GUIA DE USO

MODEL[LUMAChain] Land Use Multitemporal Analysis Chain

Preparación del entorno Google Drive

Para el proceso necesario crear una serie de carpetas en Google drive para el correcto funcionamiento del modelo sin alterar la estructura principal.

Esta es la serie de carpetas que debe crear el usuario en Mi Unidad:

Model_LUMAchain (principal)

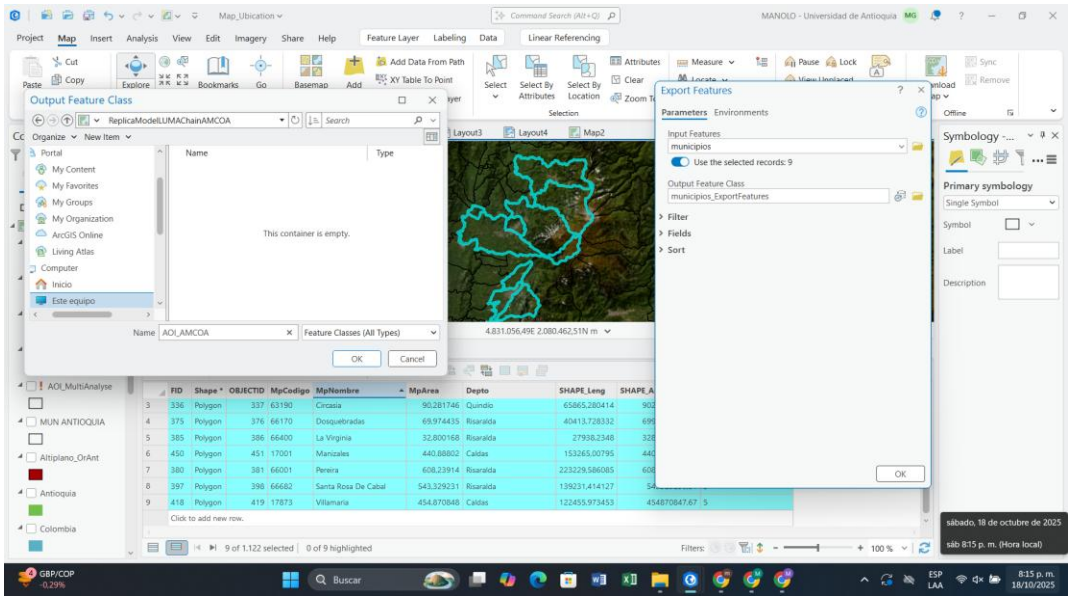
- ⇑ Cartografia_Base
- ⇑ 0_Validacion_CoberturaLandSat
- ⇑ 1_Composiciones_Multibanda
- ⇑ 2_Calculo_IndicesNormalizados
- ⇑ 4_Binarios_Resultados
- ⇑ 5_Estadisticas_Zonales
- ⇑ 6_Scripts_ModelLUMAChain

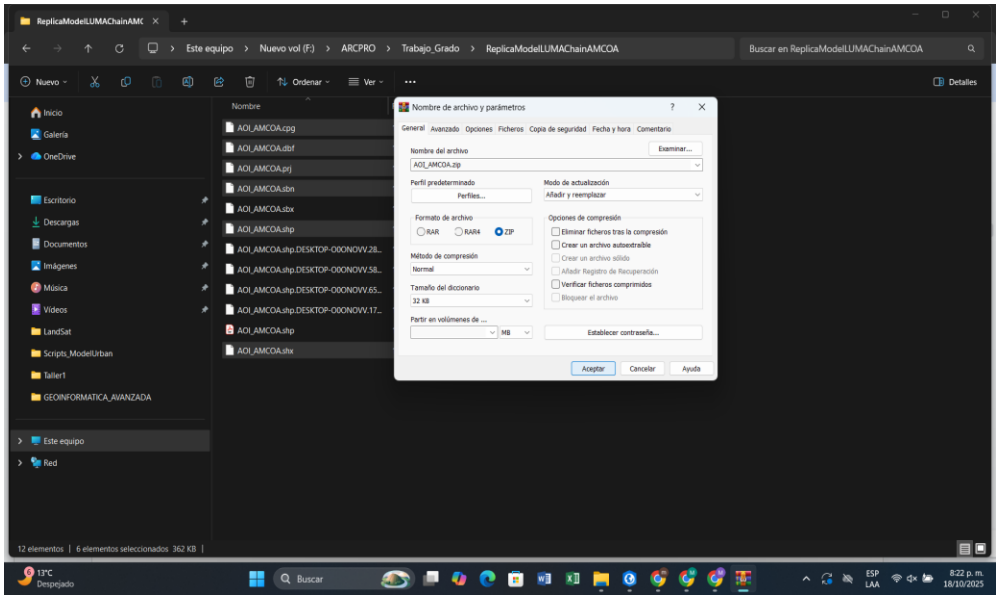
Nota: Los nombres de las carpetas son opcionales, pero se recomienda contar con el mismo numero y destinación de carpetas.

Descripción:

- ⇑ Cartografia_Base

Debes proporcionar tu área de estudio en Formato Shapefile, para este caso en Colombia se sugiere, crear a partir de la capa de municipios y límites políticos del IGAC (Instituto Geográfico Agustín Codazzi). Ejemplo: En este caso se realizará la réplica del Modelo propuesto, para ello se eligió el Área Metropolitana del Centro Occidente Ampliada (AMCOA).



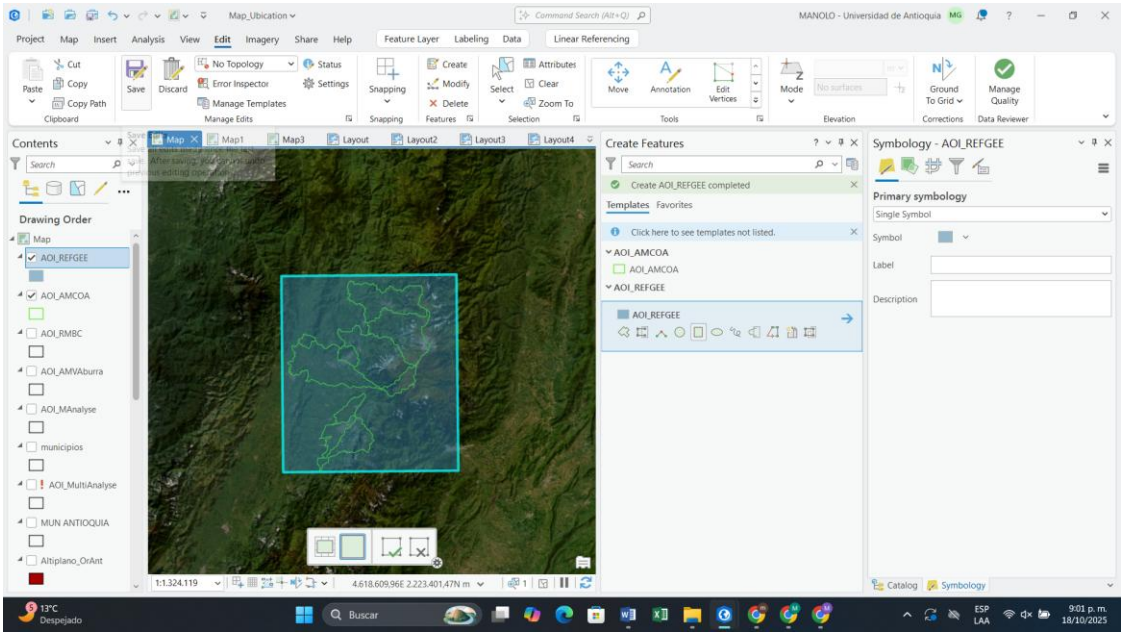


El archivo que se genera debe nombrarse como AOI_tuareareferencia para este caso AOI_AMCOA y cargarse a la carpeta: “Cartografia_Base” debe incluir los complementos que se relacionan a continuación (estrictamente esos 6).

cpg, dnf, prj, sbn, shp, shx →

Drive > My Drive > ModelLUMAChainAMCOA > Cartografia_Base”.

Luego es necesario crear un shapefile de forma rectangular que abarque todos los municipios del AOI esto para facilitar el geoprocesamiento de los datos y generar una referencia que será usada en el entorno Google Earth Engine. Debe nombrarse AOI_GEE



Es necesario cargar el AOI en los activos “Assets” de la plataforma Google Earth Engine a través del panel ubicado en la zona superior izquierda Assets → New → CSV file(.csv).

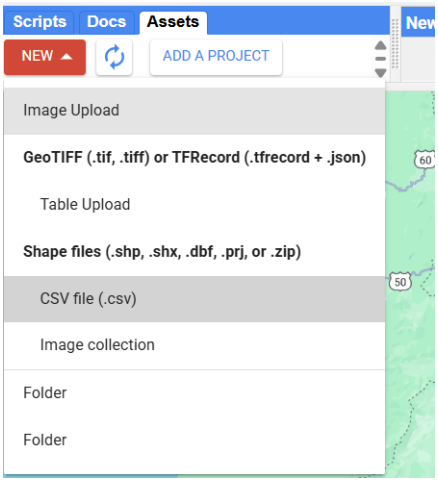


Imagen 1. Assets Google Earth Engine.

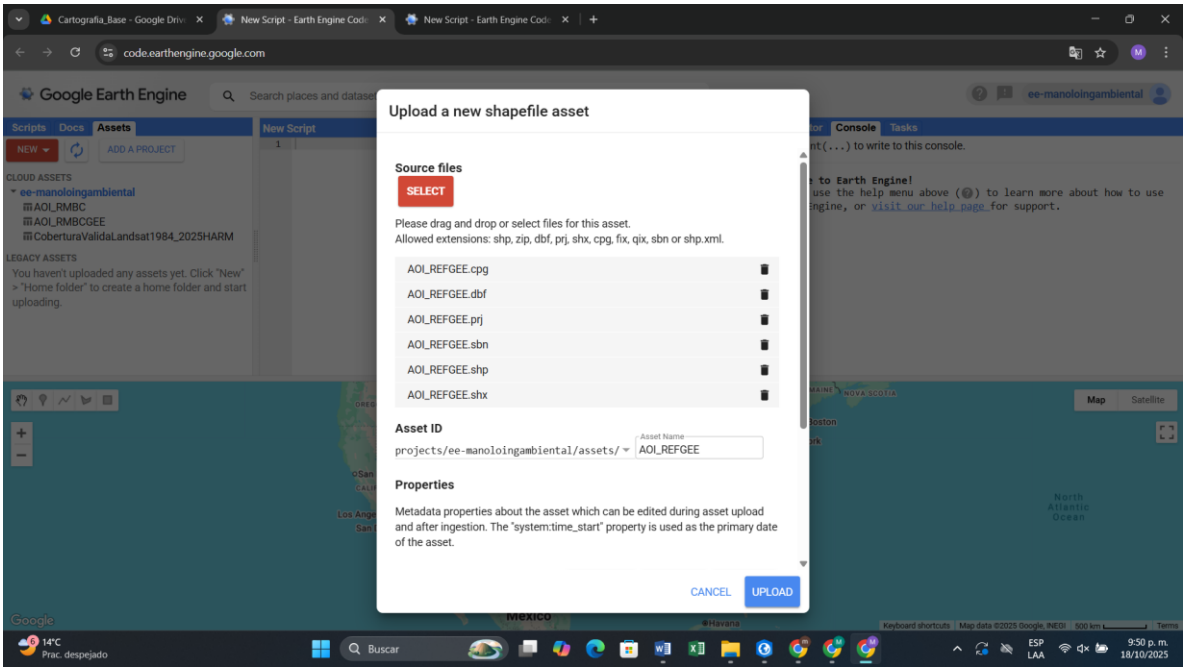


Imagen 2. Carga del AOI en Assets Google Earth Engine.

En el cuadro **Upload a new shapefile asset** se debe pulsar sobre el botón rojo **Select** → Es necesario cargar a Google Earth Engine estos 6 elementos que forman el shapefile del AOI (imagen 2) (Solo esos seis estrictamente) → Pulsar **UPLOAD**.

Cpg, dnf, prj, sbn, shp, shx.

