

**پیوست:**

**کدهای پایتون روش‌های داده‌محور**

## ایجاد نقاط معدنی با شبکه‌بندی متراکم و دقیق

```
# =====  
# □ بخش ۱: وارد کردن کتابخانه‌ها  
# =====  
import arcpy # □ برای پردازش داده‌های مکانی ArcGIS کتابخانه اصلی  
import os    # □ برای کار با مسیرها و سیستم فایل  
import math  # □ برای محاسبات ریاضی  
  
# =====  
# □ بخش ۲: تنظیمات محیطی  
# =====  
# □ Geodatabase پارامتر ورودی ۱: مسیر پایگاه داده  
# اشاره کند gdb. باید به یک فایل جا  
# کاربر باید این مسیر را اصلاح کند جا  
arcpy.env.workspace = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Gosal.gdb"  
  
# □ پارامتر ورودی ۲: اجازه رونویسی خروجی‌ها  
# باشد، فایل‌های موجود بازنویسی می‌شوند True اگر جا  
# باشد، با خطا مواجه می‌شود False اگر جا  
arcpy.env.overwriteOutput = True  
  
# =====  
# □ بخش ۳: تعریف توابع کمکی  
# =====
```

```
def create_dense_points_in_small_polygon(polygon_fc, output_name, points_per_meter=0.5):
```

```
    """
```

□ تابع ۱: ایجاد نقاط متراکم در پلیگون‌های کوچک

□ پارامترهای ورودی:

- polygon\_fc: نام Feature Class (رشته) پلیگون ورودی
- output\_name: نام Feature Class (رشته) خروجی
- points\_per\_meter: تراکم نقطه در هر متر (عدد اعشاری، پیش‌فرض: ۰,۵)

□ خروجی:

- تعداد نقاط ایجاد شده (عدد صحیح) -

□ توضیح:

این تابع یک شبکه منظم از نقاط ایجاد می‌کند  
سپس فقط نقاط داخل پلیگون را انتخاب می‌کند

```
    """
```

# □ مرحله ۱: دریافت اطلاعات پلیگون

```
desc = arcpy.Describe(polygon_fc)
```

```
extent = desc.extent # محدوده جغرافیایی پلیگون
```

# □ مرحله ۲: محاسبه ابعاد پلیگون

```
width = extent.XMax - extent.XMin # عرض پلیگون بر حسب متر
```

```
height = extent.YMax - extent.YMin # ارتفاع پلیگون بر حسب متر
```

# □ مرحله ۳: نمایش اطلاعات

```

print(f" □ اندازه پلیگون: {width:.2f} × {height:.2f} متر")

print(f" □ مساحت: {width * height:.2f} متر مربع")

# □ مرحله ۴: محاسبه فاصله بین نقاط
point_spacing = 1.0 / points_per_meter # متر
print(f" □ فاصله نقاط: {point_spacing:.2f} متر")

# □ مرحله ۵: محاسبه تعداد نقاط در هر جهت
cols = max(2, math.ceil(width / point_spacing)) # تعداد ستون‌ها
rows = max(2, math.ceil(height / point_spacing)) # تعداد سطرها

print(f" □ شبکه: {rows} × {cols} = {rows * cols} نقطه")

# □ مرحله ۶: بررسی پلیگون‌های خیلی کوچک
if width < 1 or height < 1:
    print(" □ پلیگون بسیار کوچک - ایجاد نقطه مرکزی")
    center_point = arcpy.FeatureToPoint_management(
        polygon_fc,
        output_name,
        "CENTROID"
    )
    return 1 # فقط یک نقطه ایجاد شد

# □ مرحله ۷: ایجاد لیست مختصات نقاط
points_list = []
x_coords = []

```

```

y_coords = []

# تولید مختصات X
for col in range(cols):
    x = extent.XMin + (col * point_spacing)
    if x <= extent.XMax:
        x_coords.append(x)

# تولید مختصات Y
for row in range(rows):
    y = extent.YMin + (row * point_spacing)
    if y <= extent.YMax:
        y_coords.append(y)

# ایجاد تمام ترکیبات مختصات
for x in x_coords:
    for y in y_coords:
        points_list.append(arcpy.Point(x, y))

# موقت Feature Class مرحله ۸: ایجاد □
temp_fc = arcpy.CreateFeatureclass_management(
    "in_memory", "temp_grid", # در حافظه موقت ایجاد شود
    "POINT", # نوع هندسی: نقطه
    spatial_reference=desc.spatialReference # سیستم مختصات اصلی
)

# Feature Class مرحله ۹: درج نقاط در □

