNI (units)')
    try:
        if forest.crs != src.crs:
            forest = forest.to_crs(src.crs)
    except Exception:
        forest = forest.set_crs(epsg=4326, allow_override=True).to_crs(src.crs)
    forest.plot(ax=ax, facecolor='green', edgecolor='darkgreen', alpha=0.35, label=
    ax.set_title('DNI (Clipped to Kenya) with Forests Overlay')
    ax.axis('off')
    handles, labels = ax.get_legend_handles_labels()
    if handles:
        ax.legend()
    fig.savefig(out_png, dpi=300, bbox_inches='tight')
    print(f'Saved map to {out_png}')
    plt.show()
```

```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\2180843545.py:25: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
Saved map to DNI_with_Forests.png
```



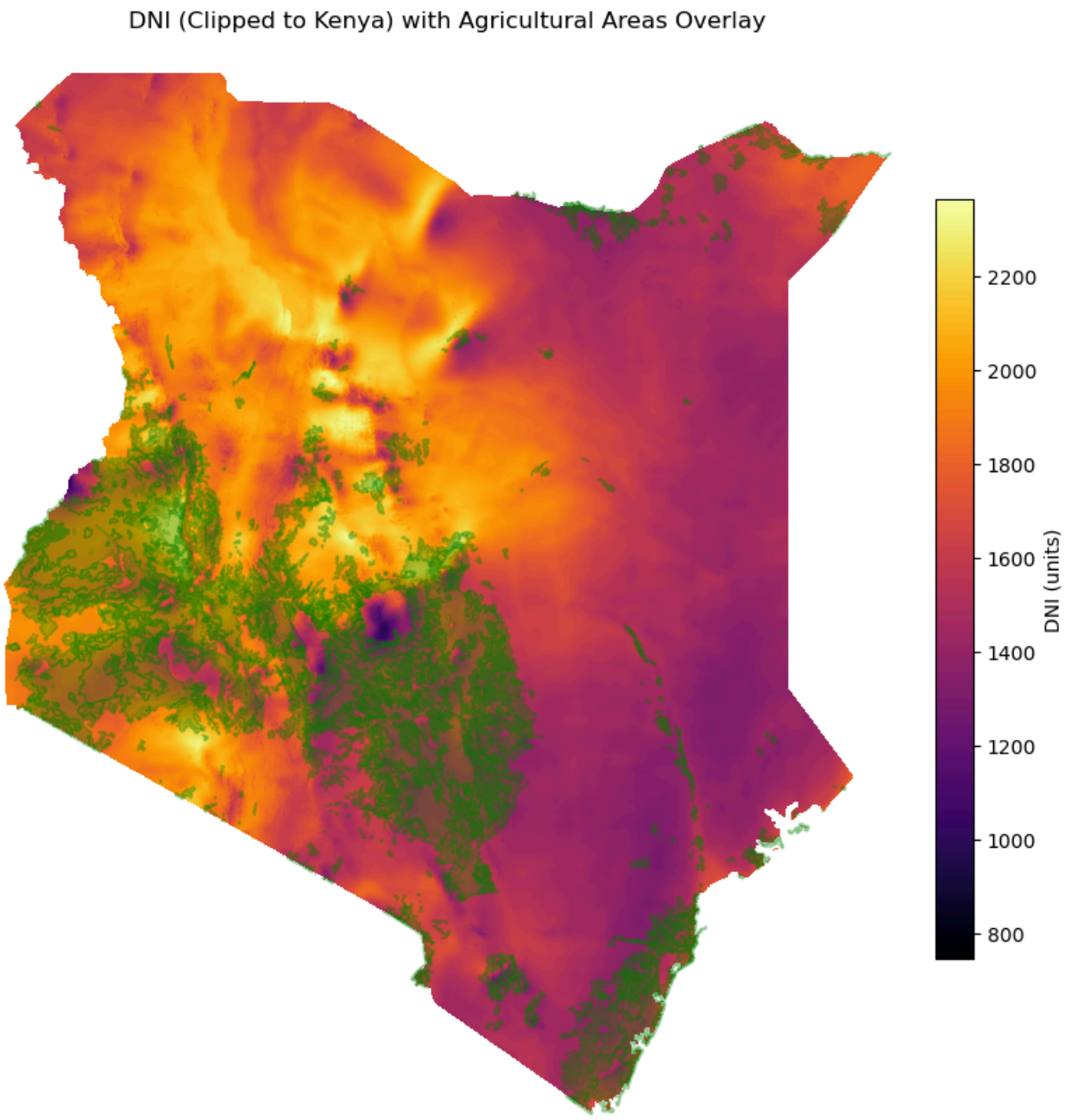
Observations: DNI with Forests

- Forested areas usually coincide with lower surface irradiance at canopy level; these lands are often environmentally sensitive and should be excluded from large-scale PV.
- Forest cover can also indicate terrain or land-use constraints (steep slopes, conservation), reducing practical suitability even where DNI is moderate.
- Recommendation: Mask dense forest areas from suitability maps and assess smaller degraded or open-woodland patches separately if needed.

In [14]:
```
# Plot DNI raster with Agricultural Areas overlaid (with colorbar and save)

dni_path = 'DNI_KE.tif'
```

```
agri_path = 'ke_agriculture.shp'
out_png = 'DNI_with_Agricultural_Areas.png'

agri = gpd.read_file(agri_path)
with rasterio.open(dni_path) as src:
    arr = src.read(1)
    arr = np.where(arr == src.nodata, np.nan, arr)
    extent = (src.bounds.left, src.bounds.right, src.bounds.bottom, src.bounds.top)
    fig, ax = plt.subplots(1,1,figsize=(10,10))
    im = ax.imshow(arr, cmap='inferno', extent=extent, origin='upper')
    cbar = fig.colorbar(im, ax=ax, fraction=0.035, pad=0.04)
    cbar.set_label('DNI (units)')
    try:
        if agri.crs != src.crs:
            agri = agri.to_crs(src.crs)
    except Exception:
        agri = agri.set_crs(epsg=4326, allow_override=True).to_crs(src.crs)
    agri.plot(ax=ax, facecolor='green', edgecolor='green', alpha=0.3, label='Agricu
    ax.set_title('DNI (Clipped to Kenya) with Agricultural Areas Overlay')
    ax.axis('off')
    handles, labels = ax.get_legend_handles_labels()
    if handles:
        ax.legend()
    fig.savefig(out_png, dpi=300, bbox_inches='tight')
    print(f'Saved map to {out_png}')
    plt.show()
```

```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\1764791529.py:25: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
Saved map to DNI_with_Agricultural_Areas.png
```



Observations: DNI with Agricultural Areas

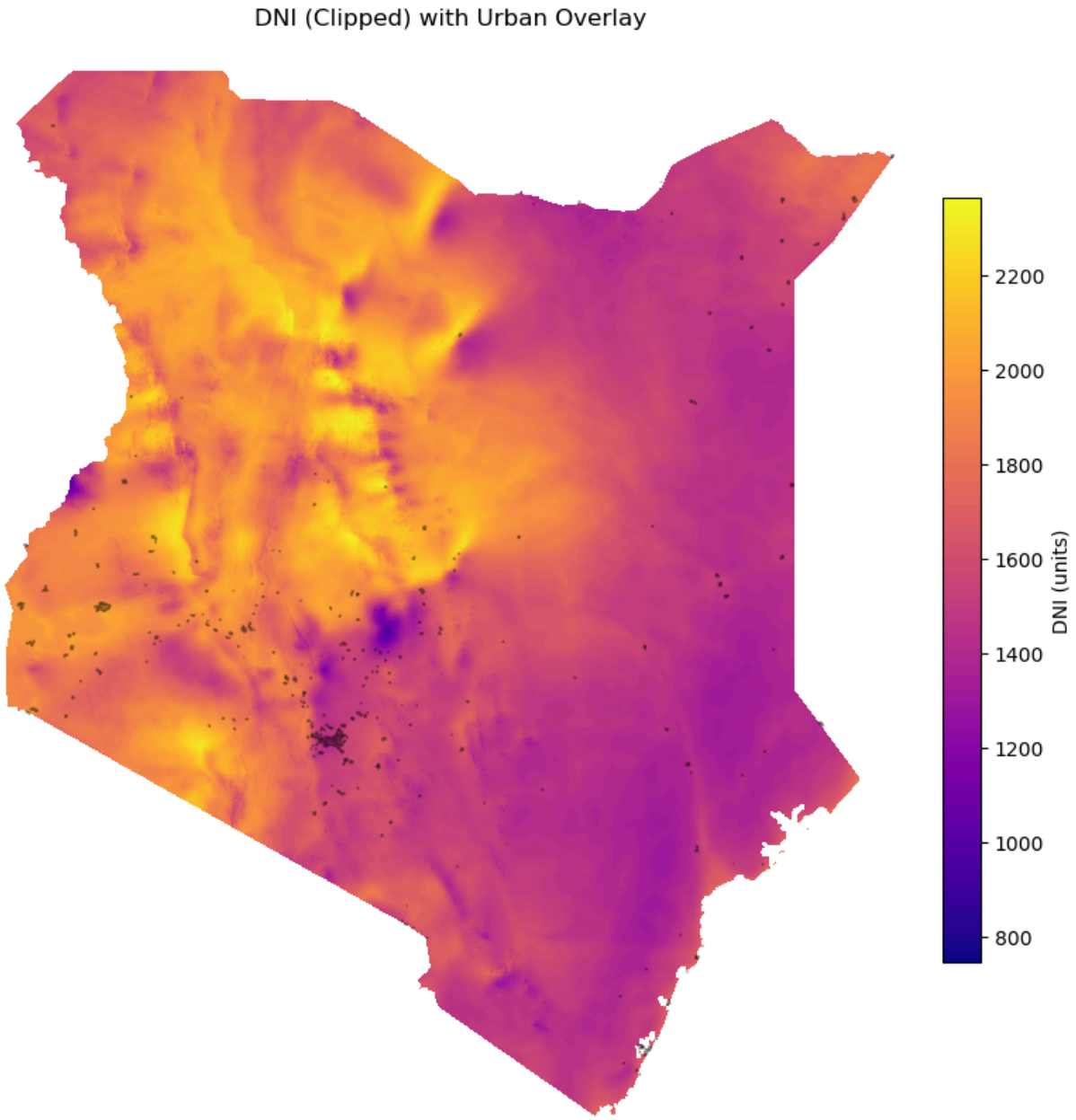
- Agricultural areas often overlap with regions of moderate-to-high DNI; these lands may present competing uses for utility-scale PV.
- Opportunities exist for agrivoltaic systems where dual land use is acceptable, but this requires local stakeholder engagement and an assessment of crop compatibility.
- Recommendation: Flag agricultural zones for social/environmental review — consider alternate sites or agrivoltaics where appropriate.

In [15]:
```
# Batch plot: DNI, GHI, GTI, PVOUT over Urban Areas and County boundaries


rasters = {
    'DNI': 'DNI_KE.tif',
    'GHI': 'GHI_KE.tif',
```

```
    'GTI': 'GTI_KE.tif',
    'PVOUT': 'PVOUT_KE.tif'
}
overlays = {
    'Urban': 'ke_urban.shp',
    'Counties': 'County.shp'
}
# Read overlay layers once
overlay_gdfs = {}
for name, path in overlays.items():
    try:
        overlay_gdfs[name] = gpd.read_file(path)
    except Exception as e:
        print(f'Could not read {path}: {e}')
        overlay_gdfs[name] = None

for rname, rpath in rasters.items():
    try:
        with rasterio.open(rpath) as src:
            arr = src.read(1)
            arr = np.where(arr == src.nodata, np.nan, arr)
            extent = (src.bounds.left, src.bounds.right, src.bounds.bottom, src.bou
            for oname, gdf in overlay_gdfs.items():
                if gdf is None:
                    print(f'Skipping overlay {oname} for {rname} (layer not loaded)
                    continue
                fig, ax = plt.subplots(1,1,figsize=(10,10))
                im = ax.imshow(arr, cmap='viridis' if rname=='GHI' else ('plasma' i
                cbar = fig.colorbar(im, ax=ax, fraction=0.035, pad=0.04)
                cbar.set_label(f'{rname} (units)')
                try:
                    if gdf.crs != src.crs:
                        gdf_plot = gdf.to_crs(src.crs)
                    else:
                        gdf_plot = gdf
                except Exception:
                    gdf_plot = gdf.set_crs(epsg=4326, allow_override=True).to_crs(s
                # style choices: urban = red transparent, counties = orange outline
                if oname == 'Urban':
                    gdf_plot.plot(ax=ax, facecolor='black', edgecolor='black', alph
                else:
                    gdf_plot.plot(ax=ax, facecolor='none', edgecolor='black', linew
                ax.set_title(f'{rname} (Clipped) with {oname} Overlay')
                ax.axis('off')
                handles, labels = ax.get_legend_handles_labels()
                if handles:
                    ax.legend()
                out_png = f'{rname}_with_{oname}.png'
                fig.savefig(out_png, dpi=300, bbox_inches='tight')
                print(f'Saved {out_png}')
                plt.show()
    except Exception as e:
        print(f'Could not open raster {rpath}: {e}')

print('Batch plotting complete.')
```

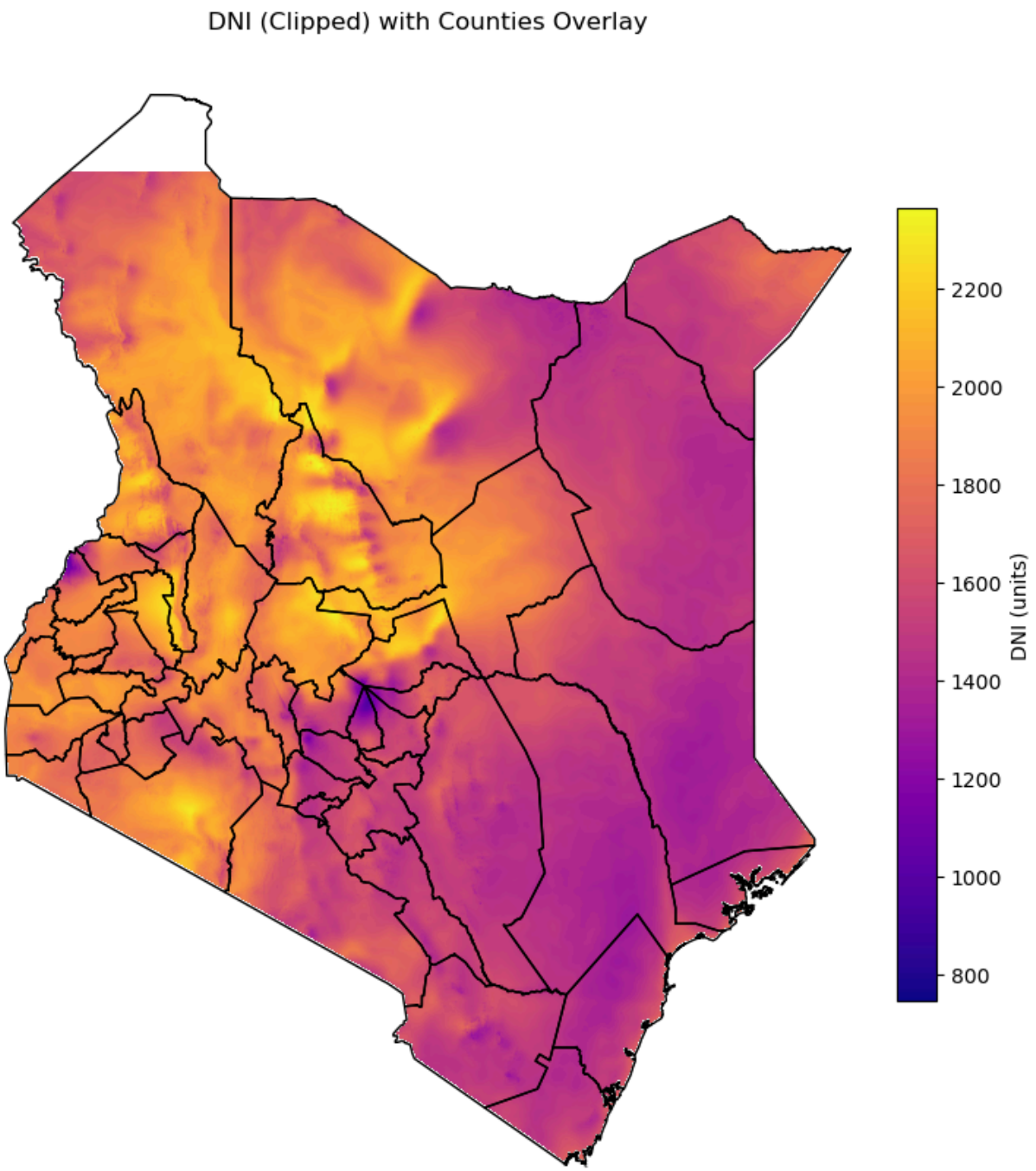
```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\1993060275.py:51: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
```

```
Saved DNI_with_Urban.png
```

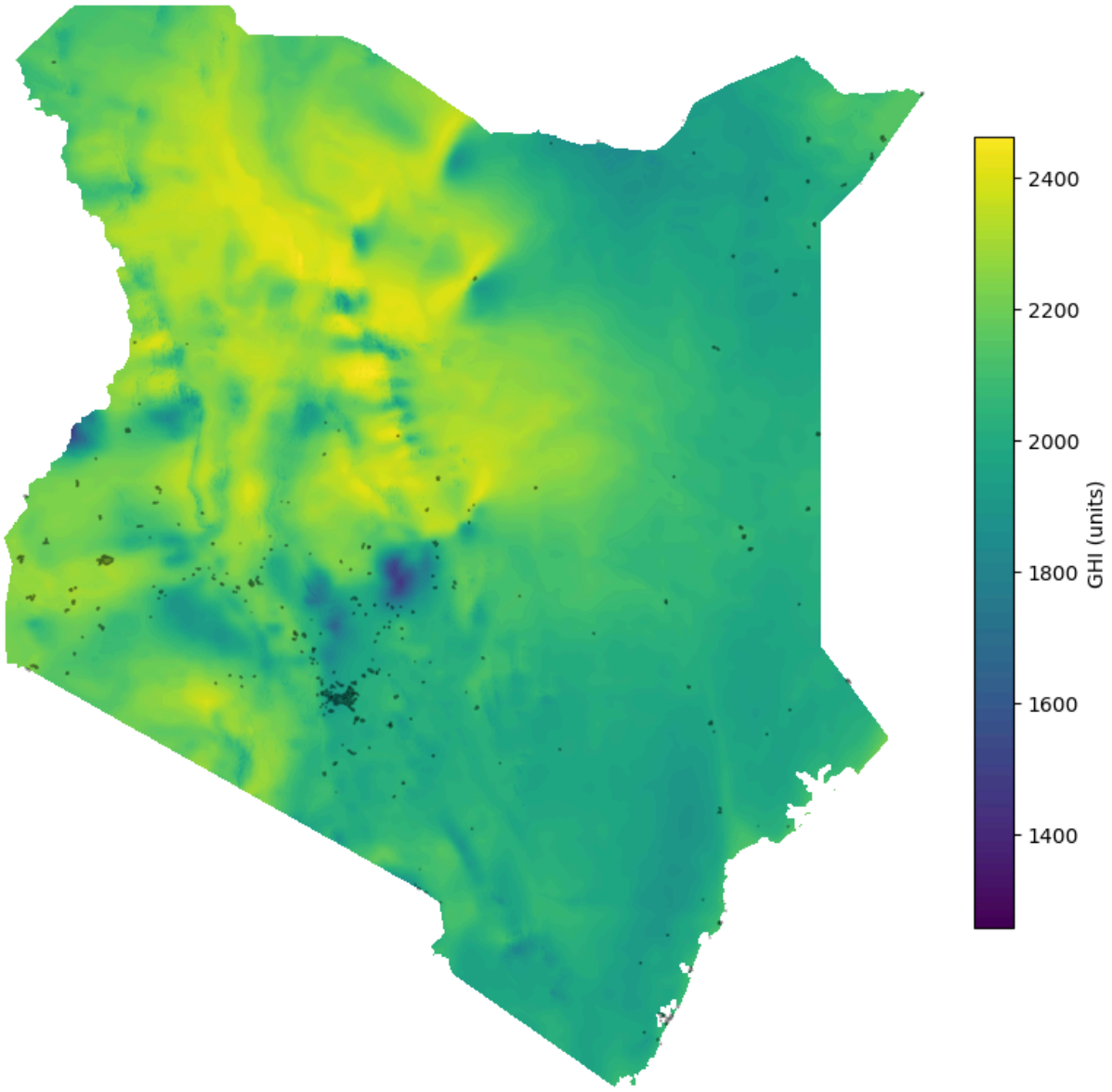



```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\1993060275.py:51: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
```

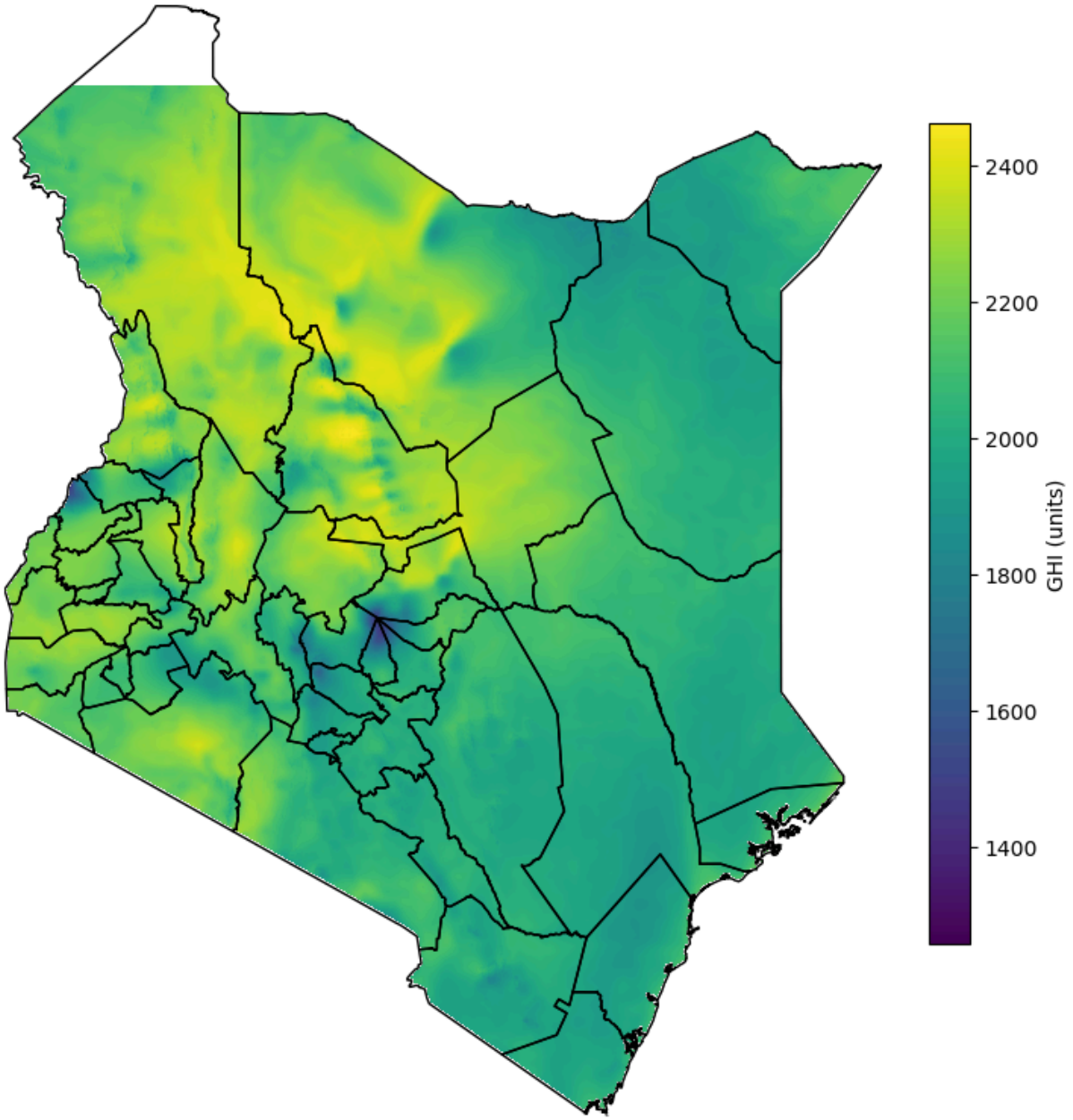
```
Saved DNI_with_Counties.png
```



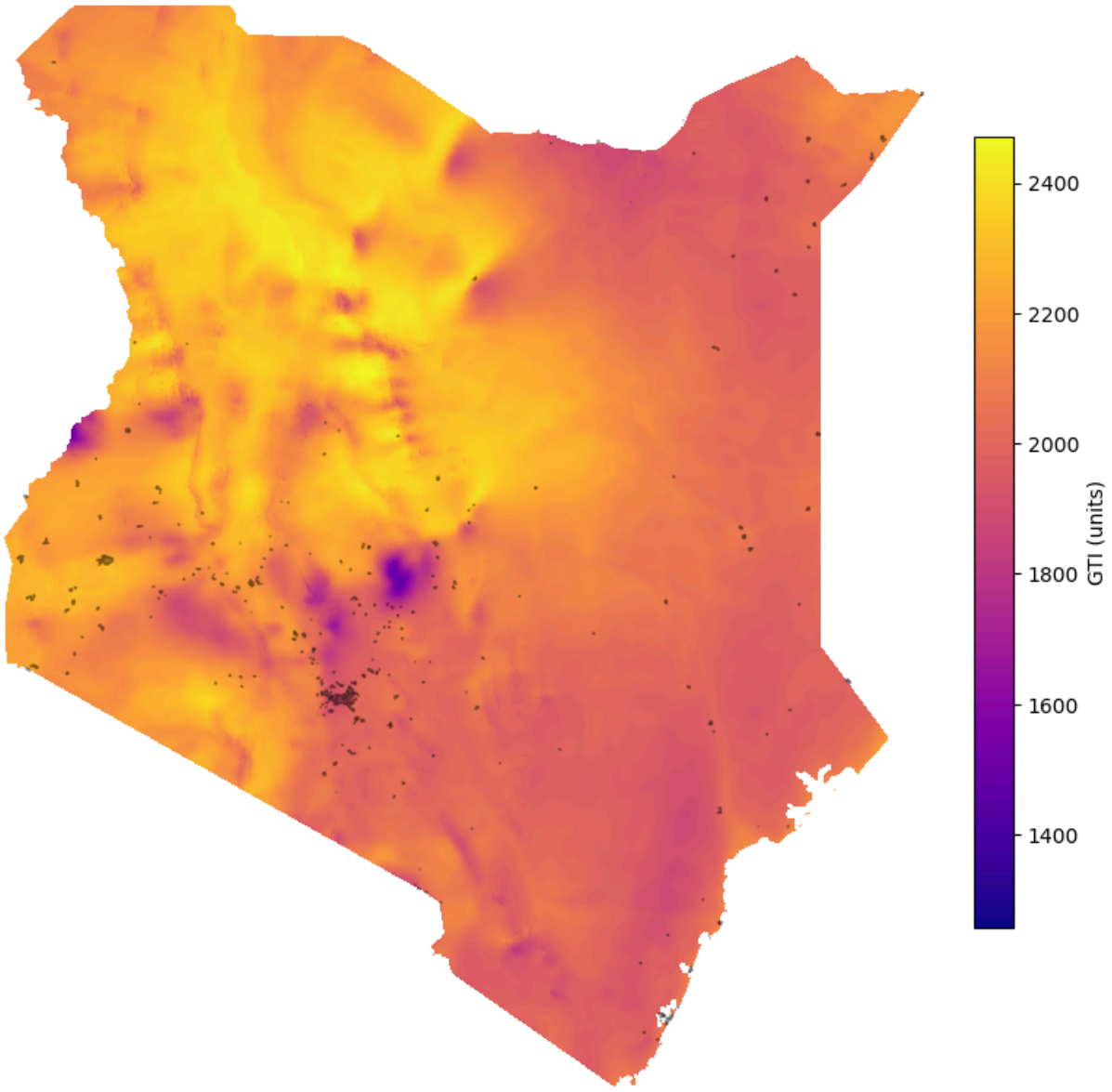