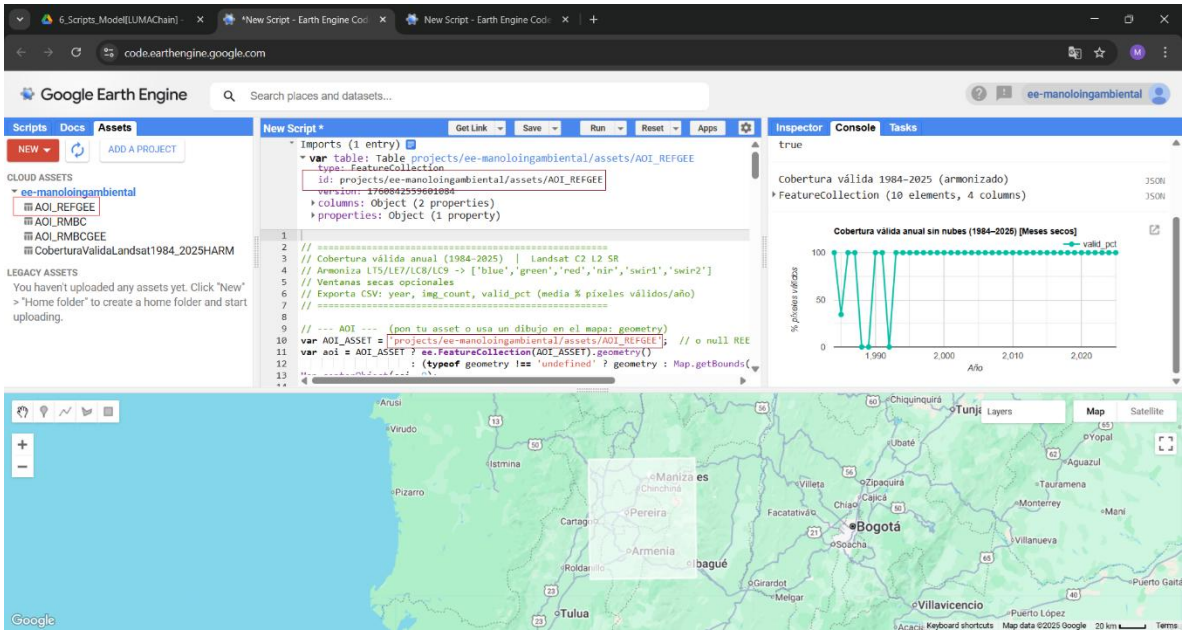
Fase 0 _ Validación_CoberturaLandSat1984_2025_GoogleEarthEngine

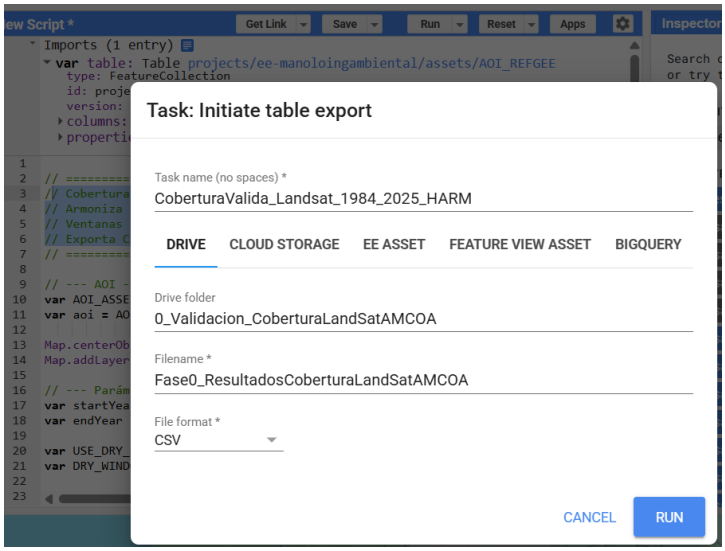
La fase cero del modelo tiene como propósito garantizar la disponibilidad y calidad de las imágenes Landsat que serán empleadas en las etapas posteriores del análisis multitemporal. Este algoritmo realiza un diagnóstico automatizado dentro del entorno Google Earth Engine, filtrando las escenas por cobertura nubosa, sombras y calidad radiométrica para el periodo 1984–2025. A partir del área de estudio (AOI) ingresada por el usuario, el script calcula el porcentaje de píxeles válidos y genera una estimación de la disponibilidad temporal de datos satelitales libres de nubosidad, permitiendo identificar los años o intervalos con mejor potencial para la generación de mosaicos. Esta verificación inicial es esencial, pues asegura que las composiciones multibanda producidas posteriormente se basen en insumos confiables, homogéneos y comparables, reduciendo el sesgo introducido por condiciones atmosféricas o vacíos de información.

```
// ================================================
// Cobertura válida anual (1984–2025) | Landsat C2 L2 SR
// Armoniza LT5/LE7/LC8/LC9 -> ['blue','green','red','nir','swir1','swir2']
// Ventanas secas opcionales
// Exporta CSV: year, img_count, valid_pct (media % píxeles válidos/año)
// ================================================

// --- AOI ---  (pon tu asset o usa un dibujo en el mapa: geometry)
var AOI_ASSET = 'projects/nucleo1mcl/assets/AOI_LCGEE';  // o null REEMPLAZA POR TU AOI RUTE
var aoi = AOI_ASSET ? ee.FeatureCollection(AOI_ASSET).geometry()
: (typeof geometry !== 'undefined' ? geometry : Map.getBounds(true));
Map.centerObject(aoi, 9);
Map.addLayer(aoi, {color:'white'}, 'AOI', false);

// --- Parámetros ---
var startYear = 1984;
var endYear   = 2025;

var USE_DRY_SEASON = true;          // true: solo ventanas secas
var DRY_WINDOWS = [[1,3],[7,9]];    // Ene–Mar y Jul–Sep

// reduceRegion para % válidos (sube si el AOI es muy grande)
var SCALE_REDUCE = 120;     // m
var TILE_SCALE   = 2;       // 1..4
var GEOM_SMOOTH  = 1000;    // m

// ---- Máscara QA_PIXEL (C2 L2) ----
function qaMask(img){
var qa = img.select('QA_PIXEL');
var dilated = qa.bitwiseAnd(1<<1).eq(0);
var cloud   = qa.bitwiseAnd(1<<3).eq(0);
var shadow  = qa.bitwiseAnd(1<<4).eq(0);
var snow    = qa.bitwiseAnd(1<<5).eq(0);
var water   = qa.bitwiseAnd(1<<7).eq(0);   // puedes omitir si no quieres retirar agua
var cirrus  = qa.bitwiseAnd(1<<9).eq(0);
var good = dilated.and(cloud).and(shadow).and(snow).and(water).and(cirrus);
return img.updateMask(good);
}

// ---- Escalado reflectancia: 0.0000275 y offset -0.2 ----
function scaleSR(img, bands){
return img.select(bands).multiply(0.0000275).add(-0.2);
}

// L5 / L7
function prepL57(img){
var s = scaleSR(img, ['SR_B1','SR_B2','SR_B3','SR_B4','SR_B5','SR_B7'])
.rename(['blue','green','red','nir','swir1','swir2']);
return qaMask(img).addBands(s, null, true)
```

```
.select(['blue','green','red','nir','swir1','swir2'])
.copyProperties(img, ['system:time_start']);
}

// L8 / L9
function prepL89(img){
var s = scaleSR(img, ['SR_B2','SR_B3','SR_B4','SR_B5','SR_B6','SR_B7'])
.rename(['blue','green','red','nir','swir1','swir2']);
return qaMask(img).addBands(s, null, true)
.select(['blue','green','red','nir','swir1','swir2'])
.copyProperties(img, ['system:time_start']);
}

// Colección armonizada LT5/LE7/LC8/LC9
function getHarmonizedCol(start, end){
var l5 = ee.ImageCollection('LANDSAT/LT05/C02/T1_L2')
.filterBounds(aoi).filterDate(start, end).map(prepL57);
var l7 = ee.ImageCollection('LANDSAT/LE07/C02/T1_L2')
.filterBounds(aoi).filterDate(start, end).map(prepL57);
var l8 = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2')
.filterBounds(aoi).filterDate(start, end).map(prepL89);
var l9 = ee.ImageCollection('LANDSAT/LC09/C02/T1_L2')
.filterBounds(aoi).filterDate(start, end).map(prepL89);

var col = l5.merge(l7).merge(l8).merge(l9)
.map(function(img){
var d = ee.Date(img.get('system:time_start'));
return img.set({year: d.get('year'), month: d.get('month')});
});

if (USE_DRY_SEASON){
var dry = ee.ImageCollection([]);
DRY_WINDOWS.forEach(function(w){
dry = dry.merge(col.filter(ee.Filter.calendarRange(w[0], w[1], 'month')));
});
col = dry;
}
return col;
}

// ---- Cálculo por año: img_count + % válidos ----
print('Procesando:', startYear, '-', endYear, '| Solo meses secos =', USE_DRY_SEASON);
var years = ee.List.sequence(startYear, endYear);

var features = years.map(function(y){
y = ee.Number(y);
var start = ee.Date.fromYMD(y,1,1);
var end   = start.advance(1,'year');

var col = getHarmonizedCol(start, end);
var nImg = col.size();

var composite = ee.Image(ee.Algorithms.If(
nImg.gt(0),
col.median().clamp(0,1),
ee.Image(0).updateMask(ee.Image(0))
));

// % válidos: media de la máscara (cualquier banda) dentro del AOI
var validPct = ee.Number(ee.Algorithms.If(
nImg.gt(0),
composite.select('red').mask().reduceRegion({
```

```
reducer: ee.Reducer.mean(),
geometry: aoi.simplify(GEOM_SMOOTH),
scale: SCALE_REDUCE,
maxPixels: 5e12,
tileScale: TILE_SCALE
}).values().get(0),
0
)).multiply(100);

return ee.Feature(null, {
year: y,
img_count: nImg,
valid_pct: validPct
});
});

var fc = ee.FeatureCollection(features).sort('year');
print('Cobertura válida 1984–2025 (armonizado)', fc.limit(10));

// ---- Gráfico rápido ----
var chart = ui.Chart.feature.byFeature(fc, 'year', ['valid_pct'])
.setOptions({
title: 'Cobertura válida anual sin nubes (1984–2025) ' + (USE_DRY_SEASON ? '[Meses secos]' : '[Todo el año]'),
hAxis: {title: 'Año'},
vAxis: {title: '% píxeles válidos', viewWindow: {min:0, max:100}},
lineWidth: 2, pointSize: 5, colors: ['#00BFA5']
});
print(chart);

// ---- Exportar CSV a Drive ----
Export.table.toDrive({
collection: fc,
description: 'CoberturaValida_Landsat_1984_2025_HARM',
fileNamePrefix: 'CoberturaValida_Landsat_1984_2025_HARM',
folder: '0_ValidacionCoberturaLandSatAMVA_AOA',
fileFormat: 'CSV'
});
```





<u>Nota: Los nombres de guardado de los resultados son del criterio del usuario sin embargo los nombres de las carpetas deben contener la misma coherencia numérica de cada fase.</u>

Fase 1_Composicion_MosaicosLandSat1984_2025_GoogleEarthEngine

El algoritmo automatiza la construcción de mosaicos multitemporales Landsat y, antes de componer, lee el diagnóstico de cobertura válida generado en la Fase 0 (porcentaje anual de píxeles libres de nubes para el AOI). Con esa información aplica una regla adaptativa que decide de forma automática el intervalo temporal óptimo (p. ej., 3/6/9 años) garantizando que cada bloque cumpla umbrales mínimos de calidad (por año y promedio del bloque). Para cada periodo seleccionado, integra Landsat 4/5/7/8/9 (C2 T1 L2), enmascara nubes, sombras, nieve y agua con QA_PIXEL, armoniza reflectancia de superficie y selecciona bandas ópticas e infrarrojo (2–7). La composición se obtiene por mediana o medoide priorizando ventanas secas (enero–marzo y julio–septiembre) para maximizar estabilidad radiométrica. Los mosaicos resultantes se reproyectan a MAGNA-SIRGAS EPSG:3116 (consistencia métrica para estadísticas zonales) y se exportan automáticamente a Drive o Assets con metadatos y nombres estandarizados (período, regla de sensor, método de composición).

