```
import geopandas as gpd
import numpy as np
import pandas as pd
import rasterio
import streamlit as st
from rasterio.features import rasterize, shapes
from rasterio.io import MemoryFile
from rasterio.warp import Resampling, calculate_default_transform, reproject
from shapely.geometry import shape

# --- CONSTANTES Y DICCIONARIOS ---
holdridge_zone_map_simplified = {
    "Nival": 1,
    "Tundra pluvial (tp-A)": 2,
    "Tundra húmeda (th-A)": 3,
    "Tundra seca (ts-A)": 4,
    "Páramo pluvial subalpino (pp-SA)": 5,
    "Páramo muy húmedo subalpino (pmh-SA)": 6,
    "Páramo seco subalpino (ps-SA)": 7,
    "Bosque pluvial Montano (bp-M)": 8,
    "Bosque muy húmedo Montano (bmh-M)": 9,
    "Bosque húmedo Montano (bh-M)": 10,
    "Bosque seco Montano (bs-M)": 11,
    "Monte espinoso Montano (me-M)": 12,
    "Bosque pluvial Premontano (bp-PM)": 13,
    "Bosque muy húmedo Premontano (bmh-PM)": 14,
    "Bosque húmedo Premontano (bh-PM)": 15,
    "Bosque seco Premontano (bs-PM)": 16,
    "Monte espinoso Premontano (me-PM)": 17,
    "Bosque pluvial Tropical (bp-T)": 18,
    "Bosque muy húmedo Tropical (bmh-T)": 19,
    "Bosque húmedo Tropical (bh-T)": 20,
    "Bosque seco Tropical (bs-T)": 21,
    "Monte espinoso Tropical (me-T)": 22,
    "Zona Desconocida": 0,
}

# Invertir para buscar nombre por ID
holdridge_int_to_name_simplified = {
    v: k for k, v in holdridge_zone_map_simplified.items()
}

# --- PALETA DE COLORES OFICIAL (HEX) ---
holdridge_colors = {
    1: "#FFFFFF",  # Nival
    2: "#B0E0E6",
    3: "#87CEEB",
    4: "#708090",  # Alpino
    5: "#8A2BE2",
    6: "#9370DB",
    7: "#D8BFD8",  # Páramo
    8: "#00008B",
    9: "#006400",
    10: "#228B22",
    11: "#9ACD32",
    12: "#F0E68C",  # Montano
    13: "#0000CD",
    14: "#008000",
    15: "#32CD32",
    16: "#FFFF00",
    17: "#DAA520",  # Premontano
    18: "#191970",
    19: "#2E8B57",
    20: "#7CFC00",
    21: "#FFA500",
    22: "#FF4500",  # Tropical
    0: "#000000",
}


def classify_life_zone_alt_ppt(altitude, ppt):
    """
    Clasifica una celda según su altitud (m) y precipitación anual (mm).
    """
    if pd.isna(altitude) or pd.isna(ppt) or altitude < 0 or ppt <= 0:
        return 0

    # 1. NIVAL
    if altitude >= 4500:
        return 1
    # 2. ALPINO
    if altitude >= 3800:
        if ppt >= 1000:
            return 2
        elif ppt >= 500:
            return 3
        else:
            return 4
    # 3. SUBALPINO (Páramo)
    if altitude >= 3000:
        if ppt >= 2000:
            return 5
        elif ppt >= 1000:
            return 6
        else:
            return 7
    # 4. MONTANO
    if altitude >= 2000:
        if ppt >= 4000:
            return 8
        elif ppt >= 2000:
            return 9
        elif ppt >= 1000:
            return 10
        elif ppt >= 500:
            return 11
        else:
            return 12
    # 5. PREMONTANO
    if altitude >= 1000:
        if ppt >= 4000:
            return 13
        elif ppt >= 2000:
            return 14
        elif ppt >= 1000:
            return 15
        elif ppt >= 500:
            return 16
        else:
            return 17
    # 6. TROPICAL
    if ppt >= 8000:
        return 18
    elif ppt >= 4000:
        return 19
    elif ppt >= 2000:
        return 20
    elif ppt >= 1000:
        return 21
    else:
        return 22


# Vectorización para rendimiento
_vectorized_classify = np.vectorize(classify_life_zone_alt_ppt)


def generate_life_zone_map(
    dem_path, precip_raster_path, mask_geometry=None, downscale_factor=4
):
    """
    Genera mapa raster clasificado de Zonas de Vida y devuelve el array y el perfil.
