import geopandas as gpd

import rasterio

import numpy as np

import pandas as pd

from rasterio.features import rasterize

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import cross\_val\_score, StratifiedKFold

import matplotlib.pyplot as plt

def filter\_classes(dataframe, valid\_classes):

"""

Filtre les classes de la BD Forêt selon une liste de classes valides.

"""

return dataframe[dataframe['TFV'].isin(valid\_classes)]

def count\_polygons\_by\_class(dataframe, class\_column='classif\_pixel'):

"""

Compte le nombre de polygones par classe.

"""

return dataframe.groupby(class\_column).size().reset\_index(name='count')

def count\_pixels\_by\_class(dataframe, raster\_path, class\_column='classif\_pixel'):

"""

Compte le nombre de pixels par classe dans un raster donné.

"""

with rasterio.open(raster\_path) as src:

data = src.read(1)

unique, counts = np.unique(data, return\_counts=True)

return pd.DataFrame({'class': unique, 'count': counts})

def compute\_ndvi(red\_band, nir\_band):

"""

Calcule le NDVI à partir des bandes rouge et proche infrarouge.

"""

return (nir\_band - red\_band) / (nir\_band + red\_band)

def calculate\_spectral\_variability(samples, raster\_path):

"""

Calcule la distance moyenne au centroïde par classe pour un raster donné.

"""

with rasterio.open(raster\_path) as src:

data = src.read()

results = {}

for class\_label in np.unique(samples['class']):

pixels = data[samples['class'] == class\_label]

centroid = np.mean(pixels, axis=0)

distances = np.sqrt(np.sum((pixels - centroid) \*\* 2, axis=1))

results[class\_label] = np.mean(distances)

return results

def train\_random\_forest(samples, features, target):

"""

Entraîne un modèle Random Forest sur les échantillons.

"""

model = RandomForestClassifier(max\_depth=50, oob\_score=True, max\_samples=0.75, class\_weight='balanced')

model.fit(features, target)

return model

def save\_classification(model, features, output\_file):

"""

Applique un modèle de classification et sauvegarde la carte en raster.

"""

predictions = model.predict(features)

with rasterio.open(output\_file, 'w', \*\*features.meta) as dst:

dst.write(predictions, 1)

def plot\_violin(data, output\_file):

"""

Produit un violin plot pour visualiser les distributions.

"""

plt.violinplot(data, showmeans=True)

plt.savefig(output\_file)