Run_Fase1

Para correr el codigo es necesario cargar el AOI desde Assets y cambiar las lineas que defien la variable AOI, pulsar RUN.

```
/**********************************************************
* LUMAChain – Fase 1 (SILENCIOSO)
* Periodos 3/6/9 desde QA Fase 0 + DEM ALOS, solo exporta
**********************************************************/

/* [S0] PARÁMETROS GENERALES (SIN CAMBIOS) */
var AOI_MODE   = 'ASSET';
var AOI_ASSET  = 'projects/nucleo1mcl/assets/AOI_LCGEE';

var USE_DRY_SEASON = true;              // usa meses secos
```

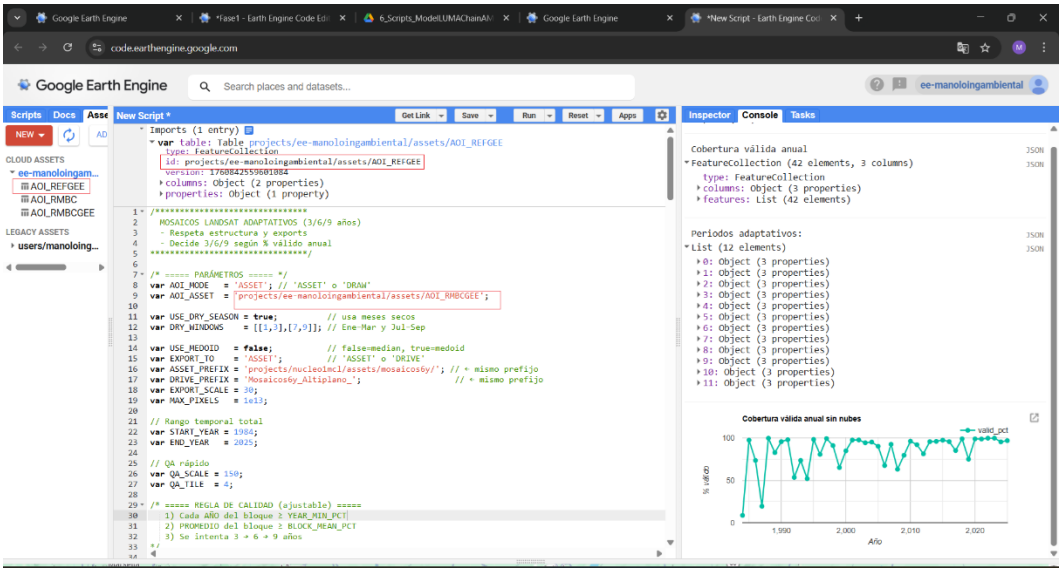
```
var DRY_WINDOWS    = [[1,3],[7,9]];      // Ene–Mar y Jul–Sep

var USE_MEDOID   = false;                // false=median, true=medoid  ← si ves pixelado, prueba
true
var EXPORT_TO    = 'DRIVE';              // 'ASSET' o 'DRIVE'
var ASSET_PREFIX = 'projects/nucleo1mcl/assets/mosaicos6y/';
var DRIVE_PREFIX = 'Mosaicos6y_Altiplano_';
var DRIVE_FOLDER = '1_ComposicionesMultibandaAMVA_AOA';

var EXPORT_SCALE = 30;
var MAX_PIXELS   = 1e13;

// Rango temporal total
var START_YEAR = 1984;
var END_YEAR   = 2025;

// Reglas adaptativas
var YEAR_MIN_PCT    = 40;
var BLOCK_MEAN_PCT  = 70;
var CANDIDATE_SIZES = [3, 6, 9];

// QA Fase 0 (tu tabla ya calculada)
var QA_TABLE_ASSET  =
'projects/nucleo1mcl/assets/CoberturaValida_Landsat_1984_2025_HARM';

/* [S1] CRS / Escala */
var OUT_CRS   = 'EPSG:3116';
var OUT_SCALE = 30;

/* [S2] AOI */
var aoi_full;
if (AOI_MODE === 'ASSET') {
aoi_full = ee.FeatureCollection(AOI_ASSET).geometry();
} else {
try { aoi_full = geometry; }
catch (e) { aoi_full = ee.Geometry.Rectangle([-180,-80,180,84], null, false); }
}
var aoi = aoi_full.simplify(1500);

/* [S3] HELPERS Landsat */
function maskQA_LS(img){
var qa = img.select('QA_PIXEL');
var mask = qa.bitwiseAnd(1<<1).eq(0)
.and(qa.bitwiseAnd(1<<3).eq(0))
.and(qa.bitwiseAnd(1<<4).eq(0))
.and(qa.bitwiseAnd(1<<5).eq(0))
.and(qa.bitwiseAnd(1<<7).eq(0));
return mask;
}
function prepL57(img){
return img.updateMask(maskQA_LS(img))
```

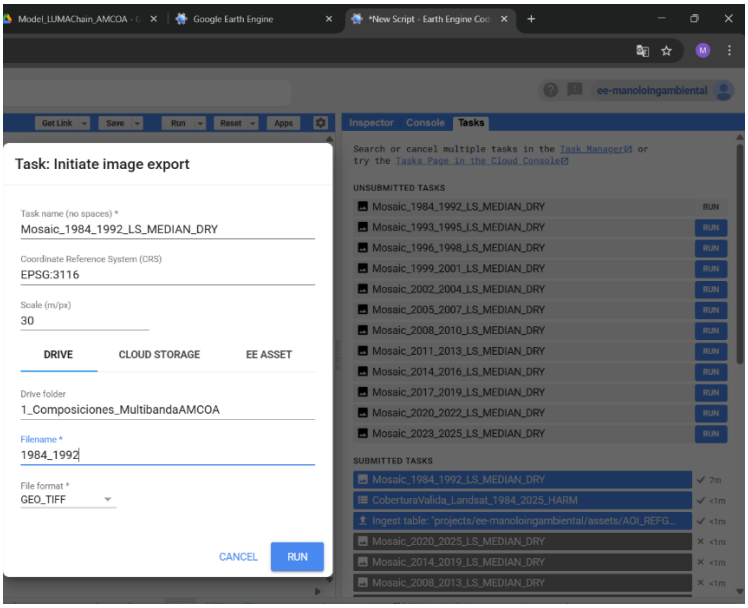
```
.select(['SR_B1','SR_B2','SR_B3','SR_B4','SR_B5','SR_B7'],
['blue','green','red','nir','swir1','swir2'])
.multiply(0.0000275).add(-0.2)
.copyProperties(img, ['system:time_start']);
}
function prepL8_9(img){
return img.updateMask(maskQA_LS(img))
.select(['SR_B2','SR_B3','SR_B4','SR_B5','SR_B6','SR_B7'],
['blue','green','red','nir','swir1','swir2'])
.multiply(0.0000275).add(-0.2)
.copyProperties(img, ['system:time_start']);
}
function getLandsatCol(start, end){
start = ee.Date(start); end = ee.Date(end);
var l4 = ee.ImageCollection('LANDSAT/LT04/C02/T1_L2').filterBounds(aoi).filterDate(start,
end).map(prepL57);
var l5 = ee.ImageCollection('LANDSAT/LT05/C02/T1_L2').filterBounds(aoi).filterDate(start,
end).map(prepL57);
var l7 = ee.ImageCollection('LANDSAT/LE07/C02/T1_L2').filterBounds(aoi).filterDate(start,
end).map(prepL57);
var l8 = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2').filterBounds(aoi).filterDate(start,
end).map(prepL8_9);
var l9 = ee.ImageCollection('LANDSAT/LC09/C02/T1_L2').filterBounds(aoi).filterDate(start,
end).map(prepL8_9);
var col = l4.merge(l5).merge(l7).merge(l8).merge(l9);
if (USE_DRY_SEASON){
var dry = ee.ImageCollection([]);
DRY_WINDOWS.forEach(function(w){
dry = dry.merge(col.filter(ee.Filter.calendarRange(w[0], w[1], 'month')));
});
col = dry;
}
return col;
}
function medoid(ic){
var median = ic.median();
var diffs = ic.map(function(img){
var d = img.subtract(median).pow(2);
var s = d.reduce(ee.Reducer.sum());
return s.addBands(img);
});
return diffs.qualityMosaic('sum').select(ic.first().bandNames());
}

/* [S4] Periodos 3/6/9 desde QA Fase 0 (1 sola agregación + 1 getInfo) */
function buildPeriodsFromQATable(qat){
// lista de pares [year, valid_pct]
var pairsList = ee.List(
qat.reduceColumns({
reducer: ee.Reducer.toList(2),
selectors: ['year','valid_pct']
```

```
}).get('list')
);
var pairs = pairsList.getInfo(); // una sola llamada al cliente

// mapa año -> %
var vmap = {};
for (var i=0; i<pairs.length; i++){
var yy = Math.floor(pairs[i][0]);
var vp = pairs[i][1] != null ? pairs[i][1] : 0;
vmap[yy] = vp;
}
// reglas 3/6/9
var out = [];
var y = START_YEAR;
while (y <= END_YEAR){
var chosen = null;
for (var k=0; k<CANDIDATE_SIZES.length; k++){
var w    = CANDIDATE_SIZES[k];
var yEnd = Math.min(END_YEAR, y + w - 1);
var vals = [];
for (var yy=y; yy<=yEnd; yy++){ vals.push(vmap[yy] || 0); }
var minPct  = Math.min.apply(null, vals);
var meanPct = vals.reduce(function(a,b){return a+b;},0) / vals.length;
if (minPct >= YEAR_MIN_PCT && meanPct >= BLOCK_MEAN_PCT){
chosen = { name: (y + '_' + yEnd), start: y + '-01-01', end: yEnd + '-12-31' };
break;
}
}
if (!chosen){
var forcedEnd = Math.min(END_YEAR, y + 9 - 1);
chosen = { name: (y + '_' + forcedEnd), start: y + '-01-01', end: forcedEnd + '-12-31' };
}
out.push(chosen);
y = parseInt(chosen.name.split('_')[1],10) + 1;
}
return out;
}

/* [S5] DEM ALOS AW3D30 (30 m, banda DSM) → export */
function exportDEM(){
var dem = ee.ImageCollection('JAXA/ALOS/AW3D30/V3_2')
.select('DSM')
.mosaic()
.clip(aoi_full)
.reproject({crs: OUT_CRS, scale: OUT_SCALE})
.rename('DEM');

var demName = 'APDEM';
if (EXPORT_TO === 'ASSET') {
Export.image.toAsset({
image: dem, description: demName, assetId: ASSET_PREFIX + demName,
```

```
region: aoi_full, scale: OUT_SCALE, crs: OUT_CRS, maxPixels: MAX_PIXELS
});
} else {
Export.image.toDrive({
image: dem, description: demName, fileNamePrefix: DRIVE_PREFIX + demName,
region: aoi_full, scale: OUT_SCALE, crs: OUT_CRS, maxPixels: MAX_PIXELS,
folder: DRIVE_FOLDER
});
}
print('OK: DEM Save →', demName);
}

/* [S6] Mosaicos + export (SIN capas ni gráficos) */
function runAll(){
var qaTable = ee.FeatureCollection(QA_TABLE_ASSET);
var PERIODS = buildPeriodsFromQATable(qaTable);

PERIODS.forEach(function (p) {
var pstart = p.start, pend = p.end, pname = p.name;

var col   = getLandsatCol(pstart, pend);
var count = col.size();

var composite = ee.Image(ee.Algorithms.If(
count.gt(0),
(USE_MEDOID ? medoid(col) : col.median()).clamp(0, 1),
ee.Image(0).updateMask(ee.Image(0))
)).reproject({ crs: OUT_CRS, scale: OUT_SCALE })
.set({ period: pname, img_count: count, sensor_rule: 'LANDSAT_ONLY' });

var outName = 'Mosaic_' + pname + '_LS_' +
(USE_MEDOID ? 'MEDOID' : 'MEDIAN') + '_' +
(USE_DRY_SEASON ? 'DRY' : 'ALLYEAR');

if (EXPORT_TO === 'ASSET') {
Export.image.toAsset({
image: composite,
description: outName,
assetId: ASSET_PREFIX + outName,
region: aoi_full,
scale: OUT_SCALE,
crs: OUT_CRS,
maxPixels: MAX_PIXELS
});
} else {
Export.image.toDrive({
image: composite,
description: outName,
fileNamePrefix: DRIVE_PREFIX + outName,
region: aoi_full,
scale: OUT_SCALE,
```

```
crs: OUT_CRS,
folder: DRIVE_FOLDER,
maxPixels: MAX_PIXELS
});
}
print('OK: mosaico Save →', outName);
});
}

/* [S7] Ejecutar (solo “OK” en consola) */
exportDEM();
runAll();
```





Posteriormente se generan las capas multibanda correspondientes a cada periodo en la pestaña Task se deben correr individualmente cada composición multibanda, y guardar en drive:

Folder (carpeta): 1_Composiciones_MultibandaAMCOA

File Name: Año(i)guionbajoAño(f) = 1984_1992 → los nombres son opcionales y se pueden editar a petición del usuario sin embargo es necesario que conserven la coherencia de este formato sugerido.