    """
    try:
        if downscale_factor is None or downscale_factor <= 0:
            downscale_factor = 1
        dst_crs = "EPSG:4326"

        # 1. Procesar DEM
        with rasterio.open(dem_path) as dem_src:
            dst_width = max(1, dem_src.width // downscale_factor)
            dst_height = max(1, dem_src.height // downscale_factor)

            dst_transform, dst_width, dst_height = calculate_default_transform(
                dem_src.crs,
                dst_crs,
                dem_src.width,
                dem_src.height,
                *dem_src.bounds,
                dst_width=dst_width,
                dst_height=dst_height,
            )

            dem_resampled = np.empty((dst_height, dst_width), dtype=np.float32)
            reproject(
                source=rasterio.band(dem_src, 1),
                destination=dem_resampled,
                src_transform=dem_src.transform,
                src_crs=dem_src.crs,
                dst_transform=dst_transform,
                dst_crs=dst_crs,
                resampling=Resampling.bilinear,
            )

        # 2. Procesar Precipitación
        with rasterio.open(precip_raster_path) as ppt_src:
            ppt_resampled = np.empty((dst_height, dst_width), dtype=np.float32)
            reproject(
                source=rasterio.band(ppt_src, 1),
                destination=ppt_resampled,
                src_transform=ppt_src.transform,
                src_crs=ppt_src.crs,
                dst_transform=dst_transform,
                dst_crs=dst_crs,
                resampling=Resampling.average,
            )

        # 3. Clasificación
        dem_mask = np.isnan(dem_resampled)
        ppt_mask = np.isnan(ppt_resampled)
        valid_mask = (
            (~dem_mask) & (~ppt_mask) & (dem_resampled > -500) & (ppt_resampled >= 0)
        )

        classified_raster = np.zeros((dst_height, dst_width), dtype=np.int16)

        if np.any(valid_mask):
            zone_ints = _vectorized_classify(
                dem_resampled[valid_mask], ppt_resampled[valid_mask]
            )
            classified_raster[valid_mask] = zone_ints.astype(np.int16)

        # 4. Máscara Geometría
        if mask_geometry is not None and not mask_geometry.empty:
            try:
                mask_reproj = (
                    mask_geometry.to_crs(dst_crs)
                    if mask_geometry.crs != dst_crs
                    else mask_geometry
                )
                shapes_list = [(geom, 1) for geom in mask_reproj.geometry]
                mask_raster = rasterize(
                    shapes_list,
                    out_shape=(dst_height, dst_width),
                    transform=dst_transform,
                    fill=0,
                    dtype=np.uint8,
                )
                classified_raster = np.where(mask_raster == 1, classified_raster, 0)
            except Exception as e:
                st.warning(f"Error máscara: {e}")

        # 5. Perfil de salida
        output_profile = {
            "driver": "GTiff",
            "dtype": "int16",
            "nodata": 0,
            "width": dst_width,
            "height": dst_height,
            "count": 1,
            "crs": dst_crs,
            "transform": dst_transform,
            "compress": "lzw",
        }

        return (
            classified_raster,
            output_profile,
            holdridge_int_to_name_simplified,
            holdridge_colors,
        )

    except Exception as e:
        st.error(f"Error generando mapa: {e}")
        return None, None, None, None


# --- NUEVAS FUNCIONES PARA DESCARGA (SHP/JSON/TIFF) ---


def vectorize_raster_to_gdf(raster_array, transform, crs):
    """
    Convierte el Raster clasificado a un GeoDataFrame (Polígonos).
    Útil para descargar como Shapefile o GeoJSON.
    """
    try:
        # Extraer formas (polígonos) del raster donde el valor != 0
        mask = raster_array != 0
        results = (
            {"properties": {"id_zona": v}, "geometry": s}
            for i, (s, v) in enumerate(
                shapes(raster_array, mask=mask, transform=transform)
            )
        )

        # Crear lista de geometrías y atributos
        geoms = []
        ids = []
        for r in results:
            geoms.append(shape(r["geometry"]))
            ids.append(r["properties"]["id_zona"])

        if not geoms:
            return gpd.GeoDataFrame()

        # Crear GeoDataFrame
        gdf = gpd.GeoDataFrame({"id_zona": ids}, geometry=geoms, crs=crs)

        # Agregar nombre de la zona
        gdf["zona_vida"] = gdf["id_zona"].map(holdridge_int_to_name_simplified)

        return gdf
    except Exception as e:
        st.error(f"Error vectorizando mapa: {e}")
        return gpd.GeoDataFrame()


def get_raster_bytes(raster_array, profile):
    """
    Escribe el array raster a un objeto BytesIO (memoria) formato TIFF.
    Útil para descargar con st.download_button.
    """
    try:
        mem_file = MemoryFile()
        with mem_file.open(**profile) as dataset:
            dataset.write(raster_array, 1)

        return mem_file.read()  # Devuelve los bytes
    except Exception as e:
        st.error(f"Error preparando descarga TIFF: {e}")
        return None
```