```

```

with arcpy.da.InsertCursor(temp_fc, ["SHAPE@"]) as cursor:

    for point in points_list:

        cursor.insertRow([point])

# □ مرحله ۱۰: انتخاب نقاط داخل پلیگون

points_layer = "in_memory/points_layer"

arcpy.MakeFeatureLayer_management(temp_fc, points_layer)

arcpy.SelectLayerByLocation_management(
    points_layer,
    "WITHIN", # شرط مکانی: داخل پلیگون
    polygon_fc
)

# □ مرحله ۱۱: شمارش نقاط انتخاب شده

count = int(arcpy.GetCount_management(points_layer)[0])

if count > 0:

    # □ نقاط داخل پلیگون وجود دارند

    arcpy.CopyFeatures_management(points_layer, output_name)

else:

    # □ هیچ نقطه‌ای داخل پلیگون نیست

    print(" □ هیچ نقطه‌ای داخل پلیگون نبود - ایجاد مرکز")

    arcpy.FeatureToPoint_management(
        polygon_fc,
        output_name,
        "CENTROID"
    )

```

)

count = 1

# مرحله ۱۲: پاکسازی حافظه موقت □

arcpy.Delete\_management("in\_memory")

return count

def create\_points\_using\_fishnet(polygon\_fc, output\_name, cell\_size=2):

"""

□ برای پلیگون‌های کوچک Fishnet تابع ۲: استفاده از

□ پارامترهای ورودی:

- polygon\_fc: نام Feature Class (رشته) پلیگون ورودی
- output\_name: نام Feature Class (رشته) خروجی
- cell\_size: اندازه سلول شبکه (عدد اعشاری، پیش‌فرض: ۲ متر)

□ خروجی:

- تعداد نقاط ایجاد شده (عدد صحیح)

□ توضیح:

برای ایجاد شبکه استفاده می‌کند Fishnet این تابع از ابزار

مناسب برای پلیگون‌های با اندازه متوسط

"""

# □ مرحله ۱: دریافت اطلاعات پلیگون

```
desc = arcpy.Describe(polygon_fc)
```

```
extent = desc.extent
```

# □ مرحله ۲: محاسبه ابعاد پلیگون

```
width = extent.XMax - extent.XMin
```

```
height = extent.YMax - extent.YMin
```

# □ برای پلیگون‌های خیلی کوچک cell size مرحله ۳: تنظیم

```
if width < cell_size or height < cell_size:
```

```
    cell_size = min(width, height) / 2
```

# □ مرحله ۴: محاسبه تعداد سطر و ستون

```
rows = max(2, math.ceil(height / cell_size))
```

```
cols = max(2, math.ceil(width / cell_size))
```

```
print(f" □ ایجاد Fishnet: {rows}×{cols} (اندازه سلول: {cell_size} متر)")
```

# □ شبکه ماهیگیری fishnet مرحله ۵: ایجاد

```
fishnet = arcpy.CreateFishnet_management(
```

```
    out_feature_class="in_memory/fishnet", # خروجی موقت
```

```
    origin_coord=f"{extent.XMin} {extent.YMin}", # مختصات مبدا
```

```
    y_axis_coord=f"{extent.XMin} {extent.YMin + cell_size}", # محور Y
```

```
    cell_width=cell_size, # عرض سلول
```

```
    cell_height=cell_size, # ارتفاع سلول
```

```
    number_rows=rows, # تعداد سطرها
```



```

number_columns=cols, # تعداد ستون ها

labels="LABELS", # ایجاد برچسب (نقاط)

geometry_type="POLYGON" # نوع هندسی: پلیگون

)[0]

# مرحله ۶: استخراج نقاط از شبکه
fishnet_labels = "in_memory/fishnet_labels"
arcpy.FeatureToPoint_management(fishnet, fishnet_labels, "CENTROID")

# مرحله ۷: انتخاب نقاط داخل پلیگون
points_inside = arcpy.SelectLayerByLocation_management(
    fishnet_labels,
    "WITHIN",
    polygon_fc
)

# مرحله ۸: شمارش نقاط
count = int(arcpy.GetCount_management(points_inside)[0])

if count > 0:
    # کپی نقاط به خروجی
    arcpy.CopyFeatures_management(points_inside, output_name)
else:
    # ایجاد نقطه مرکزی
    arcpy.FeatureToPoint_management(
        polygon_fc,
        output_name,