Fase 2_Calculo_IndicesNormalizados_GoogleColab

Preprocesados los datos en la plataforma Google Earth Engine y descargados a Google Drive procedemos al entorno Google Colab, creando un proyecto con los nombres predestinados, posteriormente se corre el algoritmo Fase 2 tras cambiar las rutas. Dando acceso a la carpeta que contiene las composiciones multibanda descargadas desde GEE. Y seguida la ruta donde se guardarán los índices normalizados.

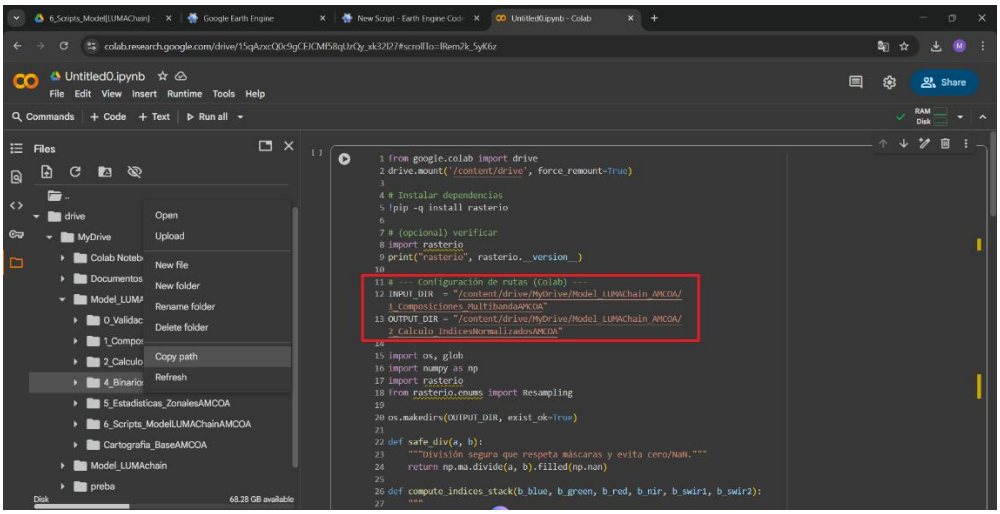
```
# === Fase 2: Cálculo de índices NORMALIZADOS (salida en archivos individuales) + SLOPE desde DEM ===
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Dependencias
!pip -q install rasterio

import os, glob
import numpy as np
import rasterio
from rasterio.enums import Resampling

print("rasterio", rasterio.__version__)

# --- Rutas ---
INPUT_DIR  = "/content/drive/MyDrive/Model_LUMAChainAMVA_AOA/1_ComposicionesMultibandaAMVA_AOA"
OUTPUT_DIR = "/content/drive/MyDrive/Model_LUMAChainAMVA_AOA/2_CalculoIndicesAMVA_AOA"
os.makedirs(OUTPUT_DIR, exist_ok=True)

# --- Parámetro NoData para escritura ---
NODATA_VAL = -9999.0

# --- Helpers ---
def safe_div(a, b):
    """División segura que respeta máscaras y evita ±inf/NaN devolviendo MaskedArray."""
    with np.errstate(divide='ignore', invalid='ignore'):
        out = np.ma.divide(a, b)  # mantiene máscara
    return out

def clip_idx(arr, lo=-1.0, hi=1.0):
    """Clipa índice a [-1, 1] manteniendo máscara."""
    return np.ma.clip(arr, lo, hi)

def compute_indices_dict(b_blue, b_green, b_red, b_nir, b_swir1, b_swir2):
    """
    Retorna un dict con arrays float32 enmascarados: NDVI, NDBI, NDWI, MNDWI, IBI, SAVI
    Entradas: np.ma.MaskedArray (float32).
    """
    # NDVI = (NIR - RED) / (NIR + RED)
    ndvi  = safe_div(b_nir - b_red,   b_nir + b_red)

    # NDBI = (SWIR1 - NIR) / (SWIR1 + NIR)
    ndbi  = safe_div(b_swir1 - b_nir, b_swir1 + b_nir)
```

```
    # NDWI (McFeeters) = (GREEN - NIR) / (GREEN + NIR)
    ndwi  = safe_div(b_green - b_nir, b_green + b_nir)

    # MNDWI (Xu) = (GREEN - SWIR1) / (GREEN + SWIR1)
    mndwi = safe_div(b_green - b_swir1, b_green + b_swir1)

    # IBI (Xu 2008)
    ibi_num = ndbi - (ndvi + mndwi) / 2.0
    ibi_den = ndbi + (ndvi + mndwi) / 2.0
    ibi     = safe_div(ibi_num, ibi_den)

    # SAVI (L=0.5)
    L = 0.5
    savi = safe_div((b_nir - b_red) * (1.0 + L), (b_nir + b_red + L))

    # Clip a [-1,1] para evitar outliers que revientan el estiramiento
    return {
        "NDVI":  clip_idx(ndvi ).astype("float32"),
        "NDBI":  clip_idx(ndbi ).astype("float32"),
        "NDWI":  clip_idx(ndwi ).astype("float32"),
        "MNDWI": clip_idx(mndwi).astype("float32"),
        "IBI":   clip_idx(ibi  ).astype("float32"),
        "SAVI":  clip_idx(savi ).astype("float32"),
    }

def write_single_band(out_path, arr_masked, base_profile, desc_name):
    """
    Escribe un GeoTIFF de 1 banda (float32, LZW, nodata=-9999) manteniendo georreferenciación.
    Rellena la máscara con NODATA_VAL antes de escribir.
    """
    arr = arr_masked.filled(NODATA_VAL).astype("float32")

    profile = base_profile.copy()
    profile.update(
        dtype="float32",
        count=1,
        nodata=NODATA_VAL,
        compress="lzw"
    )
    with rasterio.open(out_path, "w", **profile) as dst:
        dst.write(arr, 1)
        dst.set_band_description(1, desc_name)

def _qa_print(name, arr_masked):
    """QA rápido para chequear rangos."""
    arr = arr_masked.compressed()
    if arr.size == 0:
        print(f"  {name}: sin datos válidos")
        return
    p2, p98 = np.percentile(arr, [2, 98])
    print(f"  {name}: min={arr.min():.3f}  max={arr.max():.3f}  p2={p2:.3f}  p98={p98:.3f}")

def process_one_tif(path):
    base = os.path.splitext(os.path.basename(path))[0]

    # Evitar tomar el DEM si está en la misma carpeta por error
    if base.upper().startswith("DEM"):
        print(f"• Omitido DEM: {path}")
        return

    with rasterio.open(path) as src:
        profile = src.profile.copy()
```

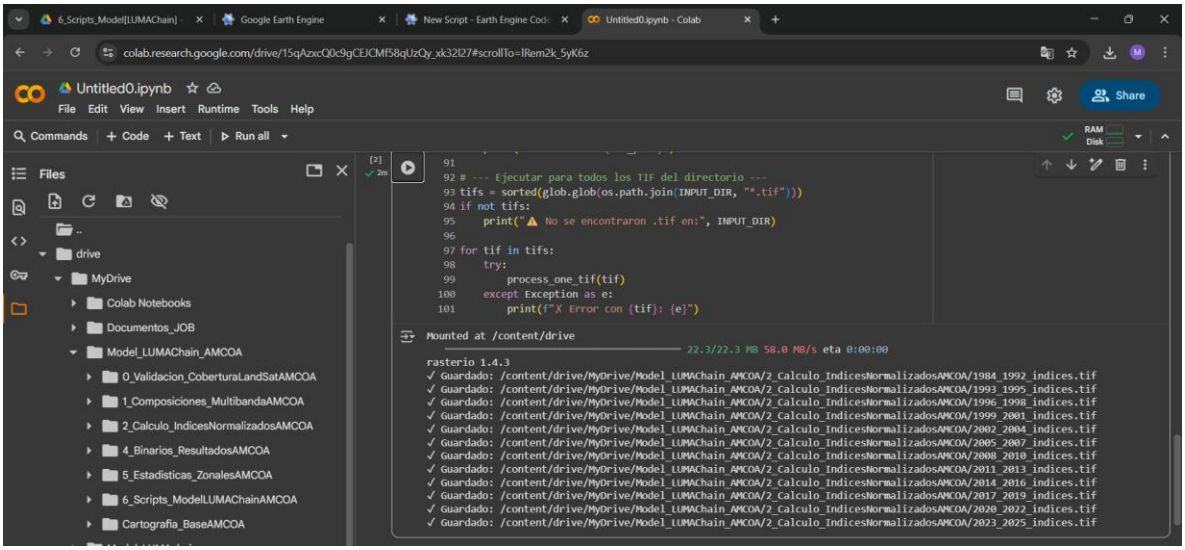
```
        # Lee bandas como masked arrays (respeta máscara/nodata del multibanda)
        b1 = src.read(1, masked=True).astype("float32")  # blue
        b2 = src.read(2, masked=True).astype("float32")  # green
        b3 = src.read(3, masked=True).astype("float32")  # red
        b4 = src.read(4, masked=True).astype("float32")  # nir
        b5 = src.read(5, masked=True).astype("float32")  # swir1
        b6 = src.read(6, masked=True).astype("float32")  # swir2

        idx = compute_indices_dict(b1, b2, b3, b4, b5, b6)

        # QA rápido
        print(f"QA {base}:")
        for k in ["NDVI","NDBI","NDWI","MNDWI","IBI","SAVI"]:
            _qa_print(k, idx[k])

        # Guarda 6 archivos individuales
        for name, arr in idx.items():
            out_path = os.path.join(OUTPUT_DIR, f"{base}_{name.lower()}.tif")
            write_single_band(out_path, arr, profile, name)
            print(f"✓ {name:<5} → {out_path}")

# ================== NUEVO: cálculo de pendiente (slope) desde DEM ==================
def _horn_slope_deg(dem_masked, transform):
    """
    Calcula pendiente (grados) con el operador de Horn usando una ventana 3x3.
    Respeta la máscara: requiere vecindad 3x3 válida, sino enmascara.
    """
    dem = dem_masked  # np.ma.MaskedArray
    if dem.mask is np.ma.nomask:
        mask = np.zeros_like(dem, dtype=bool)
    else:
        mask = dem.mask.astype(bool)

    # Tamaño de píxel (m)
    cell_x = abs(transform.a)
    cell_y = abs(transform.e)

    # Preparar salida
    h, w = dem.shape
    slope = np.ma.masked_all((h, w), dtype="float32")

    # Construir vecinos (índices 1..9 estilo Horn)
    z1 = dem[:-2, :-2]
    z2 = dem[:-2, 1:-1]
    z3 = dem[:-2, 2:]
    z4 = dem[1:-1, :-2]
    z5 = dem[1:-1, 1:-1]
    z6 = dem[1:-1, 2:]
    z7 = dem[2:, :-2]
    z8 = dem[2:, 1:-1]
    z9 = dem[2:, 2:]

    # Máscara válida si todos los 3x3 son válidos
    m1 = mask[:-2, :-2]; m2 = mask[:-2, 1:-1]; m3 = mask[:-2, 2:]
    m4 = mask[1:-1, :-2]; m5 = mask[1:-1, 1:-1]; m6 = mask[1:-1, 2:]
    m7 = mask[2:, :-2];  m8 = mask[2:, 1:-1];  m9 = mask[2:, 2:]
    valid = ~(m1 | m2 | m3 | m4 | m5 | m6 | m7 | m8 | m9)

    # Gradientes (Horn 1981)
    dzdx = ((z3 + 2*z6 + z9) - (z1 + 2*z4 + z7)) / (8.0 * cell_x)
    dzdy = ((z7 + 2*z8 + z9) - (z1 + 2*z2 + z3)) / (8.0 * cell_y)
```

```
    # Pendiente en radianes y luego grados
    slope_inner = np.arctan(np.hypot(dzdx, dzdy))
    slope_deg = np.degrees(slope_inner).astype("float32")

    # Rellenar la zona central
    out = np.ma.masked_all((h, w), dtype="float32")
    out[1:-1, 1:-1] = np.ma.array(slope_deg, mask=~valid)
    return out

def _write_slope(out_path, slope_masked, base_profile):
    arr = slope_masked.filled(NODATA_VAL).astype("float32")
    profile = base_profile.copy()
    profile.update(dtype="float32", count=1, nodata=NODATA_VAL, compress="lzw")
    with rasterio.open(out_path, "w", **profile) as dst:
        dst.write(arr, 1)
        dst.set_band_description(1, "SLOPE_DEGREES")

def _looks_like_dem(name):
    """Heurística simple por nombre."""
    u = name.upper()
    # Cubre nombres típicos: DEM, APDEM, MDE, MDT, SRTM, ALTIMETRY, ELEVATION
    keys = ["DEM", "APDEM", "MDE", "MDT", "ELEV", "SRTM", "ALTIM"]
    return any(k in u for k in keys)

def find_dem_in_dir(folder):
    """Busca un .tif que parezca DEM. Prioriza nombre, luego 1 banda."""
    cands = sorted(glob.glob(os.path.join(folder, "*.tif")))
    # 1) por nombre
    by_name = [p for p in cands if _looks_like_dem(os.path.basename(p))]
    if by_name:
        return by_name[0]
    # 2) por estructura (1 banda, float/int, no multibanda clásico)
    for p in cands:
        try:
            with rasterio.open(p) as src:
                if src.count == 1 and src.dtypes[0] in ("int16","int32","float32","float64"):
                    return p
        except Exception:
            pass
    return None

def process_dem_slope(input_dir, output_dir):
    dem_path = find_dem_in_dir(input_dir)
    if dem_path is None:
        print("⚠ No se encontró DEM en:", input_dir)
        return
    base_dem = os.path.splitext(os.path.basename(dem_path))[0]
    print(f"• DEM detectado: {dem_path}")

    with rasterio.open(dem_path) as src:
        profile = src.profile.copy()
        dem = src.read(1, masked=True).astype("float32")
        slope_deg = _horn_slope_deg(dem, src.transform)

        # QA rápido de la pendiente
        arr = slope_deg.compressed()
        if arr.size:
            p2, p98 = np.percentile(arr, [2, 98])
            print(f"QA SLOPE {base_dem}: min={arr.min():.2f}  max={arr.max():.2f}  p2={p2:.2f}  p98={p98:.2f}")
        else:
            print(f"QA SLOPE {base_dem}: sin datos válidos")
```

```
        out_slope = os.path.join(output_dir, f"{base_dem}_slope_deg.tif")
        _write_slope(out_slope, slope_deg, profile)
        print(f"✓ SLOPE → {out_slope}")

# --- Lote índices ---
tifs = sorted(glob.glob(os.path.join(INPUT_DIR, "*.tif")))
if not tifs:
    print("⚠ No se encontraron .tif en:", INPUT_DIR)

for tif in tifs:
    try:
        process_one_tif(tif)
    except Exception as e:
        print(f"✗ Error con {tif}: {e}")

# --- Cálculo de pendiente desde DEM (en la misma carpeta de INPUT_DIR) ---
try:
    process_dem_slope(INPUT_DIR, OUTPUT_DIR)
except Exception as e:
    print(f"✗ Error calculando SLOPE desde DEM: {e}")
```