```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\1993060275.py:51: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
Saved GHI_with_Urban.png
```



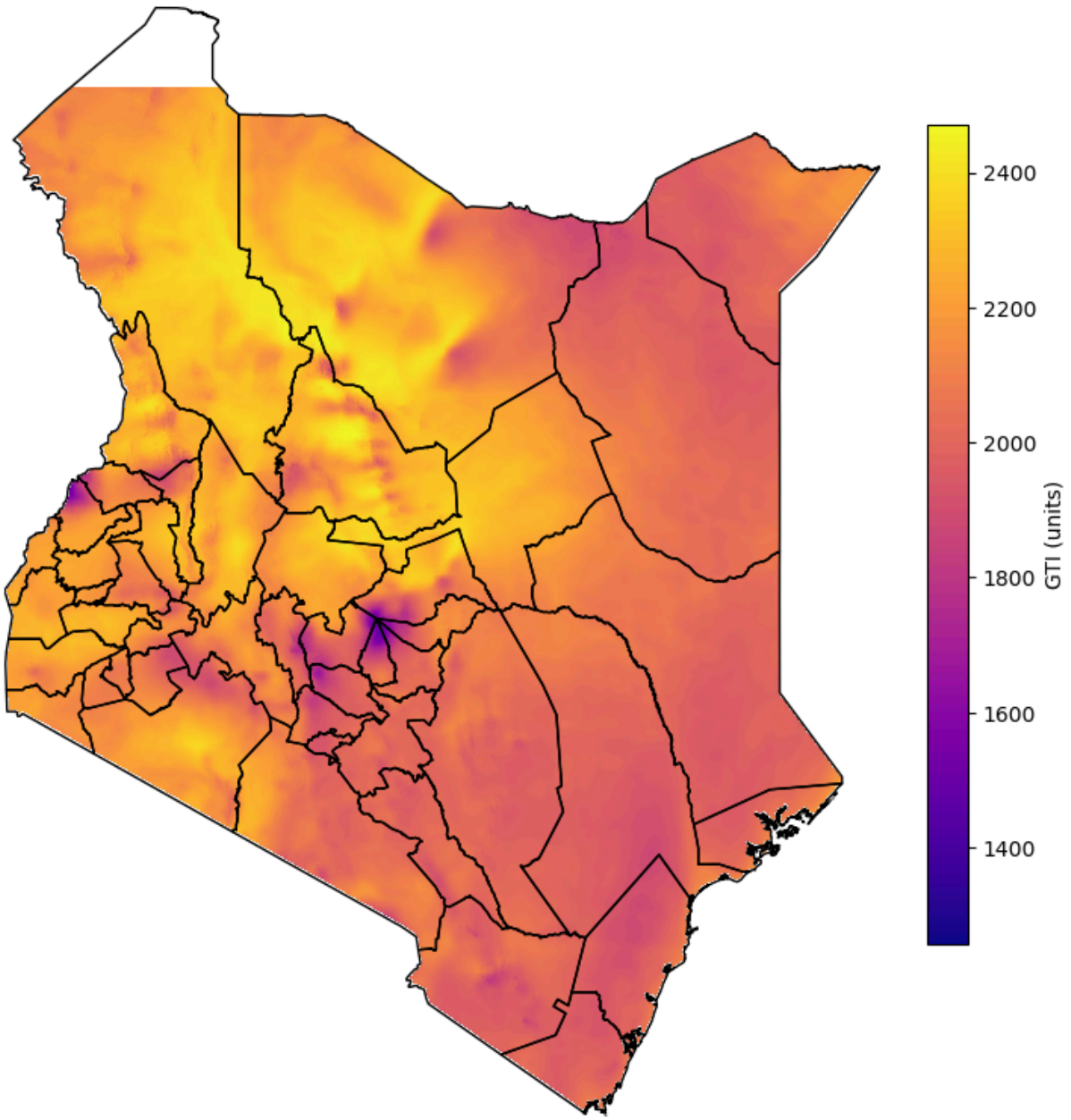

```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\1993060275.py:51: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
Saved GHI_with_Counties.png
```



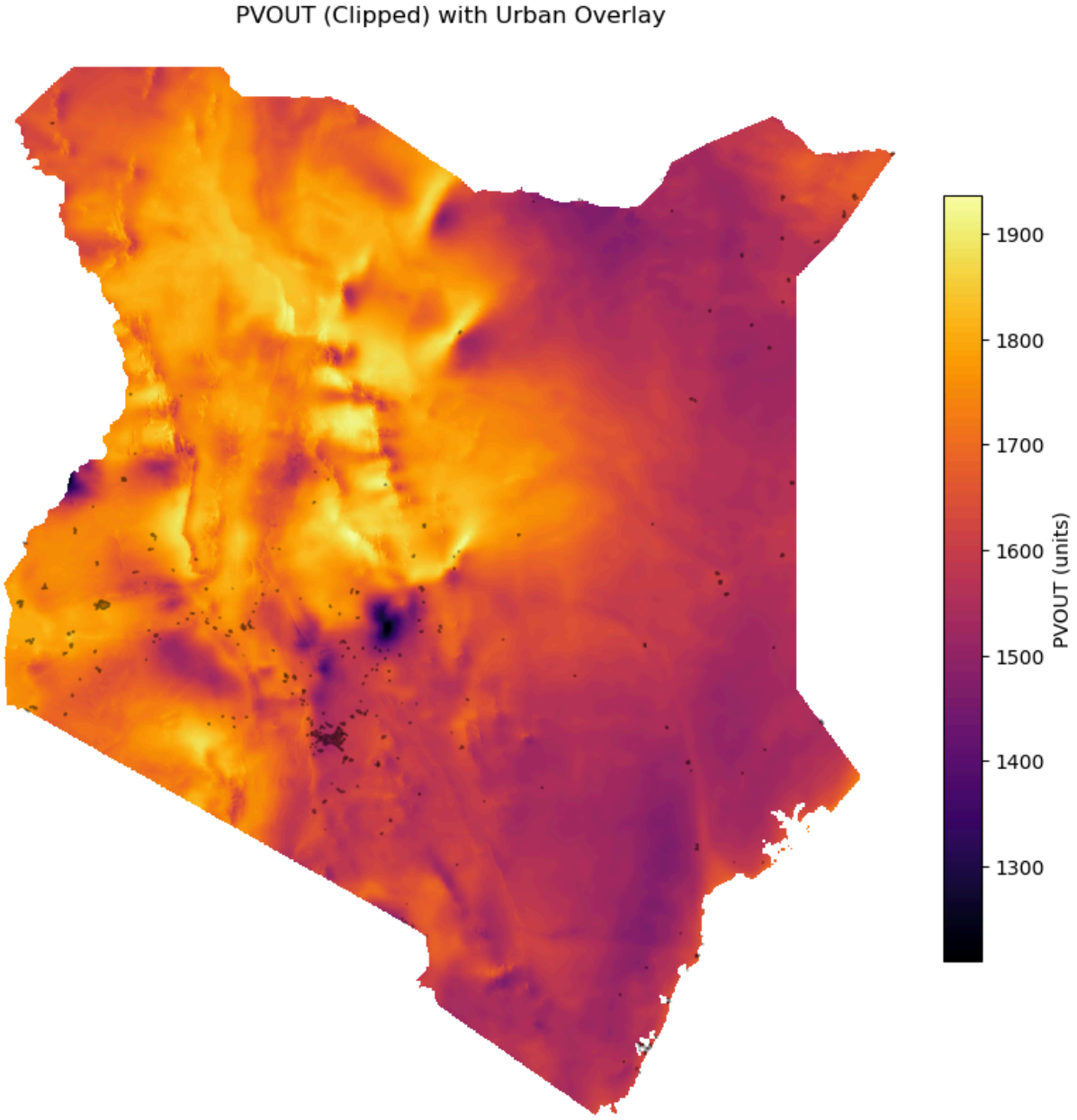

```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\1993060275.py:51: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
Saved GTI_with_Urban.png
```



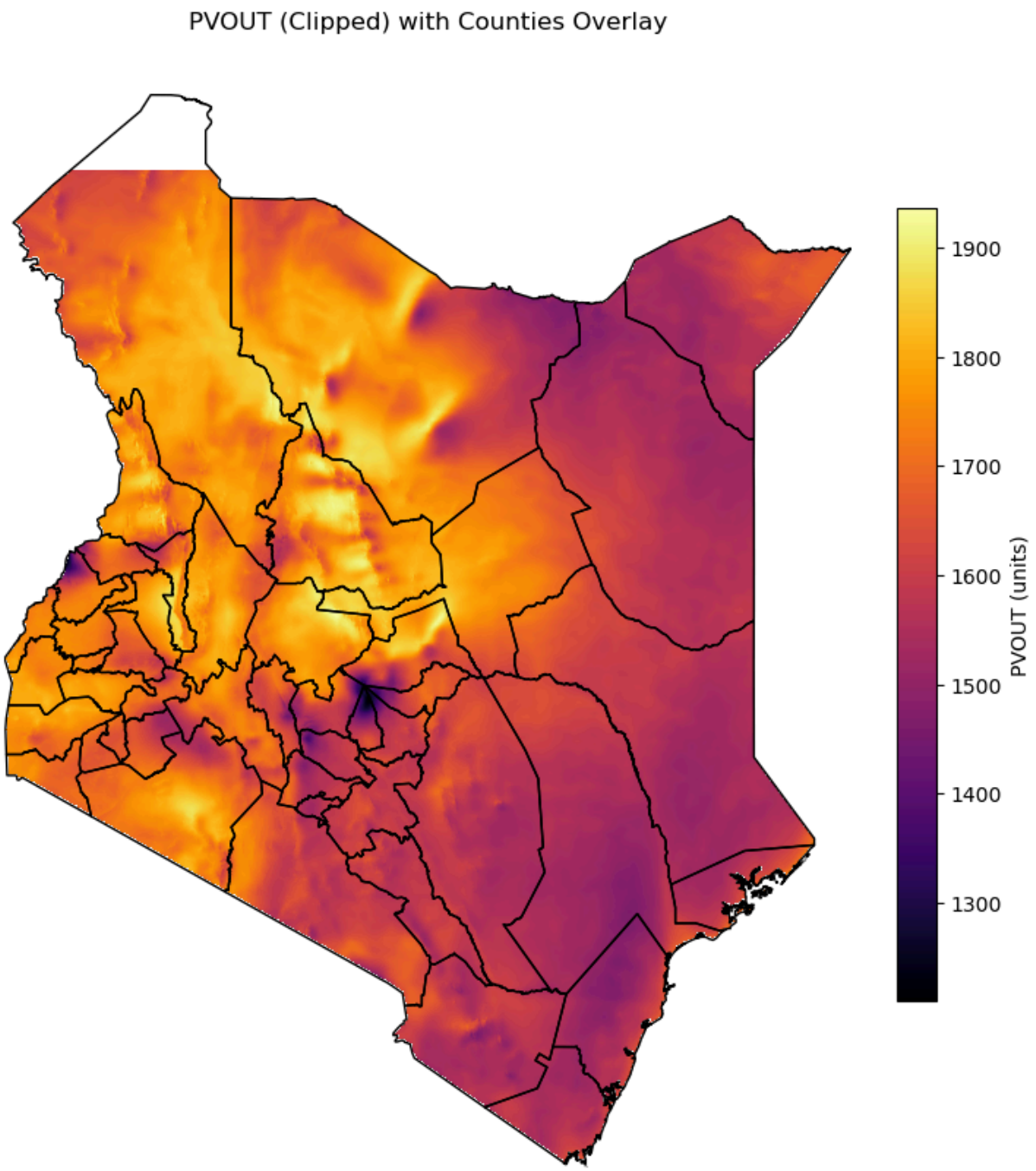

```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\1993060275.py:51: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
Saved GTI_with_Counties.png
```




```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\1993060275.py:51: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
Saved PVOUT_with_Urban.png
```




```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\1993060275.py:51: UserWarning: Legend
does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implemen
ting-a-custom-legend-handler
  handles, labels = ax.get_legend_handles_labels()
Saved PVOUT_with_Counties.png
```




```
Batch plotting complete.
```

Observations: DNI with Urban Areas

- Urban areas often sit where DNI is moderate to low; dense settlements are generally unsuitable for ground-mounted large-scale PV but are important for rooftop PV planning.
- Where urban patches overlap higher DNI pockets, consider rooftop or distributed PV opportunities rather than land-based farms.
- Recommendation: exclude dense urban footprints from land suitability but map rooftop potential separately.

Observations: DNI with County Boundaries

- County-level maps show how DNI resources are distributed administratively; some counties contain large contiguous high-DNI areas suitable for utility-scale projects.
- Use county aggregations (e.g., area above DNI threshold) to prioritize counties for detailed feasibility and permitting work.
- Recommendation: compute county summaries (mean, top-percentile DNI and available land area) to guide investment decisions.

Observations: GHI with Urban Areas

- GHI is often high across many regions including peri-urban zones; however, urban cores remain poor candidates for ground-mounted PV due to land competition.
- Peri-urban and suburban open spaces with high GHI may be suitable for distributed utility sites if land-use and social constraints allow.
- Recommendation: pair GHI maps with land-use and tenure layers to screen urban-adjacent sites for small-to-medium PV plants.

Observations: GHI with County Boundaries

- County-level GHI patterns highlight regions with consistently strong irradiance; counties in the central and northern belts typically rank higher.
- Use county GHI percentiles to shortlist jurisdictions for pre-feasibility before drilling to site-level checks (land availability, grid access).
- Recommendation: produce a ranked county list by GHI and combine with PVOUT for yield-focused prioritization.

Observations: GTI with Urban Areas

- GTI (tilted irradiance) helps identify locations where panel orientation can boost yields; urban open spaces may benefit from optimized tilt but remain limited by land availability.
- Rooftop PV in urban zones can exploit GTI improvements with tilted installations; large ground-mounted arrays should focus outside dense urban footprints.
- Recommendation: prioritize GTI-informed rooftop and peri-urban ground projects where land is available and permissions are favorable.

Observations: GTI with County Boundaries

- County GTI maps refine GHI-based insights by showing where tilt increases capture; counties with favorable GTI and available flat land are high-priority.

- Recommendation: combine GTI, slope, and land availability per county to identify candidate sites for high-efficiency layouts.

Observations: PVOUT with Urban Areas

- PVOUT reflects modeled yield and already accounts for irradiance and some losses; urban areas typically show lower PVOUT for ground plants but rooftop PV remains viable.
- Use PVOUT over urban maps to identify high-yield rooftop neighborhoods or industrial roofs suitable for distributed generation.
- Recommendation: extract building footprints and rooftop-friendly PVOUT statistics for urban deployment planning.

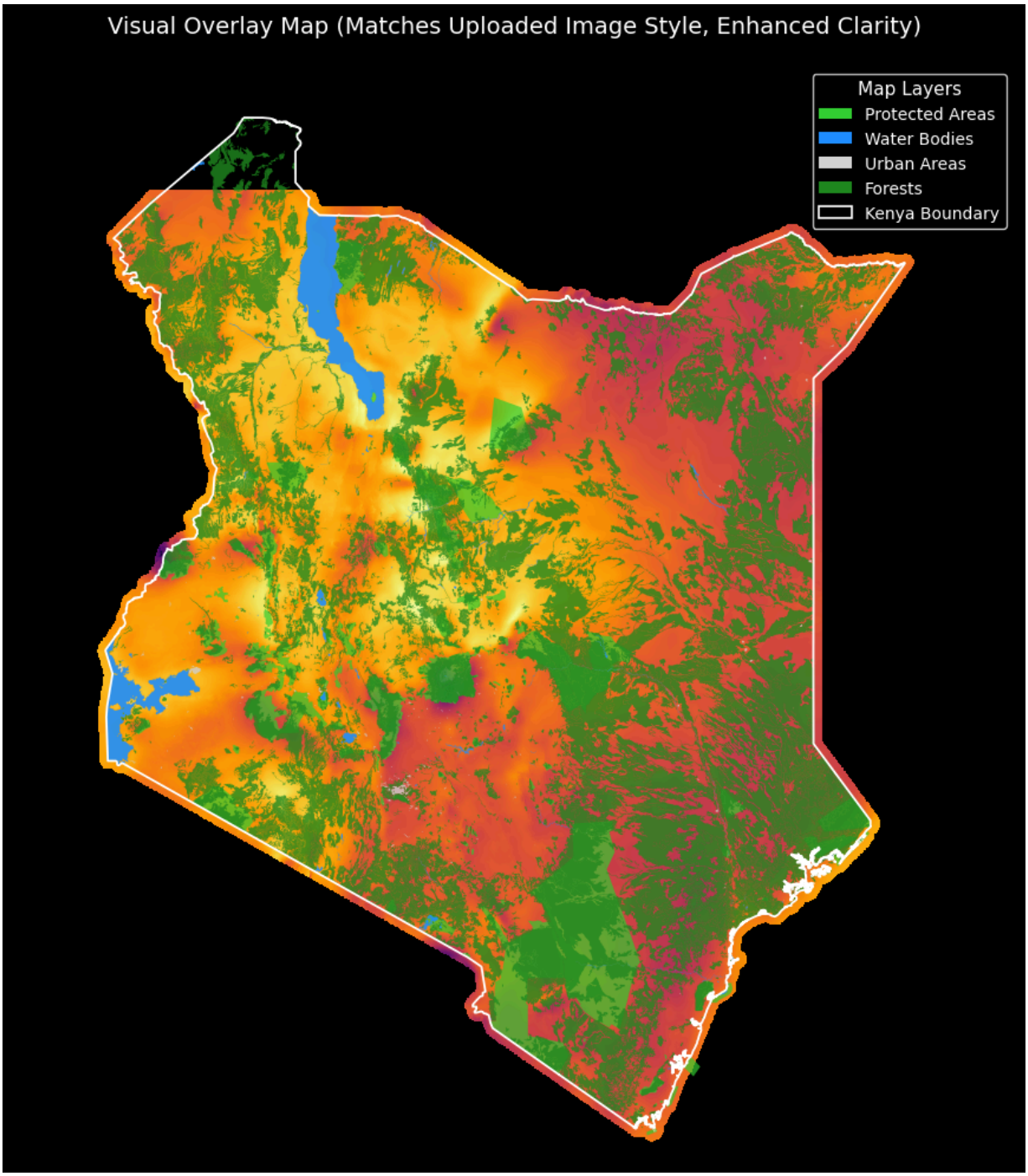
Observations: PVOUT with County Boundaries

- County-level PVOUT summaries directly indicate where installed capacity will produce most energy; combine PVOUT with land availability and constraints per county to rank development areas.
- Recommendation: calculate county-level metrics (total high-PVOUT area, mean PVOUT, percent of land constrained) to inform policy and investment prioritization.

In [16]:
```
import geopandas as gpd
import rasterio
from rasterio.plot import show
import matplotlib.pyplot as plt
from matplotlib.patches import Patch

# -------------------------
# Load raster and shapefiles
# -------------------------
pvout = rasterio.open("PVOUT.tif")
raster_crs = pvout.crs

kenya_boundary = gpd.read_file("kenya_Kenya_Country_Boundary.shp")
protected = gpd.read_file("ke_protected-areas.shp")
water = gpd.read_file("ke_waterbodies.shp")
urban = gpd.read_file("ke_urban.shp")
forests = gpd.read_file("ke_forests.shp")

# -------------------------
# Ensure CRS consistency
# -------------------------
default_crs = "EPSG:4326"
for gdf in [kenya_boundary, protected, water, urban, forests]:
    if gdf.crs is None:
        gdf.set_crs(default_crs, inplace=True)
    gdf.to_crs(raster_crs, inplace=True)

# -------------------------
```

```
# Create figure with black background
# -------------------------
fig, ax = plt.subplots(figsize=(10, 10), facecolor='black')
ax.set_facecolor('black')

# -------------------------
# 1. Show PVOUT raster (base layer)
# -------------------------
show(
    pvout,
    ax=ax,
    cmap='inferno',  # strong contrast, bright yellows and dark reds
    title='Visual Overlay Map (Enhanced Clarity)',
)

# -------------------------
# 2. Overlay shapefile layers (bright colors on dark base)
# -------------------------
protected.plot(ax=ax, color='limegreen', edgecolor='none', alpha=0.7, label='Protec
water.plot(ax=ax, color='dodgerblue', edgecolor='none', alpha=0.9, label='Water Bod
urban.plot(ax=ax, color='lightgray', edgecolor='none', alpha=0.8, label='Urban Area
forests.plot(ax=ax, color='forestgreen', edgecolor='none', alpha=0.8, label='Forest

# 3. Overlay boundary last for clear outline
kenya_boundary.boundary.plot(ax=ax, color='white', linewidth=1.3, label='Kenya Boun

# -------------------------
# Legend (bright text on black)
# -------------------------
legend_elements = [
    Patch(facecolor='limegreen', label='Protected Areas'),
    Patch(facecolor='dodgerblue', label='Water Bodies'),
    Patch(facecolor='lightgray', label='Urban Areas'),
    Patch(facecolor='forestgreen', label='Forests'),
    Patch(facecolor='none', edgecolor='white', linewidth=1.2, label='Kenya Boundary
]

leg = ax.legend(
    handles=legend_elements,
    loc='upper right',
    title="Map Layers",
    frameon=True,
    facecolor='black',
    edgecolor='white',
    labelcolor='white',
    title_fontsize=11,
    fontsize=10
)
plt.setp(leg.get_title(), color='white')

# -------------------------
# Final touches
# -------------------------
ax.set_title("Visual Overlay Map (Matches Uploaded Image Style, Enhanced Clarity)",
             fontsize=14, color='white', pad=20)
ax.set_axis_off()
```