En esta fase, se toman los mosaicos multibanda generados en la Fase 1 y, de forma automática y por periodo, se reproyectán y alinean a una grilla común (EPSG:3116, 30 m) dentro del AOI. Sobre las bandas estandarizadas (blue, green, red, NIR, SWIR1, SWIR2) el script calcula los índices clave para detección urbana y cobertura: NDVI (vegetación), NDBI (intensidad constructiva), MNDWI (agua) y IBI (índice urbano integrado). Los índices se generan en formato float32 (rango típico −1 a 1), se aplican máscaras de validez/ AOI y se guardan en disco siguiendo la nomenclatura por periodo, junto con capas de control de calidad (QC) cuando corresponde. El procesamiento es batch (todos los periodos en una corrida), garantiza coherencia espacial entre años y deja listos los insumos que usarán las fases siguientes para umbrales adaptativos, binarización urbana y estadísticas.



Fase 3. Clasificación Supervisada

3_ClasificacionSupervisadaAMVA_AOAGoogleColab

Este condigo consiste en la definición de la base binaria y reglas de clasificación (ArcPy) constituye el paso en el que se transforman los índices espectrales generados en la Fase 2 en una representación categórica simplificada de la cobertura urbana. Mediante el uso de ArcPy y herramientas del módulo Spatial Analyst, se ejecuta una clasificación no supervisada (IsoCluster) sobre las composiciones multibanda, utilizando los índices NDVI, NDBI y NDWI como variables de entrada. Posteriormente, se aplica una máscara del área de estudio (AOI) y un proceso de filtrado mayoritario (Majority Filter) que elimina ruido espectral y consolida las agrupaciones más homogéneas. El resultado es una capa binaria base (urbano / no urbano) que servirá como insumo de referencia para la comparación temporal de coberturas y el análisis de expansión urbana en las fases posteriores.

Es necesario cargar en un nuevo proyecto del entorno ArcGISPro los siguientes datos:

- 1984_1989.tif correspondiente a la composición multibanda del periodo inicial
- 1984_1989_Indices.tif correspondiente a la composición de índices normalizados
- AOI_AMVA+AOA correspondiente al área de estudio en formato shapefile.

3.2_RUN Code “3_ ClasificacionSupervisadaAMVA_AOAGoogleColab”

```
# === Fase 3 (batch): combinar índices + pendiente para TODOS los periodos detectados ===
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

!pip -q install rasterio

import os, glob, re
import numpy as np
import rasterio
from rasterio.enums import Resampling
from rasterio.warp import reproject

print("rasterio", rasterio.__version__)

# ---------------- RUTAS ----------------
INPUT_DIR  = "/content/drive/MyDrive/Model_LUMAChainAMVA_AOA/2_CalculoIndicesAMVA_AOA"
```

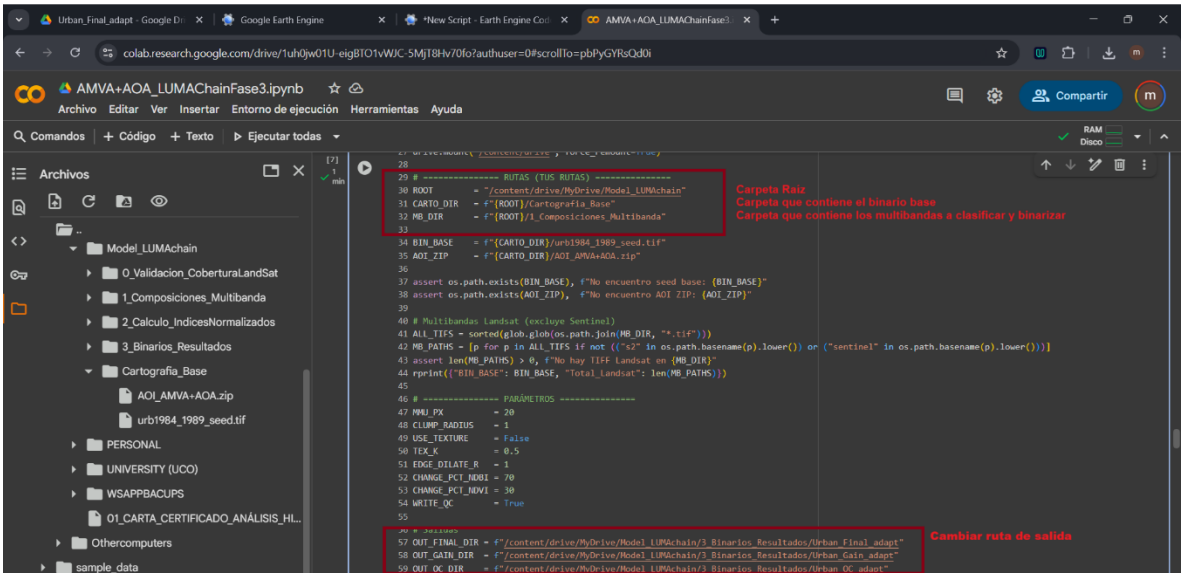
```
OUTPUT_DIR = "/content/drive/MyDrive/Model_LUMAChainAMVA_AOA/3_MuestrasEntrenamientoAMVA_AOA"
os.makedirs(OUTPUT_DIR, exist_ok=True)

NODATA_VAL = -9999.0

# ---------- HELPERS ----------
def _pick(pattern):
    c = sorted(glob.glob(pattern))
    return c[0] if c else None

def read_masked(path):
    with rasterio.open(path) as src:
        arr = src.read(1, masked=True).astype("float32")
        prof = src.profile.copy()
    return arr, prof

def reproject_like(arr_src, prof_src, prof_dst, resampling=Resampling.bilinear):
    dst = np.empty((prof_dst["height"], prof_dst["width"]), dtype="float32")
    reproject(
        source=arr_src.filled(np.nan),
        destination=dst,
        src_transform=prof_src["transform"],
        src_crs=prof_src["crs"],
        dst_transform=prof_dst["transform"],
        dst_crs=prof_dst["crs"],
        resampling=resampling,
        src_nodata=np.nan,
        dst_nodata=np.nan
    )
    return np.ma.array(dst, mask=np.isnan(dst))

def align_to(ref_prof, arr, prof, resampling=Resampling.bilinear):
    same = (prof["crs"] == ref_prof["crs"] and
            prof["transform"] == ref_prof["transform"] and
            prof["width"] == ref_prof["width"] and
            prof["height"] == ref_prof["height"])
    return arr if same else reproject_like(arr, prof, ref_prof, resampling=resampling)

def base_profile_like(ref_prof, dtype="float32"):
    prof = ref_prof.copy()
    prof.update(dtype=dtype, count=1, nodata=NODATA_VAL, compress="lzw")
    return prof

def write_tif(path, arr_masked, base_prof, band_desc=None):
    arr = arr_masked.filled(NODATA_VAL).astype("float32")
    prof = base_profile_like(base_prof, dtype="float32")
    with rasterio.open(path, "w", **prof) as dst:
        dst.write(arr, 1)
        if band_desc:
            dst.set_band_description(1, band_desc)

def con(mask_bool, a_true, a_false, *mas_for_mask):
    m = np.zeros_like(mask_bool, dtype=bool)
    for a in mas_for_mask:
        m |= np.ma.getmaskarray(a)
    res = np.where(mask_bool, a_true, a_false).astype("float32")
    return np.ma.array(res, mask=m)

def combine_masks(*mas):
    m = np.zeros_like(mas[0], dtype=bool)
    for a in mas:
        m |= np.ma.getmaskarray(a)
```