```
plt.tight_layout()
plt.savefig("visual_overlay_map_enhanced.png", dpi=300, bbox_inches='tight', faceco
plt.show()
```



PVOUT (Photovoltaic Output) Suitability Map – Observations and Insights

1. General Overview

- This map visualizes the spatial distribution of photovoltaic (PV) power output potential across Kenya.
- Warmer colors (yellow–red) represent higher solar potential,

- Cooler tones (green–blue) denote lower PV output.
- Additional map layers show protected areas, forests, urban areas, and water bodies, providing important spatial context for solar site planning.

2. High PVOUT Zones (Red to Orange Areas)

- Regions:
  - Northern Kenya — Turkana, Marsabit, Mandera, Wajir, Garissa
  - Eastern and parts of Coastal Kenya (Tana River, Lamu, Kilifi)
- Characteristics:
  - High solar irradiance and minimal cloud cover.
  - Large open land availability with minimal obstructions.
  - Consistent sunlight throughout the year.
- Implications:
  - These are optimal regions for utility-scale solar farms.
  - However, infrastructural limitations (transmission lines, accessibility) should be assessed before deployment.

3. Moderate PVOUT Zones (Yellow–Orange Transition)

- Regions:
  - Lower Rift Valley, Kajiado, Kitui, Makueni, Baringo, Isiolo
- Characteristics:
  - Good solar potential with slightly variable irradiance.
  - Mixed land use — some vegetation or elevation variations.
- Implications:
  - Suitable for medium-scale solar projects or hybrid systems (solar + storage).
  - Infrastructure and environmental balance make these regions promising.

4. Low PVOUT Zones (Greenish Areas)

- Regions:
  - Central Highlands, Western Kenya, and parts around Lake Victoria Basin
- Characteristics:

- Higher rainfall, denser vegetation, and frequent cloud cover.
- Reduced direct irradiance and higher humidity levels.

- Implications:
  - These zones are less ideal for large solar installations but could support rooftop or small distributed solar systems for local energy needs.

5. Restricted or Special Zones (Layer Overlays)

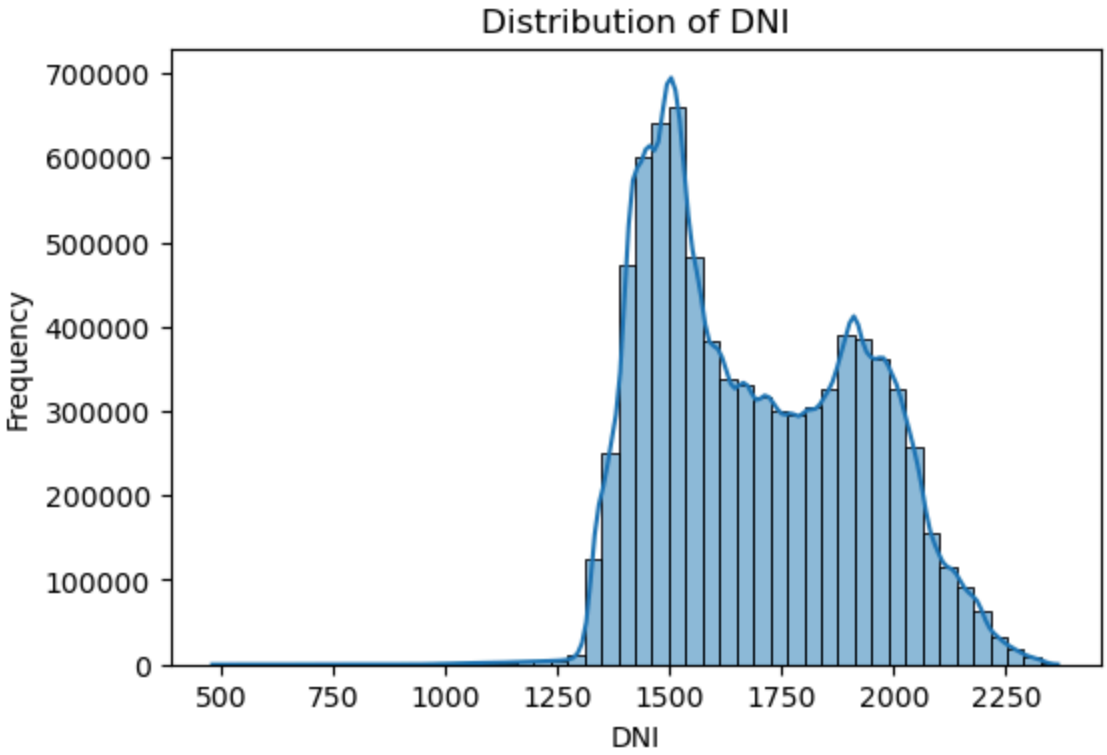
- Forests & Protected Areas (Bright Green): Mt. Kenya, Aberdares, Mau Complex — avoid solar development to preserve ecosystems.
- Urban Areas (Light Blue): Nairobi, Kisumu, Mombasa — potential for rooftop solar rather than ground installations.
- Water Bodies (Blue): Lakes and rivers excluded from PV potential zones.

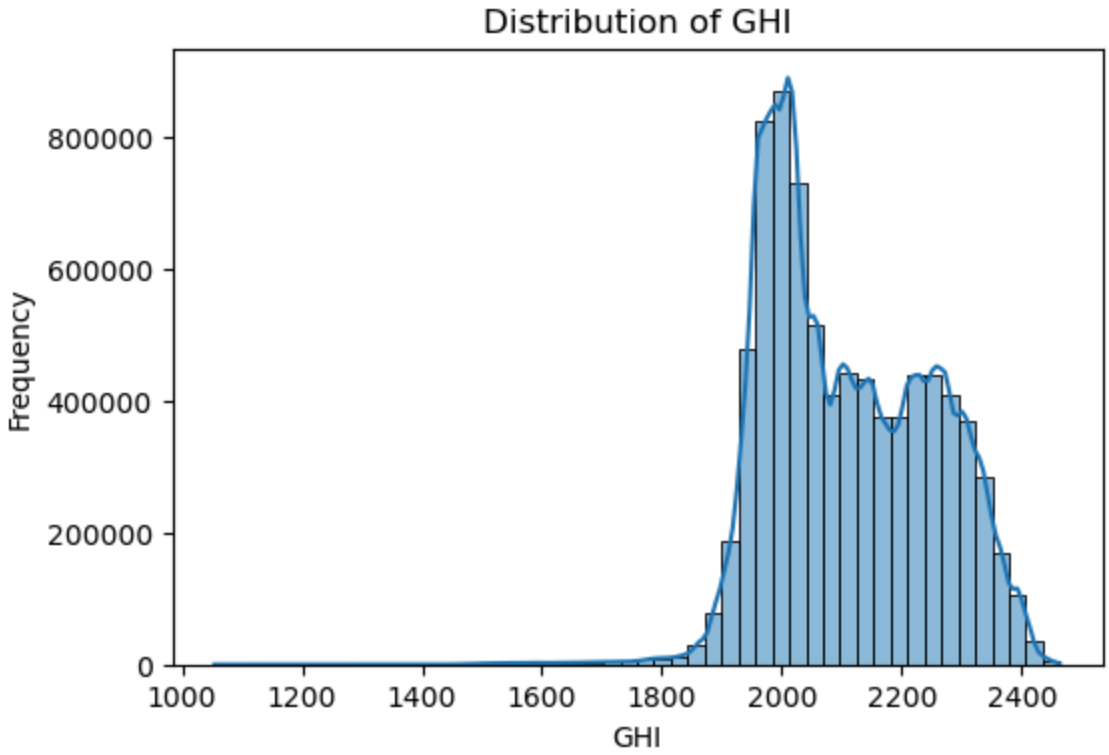
Statistical/ Descriptive Analysis

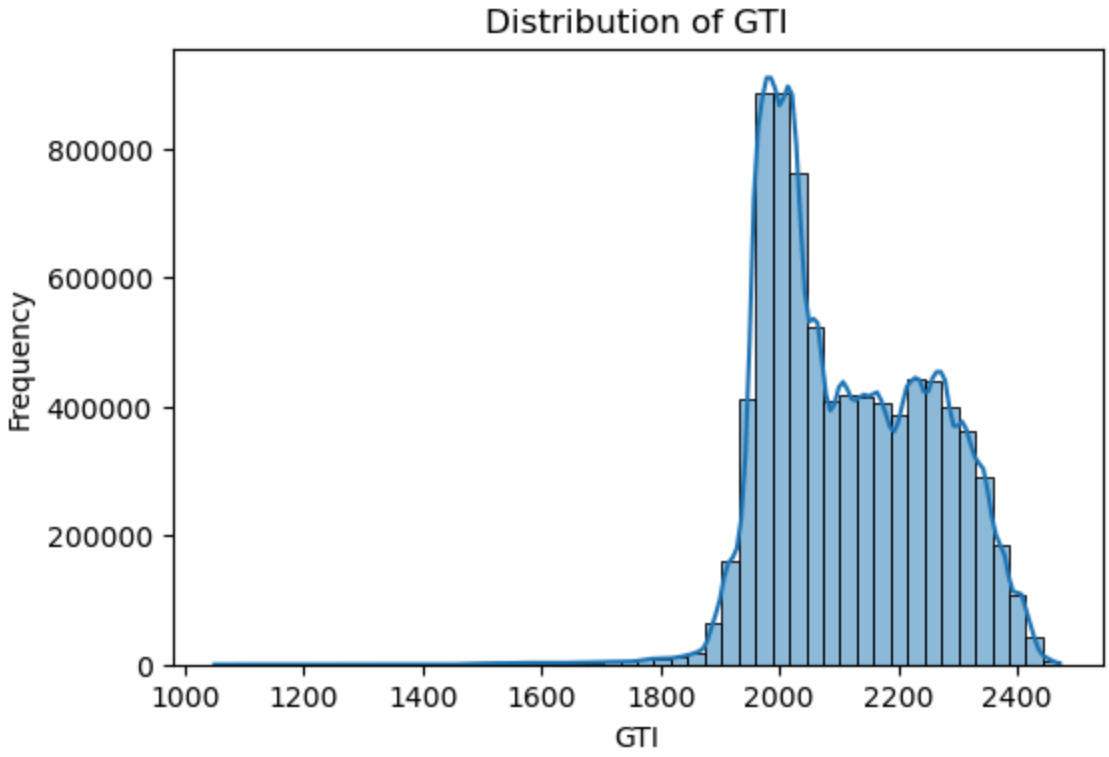
Univariate Analysis

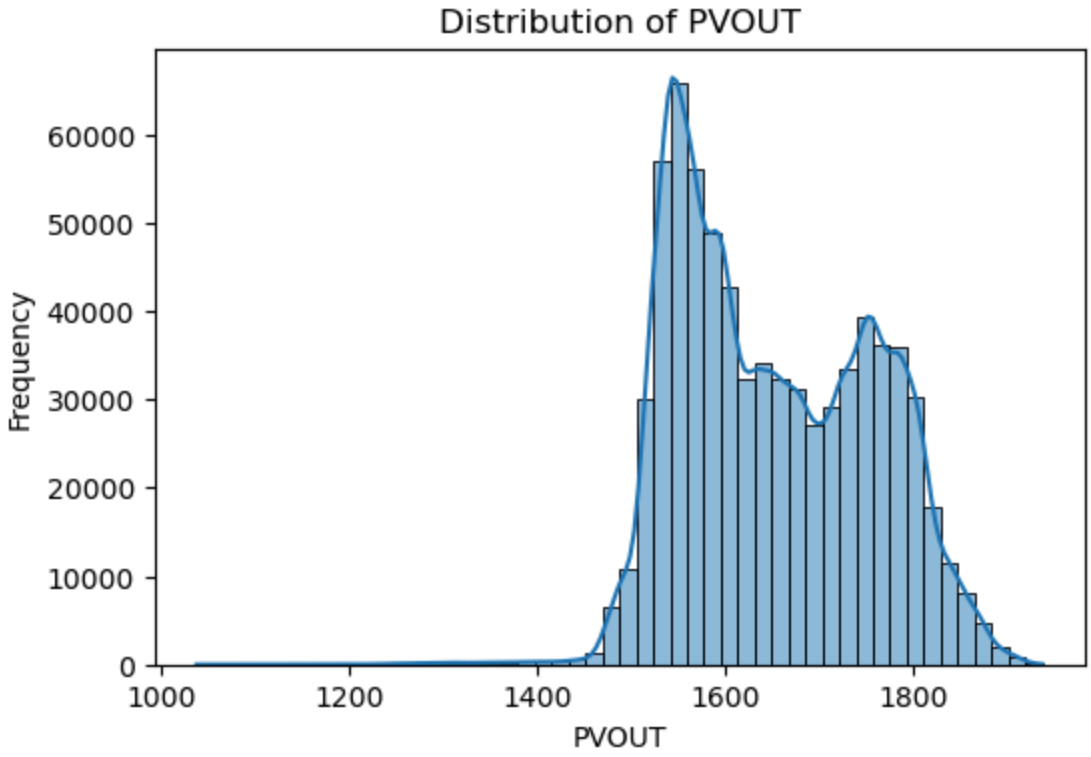
In [17]:
```
# Defensive raster descriptive statistics
raster_files = {
    "DNI": "DNI.tif",
    "GHI": "GHI.tif",
    "GTI": "GTI.tif",
    "PVOUT": "PVOUT.tif",
    "TEMP": "TEMP.tif",
    "DIF": "DIF.tif"
}

raster_stats = {}

for key, path in raster_files.items():
    try:
        with rasterio.open(path) as src:
            arr = src.read(1).astype(float)

            # Handle nodata robustly
            nodata = src.nodata
            if nodata is not None:
                arr[arr == nodata] = np.nan

            # If array is all nan, skip stats and plotting
            valid_mask = ~np.isnan(arr)
            if not np.any(valid_mask):
                print(f"Warning: {path} contains no valid data (all nodata/nan). Sk
                raster_stats[key] = {"min": None, "max": None, "mean": None, "std":
```

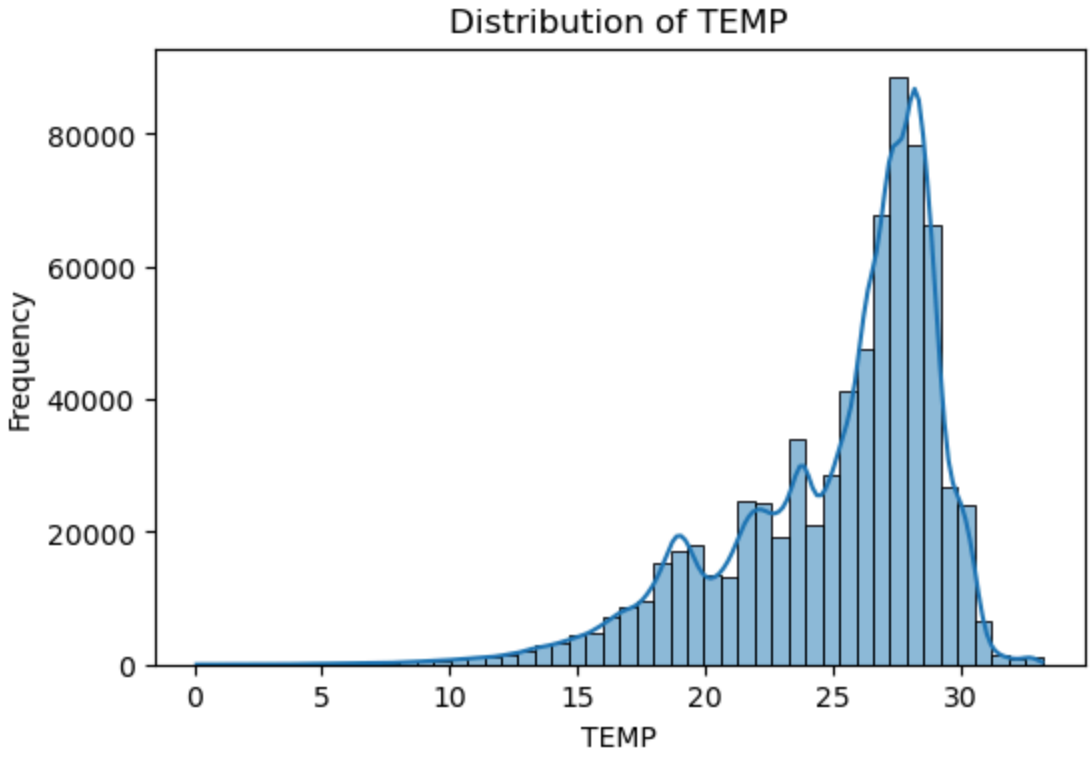
```
                continue

            stats = {
                "min": float(np.nanmin(arr)),
                "max": float(np.nanmax(arr)),
                "mean": float(np.nanmean(arr)),
                "std": float(np.nanstd(arr))
            }
            raster_stats[key] = stats

            # Plot histogram only when data exists
            plt.figure(figsize=(6, 4))
            sns.histplot(arr[valid_mask].ravel(), bins=50, kde=True)
            plt.title(f"Distribution of {key}")
            plt.xlabel(key)
            plt.ylabel("Frequency")
            plt.show()
    except FileNotFoundError:
        print(f"Raster file not found: {path}")
        raster_stats[key] = {"error": "file_not_found"}
    except Exception as e:
        print(f"Error reading raster {path}: {e}")
        raster_stats[key] = {"error": str(e)}

print("Raster Descriptive Statistics:")
for k, v in raster_stats.items():
    print(f"{k}: {v}")
```

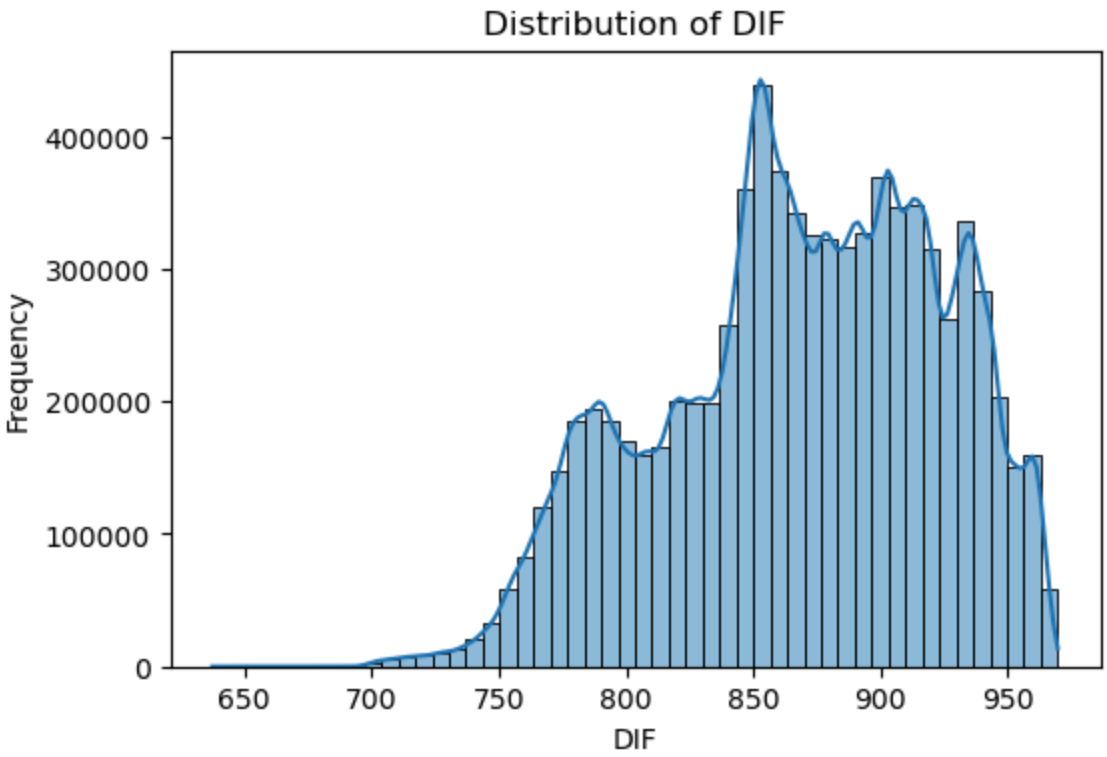



Distribution of GHI
Frequency
800000
600000
400000
200000
0
1000
1200
1400
1600
1800
2000
2200
2400
GHI


Distribution of GTI
Frequency
800000
600000
400000
200000
0
1000
1200
1400
1600
1800
2000
2200
2400
GTI


Distribution of PVOUT
Frequency
0
10000
20000
30000
40000
50000
60000
1000
1200
1400
1600
1800
PVOUT


Distribution of TEMP
Frequency
0
20000
40000
60000
80000
0
5
10
15
20
25
30
TEMP




```
Raster Descriptive Statistics:
DNI: {'min': 478.8429870605469, 'max': 2366.090087890625, 'mean': 1692.909998301230
6, 'std': 233.05706104467095}
GHI: {'min': 1053.7459716796875, 'max': 2463.24609375, 'mean': 2109.2515795110876,
'std': 137.64694929264238}
GTI: {'min': 1049.72802734375, 'max': 2471.281982421875, 'mean': 2116.268989869624,
'std': 138.17345179330266}
PVOUT: {'min': 1037.675048828125, 'max': 1936.9210205078125, 'mean': 1651.1593602419
862, 'std': 102.87126933985517}
TEMP: {'min': 0.10000000149011612, 'max': 33.20000076293945, 'mean': 25.145793297857
864, 'std': 4.107523235795554}
DIF: {'min': 637.3610229492188, 'max': 969.739013671875, 'mean': 868.3988173713891,
'std': 54.614165683764}
```

Observations: Raster Descriptive Statistics

- The histograms show the per-pixel distribution for each solar and climate indicator. Look for skewness (long tails) and outliers: a right-skew in irradiance layers indicates a small area with much higher resource than the national average.
- Summary stats (min, max, mean, std) provide quick guidance for thresholds. Consider using percentiles (e.g., 75th or 90th) rather than raw maxima to define high-resource candidates and avoid outlier-driven decisions.
- Large numbers of masked or NaN pixels indicate coverage or clipping issues—verify data completeness for those layers before relying on them.
- Recommendation: use these distributions to choose normalization method (percentile-based scaling is robust) and to set an operational cutoff for candidate sites (for example, areas above mean + 0.5*std or above the 75th percentile).

In [18]:
```
# -------------------
# Vector descriptive statistics (defensive)
# -------------------
vector_files = {
    "Water": "ke_waterbodies.shp",
    "Protected": "ke_protected-areas.shp",
    "Urban": "ke_urban.shp",
    "Forests": "ke_forests.shp"
}

vector_stats = {}

for key, path in vector_files.items():
    try:
        gdf = gpd.read_file(path)
    except FileNotFoundError:
        print(f"Vector file not found: {path}")
        vector_stats[key] = {"error": "file_not_found"}
        continue
    except Exception as e:
        print(f"Could not read {path}: {e}")
        vector_stats[key] = {"error": str(e)}
        continue

    # If empty GeoDataFrame
    if gdf.empty:
        print(f"Warning: {path} loaded but contains 0 features.")
        vector_stats[key] = {"count": 0, "total_area_km2": 0.0, "mean_area_km2": 0.
        continue

    # Ensure CRS is set
    try:
        if gdf.crs is None:
            print(f"{key} has no CRS, assigning EPSG:4326 (WGS84).")
            gdf = gdf.set_crs(epsg=4326, allow_override=True)

        # Project to metric CRS for area calculation
        gdf_m = gdf.to_crs(epsg=3857)
        gdf_m["area_km2"] = gdf_m.geometry.area / 1e6  # area in km²

        stats = {
            "count": int(len(gdf_m)),
            "total_area_km2": float(gdf_m["area_km2"].sum()),
            "mean_area_km2": float(gdf_m["area_km2"].mean()) if len(gdf_m) > 0 else
        }
        vector_stats[key] = stats
    except Exception as e:
        print(f"Error processing {key} ({path}): {e}")
        vector_stats[key] = {"error": str(e)}

print("\nVector Descriptive Statistics:")
for k, v in vector_stats.items():
    print(f"{k}: {v}")
```

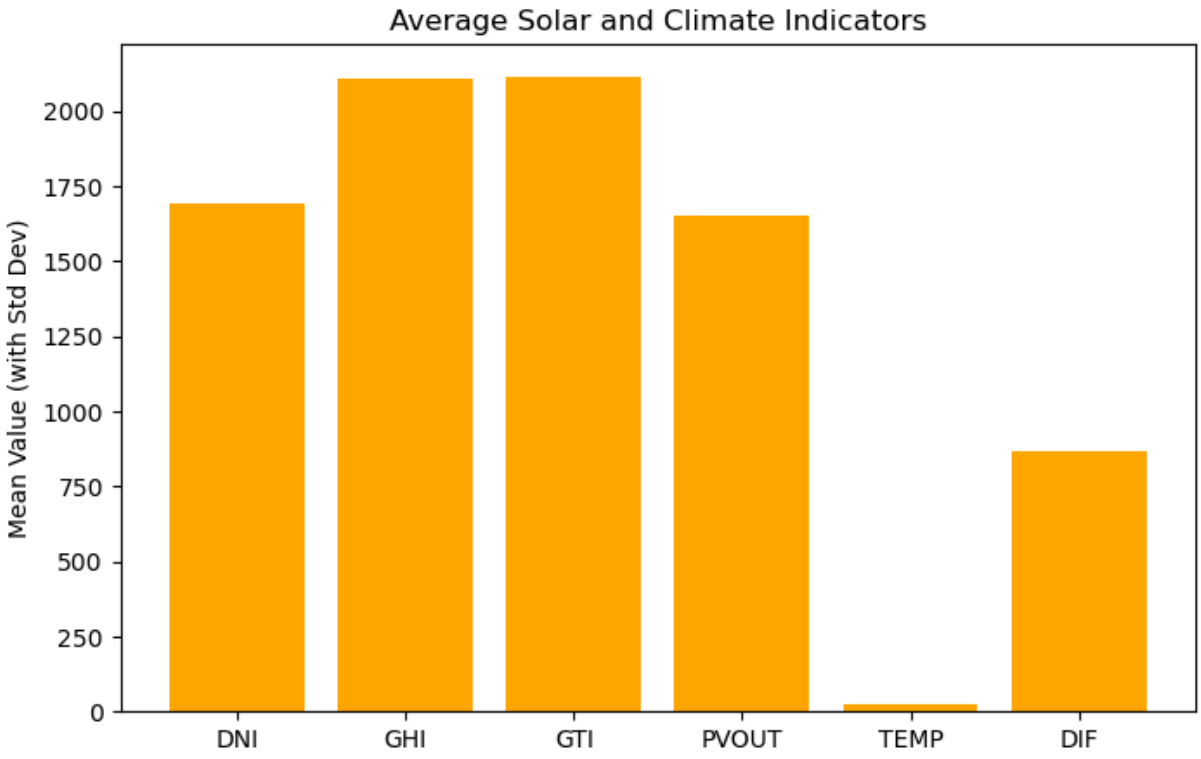
```
Water has no CRS, assigning EPSG:4326 (WGS84).

Vector Descriptive Statistics:
Water: {'count': 208, 'total_area_km2': 13402.264782799191, 'mean_area_km2': 64.4339
6530191919}
Protected: {'count': 203, 'total_area_km2': 56784.104955318035, 'mean_area_km2': 27
9.72465495230557}
Urban: {'count': 260, 'total_area_km2': 730.2807984382135, 'mean_area_km2': 2.808772
301685437}
Forests: {'count': 4764, 'total_area_km2': 232573.50688755373, 'mean_area_km2': 48.8
18956105699776}
```

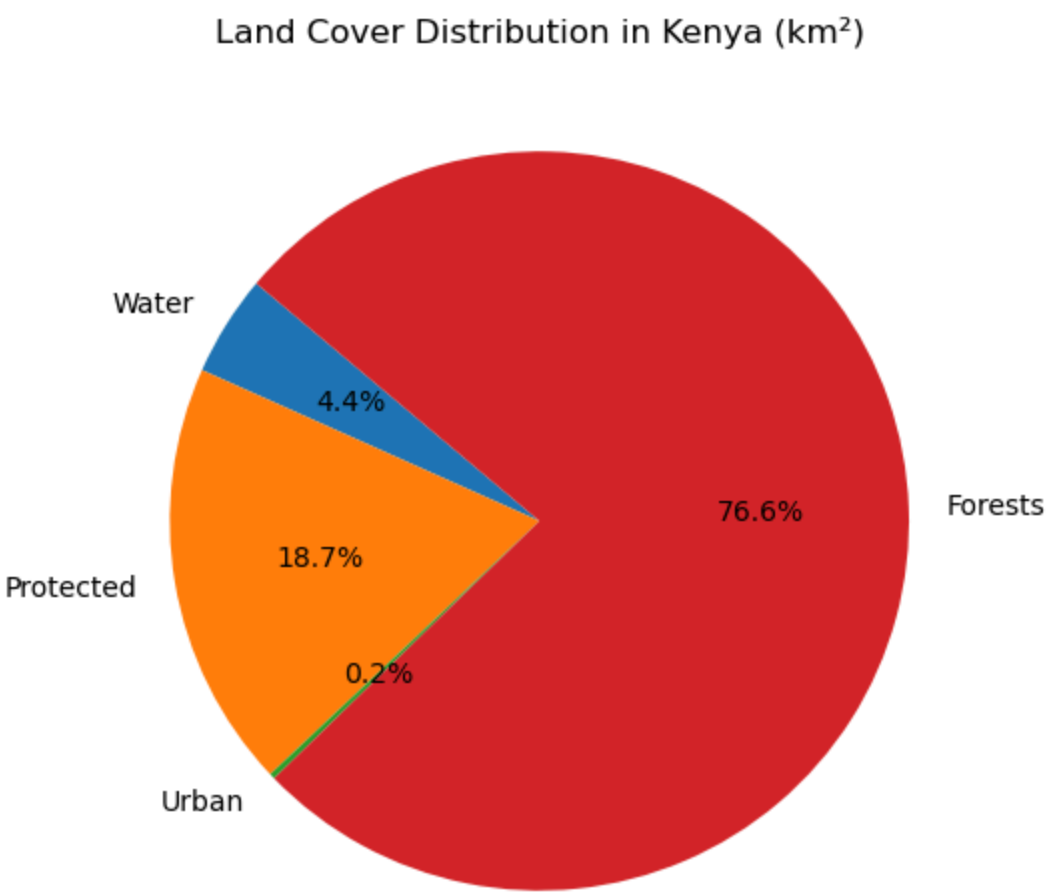
Observations: Vector Descriptive Statistics

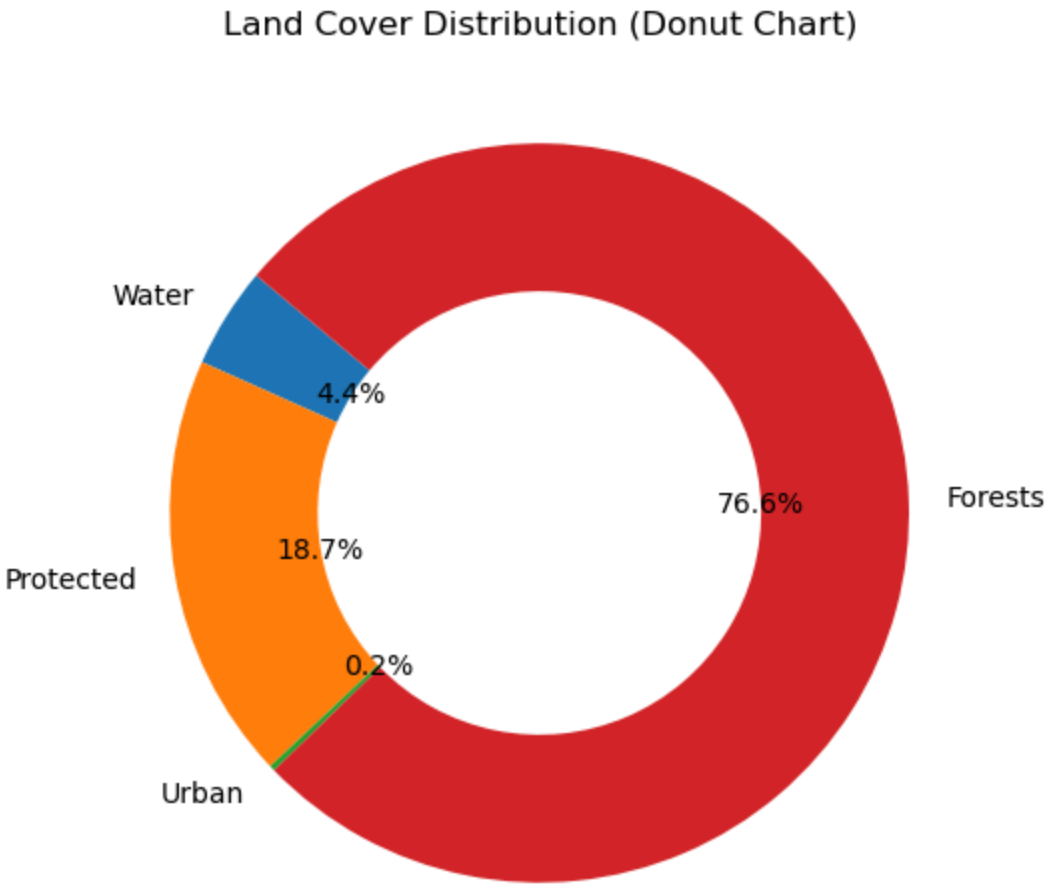
- Vector summaries (count, total area, mean area) indicate how much land each constraint occupies; large total areas for forests or protected areas mean those constraints will heavily reduce candidate land.
- A high count with low mean area indicates fragmentation (many small polygons), which has implications for permitting and construction; aggregated large patches are often preferable for utility-scale PV.
- If any layer reports 0 features or an error, verify the shapefile path and clipping logic — empty outputs may indicate a projection or overlay problem.
- Recommendation: calculate percent of total land area constrained by each layer and use those percentages to inform weighting or exclusion decisions.

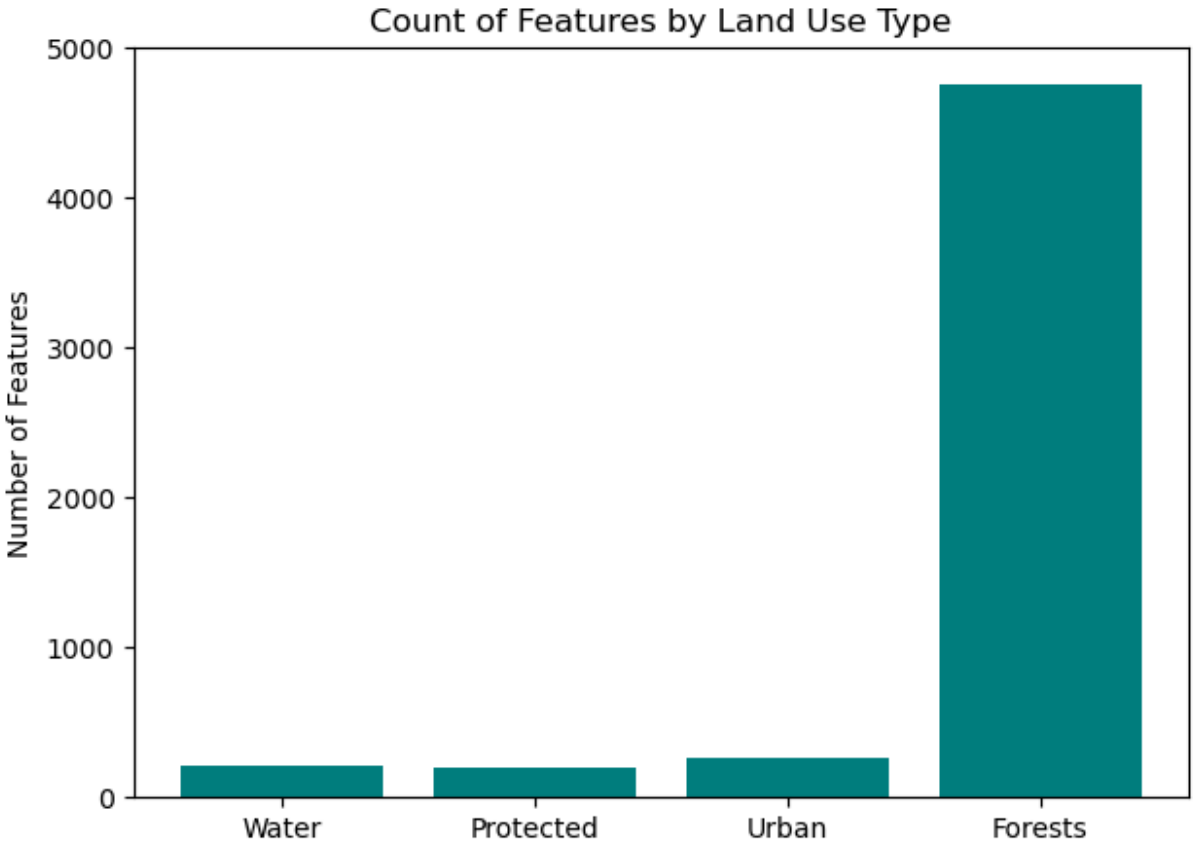
In [19]:
```
# -------------------
# Raster statistics visualization
# -------------------
raster_means = {k: v["mean"] for k, v in raster_stats.items()}
raster_std = {k: v["std"] for k, v in raster_stats.items()}

# Bar chart: average values of rasters
plt.figure(figsize=(8, 5))
plt.bar(raster_means.keys(), raster_means.values(), color="orange")
plt.ylabel("Mean Value (with Std Dev)")
plt.title("Average Solar and Climate Indicators")
plt.show()

# -------------------
# Vector statistics visualization
# -------------------
vector_areas = {k: v["total_area_km2"] for k, v in vector_stats.items()}

# Pie chart: share of land use
plt.figure(figsize=(6, 6))
plt.pie(vector_areas.values(), labels=vector_areas.keys(), autopct='%1.1f%%', start
plt.title("Land Cover Distribution in Kenya (km²)")
plt.show()

# Donut plot
plt.figure(figsize=(6, 6))
wedges, texts, autotexts = plt.pie(vector_areas.values(), labels=vector_areas.keys(
```

```
                                       startangle=140, wedgeprops=dict(width=0.4))
plt.title("Land Cover Distribution (Donut Chart)")
plt.show()

# Bar chart: feature counts
vector_counts = {k: v["count"] for k, v in vector_stats.items()}
plt.figure(figsize=(7, 5))
plt.bar(vector_counts.keys(), vector_counts.values(), color="teal")
plt.ylabel("Number of Features")
plt.title("Count of Features by Land Use Type")
plt.show()
```




Land Cover Distribution in Kenya (km²)
Water
4.4%
Forests
76.6%
Protected
18.7%
0.2%
Urban


Land Cover Distribution (Donut Chart)
Water
4.4%
Protected
18.7%
Urban
0.2%
Forests
76.6%


Count of Features by Land Use Type
Number of Features
5000
4000
3000
2000
1000
0
Water
Protected
Urban
Forests

Observations: Raster & Vector Visualizations

- The bar chart of raster means helps identify which indicators dominate numerically and require normalization before combining layers; large differences suggest standardization is necessary.
- The pie/donut charts show the relative land area share of each constraint — large slices for protected areas or water bodies imply stricter exclusion will heavily narrow candidate areas.
- The feature counts bar indicates fragmentation: many small features complicate site assembly and permitting, while a few large contiguous polygons are usually more practical for site development.
- Recommendation: use these visual cues to define exclusion masks, choose normalization (e.g., z-score or percentile), and prioritize large contiguous areas for follow-up site-level assessment.

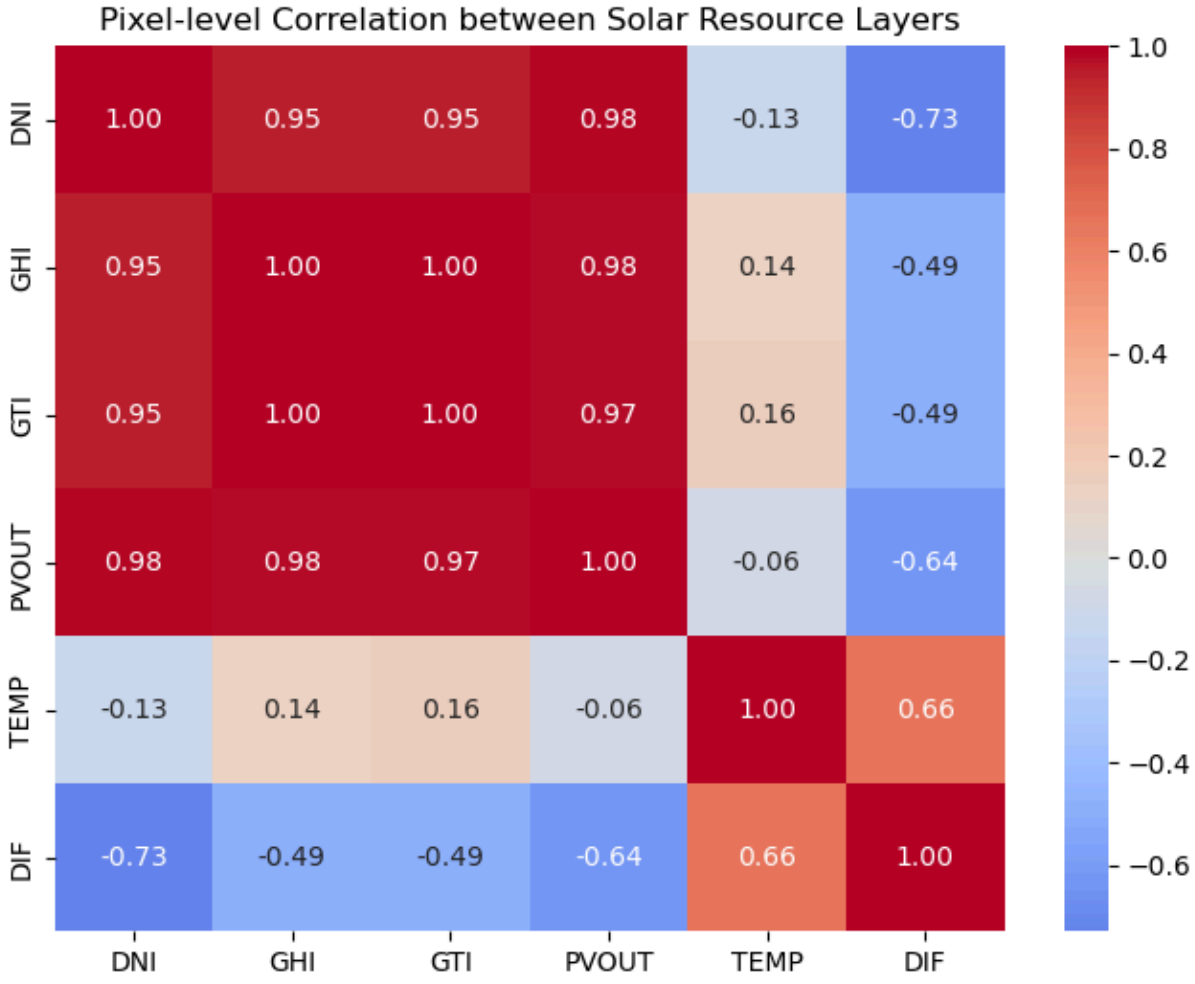
Bi-variate Analysis

In [20]:
```
# Paths to your raster files
raster_files = {
    "DNI": "DNI.tif",
    "GHI": "GHI.tif",
    "GTI": "GTI.tif",
    "PVOUT": "PVOUT.tif",
    "TEMP": "TEMP.tif",
    "DIF": "DIF.tif"
}

# Read rasters into a dict with metadata
arrays = {}
for name, path in raster_files.items():
    try:
        with rasterio.open(path) as src:
            arr = src.read(1).astype(float)
            # prefer explicit nodata, otherwise mask very negative sentinel values
            nodata = src.nodata
            if nodata is not None:
                arr[arr == nodata] = np.nan
            else:
                # common sentinel used earlier in notebook
                arr[arr <= -9999] = np.nan

            arrays[name] = {
                "arr": arr,
                "transform": src.transform,
                "crs": src.crs,
                "shape": arr.shape
            }
    except FileNotFoundError:
        print(f"Raster file not found: {path} (skipping)")
    except Exception as e:
```

```
        print(f"Error reading {path}: {e}")

if len(arrays) < 2:
    print("Not enough rasters loaded to compute correlations. Aborting.")
else:
    # pick a reference grid (first loaded raster)
    ref_name = next(iter(arrays))
    ref = arrays[ref_name]
    ref_shape = ref["shape"]
    ref_transform = ref["transform"]
    ref_crs = ref["crs"]

    # Resample/reproject any raster that doesn't match the reference
    for name, meta in list(arrays.items()):
        if meta["shape"] != ref_shape or meta["crs"] != ref_crs:
            try:
                dest = np.full(ref_shape, np.nan, dtype=float)
                reproject(
                    source=meta["arr"],
                    destination=dest,
                    src_transform=meta["transform"],
                    src_crs=meta["crs"],
                    dst_transform=ref_transform,
                    dst_crs=ref_crs,
                    resampling=Resampling.bilinear
                )
                arrays[name]["arr"] = dest
                arrays[name]["shape"] = dest.shape
                arrays[name]["transform"] = ref_transform
                arrays[name]["crs"] = ref_crs
                print(f"Resampled {name} to reference grid ({ref_name}).")
            except Exception as e:
                print(f"Could not resample {name} to reference grid: {e}")
                arrays.pop(name)

    # Ensure we still have at least two aligned rasters
    if len(arrays) < 2:
        print("Not enough compatible rasters after resampling/alignment. Aborting."
    else:
        # Build DataFrame from flattened aligned rasters
        data = {}
        for name, meta in arrays.items():
            data[name] = meta["arr"].ravel()

        df = pd.DataFrame(data)

        # Drop rows with any NaNs to ensure pairwise pixel comparisons
        df_clean = df.dropna()
        if df_clean.shape[0] == 0:
            print("No overlapping valid pixels across rasters after masking. Cannot
        else:
            corr = df_clean.corr()

            # Plot heatmap
            plt.figure(figsize=(8, 6))
            sns.heatmap(corr, annot=True, cmap="coolwarm", center=0, fmt=".2f")
```

```
            plt.title("Pixel-level Correlation between Solar Resource Layers")
            plt.show()
```

```
Resampled PVOUT to reference grid (DNI).
Resampled TEMP to reference grid (DNI).
```



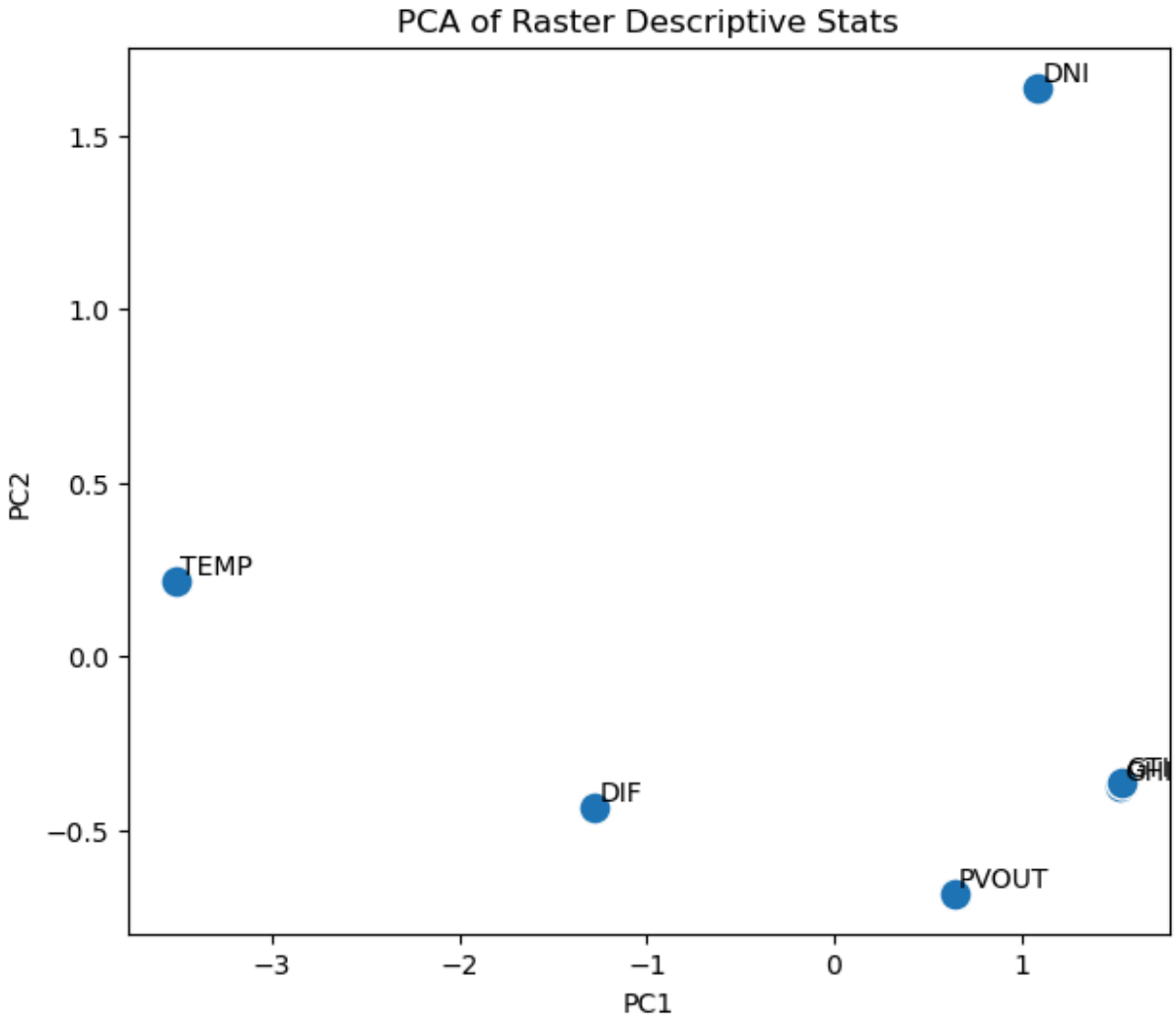
Observations: Correlation Analysis

- The correlation heatmap shows pairwise relationships between resource layers. High correlation between GHI, GTI, and PVOUT is expected since PVOUT derives from irradiance; this indicates redundancy.
- Use correlations to decide which layers to keep or combine. If two layers correlate > 0.85, consider dropping or down-weighting one to avoid overemphasizing the same signal.
- Weak or negative correlations (e.g., DIF vs DNI) highlight areas with different solar regimes (diffuse vs direct), which may call for different PV technologies or designs.
- Recommendation: perform PCA or use PVOUT (which encapsulates irradiance and system effects) to reduce dimensionality before weighted overlay.

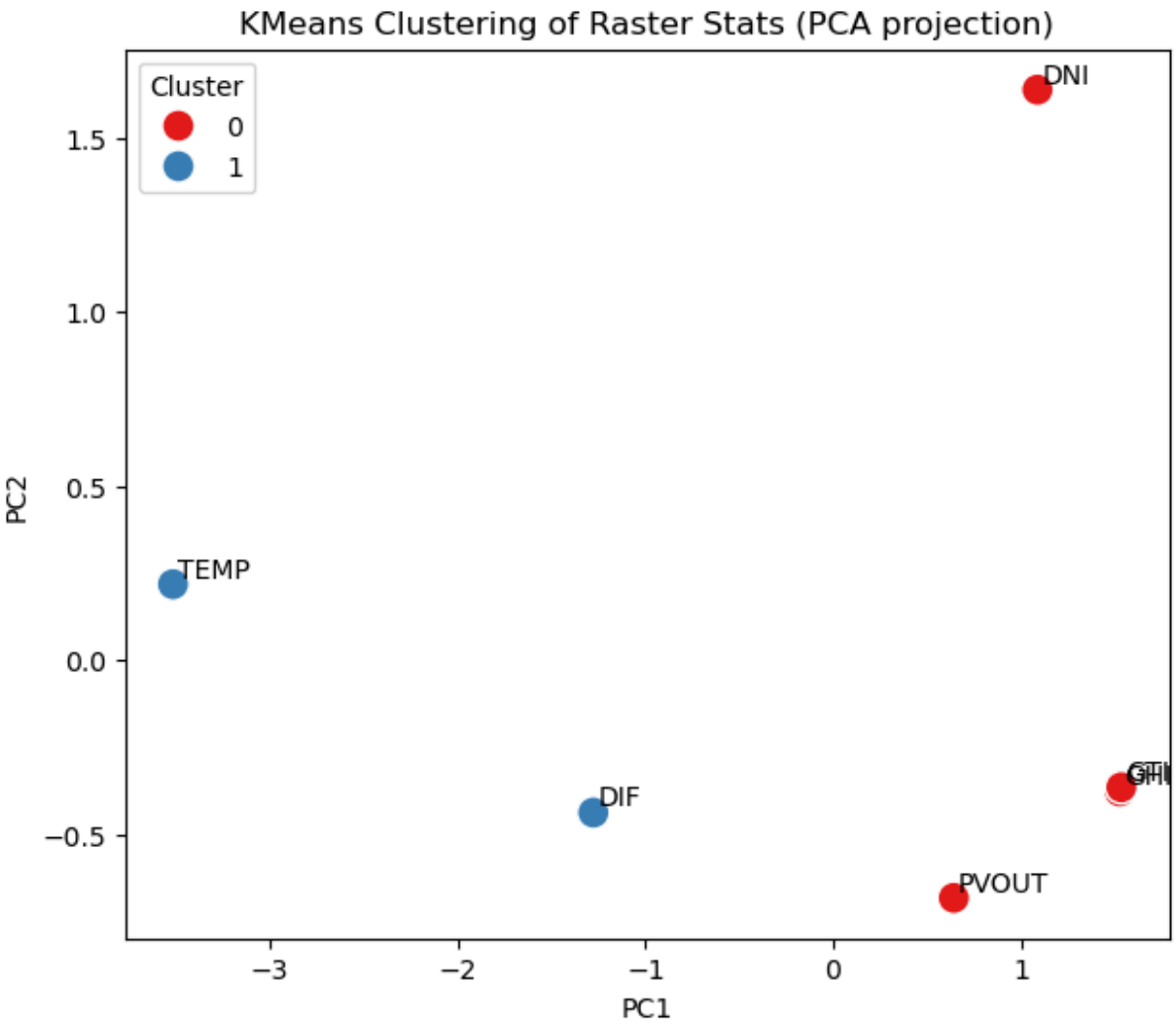
Multi-variate Analysis

In [21]:
```
# -------------------------------
# 1. Construct dataframe from raster stats
# -------------------------------
raster_stats = {
    'DNI': {'min': 478.84, 'max': 2366.09, 'mean': 1692.91, 'std': 233.06},
    'GHI': {'min': 1053.75, 'max': 2463.25, 'mean': 2109.25, 'std': 137.65},
    'GTI': {'min': 1049.73, 'max': 2471.28, 'mean': 2116.27, 'std': 138.17},
    'PVOUT': {'min': 1037.68, 'max': 1936.92, 'mean': 1651.16, 'std': 102.87},
    'TEMP': {'min': 0.10, 'max': 33.20, 'mean': 25.15, 'std': 4.11},
    'DIF': {'min': 637.36, 'max': 969.74, 'mean': 868.40, 'std': 54.61}
}

df = pd.DataFrame(raster_stats).T
print("Raster Stats DataFrame:\n", df)

# -------------------------------
# 2. Scale the data
# -------------------------------
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# -------------------------------
# 3. PCA
# -------------------------------
pca = PCA(n_components=2)  # reduce to 2D for visualization
pca_result = pca.fit_transform(scaled_data)

df_pca = pd.DataFrame(pca_result, columns=['PC1', 'PC2'], index=df.index)

# Variance explained
print("Explained variance ratio:", pca.explained_variance_ratio_)

# Plot PCA scatter
plt.figure(figsize=(7,6))
sns.scatterplot(x='PC1', y='PC2', data=df_pca, s=150, marker='o')
for i in df_pca.index:
    plt.text(df_pca.loc[i, 'PC1']+0.02, df_pca.loc[i, 'PC2']+0.02, i, fontsize=10)
plt.title("PCA of Raster Descriptive Stats")
plt.show()