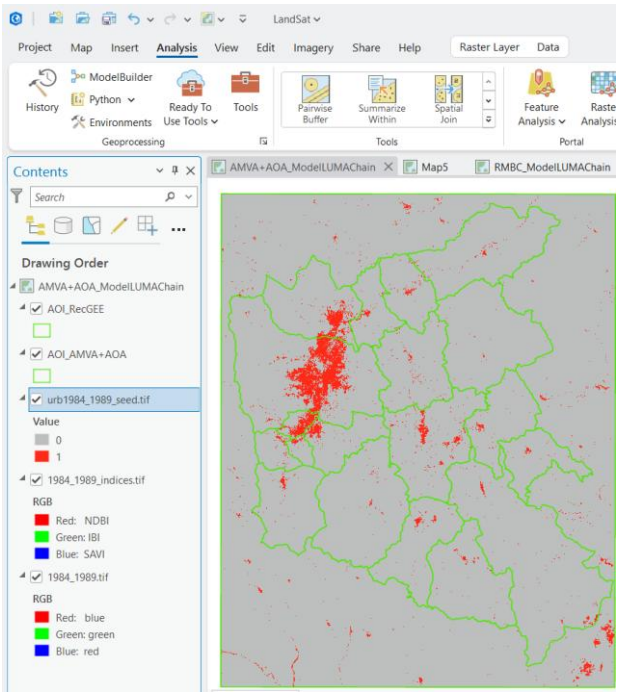
```
    return m

def list_periods(input_dir):
    # detecta nombres como "YYYY_YYYY_ndvi.tif" o "<PERIODO>_ndvi.tif"
    ndvi_files = glob.glob(os.path.join(input_dir, "*_ndvi.tif"))
    periods = []
    for p in ndvi_files:
        base = os.path.basename(p)
        per = re.sub(r"_ndvi\.tif$", "", base, flags=re.IGNORECASE)
        periods.append(per)
    return sorted(set(periods))

# ----------------- UMBRALES (ajustables, mismos para todos) -----------------
UM_U_NDBI_MIN   = 0.10
UM_U_NDVI_MAX   = 0.25
UM_U_MNDWI_MAX  = 0.05
UM_U_SLOPE_MAX  = 25

UM_W_MNDWI_MIN  = 0.20
UM_W_NDVI_MAX   = 0.10

UM_V_NDVI_MIN   = 0.40
UM_V_SAVI_MIN   = 0.50

UM_B_NDVI_MIN   = 0.10
UM_B_NDVI_MAX   = 0.35
UM_B_MNDWI_MAX  = 0.00
UM_B_NDBI_MIN   = -0.10
UM_B_NDBI_MAX   = 0.20

UM_SH_SLOPE_MIN = 30
UM_SH_NDVI_MAX  = 0.15
UM_SH_MNDWI_MAX = 0.00

W_NDBI, W_NDVIi, W_MNDWIi, W_SLOPEp = 0.50, 0.30, 0.20, 0.10

# ----------------- DETECTAR PERIODOS -----------------
PERIODS = list_periods(INPUT_DIR)
if not PERIODS:
    raise RuntimeError(f"No encontré *_ndvi.tif en {INPUT_DIR}")
print("Periodos detectados:", PERIODS)

# For stack: referencia global = primer NDVI del primer periodo
ref_prof_global = None
stack_bands = []
stack_names = []

# ----------------- LOOP POR PERIODO -----------------
for PERIODO in PERIODS:
    print(f"\n— Procesando periodo: {PERIODO}")

    ndvi_p  = _pick(os.path.join(INPUT_DIR, f"{PERIODO}_ndvi.tif"))
    ndbi_p  = _pick(os.path.join(INPUT_DIR, f"{PERIODO}_ndbi.tif"))
    mndwi_p = _pick(os.path.join(INPUT_DIR, f"{PERIODO}_mndwi.tif"))
    savi_p  = _pick(os.path.join(INPUT_DIR, f"{PERIODO}_savi.tif"))  # opcional
    ibi_p   = _pick(os.path.join(INPUT_DIR, f"{PERIODO}_ibi.tif"))   # opcional

    # slope: común (un solo DEM) o por periodo si existe
    slope_p = _pick(os.path.join(INPUT_DIR, "*slope*deg*.tif")) or \
              _pick(os.path.join(INPUT_DIR, f"{PERIODO}_slope_deg.tif"))

    if not (ndvi_p and ndbi_p and mndwi_p and slope_p):
```

```
        print(f"   ⚠ Faltan insumos en {PERIODO}. Saltando.")
        continue

    NDVI, prof_ref = read_masked(ndvi_p)
    NDBI,p = read_masked(ndbi_p);   NDBI  = align_to(prof_ref, NDBI, p, Resampling.bilinear)
    MNDWI,p= read_masked(mndwi_p);  MNDWI = align_to(prof_ref, MNDWI, p, Resampling.bilinear)
    SLOPE,p= read_masked(slope_p);  SLOPE = align_to(prof_ref, SLOPE, p, Resampling.bilinear)

    SAVI = None
    if savi_p:
        SAVI,p = read_masked(savi_p)
        SAVI = align_to(prof_ref, SAVI, p, Resampling.bilinear)

    # Reescalados
    NDBI_pos  = (NDBI + 1.0) / 2.0
    NDVI_inv  = 1.0 - ((NDVI + 1.0) / 2.0)
    MNDWI_inv = 1.0 - ((MNDWI + 1.0) / 2.0)
    SLOPE_pen = np.minimum(SLOPE / 45.0, 1.0)

    # Máscaras
    UrbC = con((NDBI >= UM_U_NDBI_MIN) & (NDVI <= UM_U_NDVI_MAX) &
               (MNDWI <= UM_U_MNDWI_MAX) & (SLOPE <= UM_U_SLOPE_MAX), 1, 0,
               NDBI, NDVI, MNDWI, SLOPE)

    AguaC = con((MNDWI >= UM_W_MNDWI_MIN) & (NDVI <= UM_W_NDVI_MAX), 1, 0, MNDWI, NDVI)

    if SAVI is not None:
        VegC = con((NDVI >= UM_V_NDVI_MIN) | (SAVI >= UM_V_SAVI_MIN), 1, 0, NDVI, SAVI)
    else:
        VegC = con((NDVI >= UM_V_NDVI_MIN), 1, 0, NDVI)

    SueloC = con((NDVI > UM_B_NDVI_MIN) & (NDVI < UM_B_NDVI_MAX) &
                 (MNDWI < UM_B_MNDWI_MAX) &
                 (NDBI > UM_B_NDBI_MIN) & (NDBI < UM_B_NDBI_MAX), 1, 0, NDVI, MNDWI, NDBI)

    SombraC = con((SLOPE >= UM_SH_SLOPE_MIN) & (NDVI <= UM_SH_NDVI_MAX) &
                  (MNDWI < UM_SH_MNDWI_MAX), 1, 0, SLOPE, NDVI, MNDWI)

    # UrbScore 0..100
    Score = 100.0 * (W_NDBI*NDBI_pos + W_NDVIi*NDVI_inv + W_MNDWIi*MNDWI_inv -
W_SLOPEp*SLOPE_pen)
    UrbScore = np.ma.clip(Score.astype("float32"), 0.0, 100.0)

    # CLS multicategoría (Agua>Urb>Veg>Suelo>Sombra>0)
    mask_union = combine_masks(UrbC, AguaC, VegC, SueloC, SombraC)
    CLS = np.ma.array(np.zeros(NDVI.shape, dtype="float32"), mask=mask_union)
    CLS = np.where(AguaC.filled(0)==1, 2, CLS)
    CLS = np.where((UrbC.filled(0)==1) & (CLS==0), 1, CLS)
    CLS = np.where((VegC.filled(0)==1) & (CLS==0), 3, CLS)
    CLS = np.where((SueloC.filled(0)==1) & (CLS==0), 4, CLS)
    CLS = np.where((SombraC.filled(0)==1) & (CLS==0), 5, CLS)
    CLS = np.ma.array(CLS, mask=mask_union)

    # Guardar salidas por periodo
    write_tif(os.path.join(OUTPUT_DIR, f"UrbC_{PERIODO}.tif"),     UrbC,     prof_ref, "UrbC")
    write_tif(os.path.join(OUTPUT_DIR, f"AguaC_{PERIODO}.tif"),    AguaC,    prof_ref, "AguaC")
    write_tif(os.path.join(OUTPUT_DIR, f"VegC_{PERIODO}.tif"),     VegC,     prof_ref, "VegC")
    write_tif(os.path.join(OUTPUT_DIR, f"SueloC_{PERIODO}.tif"),   SueloC,   prof_ref, "SueloC")
    write_tif(os.path.join(OUTPUT_DIR, f"SombraC_{PERIODO}.tif"),  SombraC,  prof_ref, "SombraC")
    write_tif(os.path.join(OUTPUT_DIR, f"UrbScore_{PERIODO}.tif"), UrbScore, prof_ref, "UrbScore")
    write_tif(os.path.join(OUTPUT_DIR, f"CLS_{PERIODO}.tif"),      CLS,      prof_ref, "CLS")
```

```
    # Preparar stack (multibanda) alineado a la referencia del primer periodo
    if ref_prof_global is None:
        ref_prof_global = prof_ref.copy()
        # asegurar dtype float32/nodata
        ref_prof_global.update(dtype="float32", count=1, nodata=NODATA_VAL, compress="lzw")
    # reproyecta CLS al grid global si es necesario
    CLS_g = CLS if (prof_ref["transform"]==ref_prof_global["transform"] and
                    prof_ref["crs"]==ref_prof_global["crs"] and
                    prof_ref["width"]==ref_prof_global["width"] and
                    prof_ref["height"]==ref_prof_global["height"]) \
            else reproject_like(CLS, prof_ref, ref_prof_global, Resampling.nearest)

    stack_bands.append(CLS_g.filled(NODATA_VAL).astype("float32"))
    stack_names.append(PERIODO)

# ---------- GUARDAR STACK MULTIBANDA (CLS compuesto) ----------
if stack_bands:
    stack_bands_np = np.stack(stack_bands, axis=0)  # (bands, rows, cols)
    out_stack = os.path.join(OUTPUT_DIR, "CLS_stack_all_periods.tif")
    prof_stack = ref_prof_global.copy()
    prof_stack.update(count=stack_bands_np.shape[0])
    with rasterio.open(out_stack, "w", **prof_stack) as dst:
        dst.write(stack_bands_np)
        for i, nm in enumerate(stack_names, start=1):
            dst.set_band_description(i, f"CLS_{nm}")
    print(f"\n CLS compuesto guardado → {out_stack}")
    print("Bandas:", stack_names)

print("\n Fase 3 batch completada.")
print("Salidas por periodo y stack en:", OUTPUT_DIR)
```