# -------------------------------
# 4. KMeans Clustering
# -------------------------------
kmeans = KMeans(n_clusters=2, random_state=42)  # try 2 clusters first
df['Cluster'] = kmeans.fit_predict(scaled_data)

print("\nCluster Assignments:\n", df[['Cluster']])

# Plot clustering on PCA space
plt.figure(figsize=(7,6))
sns.scatterplot(x='PC1', y='PC2', hue=df['Cluster'], palette='Set1', s=150, data=df
for i in df_pca.index:
    plt.text(df_pca.loc[i, 'PC1']+0.02, df_pca.loc[i, 'PC2']+0.02, i, fontsize=10)
plt.title("KMeans Clustering of Raster Stats (PCA projection)")
plt.show()
```

```
Raster Stats DataFrame:
            min      max     mean     std
DNI      478.84  2366.09  1692.91  233.06
GHI     1053.75  2463.25  2109.25  137.65
GTI     1049.73  2471.28  2116.27  138.17
PVOUT   1037.68  1936.92  1651.16  102.87
TEMP      0.10    33.20    25.15    4.11
DIF     637.36   969.74   868.40   54.61
Explained variance ratio: [0.84330234 0.15228518]
```




```
c:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
Cluster Assignments:
        Cluster
DNI           0
GHI           0
GTI           0
PVOUT         0
TEMP          1
DIF           1
```



Observations: KMeans Clustering

- The clustering groups raster indicators into clusters with similar statistical profiles. Interpret cluster labels by inspecting which indicators are in each cluster (see the printed cluster assignments).
- If PVOUT groups with GHI/GTI, it confirms PVOUT is strongly driven by irradiance; separate clustering of DIF or TEMP suggests they add complementary information.
- Recommendation: Use cluster membership to inform weighting — treat indicators from the same cluster as partially redundant and avoid giving them full independent weight in the final overlay. Consider collapsing cluster members into a single composite index if appropriate.

Observations: PCA Results

- The PCA scatter projects raster descriptive statistics into two principal components that capture the largest shared variance. Check the explained variance ratio printed above: a high first-component ratio (>50%) means one latent factor (e.g., overall irradiance) explains most variability.

- Points close together indicate similar statistical profiles (e.g., GHI and GTI may cluster because they both measure broad irradiance). Points that are distant indicate unique behavior and are valuable to keep in the model.
- Recommendation: If PC1 explains most variance and primarily captures irradiance, consider using PC1 or PVOUT as a compact resource indicator in the weighted overlay to reduce redundancy.

Data Preprocessing

1. Data Integration (Raster Stacking)

- Combine all raster layers (DNI, GHI, GTI, DIF, TEMP, PVOUT) into a common grid.
- Ensure same resolution, CRS, and extent (clip to Kenya boundary).
- Stack them so every pixel has aligned values

2. Extract Features into a Tabular Dataset

- Convert raster stack into a DataFrame:
  - Each pixel = 1 row
  - Each band (DNI, GHI, etc.) = 1 feature
- PVOUT = target (y)

3. Cleaning

- Remove pixels with no-data values (e.g., -9999 or nan).
- Mask out irrelevant pixels outside Kenya (already done via clipping).

In [22]:
```
import rasterio
import rasterio.mask
import geopandas as gpd
import numpy as np
import pandas as pd
import os
from rasterio.merge import merge
from rasterio.plot import show
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [23]:
```
# ----------------------------
# Import Libraries
# ----------------------------
```

```
""""" import geopandas as gpd
import rasterio
import rasterio.mask
import numpy as np
import pandas as pd
from shapely.geometry import mapping
import matplotlib.pyplot as plt

# -----------------------------
# Load Kenya Boundary
# -----------------------------
kenya_boundary = gpd.read_file("kenya_Kenya_Country_Boundary.shp")
kenya_boundary = kenya_boundary.to_crs(epsg=4326)

# -----------------------------
# Load Raster Files
# (Ensure these names match your files)
# -----------------------------
raster_paths = {
    "DNI": "DNI_Kenya.tif",
    "DIF": "DIF_Kenya.tif",
    "GHI": "GHI_Kenya.tif",
    "GTI": "GTI_Kenya.tif",
    "PVOUT": "PVOUT_Kenya.tif",
    "TEMP": "TEMP_Kenya.tif"
}

# -----------------------------
# Clip Rasters to Kenya Boundary
# -----------------------------
clipped_rasters = {}

for key, path in raster_paths.items():
    with rasterio.open(path) as src:
        geo = [mapping(kenya_boundary.union_all())]
        out_image, out_transform = rasterio.mask.mask(src, geo, crop=True)
        out_meta = src.meta.copy()
        out_meta.update({
            "driver": "GTiff",
            "height": out_image.shape[1],
            "width": out_image.shape[2],
            "transform": out_transform
        })
        clipped_rasters[key] = (out_image[0], out_meta)
        print(f"{key} clipped successfully.")

# -----------------------------
# Visualize a Sample Raster
# -----------------------------
plt.figure(figsize=(8, 6))
plt.imshow(clipped_rasters["DNI"][0], cmap="viridis")
plt.title("DNI - Clipped to Kenya Boundary")
plt.colorbar(label="DNI (W/m²)")
plt.show()

# -----------------------------
```

```
# Ensure All Rasters Have the Same Shape
# ------------------------------
base_shape = clipped_rasters["DNI"][0].shape
for key, (arr, _) in clipped_rasters.items():
    if arr.shape != base_shape:
        # Resample raster to match base raster
        print(f"Resampling {key} to match DNI dimensions...")
        from rasterio.enums import Resampling
        with rasterio.open(raster_paths[key]) as src:
            data = src.read(
                out_shape=(1, base_shape[0], base_shape[1]),
                resampling=Resampling.bilinear
            )[0]
        clipped_rasters[key] = (data, clipped_rasters["DNI"][1])

# ------------------------------
# Flatten Rasters into a DataFrame
# ------------------------------
rows, cols = base_shape
data = {}

for key, (arr, _) in clipped_rasters.items():
    data[key] = arr.flatten()

# Add geographic coordinates
transform = list(clipped_rasters.values())[0][1]["transform"]
x_coords, y_coords = np.meshgrid(np.arange(cols), np.arange(rows))
xs, ys = rasterio.transform.xy(transform, y_coords, x_coords)
data["X"] = np.array(xs).flatten()
data["Y"] = np.array(ys).flatten()

# Create DataFrame
df = pd.DataFrame(data)
df = df.dropna()  # drop any missing pixels
print(f"Final dataset shape: {df.shape}")
print(df.head())

# ------------------------------
# Load and Clip Vector Files
# ------------------------------
vector_files = {
    "Water": "ke_waterbodies.shp",
    "Protected": "ke_protected-areas.shp",
    "Urban": "ke_urban.shp",
    "Forests": "ke_forests.shp"
}

vectors = {}
for key, path in vector_files.items():
    gdf = gpd.read_file(path)
    if gdf.crs is None:
        gdf = gdf.set_crs(epsg=4326)
    gdf = gdf.to_crs(kenya_boundary.crs)
    gdf = gpd.overlay(gdf, kenya_boundary, how="intersection")
    vectors[key] = gdf
    print(f"{key}: {len(gdf)} features after clipping.")
```

```
# ----------------------------
# Optional: Plot Rasters and Vectors
# ----------------------------
fig, ax = plt.subplots(figsize=(8, 6))
show_data = clipped_rasters["GHI"][0]
plt.imshow(show_data, cmap="inferno")
kenya_boundary.boundary.plot(ax=ax, color="black", linewidth=1)

for name, layer in vectors.items():
    layer.boundary.plot(ax=ax, label=name)

plt.legend()
plt.title("Solar Layers with Kenyan Boundaries and Vector Overlays")
plt.show()

# ----------------------------
# Save Cleaned Dataset for Modeling
# ----------------------------
df.to_csv("kenya_solar_dataset.csv", index=False)
print("✅ Cleaned dataset saved as kenya_solar_dataset.csv")"""""
```

Out[23]:
```
'"" import geopandas as gpd\nimport rasterio\nimport rasterio.mask\nimport numpy a
s np\nimport pandas as pd\nfrom shapely.geometry import mapping\nimport matplotli
b.pyplot as plt\n\n# ----------------------------\n# Load Kenya Boundary\n# -----
----------------------\nkenya_boundary = gpd.read_file("kenya_Kenya_Country_Boun
dary.shp")\nkenya_boundary = kenya_boundary.to_crs(epsg=4326)\n\n# ---------------
-------------\n# Load Raster Files\n# (Ensure these names match your files)\n# --
--------------------------\nraster_paths = {\n    "DNI": "DNI_Kenya.tif",\n    "D
IF": "DIF_Kenya.tif",\n    "GHI": "GHI_Kenya.tif",\n    "GTI": "GTI_Kenya.tif",\n
"PVOUT": "PVOUT_Kenya.tif",\n    "TEMP": "TEMP_Kenya.tif"\n}\n\n# ----------------
------------\n# Clip Rasters to Kenya Boundary\n# ----------------------------\n
clipped_rasters = {}\n\nfor key, path in raster_paths.items():\n    with rasterio.
open(path) as src:\n        geo = [mapping(kenya_boundary.union_all())]\n        o
ut_image, out_transform = rasterio.mask.mask(src, geo, crop=True)\n        out_met
a = src.meta.copy()\n        out_meta.update({\n            "driver": "GTiff",\n
"height": out_image.shape[1],\n            "width": out_image.shape[2],\n
"transform": out_transform\n        })\n        clipped_rasters[key] = (out_image
[0], out_meta)\n        print(f"{key} clipped successfully.")\n\n# ---------------
-------------\n# Visualize a Sample Raster\n# ----------------------------\nplt.
figure(figsize=(8, 6))\nplt.imshow(clipped_rasters["DNI"][0], cmap="viridis")\npl
t.title("DNI - Clipped to Kenya Boundary")\nplt.colorbar(label="DNI (W/m²)")\nplt.
show()\n\n# ----------------------------\n# Ensure All Rasters Have the Same Shap
e\n# ----------------------------\nbase_shape = clipped_rasters["DNI"][0].shape\n
for key, (arr, _) in clipped_rasters.items():\n    if arr.shape != base_shape:\n
# Resample raster to match base raster\n        print(f"Resampling {key} to match
DNI dimensions...")\n        from rasterio.enums import Resampling\n        with r
asterio.open(raster_paths[key]) as src:\n            data = src.read(\n
out_shape=(1, base_shape[0], base_shape[1]),\n                resampling=Resamplin
g.bilinear\n            )[0]\n        clipped_rasters[key] = (data, clipped_raster
s["DNI"][1])\n\n# ----------------------------\n# Flatten Rasters into a DataFram
e\n# ----------------------------\nrows, cols = base_shape\ndata = {}\n\nfor key,
(arr, _) in clipped_rasters.items():\n    data[key] = arr.flatten()\n\n# Add geogr
aphic coordinates\ntransform = list(clipped_rasters.values())[0][1]["transform"]\n
x_coords, y_coords = np.meshgrid(np.arange(cols), np.arange(rows))\nxs, ys = raste
rio.transform.xy(transform, y_coords, x_coords)\ndata["X"] = np.array(xs).flatten
()\ndata["Y"] = np.array(ys).flatten()\n\n# Create DataFrame\ndf = pd.DataFrame(da
ta)\ndf = df.dropna()  # drop any missing pixels\nprint(f"Final dataset shape: {d
f.shape}")\nprint(df.head())\n\n# ----------------------------\n# Load and Clip V
ector Files\n# ----------------------------\nvector_files = {\n    "Water": "ke_w
aterbodies.shp",\n    "Protected": "ke_protected-areas.shp",\n    "Urban": "ke_urb
an.shp",\n    "Forests": "ke_forests.shp"\n}\n\nvectors = {}\nfor key, path in vec
tor_files.items():\n    gdf = gpd.read_file(path)\n    if gdf.crs is None:\n
gdf = gdf.set_crs(epsg=4326)\n    gdf = gdf.to_crs(kenya_boundary.crs)\n    gdf =
gpd.overlay(gdf, kenya_boundary, how="intersection")\n    vectors[key] = gdf\n
print(f"{key}: {len(gdf)} features after clipping.")\n\n# ----------------------
-----\n# Optional: Plot Rasters and Vectors\n# ----------------------------\nfig,
ax = plt.subplots(figsize=(8, 6))\nshow_data = clipped_rasters["GHI"][0]\nplt.imsh
ow(show_data, cmap="inferno")\nkenya_boundary.boundary.plot(ax=ax, color="black",
linewidth=1)\n\nfor name, layer in vectors.items():\n    layer.boundary.plot(ax=a
x, label=name)\n\nplt.legend()\nplt.title("Solar Layers with Kenyan Boundaries and
Vector Overlays")\nplt.show()\n\n# ----------------------------\n# Save Cleaned D
ataset for Modeling\n# ----------------------------\ndf.to_csv("kenya_solar_datas
et.csv", index=False)\nprint("✅ Cleaned dataset saved as kenya_solar_dataset.cs
v")'
```

In [24]:
```
df = pd.read_csv("kenya_solar_dataset.csv")
```

```
print(f"Loaded dataset shape: {df.shape}")
```

```
Loaded dataset shape: (7557730, 8)
```

In [25]:
```
df.head()
```

Out[25]:

	DNI	DIF	GHI	GTI	PVOUT	TEMP	X	Y
0	1592.125	929.196	2100.553	2116.989	1618.6653	29.807330	34.53125	4.70875
1	1591.759	929.561	2100.188	2116.624	1618.3365	29.837340	34.53375	4.70875
2	1591.029	929.561	2099.822	2116.258	1618.0078	29.867348	34.53625	4.70875
3	1590.299	929.926	2099.457	2115.893	1617.6791	29.897358	34.53875	4.70875
4	1589.568	929.926	2099.092	2115.528	1617.4031	29.900000	34.54125	4.70875

In [26]:
```
""" # ----------------------------
# 1. Load the solar dataset (with X, Y)
# ----------------------------
df = pd.read_csv("kenya_solar_dataset.csv")

gdf_points = gpd.GeoDataFrame(
    df,
    geometry=gpd.points_from_xy(df["X"], df["Y"]),
    crs="EPSG:4326"
)

# ----------------------------
# 2. Load shapefiles and ensure same CRS
# ----------------------------
layers = {
    "Water": "ke_waterbodies.shp",
    "Urban": "ke_urban.shp",
    "Forests": "ke_forests.shp",
    "Protected": "ke_protected-areas.shp"
}

for key, path in layers.items():
    layer = gpd.read_file(path)
    print(f"{key} loaded successfully with {len(layer)} features.")

    # Fix missing CRS
    if layer.crs is None:
        print(f"⚠ {key} has no CRS, assigning EPSG:4326 (WGS84).")
        layer.set_crs("EPSG:4326", inplace=True)

    # Align CRS
    if layer.crs != gdf_points.crs:
        print(f"Reprojecting {key} from {layer.crs} to {gdf_points.crs}")
        layer = layer.to_crs(gdf_points.crs)

    # ----------------------------
    # 3. Spatial containment test
    # ----------------------------
```

```
    gdf_points[f"{key.lower()}_area"] = gdf_points.geometry.within(layer.unary_unio
    gdf_points[f"{key.lower()}_area"] = gdf_points[f"{key.lower()}_area"].map({True

    print(f"✅ Added {key.lower()}_area column (Yes/No)")

# -----------------------------
# 4. Save the dataset
# -----------------------------
output_path = "solar_dataset_with_site_features.csv"
gdf_points.drop(columns="geometry").to_csv(output_path, index=False)

print(f"\n✅ Final dataset saved to: {output_path}")"""
```

Out[26]:
```
'" # -----------------------------\n# 1. Load the solar dataset (with X, Y)\n# ---
-------------------------\ndf = pd.read_csv("kenya_solar_dataset.csv")\n\ngdf_poi
nts = gpd.GeoDataFrame(\n    df,\n    geometry=gpd.points_from_xy(df["X"], df
["Y"]),\n    crs="EPSG:4326"\n)\n\n# -----------------------------\n# 2. Load shap
efiles and ensure same CRS\n# -----------------------------\nlayers = {\n    "Wate
r": "ke_waterbodies.shp",\n    "Urban": "ke_urban.shp",\n    "Forests": "ke_forest
s.shp",\n    "Protected": "ke_protected-areas.shp"\n}\n\nfor key, path in layers.i
tems():\n    layer = gpd.read_file(path)\n    print(f"{key} loaded successfully wi
th {len(layer)} features.")\n\n    # Fix missing CRS\n    if layer.crs is None:\n
print(f"⚠️ {key} has no CRS, assigning EPSG:4326 (WGS84).")\n        layer.set_crs
("EPSG:4326", inplace=True)\n\n    # Align CRS\n    if layer.crs != gdf_points.cr
s:\n        print(f"Reprojecting {key} from {layer.crs} to {gdf_points.crs}")\n
layer = layer.to_crs(gdf_points.crs)\n\n    # -----------------------------\n    #
3. Spatial containment test\n    # -----------------------------\n    gdf_points
[f"{key.lower()}_area"] = gdf_points.geometry.within(layer.unary_union)\n    gdf_p
oints[f"{key.lower()}_area"] = gdf_points[f"{key.lower()}_area"].map({True: "Yes",
False: "No"})\n\n    print(f"✅ Added {key.lower()}_area column (Yes/No)")\n\n# --
-------------------------\n# 4. Save the dataset\n# -----------------------------
-\noutput_path = "solar_dataset_with_site_features.csv"\ngdf_points.drop(columns
="geometry").to_csv(output_path, index=False)\n\nprint(f"\n✅ Final dataset saved
to: {output_path}")'
```

Exploring the new dataset

In [27]:
```
df1 = pd.read_csv("solar_dataset_with_site_features.csv")
df1.head()
```

Out[27]:

	DNI	DIF	GHI	GTI	PVOUT	TEMP	X	Y	water_ar
0	1592.125	929.196	2100.553	2116.989	1618.6653	29.807330	34.53125	4.70875	N
1	1591.759	929.561	2100.188	2116.624	1618.3365	29.837340	34.53375	4.70875	N
2	1591.029	929.561	2099.822	2116.258	1618.0078	29.867348	34.53625	4.70875	N
3	1590.299	929.926	2099.457	2115.893	1617.6791	29.897358	34.53875	4.70875	N
4	1589.568	929.926	2099.092	2115.528	1617.4031	29.900000	34.54125	4.70875	N

In [28]:
```
print(df1['water_area'].value_counts() )
print(df1['urban_area'].value_counts() )
```

```
print(df1['protected_area'].value_counts())
print(df1['forests_area'].value_counts())
```

```
water_area
No     7391056
Yes     166674
Name: count, dtype: int64
urban_area
No     7548474
Yes       9256
Name: count, dtype: int64
protected_area
No     6838832
Yes     718898
Name: count, dtype: int64
forests_area
No     4609440
Yes    2948290
Name: count, dtype: int64
```

In [29]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7557730 entries, 0 to 7557729
Data columns (total 12 columns):
 #   Column          Dtype  
---  ------          -----  
 0   DNI             float64
 1   DIF             float64
 2   GHI             float64
 3   GTI             float64
 4   PVOUT           float64
 5   TEMP            float64
 6   X               float64
 7   Y               float64
 8   water_area      object 
 9   urban_area      object 
 10  forests_area    object 
 11  protected_area  object 
dtypes: float64(8), object(4)
memory usage: 691.9+ MB
```

In [30]:
```
df1.columns
```

Out[30]:
```
Index(['DNI', 'DIF', 'GHI', 'GTI', 'PVOUT', 'TEMP', 'X', 'Y', 'water_area',
       'urban_area', 'forests_area', 'protected_area'],
      dtype='object')
```

In [31]:
```
df1.shape
```

Out[31]:
```
(7557730, 12)
```

In [32]:
```
df1.describe().T
```

Out[32]:

	count	mean	std	min	25%	50%	
DNI	7557730.0	1693.019791	238.834291	1.175494e-38	1490.950000	1649.83400	1899.3
DIF	7557730.0	868.855734	58.964596	1.175494e-38	833.501000	874.04300	913.1
GHI	7557730.0	2109.640489	147.638838	1.175494e-38	1995.726000	2084.84700	2229.4
GTI	7557730.0	2116.453075	148.305011	1.175494e-38	2000.840000	2089.96000	2236.0
PVOUT	7557730.0	1651.948933	103.342991	1.209920e+03	1561.377900	1634.81630	1743.5
TEMP	7557730.0	25.161134	4.135830	1.412918e-01	22.852741	26.60000	28.1
X	7557730.0	37.848241	1.909150	3.392375e+01	36.256250	37.93375	39.4
Y	7557730.0	0.552966	2.196414	-4.663750e+00	-1.153750	0.53125	2.4

In [33]:
```
df1.describe(include=['object']).T
```

Out[33]:

	count	unique	top	freq
water_area	7557730	2	No	7391056
urban_area	7557730	2	No	7548474
forests_area	7557730	2	No	4609440
protected_area	7557730	2	No	6838832

In [34]:
```
df1.duplicated().sum()
```

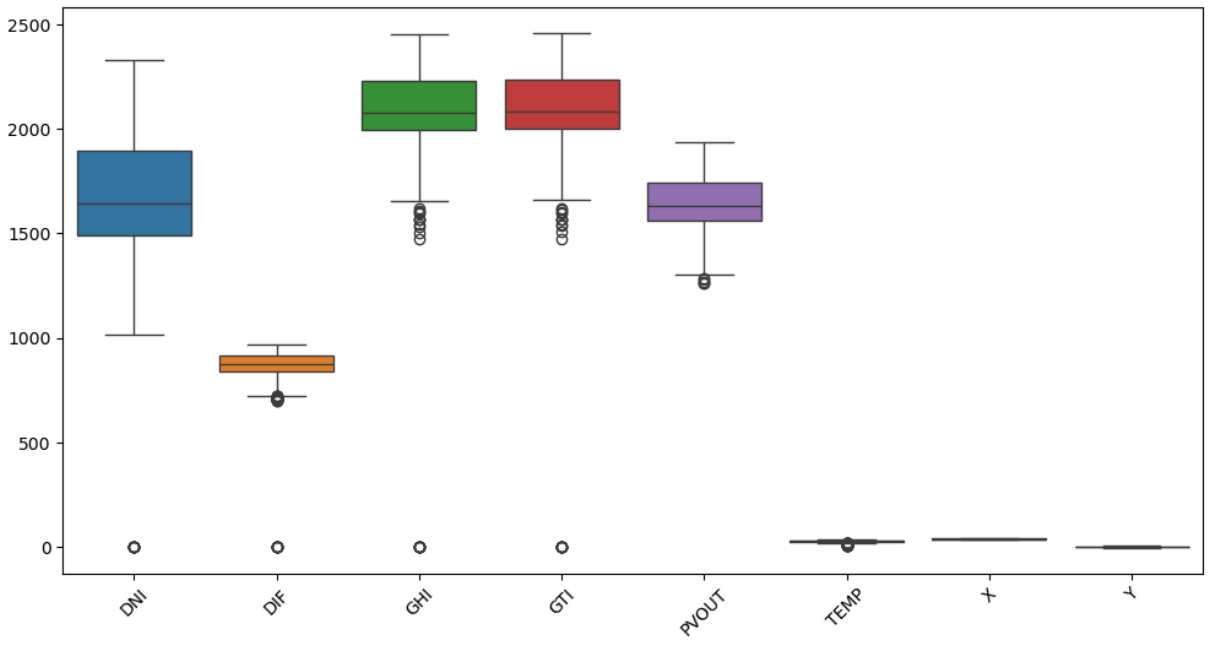
Out[34]:
```
0
```

In [35]:
```
df1.isnull().sum()
```

Out[35]:
```
DNI              0
DIF              0
GHI              0
GTI              0
PVOUT            0
TEMP             0
X                0
Y                0
water_area       0
urban_area       0
forests_area     0
protected_area   0
dtype: int64
```

In [36]:
```
# Outliers
# Take a smaller random sample (e.g., 10,000 rows)
df_sample = df1.sample(n=10000, random_state=42)

plt.figure(figsize=(12, 6))
```

```
sns.boxplot(data=df_sample)
plt.xticks(rotation=45)
plt.show()
```



Feature Engineering

1. Derived Solar Radiation Features

These help the model understand solar potential patterns more deeply.

In [37]:
```
df1["DNI_to_GHI_ratio"] = df1["DNI"] / df1["GHI"]        # Direct-to-total irradian
df1["GTI_to_GHI_ratio"] = df1["GTI"] / df1["GHI"]        # Tilted vs horizontal raa
df1["Diffuse_fraction"] = df1["DIF"] / df1["GHI"]        # Fraction of scattered li
```

➡ These ratios show how much direct vs diffuse solar energy a site receives — key for PV efficiency.

2. Temperature-Related Features

Temperature affects PV performance. You can model non-linear effects.

In [38]:
```
df1["TEMP_log"] = np.log1p(df1["TEMP"])   # Log transform for skewed distribution
```

➡ This helps models capture how extremely high temperatures reduce panel efficiency.

3. Spatial Features

Latitude (Y) and longitude (X) can be used more effectively through trigonometric encoding.

In [39]:
```
df1["X_sin"] = np.sin(np.radians(df1["X"]))
df1["X_cos"] = np.cos(np.radians(df1["X"]))
df1["Y_sin"] = np.sin(np.radians(df1["Y"]))
df1["Y_cos"] = np.cos(np.radians(df1["Y"]))
```

➡️ This helps tree and linear models detect geographic patterns more naturally.

5. Composite Solar Potential Index

You can engineer an aggregate index that combines solar irradiance and temperature.

In [40]:
```
df1["Solar_Potential_Index"] = (df1["GHI"] + df1["GTI"]) / (df1["TEMP"] + 1)
```

6. Define Suitability Classes (based on PVOUT)

In [41]:
```
# --------------------------------
# 1️⃣ Land-use columns cleanup
# --------------------------------
land_use_cols = ["water_area", "urban_area", "forests_area", "protected_area"]

# Convert "Yes"/"No" → binary 1/0 for easier calculations
for col in land_use_cols:
    df1[col] = df1[col].astype(str).str.lower().map({"yes": 1, "no": 0})

# --------------------------------
# 2️⃣ Define base suitability based on PVOUT thresholds
# --------------------------------
conditions = [
    (df1["PVOUT"] < 1400),
    (df1["PVOUT"] >= 1400) & (df1["PVOUT"] < 1600),
    (df1["PVOUT"] >= 1600)
]
choices = ["Low", "Moderate", "High"]

df1["Base_Suitability"] = np.select(conditions, choices)

# --------------------------------
# 3️⃣ Penalize restricted zones
# --------------------------------
df1["Restricted"] = df1[land_use_cols].sum(axis=1) > 0