4_ Binarizacion_EntornoGoogleColab






```
# === Fase 4 — Binarizar CLS + NO-REDUCCIÓN + CLIP AOI (.zip) — SOLO guarda UrbBinNR ===
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

!pip -q install rasterio fiona

import os, re, zipfile, tempfile
import numpy as np
import rasterio
from rasterio.enums import Resampling
from rasterio.warp import reproject, transform_geom
from rasterio.errors import RasterioIOError
from rasterio.mask import mask as rio_mask
from rasterio.io import MemoryFile
import fiona

print("rasterio", rasterio.__version__)

# ----- Rutas -----
CLS_DIR = "/content/drive/MyDrive/Model_LUMAChainAMVA_AOA/3_MuestrasEntrenamientoAMVA_AOA"  # CLS_YYYY_YYYY.tif
OUT_DIR = "/content/drive/MyDrive/Model_LUMAChainAMVA_AOA/4_BinariosResultadosAMVA_AOA"
os.makedirs(OUT_DIR, exist_ok=True)

# *** AOI (puede ser .zip/.shp/.geojson/.gpkg). ***
CLIP_TO_AOI = True
AOI_PATH    = "/content/drive/MyDrive/Model_LUMAChainAMVA_AOA/6_CartografiaBaseAMVA_AOA/AOI.zip"

# ----- Parámetros -----
URBAN_CODE = 4
NODATA_VAL = -9999.0

# ----- Qué guardar -----
SAVE_URBBINNR  = True   # ← único en True
SAVE_URBBINRAW = False  # ← no guardar
SAVE_URBACC    = False  # ← no guardar
```

```
# ====================== Helpers ======================
def list_cls_periods(folder):
    all_files = [os.path.join(folder, f) for f in os.listdir(folder)
                 if f.lower().endswith(".tif") and f.lower().startswith("cls_")]
    keep, bad = [], []
    for p in sorted(all_files):
        base = os.path.basename(p)
        m = re.fullmatch(r"CLS_(\d{4})_(\d{4})\.tif", base, flags=re.IGNORECASE)
        if not m:
            continue
        try:
            with rasterio.open(p) as src:
                if src.driver != "GTiff" or src.width == 0 or src.height == 0:
                    bad.append((base, f"driver={src.driver}, size={src.width}x{src.height}"))
                    continue
        except RasterioIOError as e:
            bad.append((base, str(e)))
            continue
        keep.append((f"{m.group(1)}_{m.group(2)}", p))
    if bad:
        print("⚠ CLS ignorados por formato/apertura:")
        for name, why in bad: print("   -", name, "→", why)
    keep.sort(key=lambda x: tuple(map(int, x[0].split("_"))))
    return keep

def read_masked(path):
    with rasterio.open(path) as src:
        if src.driver != "GTiff" or src.width == 0 or src.height == 0:
            raise RasterioIOError(f"{path} inválido (driver/size)")
        arr = src.read(1, masked=True).astype("float32")
        prof = src.profile.copy()
    return arr, prof

def same_grid(p1, p2):
    return (p1["crs"]==p2["crs"] and p1["transform"]==p2["transform"]
            and p1["width"]==p2["width"] and p1["height"]==p2["height"])

def reproject_like(arr_masked, prof_src, prof_dst, resampling=Resampling.nearest):
    dst = np.empty((prof_dst["height"], prof_dst["width"]), dtype="float32")
    reproject(
        source=arr_masked.filled(np.nan),
        destination=dst,
        src_transform=prof_src["transform"], src_crs=prof_src["crs"],
        dst_transform=prof_dst["transform"], dst_crs=prof_dst["crs"],
        resampling=resampling, src_nodata=np.nan, dst_nodata=np.nan
    )
    return np.ma.array(dst, mask=np.isnan(dst))

def base_profile_like(ref_prof, dtype="float32"):
    prof = ref_prof.copy()
    prof.update(dtype=dtype, count=1, nodata=NODATA_VAL, compress="lzw")
    return prof

def write_tif(path, arr_masked, out_prof, band_desc=None):
    arr = arr_masked.filled(NODATA_VAL).astype("float32")
    prof = base_profile_like(out_prof, "float32")
    with rasterio.open(path, "w", **prof) as dst:
        dst.write(arr, 1)
        if band_desc: dst.set_band_description(1, band_desc)

# --------- CLIP AOI con soporte .zip ----------
def _choose_layer(uri):
```

```
    try:
        layers = fiona.listlayers(uri)
        if not layers: return None
        for ly in layers:
            with fiona.open(uri, layer=ly) as src:
                gt = (src.schema.get("geometry") or "").lower()
                if "polygon" in gt:
                    return ly
        return layers[0]
    except Exception:
        return None

def _open_vector(uri, layer=None):
    return fiona.open(uri, layer=layer) if layer else fiona.open(uri)

def load_aoi_shapes(aoi_path, dst_crs):
    if not (aoi_path and os.path.exists(aoi_path)):
        return None
    is_zip = aoi_path.lower().endswith(".zip")
    uri = f"zip://{aoi_path}" if is_zip else aoi_path
    try:
        layer = _choose_layer(uri) if is_zip else None
        with _open_vector(uri, layer=layer) as src:
            src_crs = src.crs_wkt or src.crs
            shapes = []
            for feat in src:
                geom = feat["geometry"]
                if src_crs and dst_crs:
                    geom = transform_geom(src_crs, dst_crs, geom, precision=6)
                shapes.append(geom)
            if shapes: return shapes
    except Exception as e:
        print("🔁 Reintento AOI: extracción temporal por zip (", str(e), ")")

    if is_zip:
        tmpdir = tempfile.mkdtemp(prefix="aoi_zip_")
        with zipfile.ZipFile(aoi_path, 'r') as z: z.extractall(tmpdir)
        shp_path = None
        for root, _, files in os.walk(tmpdir):
            for f in files:
                if f.lower().endswith(".shp"):
                    shp_path = os.path.join(root, f); break
            if shp_path: break
        if not shp_path:
            print("⚠️ No se encontró .shp dentro del zip:", aoi_path); return None
        with fiona.open(shp_path) as src:
            src_crs = src.crs_wkt or src.crs
            shapes = []
            for feat in src:
                geom = feat["geometry"]
                if src_crs and dst_crs:
                    geom = transform_geom(src_crs, dst_crs, geom, precision=6)
                shapes.append(geom)
            if shapes: return shapes

    print("⚠️ AOI vacío o no legible:", aoi_path)
    return None

def clip_arr_profile_with_shapes(arr_masked, in_prof, shapes):
    with MemoryFile() as memfile:
        prof = base_profile_like(in_prof, "float32")
        with memfile.open(**prof) as tmp:
```

```
            tmp.write(arr_masked.filled(NODATA_VAL).astype("float32"), 1)
            out, out_transform = rio_mask(tmp, shapes, crop=True, filled=True, nodata=NODATA_VAL)
            meta = tmp.profile.copy()
            meta.update({"height": out.shape[1], "width": out.shape[2], "transform": out_transform})
    out_arr = np.ma.array(out[0], mask=(out[0] == NODATA_VAL))
    return out_arr, meta

def maybe_clip(arr_masked, prof, aoi_shapes):
    return clip_arr_profile_with_shapes(arr_masked, prof, aoi_shapes) if aoi_shapes else (arr_masked, prof)

# ===================== Cargar periodos =====================
PERIODS = list_cls_periods(CLS_DIR)
if not PERIODS:
    raise RuntimeError(f"No encontré CLS_YYYY_YYYY.tif válidos en {CLS_DIR}")
print("Periodos:", [p for p,_ in PERIODS])

# === Referencia global ===
first_period, first_path = PERIODS[0]
CLS0, ref_prof = read_masked(first_path)

# AOI en CRS del raster de referencia
AOI_SHAPES = load_aoi_shapes(AOI_PATH, ref_prof["crs"]) if (CLIP_TO_AOI and AOI_PATH) else None
if CLIP_TO_AOI and AOI_PATH and not AOI_SHAPES:
    print("⚠ No se pudo aplicar AOI; se escribirán salidas sin recorte.")

# Binario primer periodo y acumulado (siembra)
BinRAW0 = np.ma.array((CLS0 == URBAN_CODE).astype("float32"), mask=np.ma.getmaskarray(CLS0))
Acc = BinRAW0.copy()

# ---- Guardar SOLO UrbBinNR del primer periodo ----
if SAVE_URBBINNR:
    arr_w, prof_w = maybe_clip(Acc, ref_prof, AOI_SHAPES)
    write_tif(os.path.join(OUT_DIR, f"UrbBinNR_{first_period}.tif"), arr_w, prof_w, f"UrbBinNR_{first_period}")

# ===================== Siguientes periodos =====================
for period, path in PERIODS[1:]:
    CLSi, pro_i = read_masked(path)
    if not same_grid(pro_i, ref_prof):
        CLSi = reproject_like(CLSi, pro_i, ref_prof, Resampling.nearest)

    BinRAW = np.ma.array((CLSi == URBAN_CODE).astype("float32"),
                         mask=np.ma.getmaskarray(CLSi))

    prev = Acc.filled(0)
    curr = BinRAW.filled(np.nan)
    nr   = np.where(np.isnan(curr), prev, np.maximum(prev, curr))
    mask_out = np.ma.getmaskarray(Acc) & np.ma.getmaskarray(BinRAW)
    BinNR = np.ma.array(nr.astype("float32"), mask=mask_out)

    # Actualiza acumulado (siembra)
    Acc = BinNR.copy()

    # ---- Guardados según banderas ----
    if SAVE_URBBINRAW:
        arr_w, prof_w = maybe_clip(BinRAW, ref_prof, AOI_SHAPES)
        write_tif(os.path.join(OUT_DIR, f"UrbBinRAW_{period}.tif"), arr_w, prof_w, f"UrbBinRAW_{period}")

    if SAVE_URBBINNR:
        arr_w, prof_w = maybe_clip(BinNR, ref_prof, AOI_SHAPES)
        write_tif(os.path.join(OUT_DIR, f"UrbBinNR_{period}.tif"),  arr_w, prof_w, f"UrbBinNR_{period}")

    if SAVE_URBACC:
```

```
        arr_w, prof_w = maybe_clip(Acc, ref_prof, AOI_SHAPES)
        write_tif(os.path.join(OUT_DIR, f"UrbAcc_{period}.tif"),    arr_w, prof_w, f"UrbAcc_{period}")

print("🎯 Fase 4 completada (no-reducción + clip AOI).")
print("Guardado en:", OUT_DIR)
print(f" - UrbBinNR_* guardados: {SAVE_URBBINNR}")
print(f" - UrbBinRAW_* guardados: {SAVE_URBBINRAW}")
print(f" - UrbAcc_* guardados: {SAVE_URBACC}")
```