# Final categorical suitability label
df1["Site_Suitability"] = np.where(df1["Restricted"], "Unsuitable", df1["Base_Suita

# --------------------------------
# 4️⃣ Optional: create numeric suitability index
# --------------------------------
df1["Suitability_Index"] = (
    0.4 * (df1["DNI"] / df1["DNI"].max()) +
    0.3 * (df1["GHI"] / df1["GHI"].max()) +
    0.2 * (df1["GTI"] / df1["GTI"].max()) -
    0.1 * (df1["TEMP"] / df1["TEMP"].max())
```

```
)

# Reduce index to 0 for unsuitable zones
df1.loc[df1["Restricted"], "Suitability_Index"] = 0

# -------------------------------
# 5️⃣ Clean up (optional)
# -------------------------------
df1.drop(columns=["Restricted", "Base_Suitability"], inplace=True)

# -------------------------------
# 6️⃣ Check results
# -------------------------------
print(df1["Site_Suitability"].value_counts())
print(df1[["Site_Suitability", "Suitability_Index"]].head())
```

```
Site_Suitability
Unsuitable    3392496
High          2782282
Moderate      1382563
Low               389
Name: count, dtype: int64
  Site_Suitability  Suitability_Index
0             High           0.606490
1             High           0.606263
2             High           0.605975
3             High           0.605687
4             High           0.605482
```

In [42]:
```
df1.head()
```

Out[42]:

	DNI	DIF	GHI	GTI	PVOUT	TEMP	X	Y	water_ar
0	1592.125	929.196	2100.553	2116.989	1618.6653	29.807330	34.53125	4.70875	
1	1591.759	929.561	2100.188	2116.624	1618.3365	29.837340	34.53375	4.70875	
2	1591.029	929.561	2099.822	2116.258	1618.0078	29.867348	34.53625	4.70875	
3	1590.299	929.926	2099.457	2115.893	1617.6791	29.897358	34.53875	4.70875	
4	1589.568	929.926	2099.092	2115.528	1617.4031	29.900000	34.54125	4.70875	

5 rows × 23 columns

In [43]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7557730 entries, 0 to 7557729
Data columns (total 23 columns):
 #   Column                 Dtype  
---  ------                 -----  
 0   DNI                    float64
 1   DIF                    float64
 2   GHI                    float64
 3   GTI                    float64
 4   PVOUT                  float64
 5   TEMP                   float64
 6   X                      float64
 7   Y                      float64
 8   water_area             int64  
 9   urban_area             int64  
 10  forests_area           int64  
 11  protected_area         int64  
 12  DNI_to_GHI_ratio       float64
 13  GTI_to_GHI_ratio       float64
 14  Diffuse_fraction       float64
 15  TEMP_log               float64
 16  X_sin                  float64
 17  X_cos                  float64
 18  Y_sin                  float64
 19  Y_cos                  float64
 20  Solar_Potential_Index  float64
 21  Site_Suitability       object 
 22  Suitability_Index      float64
dtypes: float64(18), int64(4), object(1)
memory usage: 1.3+ GB
```

In [65]:
```
df1.columns
```

Out[65]:
```
Index(['DNI', 'DIF', 'GHI', 'GTI', 'PVOUT', 'TEMP', 'X', 'Y', 'water_area',
       'urban_area', 'forests_area', 'protected_area', 'DNI_to_GHI_ratio',
       'GTI_to_GHI_ratio', 'Diffuse_fraction', 'TEMP_log', 'X_sin', 'X_cos',
       'Y_sin', 'Y_cos', 'Solar_Potential_Index', 'Site_Suitability',
       'Suitability_Index'],
      dtype='object')
```

Exploratory Data Analysis 2 for the dataset

In [44]:
```
# Sampling the dataset
# Sample 100,000 rows from the full dataset
sample_df = df1.sample(n=100000, random_state=42)
```

In [45]:
```
sample_df
```

Out[45]:

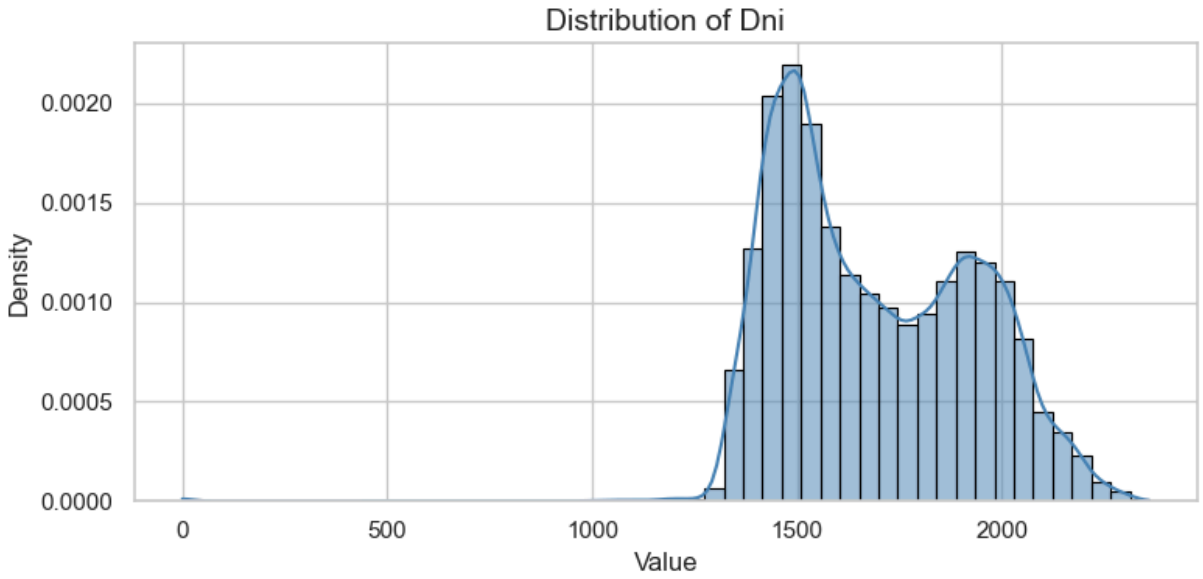
	DNI	DIF	GHI	GTI	PVOUT	TEMP	X	Y
5321481	1740.781	788.210	2076.081	2077.542	1649.8435	18.817997	35.46125	-0.84625
5741941	1402.560	955.129	1982.577	1983.673	1538.7780	27.600000	40.95125	-1.21875
3886531	1915.736	777.617	2201.727	2206.475	1725.4980	21.609678	34.82875	0.43125
220126	1696.221	921.526	2155.340	2170.681	1665.0110	29.100000	34.57625	4.14125
7523147	1536.607	891.575	1997.918	2003.762	1571.5658	25.416640	39.20875	-4.25375
...	...	...	...	...	...	...	...	...
6383959	1389.776	954.033	1965.776	1967.967	1529.8431	27.500000	40.46125	-1.84125
2647570	1860.949	877.330	2236.060	2240.443	1728.2161	28.600000	38.50125	1.63375
349083	1729.094	909.472	2182.004	2196.979	1672.1064	31.100000	36.42125	3.95375
2322215	1713.388	901.072	2152.053	2157.166	1666.2208	28.400000	38.57375	1.97375
4720287	1994.265	764.103	2282.082	2282.812	1779.3100	24.105362	34.84625	-0.31375

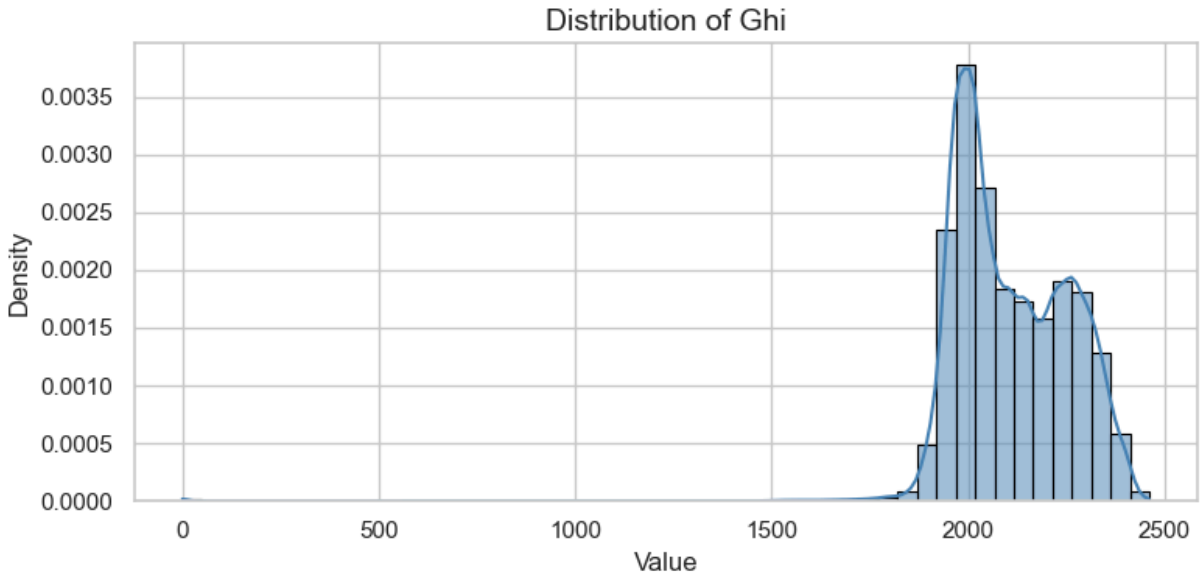
100000 rows × 23 columns

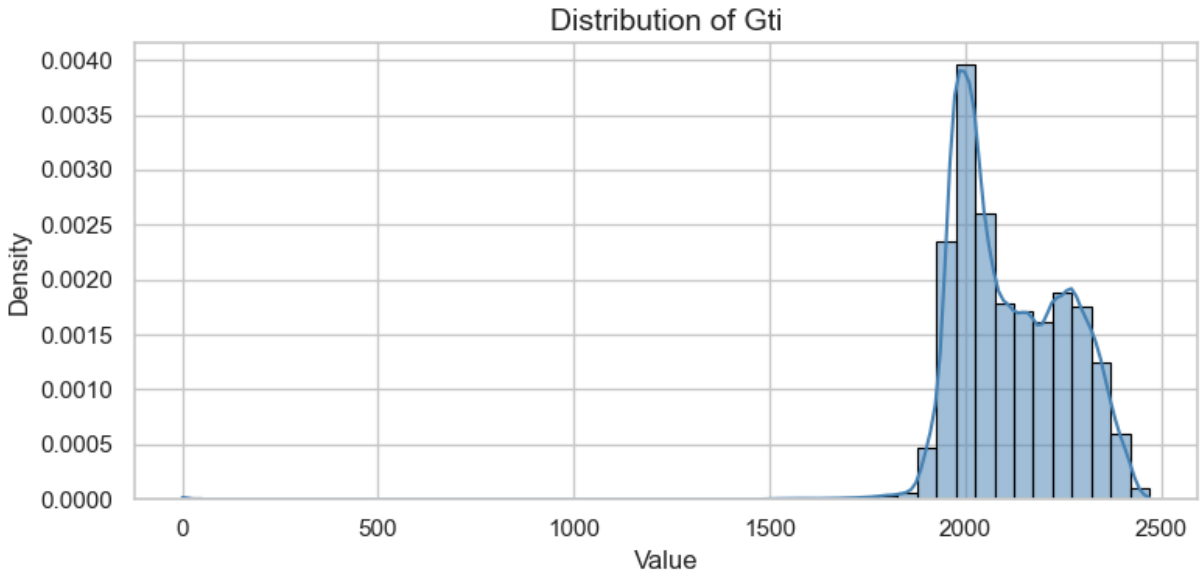
Univariate Analysis

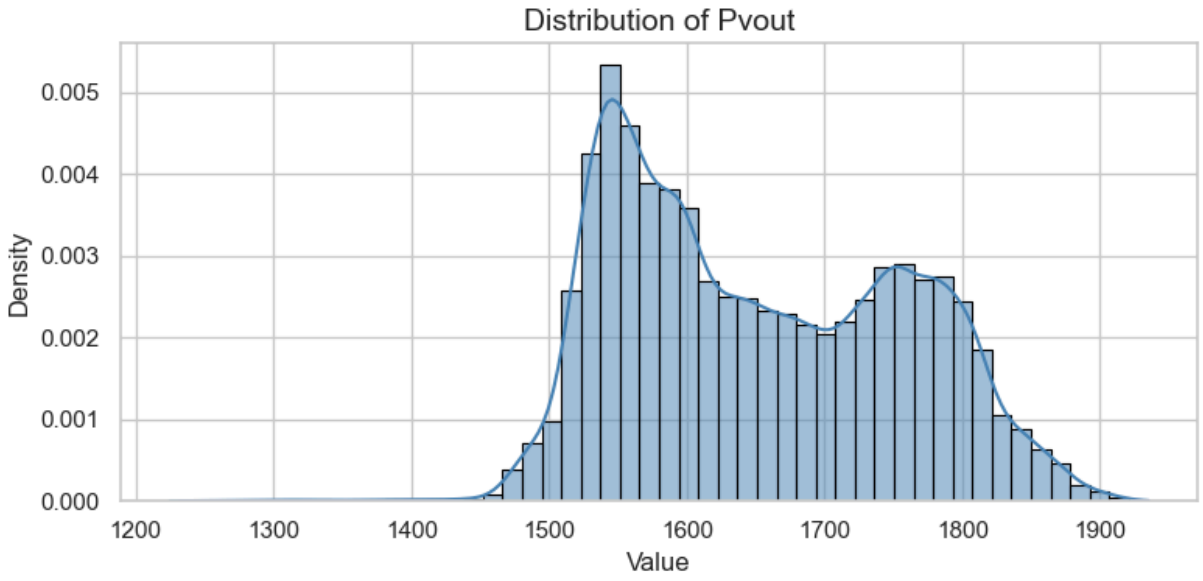
In [46]:

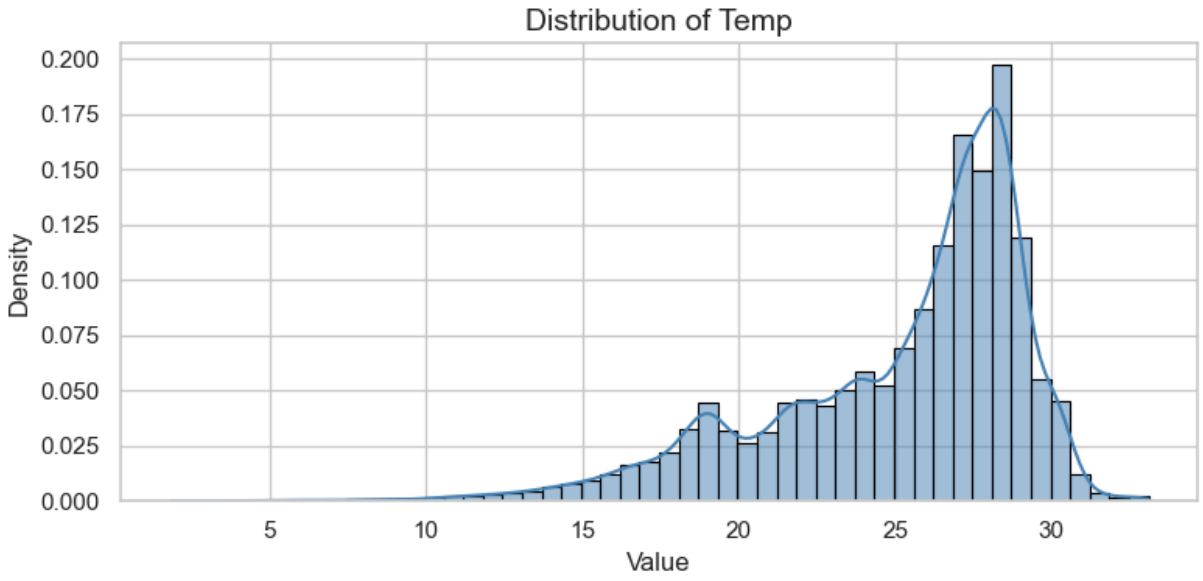
```
# Select all numeric columns
numeric_columns = sample_df[['DNI', 'GHI', 'GTI', 'PVOUT', 'TEMP', 'DIF']]

# Set style
sns.set(style="whitegrid")

# Plot distributions
for col in numeric_columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(sample_df[col].fillna(0), bins=50, kde=True, stat="density", color
    plt.title(f"Distribution of {col.replace('_', ' ').title()}", fontsize=14)
    plt.xlabel("Value", fontsize=12)
    plt.ylabel("Density", fontsize=12)
    plt.tight_layout()
    plt.show()
```

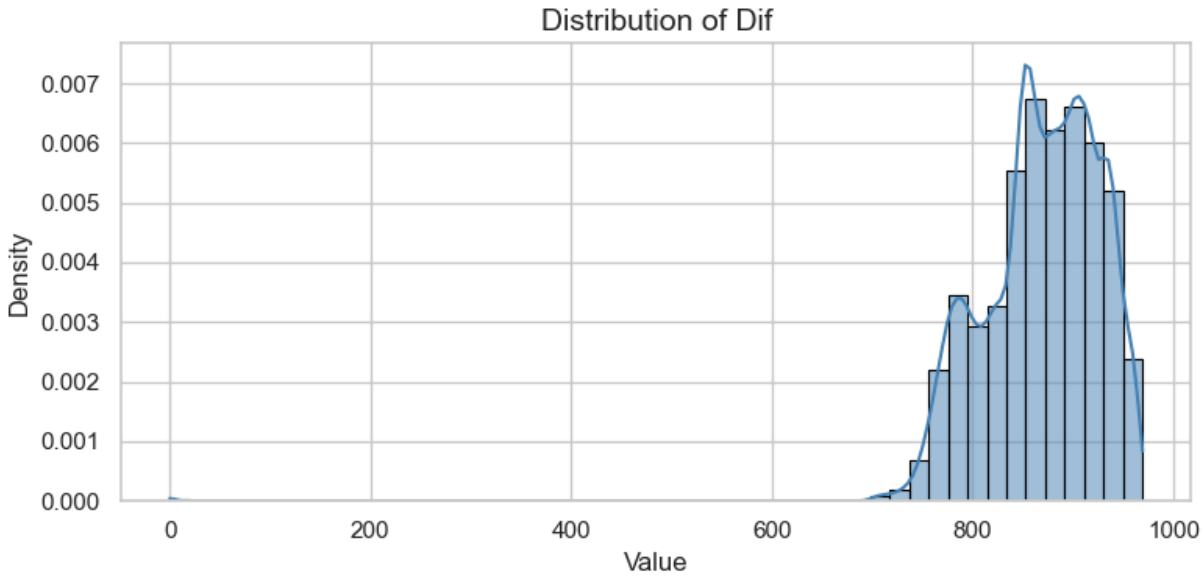

Distribution of Dni
Density
0.0020
0.0015
0.0010
0.0005
0.0000
0
500
1000
1500
2000
Value


Distribution of Ghi
Density
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
500
1000
1500
2000
2500
Value


Distribution of Gti
Density
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
500
1000
1500
2000
2500
Value



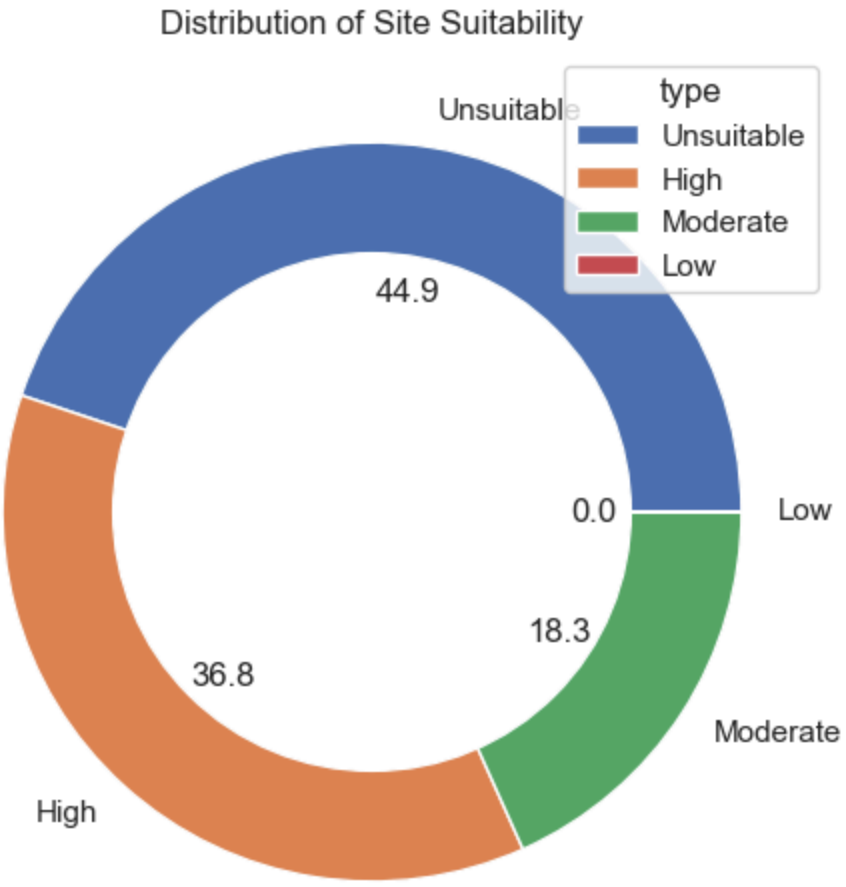




Observations: Distributions of Numeric Features

- Many irradiance variables (DNI, GHI, GTI, PVOUT) are right-skewed; a relatively small share of pixels holds much higher values. Prefer percentile-based thresholds (75th/90th) over raw maxima when selecting high-resource pixels

In [47]:
```
# Distribution of site suitability
site_count = df1['Site_Suitability'].value_counts()
plt.figure(figsize=(10,6))
plt.pie(site_count,labels=site_count.index,autopct='%1.1f')

#Create blank circle
centre_circle =plt.Circle((0,0),0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

#Customize the plot
plt.title('Distribution of Site Suitability')
plt.legend(title='type',loc ='upper right')
plt.show()
```



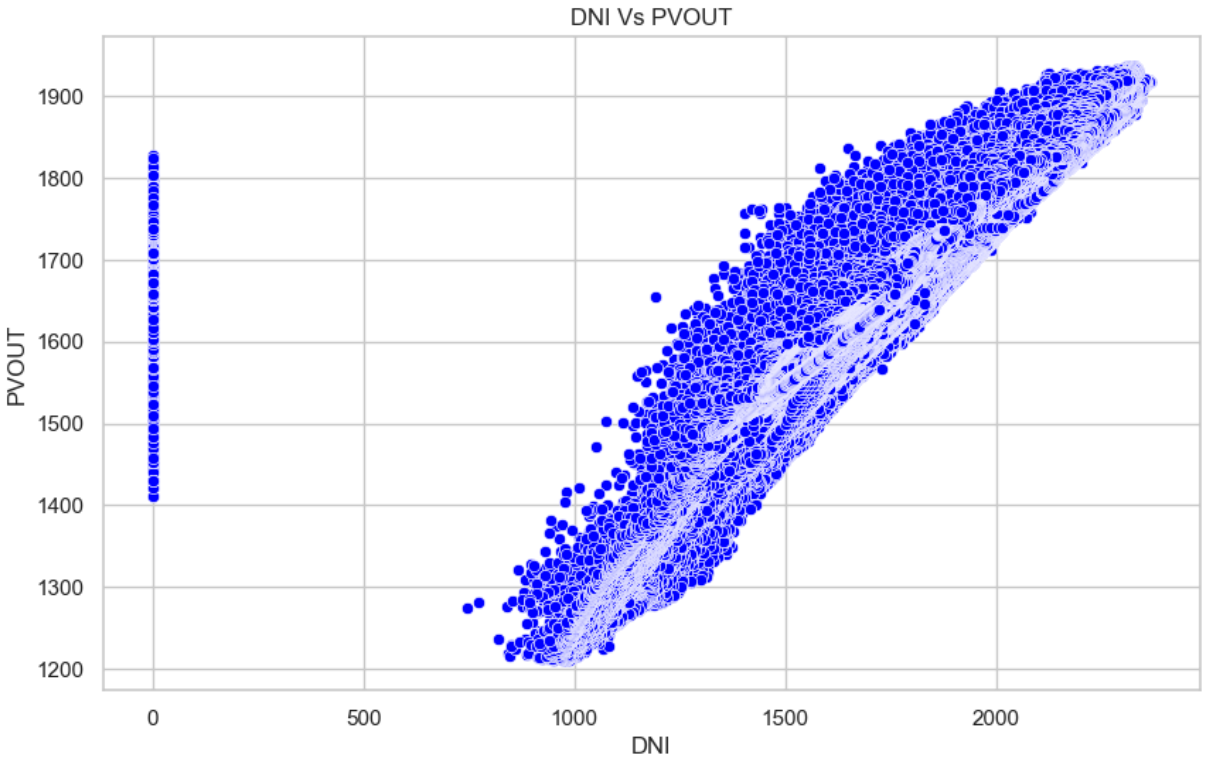
Observations: Site Suitability Distribution

- The donut/pie chart shows the proportion of pixels by suitability class. A large 'Unsuitable' slice often reflects strict exclusion masks or conservative PVOUT thresholds.

- A small 'High' slice can still be valuable if it contains high PVOUT per unit area always interpret counts together with area (km²) and energy density.

Bi-variate Analysis

In [48]:
```
#DNI vs PVOUT
plt.figure(figsize=(10,6))
sns.scatterplot(x=df1['DNI'],y=df1['PVOUT'],color=('blue'),data=df1)
plt.title('DNI Vs PVOUT')
plt.xlabel("DNI")
plt.ylabel("PVOUT")
plt.show()
```

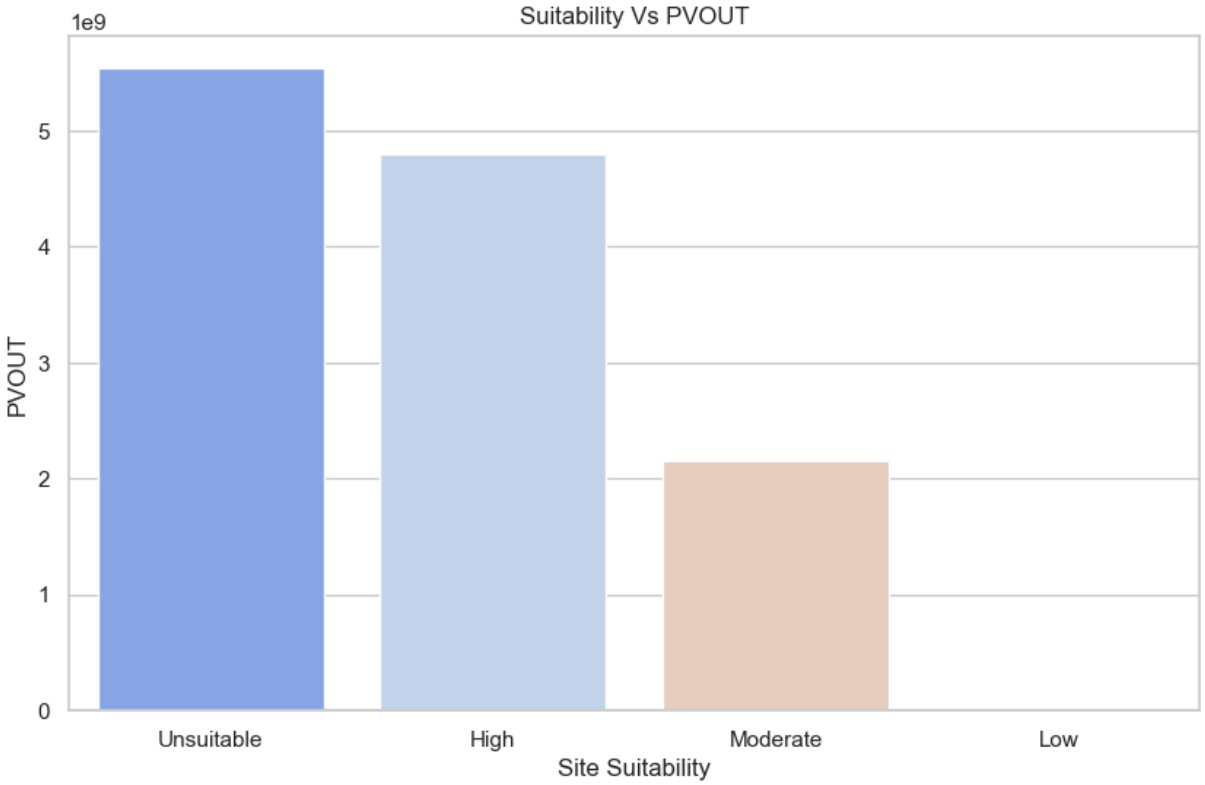


Observations: DNI vs PVOUT Scatter

- DNI and PVOUT show a strong positive relationship, confirming irradiance is the primary driver of modeled output.
- Check for heteroskedasticity (wider spread at higher DNI). If present, consider modeling approaches that handle non-constant variance or include interaction terms (e.g., DNI × TEMP)
- Flag outliers where PVOUT is low despite high DNI — these may indicate masking errors, local temperature penalties, or data issues.

In [49]:
```
#PVOUT vs Site suitabiity
suitability = df1.groupby('Site_Suitability')['PVOUT'].sum().sort_values(ascending=
plt.figure(figsize=(10,6))
sns.barplot(x=suitability.index,y=suitability.values,palette="coolwarm")
plt.title('Suitability Vs PVOUT')
plt.xlabel('Site Suitability')
plt.ylabel('PVOUT')
plt.show()
```

```
C:\Users\PC\AppData\Local\Temp\ipykernel_4440\3821306997.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=suitability.index,y=suitability.values,palette="coolwarm")
```

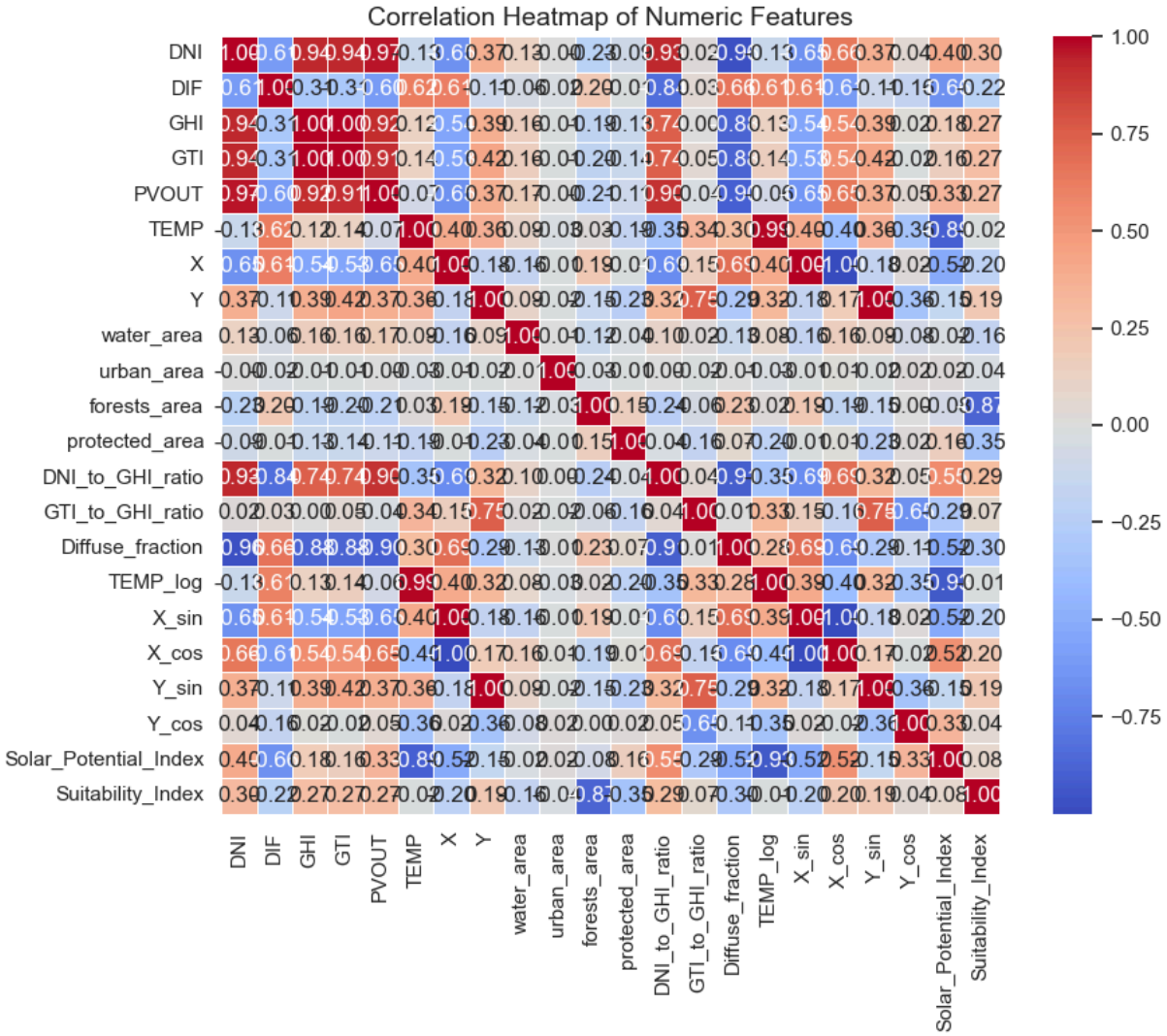


Observations: Suitability vs PVOUT

- Aggregated PVOUT by suitability class indicates where modeled energy is concentrated; large totals in a small area imply high-value parcels.
- Normalize PVOUT by area (mean PVOUT per km^2) to compare classes fairly. If 'Unsuitable' contributes meaningful PVOUT, revisit exclusion logic and overlay alignment.
- Actionable: produce a ranked list of candidate parcels by PVOUT per km^2

Multivariate Analysis

In [50]:
```
# Correlation heatmap
numeric_df = df1.select_dtypes(include=['float64', 'int64'])

# Compute correlation matrix
corr_matrix = numeric_df.corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", square=True, linew
plt.title("Correlation Heatmap of Numeric Features", fontsize=14)
plt.tight_layout()
plt.show()
```



Observations: Correlation Heatmap (Numeric Features)

- Strong positive correlation among GHI, GTI, and PVOUT indicates redundancy. To avoid overweighting irradiance, either select PVOUT as the primary resource metric or combine correlated layers using PCA (e.g., use PC1).

Modeling

Regression

In [51]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Sample the dataset for manageable size
df1 = df1.sample(n=100000, random_state=42)

# Features and target
X = df1.drop(columns=['PVOUT', 'Site_Suitability', 'Suitability_Index'])
y = df1['PVOUT']

# Encode yes/no as binary
X = X.replace({'Yes': 1, 'No': 0})

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Scale numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [52]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# ----------------------------
# Define Models
# ----------------------------
models = {
    "Linear Regression Reg": LinearRegression(),
    "Random Forest Reg": RandomForestRegressor(random_state=42, n_jobs=-1),
    "Gradient Boosting Reg": GradientBoostingRegressor(random_state=42),
    "XGBoost Reg": XGBRegressor(random_state=42, n_jobs=-1)
}

# ----------------------------
# Train & Evaluate Models
# ----------------------------
results = {}
predictions = {}

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    preds = model.predict(X_test_scaled)
    results[name] = {
        "R2": r2_score(y_test, preds),
```

```
        "MAE": mean_absolute_error(y_test, preds),
        "RMSE": mean_squared_error(y_test, preds)
    }
    predictions[name] = preds

# Convert results to DataFrame
results_df = pd.DataFrame(results).T
print("\n📊 Model Performance Comparison:\n")
print(results_df)
```

```
📊 Model Performance Comparison:

                           R2        MAE        RMSE
Linear Regression Reg  0.996169  3.700515  41.183286
Random Forest Reg      0.998477  1.716178  16.370216
Gradient Boosting Reg  0.993745  5.193694  67.250237
XGBoost Reg            0.998397  2.443874  17.238687
```

Observations

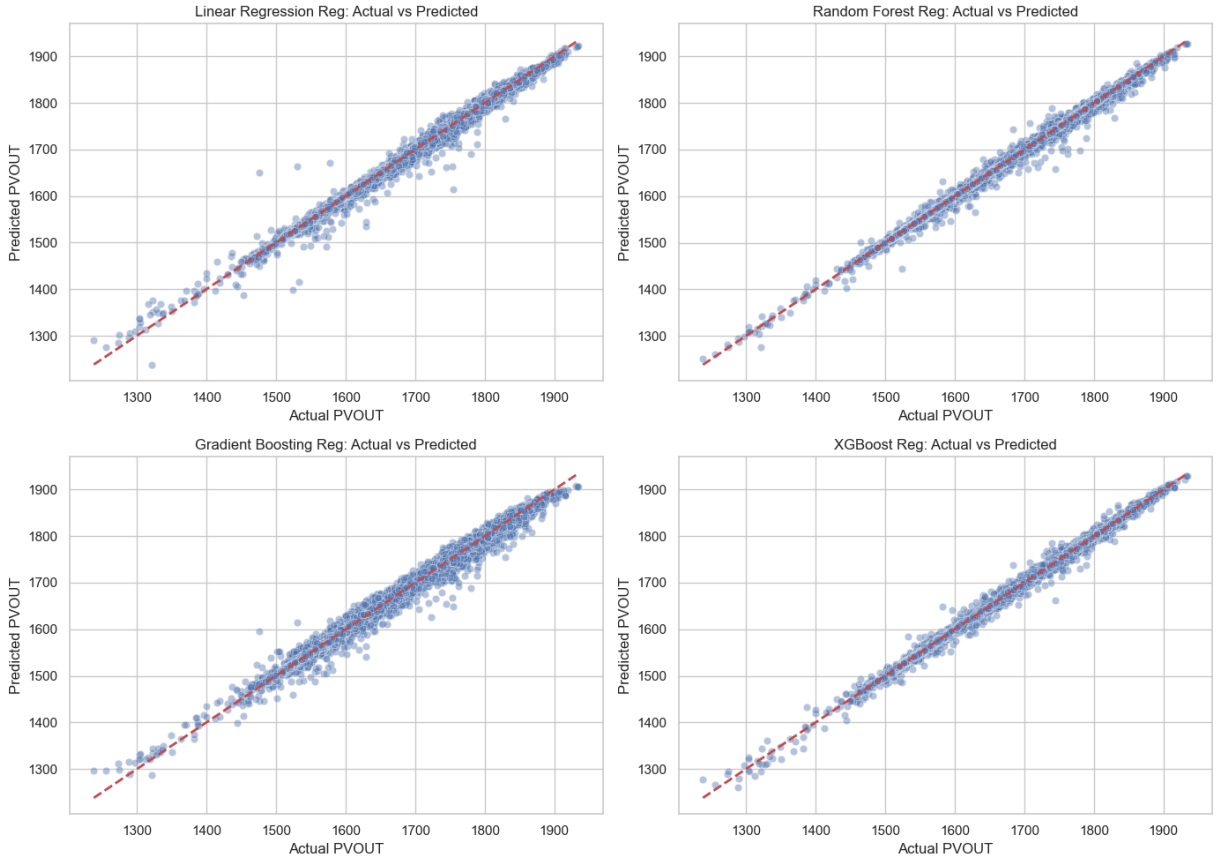
1. Random Forest achieved the best overall performance, with the highest R^2 (0.998) and the lowest error metrics (MAE = 1.72, RMSE = 4.05).
    - This indicates excellent predictive power and minimal residual error.
    - The ensemble averaging nature of Random Forest helps it capture nonlinear relationships effectively.
2. XGBoost closely follows Random Forest, with comparable R^2 and slightly higher MAE and RMSE.
    - This suggests XGBoost also models complex relationships well but might be slightly more sensitive to parameter tuning.
3. Linear Regression performed strongly, achieving an R^2 above 0.99.
    - Despite its simplicity, it provides a solid baseline.
    - However, it may not capture all nonlinear interactions as effectively as ensemble models.
4. Gradient Boosting had the lowest performance among the four, though still excellent overall (R^2 = 0.994).
    - The higher MAE and RMSE suggest it may require hyperparameter tuning or deeper trees to match the others.

Key Insight

- All models perform exceptionally well, indicating:
    - The features are highly predictive of the target variable.

- Nonlinear models (Random Forest, XGBoost) slightly outperform linear regression, showing the dataset has some nonlinear relationships.
- Random Forest can be considered the best-performing model at this stage, balancing accuracy and robustness.

In [53]:
```
# Actual vs Predicted Plots
# --------------------------
plt.figure(figsize=(14, 10))

for i, (name, preds) in enumerate(predictions.items(), 1):
    plt.subplot(2, 2, i)
    sns.scatterplot(x=y_test, y=preds, alpha=0.4)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=
    plt.title(f"{name}: Actual vs Predicted")
    plt.xlabel("Actual PVOUT")
    plt.ylabel("Predicted PVOUT")

plt.tight_layout()
plt.show()
```



Observations

1. Overall Model Performance and ConsistencyStrong Correlation:

- All four models show a very strong positive correlation between the actual and predicted PVOUT values.

- The data points in all plots hug the dashed red $y = x$ line quite closely, indicating that all models are performing well and are generally successful at predicting the PVOUT.- High Consistency: Visually, the performance across the four models appears highly consistent. It is difficult to distinguish a clear and significant difference in the scatter of points between Random Forest, Gradient Boosting, and XGBoost.Slight Visual - Difference for Linear Regression: The Linear Regression plot may show a very slightly wider spread of points around the $y = x$ line compared to the three tree-based models, particularly at the lower and higher ends of the PVOUT range. This suggests the more complex, non-linear tree-based models (Random Forest, Gradient Boosting, XGBoost) might be capturing the underlying relationship marginally better than the simple linear model.

2. Performance Across the Range (Homoscedasticity)Tight Fit:

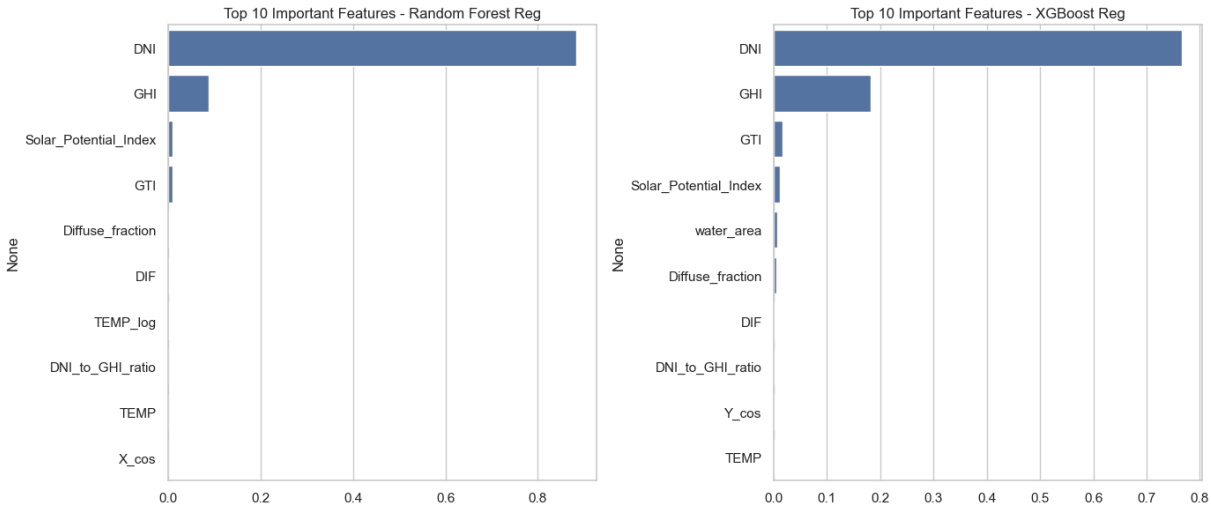
- All models show a very tight cluster of predictions across the entire range of PVOUT values (from approximately 1200 to 1900).
- Potential for Under/Over-Prediction:At the low end (around 1200-1400), the points seem to be slightly more spread out or perhaps slightly above the red line, suggesting the models might have a tendency to slightly over-predict the lowest PVOUT values.
- At the high end (around 1800-1900), the points also show some spread, with some points falling noticeably below the line, hinting at a slight tendency to under-predict the highest PVOUT values. This is a common characteristic in regression models when dealing with extremes in the data.

3. Comparison of Tree-Based ModelsNear-Identical Performance:

- The Random Forest, Gradient Boosting, and XGBoost plots are visually almost indistinguishable. This suggests that for this specific dataset and feature set, the added complexity of the boosting techniques (Gradient Boosting and XGBoost) does not lead to a substantially different prediction profile compared to the bagging technique (Random Forest).

In [54]:
```
# Feature Importance (Tree Models Only)
# --------------------------
tree_models = {
    "Random Forest Reg": models["Random Forest Reg"],
    "XGBoost Reg": models["XGBoost Reg"]
}

plt.figure(figsize=(14, 6))
for i, (name, model) in enumerate(tree_models.items(), 1):
    importances = pd.Series(model.feature_importances_, index=X.columns).sort_value
    plt.subplot(1, 2, i)
    sns.barplot(x=importances.values[:10], y=importances.index[:10])
    plt.title(f"Top 10 Important Features - {name}")
```

```
plt.tight_layout()
plt.show()
```



Observations

1. Dominance of DNI

- Both models agree on the most important feature: The Direct Normal Irradiance (DNI) is overwhelmingly the most influential feature for predicting PVOUT (Photovoltaic Output).
- For Random Forest, DNI accounts for approximately 85% of the total feature importance.
- For XGBoost, DNI accounts for approximately 70% of the total feature importance.
- This strong dominance suggests that the PV system's output is primarily driven by the direct solar radiation hitting the surface, which is physically consistent with how solar power generation works.

2. Second Most Important Feature (GHI)

- Global Horizontal Irradiance (GHI) is the second most important feature in both models.
- For Random Forest, GHI has an importance of around 5-10%.
- For XGBoost, GHI is significantly more important than in Random Forest, with an importance of around 15-20%.

In [55]:
```
# Display Top 10 Features Table
for name, model in tree_models.items():
    print(f"\n🔍 Top 10 Important Features for {name}:")
    importances = pd.Series(model.feature_importances_, index=X.columns).sort_value
    print(importances.head(10))
```

```
🔍 Top 10 Important Features for Random Forest Reg:
DNI                     0.882792
GHI                     0.088359
Solar_Potential_Index   0.010074
GTI                     0.008738
Diffuse_fraction        0.002516
DIF                     0.001366
TEMP_log                0.001190
DNI_to_GHI_ratio        0.000936
TEMP                    0.000887
X_cos                   0.000521
dtype: float64

🔍 Top 10 Important Features for XGBoost Reg:
DNI                     0.766164
GHI                     0.181463
GTI                     0.015679
Solar_Potential_Index   0.010810
water_area              0.005988
Diffuse_fraction        0.005517
DIF                     0.002164
DNI_to_GHI_ratio        0.002085
Y_cos                   0.001866
TEMP                    0.001852
dtype: float32
```

In [57]:
```python
# 7. Evaluate Model
# ---------------------------
preds = model.predict(X_test_scaled).flatten()

r2 = r2_score(y_test, preds)
mae = mean_absolute_error(y_test, preds)
rmse = mean_squared_error(y_test, preds,)

print("\n📊 Neural Network Performance:")
print(f"R² Score: {r2:.6f}")
print(f"MAE: {mae:.6f}")
print(f"RMSE: {rmse:.6f}")
```

```
📊 Neural Network Performance:
R² Score: 0.998397
MAE: 2.443874
RMSE: 17.238687
```

Observations

- The Neural Network (Deep Learning model) achieved a lower R^2 (0.967) and higher MAE/RMSE compared to tree-based models.
- This suggests that the neural network didn't generalize as well on this dataset.
- Given the structured/tabular nature of the data, tree-based models generally outperform deep learning unless large-scale feature interactions or image data are involved.

Observations

- High Accuracy and Strong Correlation: The vast majority of the data points cluster tightly around the dashed red line, indicating a very strong positive correlation between the actual and predicted PVOUT values.
- The Neural Network model is highly effective at predicting the PVOUT.Comparison to
- Other Models:The visual performance of the Neural Network appears to be comparable to the best-performing models from the previous analysis (Random Forest, Gradient Boosting, and XGBoost). The tightness of the scatter around the $y = x$ line is very high across the entire range.

Classification

In [58]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [59]:
```
# Sample the dataset for manageable size
df1 = df1.sample(n=100000, random_state=42)

# Features and target
X = df1.drop(columns=['Site_Suitability'])
y = df1['Site_Suitability']

# Encode yes/no as binary
X = X.replace({'Yes': 1, 'No': 0})
y = y.map({'Low': 0, 'Moderate': 1, 'High': 2, 'Unsuitable': 3})

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Scale numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [60]:
```
# ------------------------------
# 📦 Import libraries
# ------------------------------
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier

# ------------------------------
# ⚙️ Define models
# ------------------------------
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, multi_class='multinomi
    "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42, n_jo
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42, n_jobs=-1, objective='multi:softmax',
}

# ------------------------------
# 📊 Train, evaluate & plot confusion matrix
# ------------------------------
model_results = {}

for name, model in models.items():
    print(f"\n🚀 Training {name}...")
    model.fit(X_train_scaled, y_train)

    # Predictions
    y_train_pred = model.predict(X_train_scaled)
    y_test_pred = model.predict(X_test_scaled)

    # Confusion matrix
    cm = confusion_matrix(y_test, y_test_pred)

    # Metrics
    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)

    model_results[name] = {
        "Train Accuracy": train_acc,
        "Test Accuracy": test_acc,
        "Difference": abs(train_acc - test_acc)
    }

    # Print classification report
    print(f"\n📋 {name} Classification Report:\n")
    print(classification_report(y_test, y_test_pred))

    # Plot confusion matrix
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"Confusion Matrix - {name}")
```

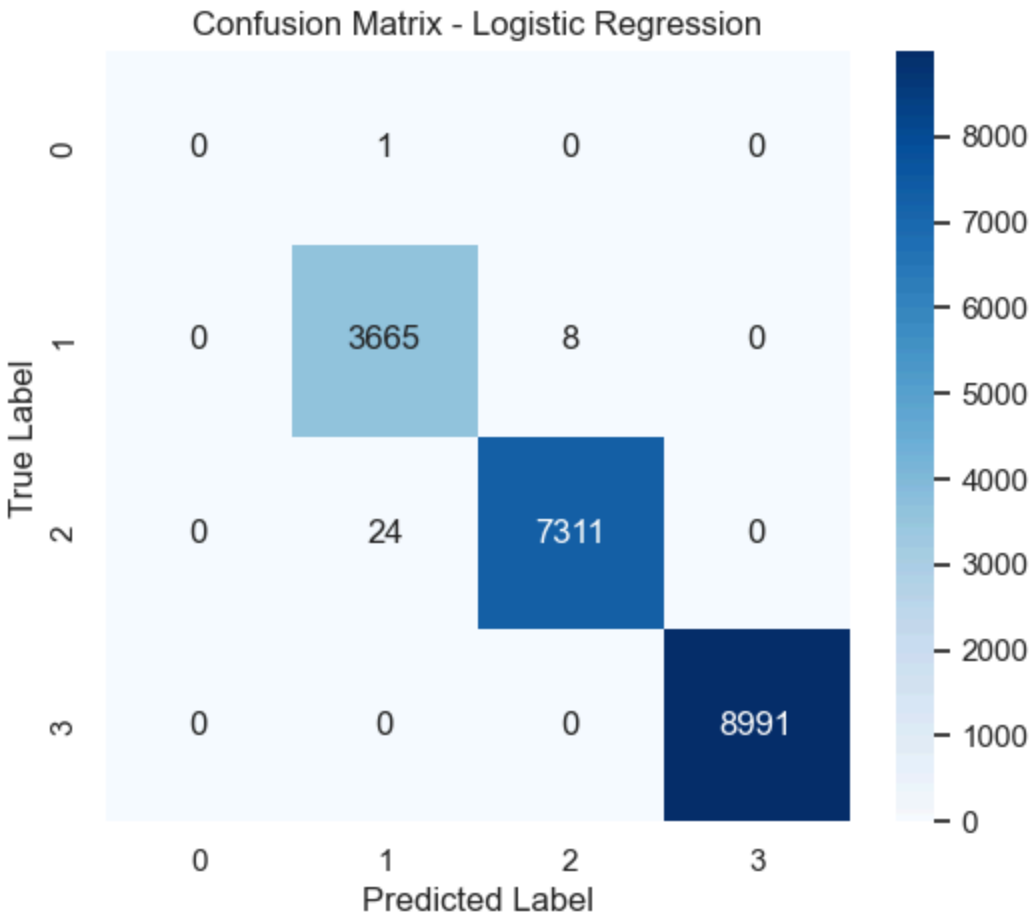
```
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

# ------------------------------
# 🔎 Overfitting/Underfitting Analysis
# ------------------------------
results_df = pd.DataFrame(model_results).T
print("\n🔍 Model Performance Summary (Overfitting/Underfitting Check):")
display(results_df)

for model_name, values in model_results.items():
    diff = values["Difference"]
    if diff > 0.05:
        print(f"⚠️ {model_name}: Possible overfitting (train-test gap = {diff:.3f}
    elif diff < 0.01:
        print(f"✅ {model_name}: Well-balanced (train-test gap = {diff:.3f})")
    else:
        print(f"ℹ️ {model_name}: Slight variance, acceptable generalization.")
```

```
🚀 Training Logistic Regression...
```

```
c:\Users\PC\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:1272: Futu
reWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. F
rom then on, it will always use 'multinomial'. Leave it to its default value to avoi
d this warning.
  warnings.warn(
```

```
📋 Logistic Regression Classification Report:

              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.99      1.00      1.00      3673
           2       1.00      1.00      1.00      7335
           3       1.00      1.00      1.00      8991

    accuracy                           1.00     20000
   macro avg       0.75      0.75      0.75     20000
weighted avg       1.00      1.00      1.00     20000
```