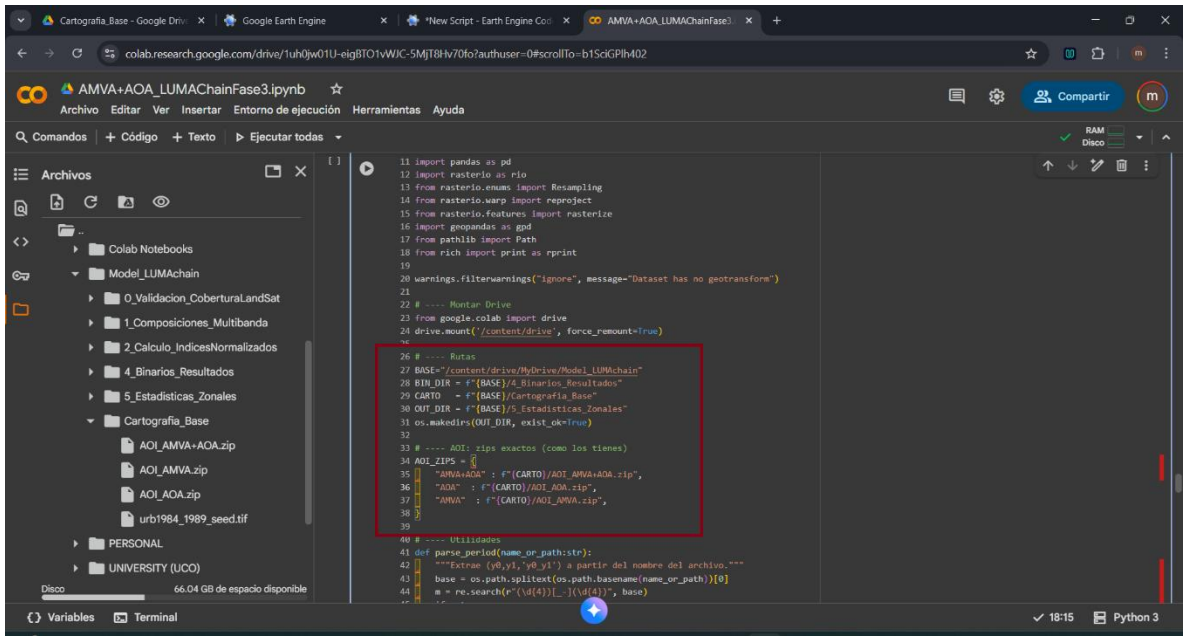
5_Estadisticas_Zonales

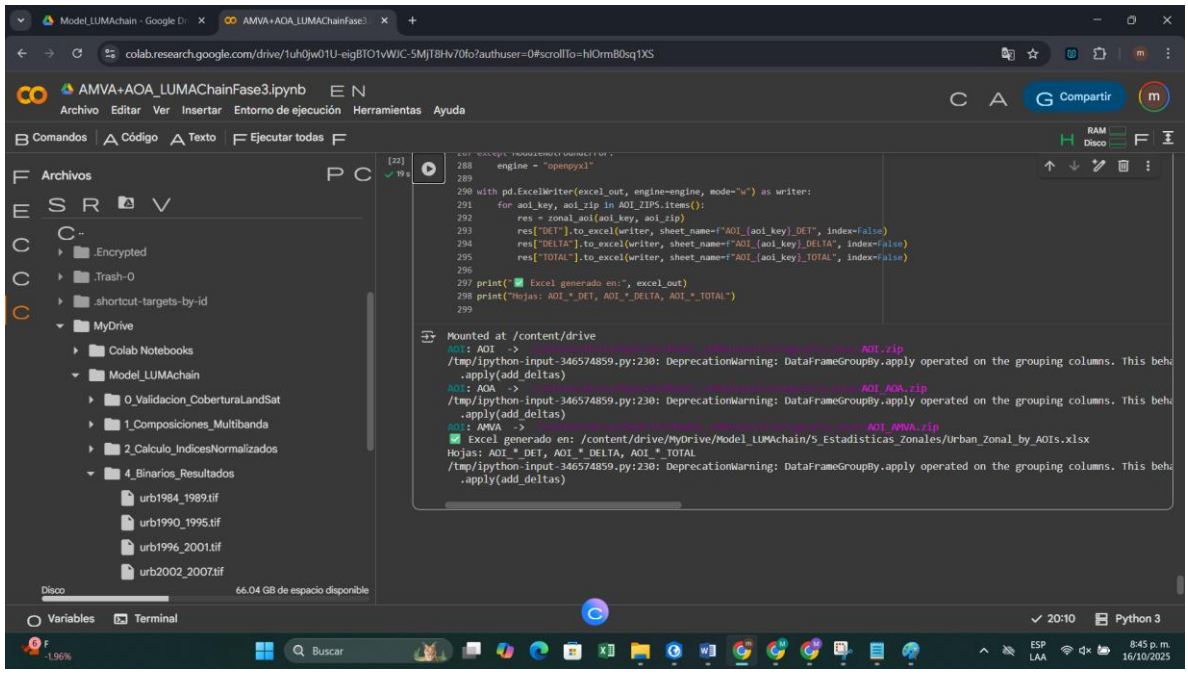
```
# ============================
# ZONAL STATS (binarios ya hechos) + DELTAS
# ============================

!pip -q install rasterio geopandas shapely fiona rich xlsxwriter openpyxl

import os, glob, zipfile, re, warnings
import numpy as np
import pandas as pd
import rasterio as rio
from rasterio.enums import Resampling
from rasterio.warp import reproject
from rasterio.features import rasterize
import geopandas as gpd
from pathlib import Path
from rich import print as rprint

warnings.filterwarnings("ignore", message="Dataset has no geotransform")

from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# ---- Rutas
BASE    = "/content/drive/MyDrive/Model_LUMAChainAMVA_AOA"
BIN_DIR = f"{BASE}/4_BinariosResultadosAMVA_AOA"   # tu carpeta de UrbBin_*.tif
CARTO   = f"{BASE}/6_CartografiaBaseAMVA_AOA"
OUT_DIR = f"{BASE}/5_EstadisticasZonalesAMVA_AOA"
os.makedirs(OUT_DIR, exist_ok=True)

# ---- AOI: zips exactos (CLAVES ÚNICAS)
AOI_ZIPS = {
   "AOI":      f"{CARTO}/AOI.zip",
   "AOI_AMVA": f"{CARTO}/AOI_AMVA.zip",
   "AOI_AOA":  f"{CARTO}/AOI_AOA.zip",
}

# ---- Utils (igual que tenías)
def parse_period(name_or_path: str):
    base = os.path.splitext(os.path.basename(name_or_path))[0]
    m = re.search(r"(\d{4})[_-](\d{4})", base)
    if not m: return (9999, 9999, base)
    y0, y1 = int(m.group(1)), int(m.group(2))
    return y0, y1, f"{y0}_{y1}"

def read_any_vector_from_zip(zip_path: str):
    z = Path(zip_path); assert z.exists(), f"No existe: {zip_path}"
    out = Path("/content/_aoi_tmp") / z.stem; out.mkdir(parents=True, exist_ok=True)
    with zipfile.ZipFile(z,'r') as zf: zf.extractall(out)
```

```
    shp = list(out.rglob("*.shp")); assert shp, f"ZIP sin .shp: {zip_path}"
    return gpd.read_file(shp[0])

def choose_name_code_fields(gdf: gpd.GeoDataFrame):
    name_fields = ["MpNombre","NOMBRE","Nombre","name","NAM","MUNICIPIO","MPNOMBRE"]
    code_fields = ["MpCodigo","CODIGO","Codigo","CODE","MPIO_CCDGO","MPIO_CCNCT","ID_MPIO"]
    name = next((c for c in name_fields if c in gdf.columns), None)
    code = next((c for c in code_fields if c in gdf.columns), None)
    gdf["muni_name"] = gdf[name].astype(str) if name else np.arange(1,len(gdf)+1).astype(str)
    gdf["muni_code"] = gdf[code].astype(str) if code else gdf["muni_name"]
    return gdf

def raster_profile_like(r_path: str):
    with rio.open(r_path) as src:
        return {"crs": src.crs, "transform": src.transform,
                "width": src.width, "height": src.height,
                "dtype": src.dtypes[0], "nodata": src.nodata}

def reproject_match(arr, src_prof, dst_prof, resampling=Resampling.nearest):
    dst = np.zeros((dst_prof["height"], dst_prof["width"]), dtype=arr.dtype)
    reproject(source=arr, destination=dst,
              src_transform=src_prof["transform"], src_crs=src_prof["crs"],
              dst_transform=dst_prof["transform"], dst_crs=dst_prof["crs"],
              resampling=resampling)
    return dst

def build_label_raster(gdf: gpd.GeoDataFrame, profile_like: dict):
    gdf = gdf.to_crs(profile_like["crs"]).loc[~gdf.geometry.is_empty].copy()
    gdf["muni_id"] = np.arange(1, len(gdf)+1, dtype=np.int32)
    shapes = list(zip(gdf.geometry, gdf["muni_id"].astype(int)))
    lbl = rasterize(shapes, out_shape=(profile_like["height"], profile_like["width"]),
                    transform=profile_like["transform"], fill=0, dtype="int32")
    id2name = dict(zip(gdf["muni_id"].astype(int), gdf["muni_name"].astype(str)))
    id2code = dict(zip(gdf["muni_id"].astype(int), gdf["muni_code"].astype(str)))
    return lbl, id2name, id2code

def load_binary_uint8(r_path: str):
    with rio.open(r_path) as src:
        arr = src.read(1); nd = src.nodata
        if nd is not None: arr = np.where(arr == nd, 0, arr)
        arr = (arr > 0).astype(np.uint8)
        prof = {"crs": src.crs, "transform": src.transform,
                "width": src.width, "height": src.height,
                "dtype": "uint8", "nodata": 0}
    return arr, prof

def pixel_area_m2_from_profile():
    any_tif = sorted(glob.glob(os.path.join(BIN_DIR, "UrbBinNR_*_*.tif")))[0]
    with rio.open(any_tif) as src:
        return abs(src.transform.a) * abs(src.transform.e)

# ---- Rastras binarios
raster_paths = sorted(glob.glob(f"{BIN_DIR}/UrbBinNR_*_*.tif"), key=lambda p: parse_period(p)[0])
assert raster_paths, f"No se encontraron UrbBinNR_*_*.tif en {BIN_DIR}"

ref_prof   = raster_profile_like(raster_paths[0])
px_area_m2 = pixel_area_m2_from_profile()

def zonal_aoi(aoi_name: str, aoi_zip: str):
    rprint(f"[bold cyan]AOI[/]: {aoi_name}  →  {aoi_zip}")
    gdf = choose_name_code_fields(read_any_vector_from_zip(aoi_zip))
    lbl, id2name, id2code = build_label_raster(gdf, ref_prof)
```

```
    nlabels = int(lbl.max())

    rows = []
    for rpath in raster_paths:
        y0, y1, per = parse_period(rpath)
        arr, prof = load_binary_uint8(rpath)
        if (prof["crs"]!=ref_prof["crs"] or prof["transform"]!=ref_prof["transform"] or
            prof["width"]!=ref_prof["width"] or prof["height"]!=ref_prof["height"]):
            arr = reproject_match(arr, prof, ref_prof, Resampling.nearest)

        valid = lbl > 0
        lab   = lbl[valid].ravel()
        vals  = arr[valid].ravel().astype(np.float32)
        urb_counts = np.bincount(lab, weights=vals, minlength=nlabels+1)

        for mid in np.nonzero(urb_counts)[0]:
            if mid == 0: continue
            urb_px = int(urb_counts[mid])
            rows.append({
                "aoi": aoi_name, "period": per,
                "y_start": y0, "y_end": y1,
                "muni_id": int(mid),
                "muni_code": id2code.get(int(mid), f"ID_{int(mid)}"),
                "muni_name": id2name.get(int(mid), f"ID_{int(mid)}"),
                "urb_px": urb_px, "urb_km2": urb_px * px_area_m2 / 1e6
            })

    det = (pd.DataFrame(rows)
           .sort_values(["muni_code","y_start","y_end"])
           .reset_index(drop=True))

    def add_deltas(df):
        df = df.copy()
        df["period_len_yrs"] = (df["y_end"] - df["y_start"]).clip(lower=1)
        df["urb_km2_prev"]   = df["urb_km2"].shift(1)
        df["gain_km2"]       = (df["urb_km2"] - df["urb_km2_prev"]).fillna(0.0)
        df["gain_py"]        = df["gain_km2"] / df["period_len_yrs"]
        return df

    det_deltas = det.groupby(["muni_code","muni_name"], group_keys=False).apply(add_deltas).reset_index(drop=True)

    tot = (det.groupby(["period","y_start","y_end"], as_index=False)["urb_km2"]
             .sum().rename(columns={"urb_km2":"urb_total_km2"}))
    det_tot = det.merge(tot, on=["period","y_start","y_end"], how="left")
    det_tot["share_pct"] = np.where(det_tot["urb_total_km2"]>0,
                                    100.0*det_tot["urb_km2"]/det_tot["urb_total_km2"], 0.0)

    total_period = tot.sort_values(["y_start","y_end"]).reset_index(drop=True)
    return {"DET": det_tot[["aoi","period","y_start","y_end","muni_code","muni_name",
                            "urb_km2","urb_total_km2","share_pct"]],
            "DELTA": det_deltas[["aoi","period","y_start","y_end","muni_code","muni_name",
                                 "urb_km2_prev","urb_km2","gain_km2","gain_py"]],
            "TOTAL": total_period}

# ---- Escribir Excel (aseguramos carpeta)
excel_out = f"{OUT_DIR}/Urban_Zonal_by_AOIs.xlsx"
os.makedirs(os.path.dirname(excel_out), exist_ok=True)

engine = "xlsxwriter"
try:
    import xlsxwriter
except ModuleNotFoundError:
```

```
    engine = "openpyxl"

with pd.ExcelWriter(excel_out, engine=engine, mode="w") as writer:
    for aoi_key, aoi_zip in AOI_ZIPS.items():
        res = zonal_aoi(aoi_key, aoi_zip)
        res["DET"].to_excel(writer,   sheet_name=f"{aoi_key}_DET",   index=False)
        res["DELTA"].to_excel(writer, sheet_name=f"{aoi_key}_DELTA", index=False)
        res["TOTAL"].to_excel(writer, sheet_name=f"{aoi_key}_TOTAL", index=False)

print("☑ Excel generado en:", excel_out)
print("Hojas: AOI_DET/DELTA/TOTAL, AOI_AMVA_DET/DELTA/TOTAL, AOI_AOA_DET/DELTA/TOTAL")
```





```
// ====================== CONFIGURACIÓN BÁSICA ======================
var AOI_ASSET   = 'projects/nucleo1mcl/assets/AOI';   // <- tu AOI
var DRIVE_FOLDER = '8_ValidacionEstadistica';         // carpeta destino en Drive

var aoi = ee.FeatureCollection(AOI_ASSET).geometry();

// Helper de exportación
function exportToDrive(img, name, scale){
  Export.image.toDrive({
    image: img.clip(aoi),
    description: name,
    folder: DRIVE_FOLDER,
    fileNamePrefix: name,
    region: aoi,
    scale: scale,                 // 100 m (GHSL) o 10 m (WSF/WorldCover)
    maxPixels: 1e13,
    fileFormat: 'GeoTIFF',
    formatOptions: {cloudOptimized: true}
  });
}

// ====================== 1) GHSL BUILT-UP (m² por celda, 100 m) ======================
var ghslYears = [1990, 2000, 2015, 2020];
ghslYears.forEach(function(y){
  var ghsl = ee.Image('JRC/GHSL/P2023A/GHS_BUILT_S/' + y)
               .select('built_surface');     // m² construidos / celda (100 m)
  exportToDrive(ghsl, 'GHSL_built_m2_' + y, 100);
});

// ====================== 2) WSF 2015 (10 m, binario) ================================
// Banda correcta = 'WSF' (0 o 255). Lo pasamos a 0/1.
var wsf2015 = ee.Image('DLR/WSF/WSF2015/v1')
               .select('WSF')
               .gt(0)                        // 1 = asentamiento
               .rename('WSF2015_built');
exportToDrive(wsf2015, 'WSF2015_built_10m', 10);

// ====================== 3) WorldCover 2021 (10 m, built-up=50) ====================
var worldcover2021_built = ee.ImageCollection('ESA/WorldCover/v200')
                            .first()       // 2021
                            .select('Map').eq(50)
                            .rename('WorldCover2021_built');
exportToDrive(worldcover2021_built, 'WorldCover2021_built_10m', 10);

// ====================== Totales rápidos en consola (km²) ==========================
var km2 = ee.Image.pixelArea().divide(1e6);

function printArea(maskImg, name, scale){
  var area = km2.updateMask(maskImg)
               .reduceRegion({
                 reducer: ee.Reducer.sum(),
                 geometry: aoi,
                 scale: scale,
                 maxPixels: 1e13
               });
  print(name + ' (km²):', area.get('area'));
}
```

```
printArea(wsf2015, 'WSF2015 built-up', 10);
printArea(worldcover2021_built, 'WorldCover2021 built-up', 10);

ghslYears.forEach(function(y){
  var ghsl = ee.Image('JRC/GHSL/P2023A/GHS_BUILT_S/' + y).select('built_surface');
  var total_km2 = ghsl.divide(1e6).reduceRegion({
      reducer: ee.Reducer.sum(), geometry: aoi, scale: 100, maxPixels: 1e13
    }).get('built_surface');
  print('GHSL built-up ' + y + ' (km²):', total_km2);
});

// (Opcional) Vista rápida en el mapa
Map.centerObject(aoi, 10);
Map.addLayer(aoi, {color:'#00FFFF'}, 'AOI', false);
Map.addLayer(wsf2015, {min:0, max:1, palette:['000000','ffffff']}, 'WSF2015 10m', false);
Map.addLayer(worldcover2021_built, {min:0, max:1, palette:['000000','ffcc00']}, 'WorldCover Built 2021', false);
```