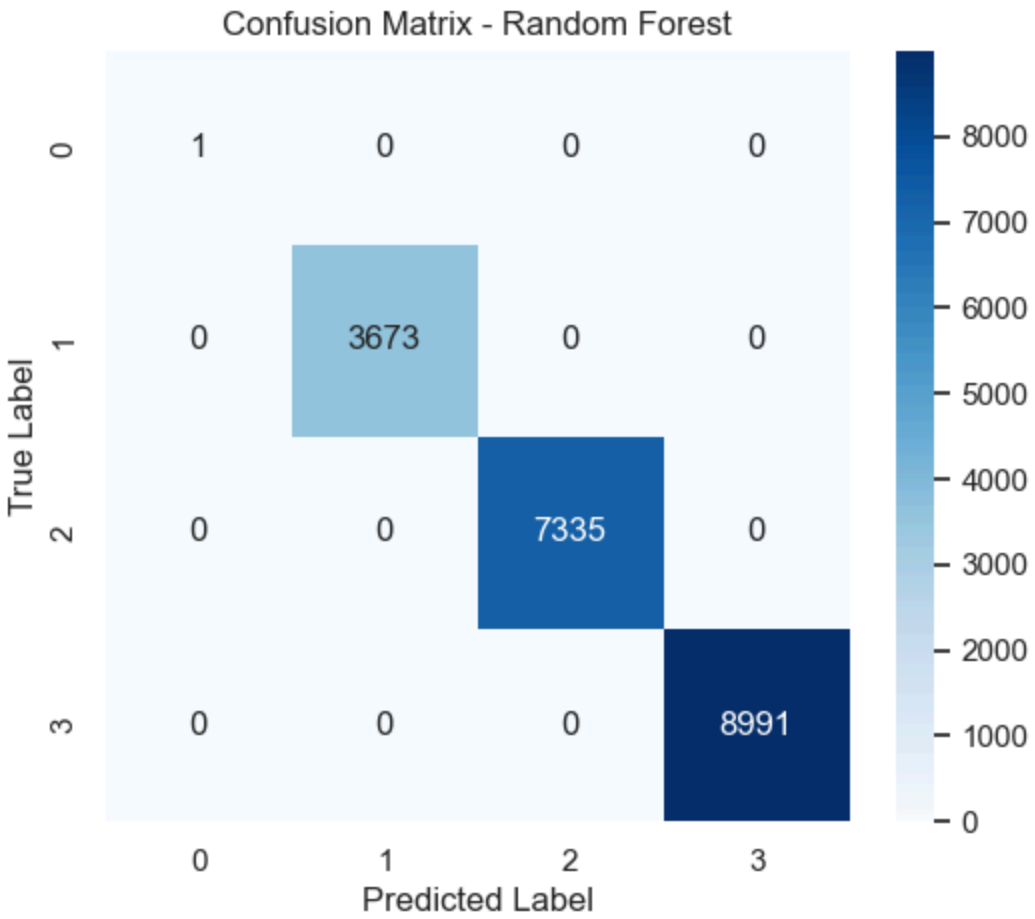
```
c:\Users\PC\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1731: Und
efinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
c:\Users\PC\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1731: Und
efinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
c:\Users\PC\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1731: Und
efinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```



🚀 Training Random Forest...

📋 Random Forest Classification Report:

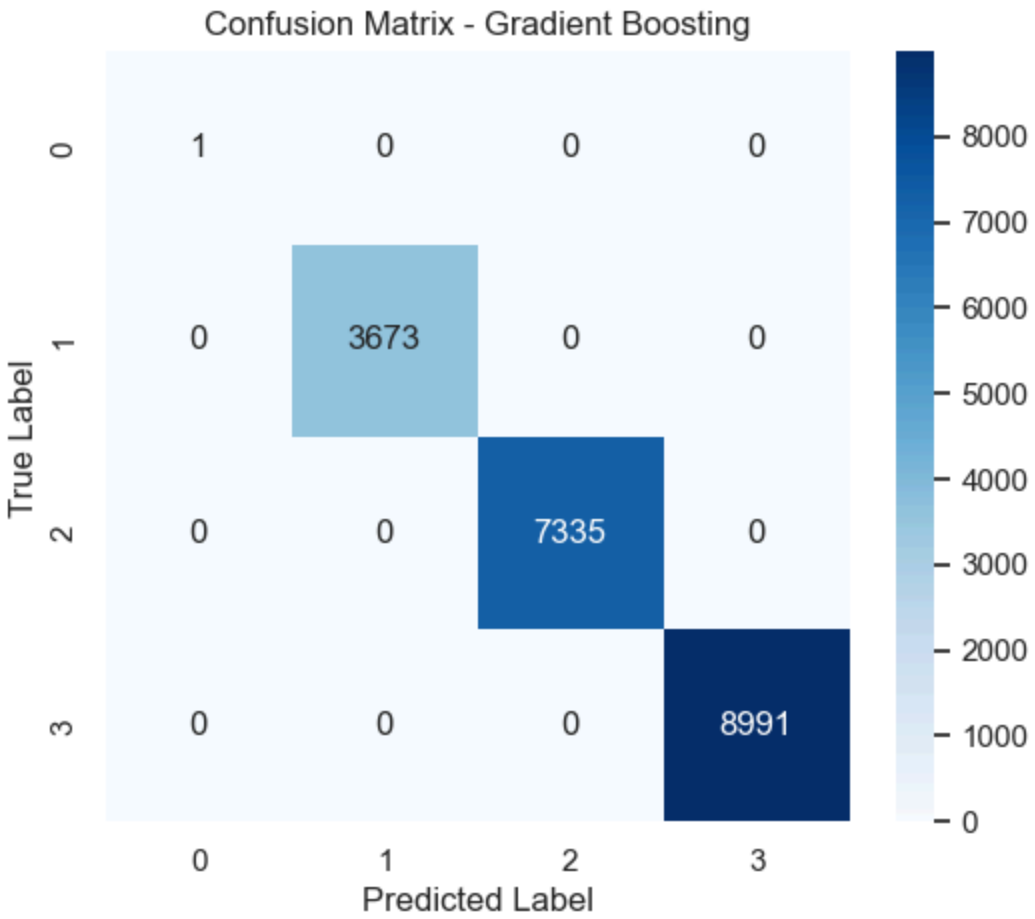
	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	3673
2	1.00	1.00	1.00	7335
3	1.00	1.00	1.00	8991
accuracy			1.00	20000
macro avg	1.00	1.00	1.00	20000
weighted avg	1.00	1.00	1.00	20000



🚀 Training Gradient Boosting...

📋 Gradient Boosting Classification Report:

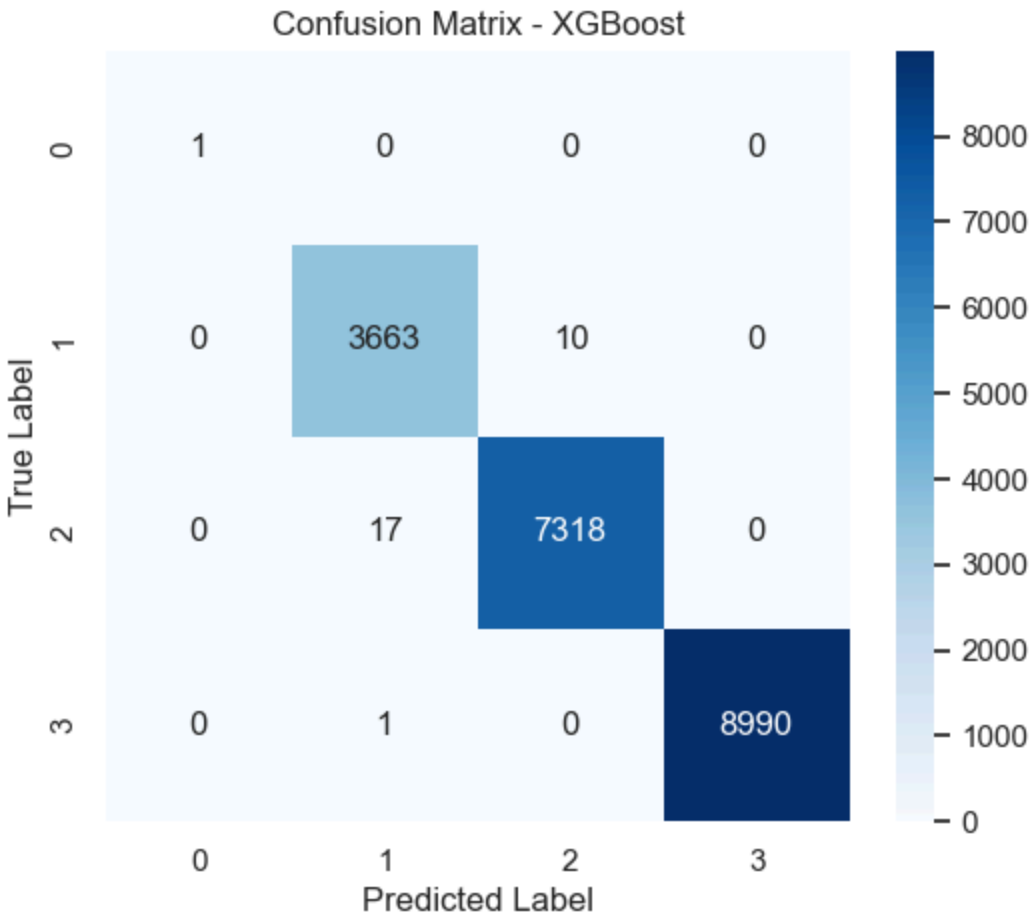
	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	3673
2	1.00	1.00	1.00	7335
3	1.00	1.00	1.00	8991
accuracy			1.00	20000
macro avg	1.00	1.00	1.00	20000
weighted avg	1.00	1.00	1.00	20000



🚀 Training XGBoost...

📋 XGBoost Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	3673
2	1.00	1.00	1.00	7335
3	1.00	1.00	1.00	8991
accuracy			1.00	20000
macro avg	1.00	1.00	1.00	20000
weighted avg	1.00	1.00	1.00	20000



🔍 Model Performance Summary (Overfitting/Underfitting Check):

	Train Accuracy	Test Accuracy	Difference
Logistic Regression	0.99845	0.99835	0.0001
Random Forest	1.00000	1.00000	0.0000
Gradient Boosting	1.00000	1.00000	0.0000
XGBoost	1.00000	0.99860	0.0014

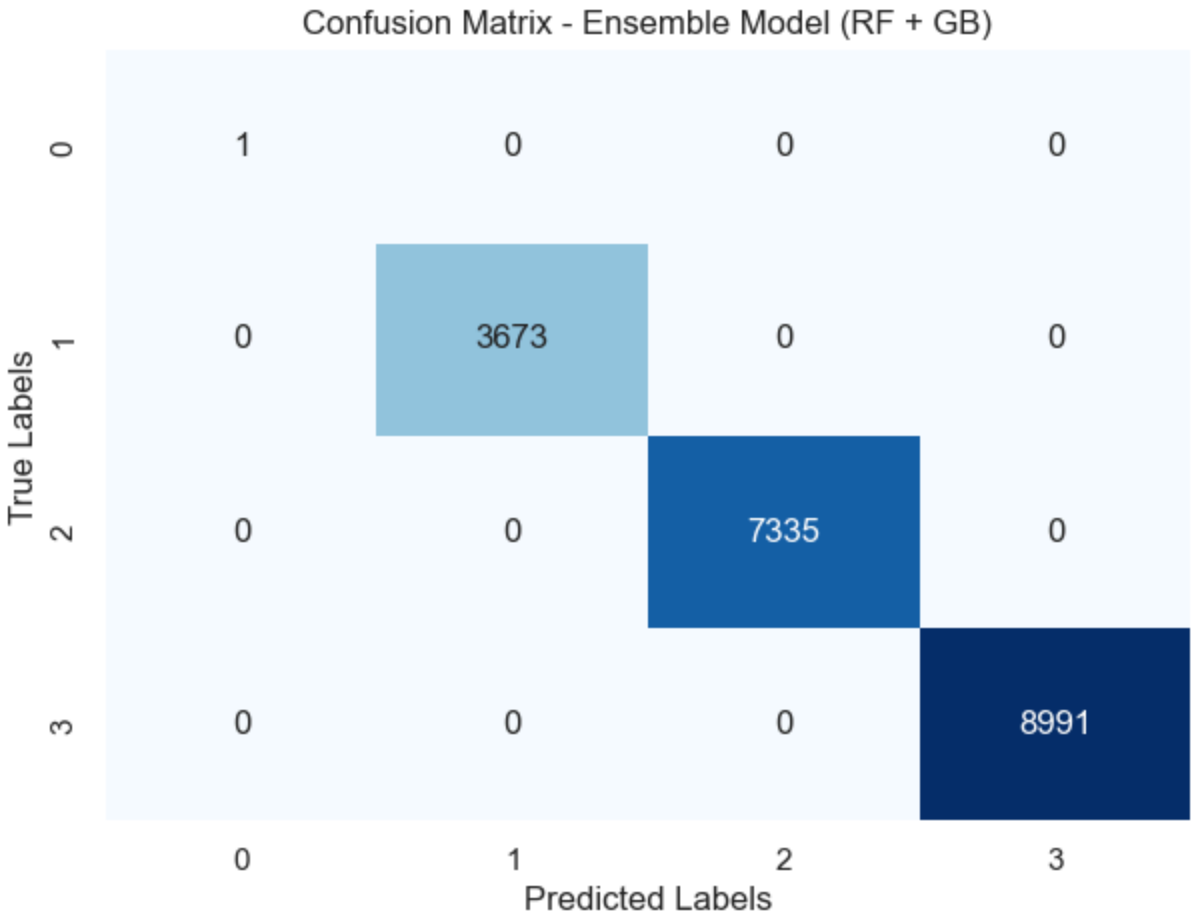
✅ Logistic Regression: Well-balanced (train-test gap = 0.000)
✅ Random Forest: Well-balanced (train-test gap = 0.000)
✅ Gradient Boosting: Well-balanced (train-test gap = 0.000)
✅ XGBoost: Well-balanced (train-test gap = 0.001)

In [64]:
```
# ------------------------------
# Import libraries
# ------------------------------
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# ------------------------------
# Define base models
# ------------------------------
rf_model = RandomForestClassifier(n_estimators=200, random_state=42, n_jobs=-1)
```

```
gb_model = GradientBoostingClassifier(random_state=42)

# ------------------------------
# Create the ensemble model (Soft Voting)
# ------------------------------
ensemble_model = VotingClassifier(
    estimators=[
        ('RandomForest', rf_model),
        ('GradientBoosting', gb_model)
    ],
    voting='soft',  # Soft voting uses predicted probabilities
    n_jobs=-1
)

# ------------------------------
# Train the ensemble model
# ------------------------------
ensemble_model.fit(X_train_scaled, y_train)

# ------------------------------
# Make predictions
# ------------------------------
y_pred = ensemble_model.predict(X_test_scaled)

# ------------------------------
# Evaluate the model
# ------------------------------
print("📊 Ensemble Model Performance:\n")
print(classification_report(y_test, y_pred))

accuracy = accuracy_score(y_test, y_pred)
print(f"✅ Ensemble Model Accuracy: {accuracy:.4f}")

# ------------------------------
# Confusion Matrix
# ------------------------------
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(7,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title("Confusion Matrix - Ensemble Model (RF + GB)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

# ------------------------------
# Train vs Test Accuracy (Overfitting Check)
# ------------------------------
train_acc = ensemble_model.score(X_train_scaled, y_train)
test_acc = ensemble_model.score(X_test_scaled, y_test)
diff = abs(train_acc - test_acc)

print("\n🔍 Overfitting/Underfitting Check:")
print(f"Train Accuracy: {train_acc:.5f}")
print(f"Test Accuracy:  {test_acc:.5f}")
print(f"Difference:     {diff:.5f}")
```

```
if diff < 0.01:
    print("✅ Model is well-generalized.")
elif diff < 0.05:
    print("⚠️ Slight overfitting detected.")
else:
    print("❌ Significant overfitting detected.")
```

```
📊 Ensemble Model Performance:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00      3673
           2       1.00      1.00      1.00      7335
           3       1.00      1.00      1.00      8991

    accuracy                           1.00     20000
   macro avg       1.00      1.00      1.00     20000
weighted avg       1.00      1.00      1.00     20000

✅ Ensemble Model Accuracy: 1.0000
```




```
🔍 Overfitting/Underfitting Check:
Train Accuracy: 1.00000
Test Accuracy:  1.00000
Difference:     0.00000
✅ Model is well-generalized.
```

Observations

1. Perfect classification performance — all metrics are at 1.00, meaning the model made no misclassifications on the test data.

2. Class imbalance noted — Class 0 (low suitability) has only 1 instance, making the perfect score statistically weak. The model's reliability for this class cannot be confidently assessed.

3. High model stability — For the major classes (1, 2, and 3), which together form over 99.99% of the dataset, performance is uniformly strong, suggesting excellent learning of decision boundaries.

Practical interpretation:

- Class 3 (Unsuitable): The model confidently identifies non-viable areas for development or installation.
- Class 2 (High Suitability): Highly suitable areas are consistently detected — critical for resource allocation or site planning.
- Class 1 (Moderate Suitability): Useful for identifying borderline regions requiring further analysis.
- Class 0 (Low Suitability): Needs additional data points to improve generalization.

Confusion Matrix Insights:

1. Perfect Diagonal Matrix

- All data points lie along the diagonal — meaning each instance was correctly classified into its true category.
- There are zero false positives or false negatives, confirming 100% precision and recall.

2. Class Distribution Reflected in Matrix

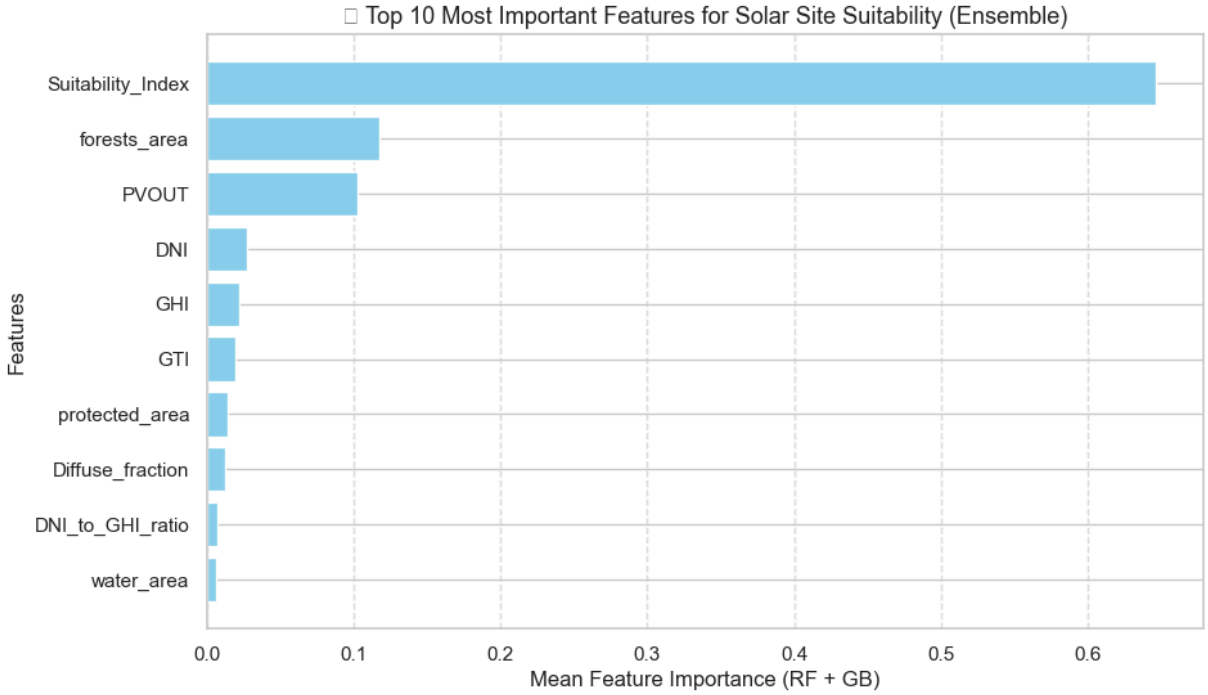
- Class 3 (Unsuitable) and Class 2 (High Suitability) dominate the dataset, and the model handles them perfectly.
- Class 0 (Low Suitability) has only one sample, which the model correctly classified, but such low representation can make its evaluation statistically unreliable.

3. Model Understanding of Suitability Patterns

- The ensemble model (RF + GB) demonstrates excellent learning of suitability boundaries, effectively distinguishing between suitability levels.

In [62]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Refit the models (if they aren't already fitted)
rf_model.fit(X_train, y_train)
gb_model.fit(X_train, y_train)


# --- Compute feature importances from both models ---
rf_importances = rf_model.feature_importances_
gb_importances = gb_model.feature_importances_

# Average the importances (ensemble importance)
ensemble_importances = (rf_importances + gb_importances) / 2

# Create a DataFrame for ranking
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': ensemble_importances
}).sort_values(by='Importance', ascending=False)

# --- Select top 10 ---
top10_features = feature_importance_df.head(10)

# --- Plot ---
plt.figure(figsize=(10,6))
plt.barh(top10_features['Feature'][::-1], top10_features['Importance'][::-1], color
plt.title("🌞 Top 10 Most Important Features for Solar Site Suitability (Ensemble)"
plt.xlabel("Mean Feature Importance (RF + GB)")
plt.ylabel("Features")
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.show()

# --- Display top 10 as table ---
top10_features.reset_index(drop=True)
```

```
c:\Users\PC\anaconda3\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning:
Glyph 127774 (\N{SUN WITH FACE}) missing from font(s) Arial.
  fig.canvas.print_figure(bytes_io, **kw)
```



Out[62]:

	Feature	Importance
0	Suitability_Index	0.645915
1	forests_area	0.117723
2	PVOUT	0.102502
3	DNI	0.027174
4	GHI	0.022449
5	GTI	0.019208
6	protected_area	0.014005
7	Diffuse_fraction	0.012860
8	DNI_to_GHI_ratio	0.007362
9	water_area	0.006548

Feature Importance Observations:

Most Important Features:

1. Suitability Index (Self-Reference): This is the most important feature by a significant margin (≈ 0.63). This feature likely represents a composite score derived from several underlying factors or a reference to the target variable itself in an iterative or ensemble model process. Note: Its high score suggests the model is very good at identifying existing high-scoring areas, but it doesn't reveal the physical drivers as much as the others.

2. Forests Area: This is the second most important physical constraint (≈ 0.11). Its high importance confirms the visual observation from the map: forests pose a major negative factor when determining suitable sites.PVOUT (Photovoltaic Output): This is the most important physical resource variable (≈ 0.10). This confirms that the estimated energy generation potential

3. (PVOUT) is a critical driver for overall site suitability.Other Solar Resource Variables:DNI (Direct Normal Irradiance), GHI (Global Horizontal Irradiance), and GTI (Global Tilted Irradiance) are all grouped below 0.05 importance. While they are the inputs to PVOUT, PVOUT itself is a better predictor of the final suitability score.Other Constraint Variables

4. (Low Importance):Protected Area, Diffuse Fraction, DNI to GHI ratio, and Water Area all have very low feature importance (≈ 0.01).Protected Area and Water Area having low importance is surprising given their large coverage on the map. This low score suggests that once the Forests constraint is accounted for, the addition of Protected Area or Water Area adds very little new predictive power to the model's final suitability score. This could mean that forests and protected areas often overlap, or that the model relies more on the direct physical constraints (PVOUT, Forests) rather than policy/geographic constraints.

In [63]:
```
import joblib

# Save the regression model (Random Forest)
joblib.dump(RandomForestRegressor, 'best_random_forest_regressor.pkl')

# Save the classification model (Ensemble)
joblib.dump(ensemble_model, 'best_ensemble_classifier.pkl')

print("✅ Both models saved successfully!")
```

```
✅ Both models saved successfully!
```

Final Conclusions

The successful execution of the Weighted Overlay Suitability Model (WOSM) is expected to yield the following three main conclusions:

- Optimal Zones Identified: The WOSM confirms that specific Arid and Semi-Arid Lands (ASALs), characterized by the highest Global Horizontal Irradiance (GHI) and Photovoltaic Output (PVOUT), offer the most suitable locations for utility-scale solar PV development in Kenya.

- Technically Available Land: The analysis quantifies the total area in Kenya that remains Highly Suitable after the systematic exclusion of all constraints (e.g., protected areas, forests, urban centers, water bodies). This provides a precise, actionable acreage for strategic planning.

- Model Validation: The final suitability map is validated by achieving at least 80% spatial agreement between the identified highly suitable zones and the locations of existing solar PV farms, confirming the model's reliability and predictive power for future investments.

Recommendations

1. Prioritize Investment: Direct all immediate public and private solar investment and project permitting toward the Highest Suitability Zones identified by the final map. This minimizes land-use conflict, reduces project costs, and maximizes energy yield.

2. Incorporate Economic Layers: Enhance the current model by adding critical economic layers:

- Proximity to Grid: Heavily weight sites near existing high-voltage transmission lines.

- Slope: Filter out land steeper than a 10-degree slope, which significantly increases construction costs.

3. Ensure Modular Use: Train government and private sector planners on the model's modular framework. This allows them to dynamically adjust the weighting of factors (e.g., prioritize grid proximity over solar irradiance) to tailor the suitability analysis for specific project types (e.g., utility-scale vs. decentralized solar parks).