```

```
"CENTROID"
```

```
)
```

```
count = 1
```

```
# مرحله ۹: پاکسازی □
```

```
arcpy.Delete_management("in_memory")
```

```
return count
```

```
def simple_grid_points(polygon_fc, output_name, spacing=0.5):
```

```
"""
```

```
□ تابع ۳: روش ساده برای ایجاد نقاط شبکه‌ای
```

```
□ پارامترهای ورودی:
```

```
- polygon_fc: نام Feature Class (رشته) پلیگون ورودی
```

```
- output_name: نام Feature Class (رشته) خروجی
```

```
- spacing: فاصله بین نقاط (عدد اعشاری، پیش‌فرض: ۰.۵ متر)
```

```
□ خروجی:
```

```
- تعداد نقاط ایجاد شده (عدد صحیح)
```

```
□ توضیح:
```

```
این تابع یک روش ساده و مستقیم برای ایجاد شبکه نقاط ارائه می‌دهد
```

```
مناسب برای پلیگون‌های کوچک
```

```
"""
```

# □ مرحله ۱: دریافت اطلاعات پلیگون

```
desc = arcpy.Describe(polygon_fc)
```

```
extent = desc.extent
```

# □ مرحله ۲: تعریف محدوده

```
x_min, y_min = extent.XMin, extent.YMin
```

```
x_max, y_max = extent.XMax, extent.YMax
```

# □ مرحله ۳: ایجاد لیست نقاط

```
points = []
```

```
x = x_min
```

```
while x <= x_max:
```

```
    y = y_min
```

```
    while y <= y_max:
```

```
        points.append(arcpy.Point(x, y))
```

```
        y += spacing # افزایش مختصات Y
```

```
    x += spacing # افزایش مختصات X
```

# □ خروجی Feature Class مرحله ۴: ایجاد

```
out_fc = arcpy.CreateFeatureclass_management(
```

```
    arcpy.env.workspace, # مسیر جی دی بی
```

```
    output_name, # نام خروجی
```

```
    "POINT", # نوع هندسی
```

```
    spatial_reference=desc.spatialReference # سیستم مختصات
```

```
)
```

# □ مرحله ۵: درج تمام نقاط

```
with arcpy.da.InsertCursor(out_fc, ["SHAPE@"]) as cursor:
```

```
    for point in points:
```

```
        cursor.insertRow([point])
```

# □ مرحله ۶: انتخاب نقاط داخل پلیگون

```
temp_layer = "temp_points_layer"
```

```
arcpy.MakeFeatureLayer_management(out_fc, temp_layer)
```

```
arcpy.SelectLayerByLocation_management(
```

```
    temp_layer,
```

```
    "WITHIN",
```

```
    polygon_fc
```

```
)
```

# □< مرحله ۷: حذف نقاط خارج از پلیگون

```
arcpy.CopyFeatures_management(temp_layer, f"{output_name}_final")
```

```
arcpy.Delete_management(out_fc) # حذف نسخه قدیمی
```

```
arcpy.Rename_management(f"{output_name}_final", output_name) # تغییر نام
```

# □ مرحله ۸: شمارش نهایی

```
count = int(arcpy.GetCount_management(output_name)[0])
```

# □ مرحله ۹: پاکسازی

```
if arcpy.Exists(temp_layer):
```

```
    arcpy.Delete_management(temp_layer)
```

```
return count
```

```
# =====
```

```
# □ بخش ۴: اجرای اصلی برنامه
```

```
# =====
```

```
try:
```

```
    print("=" * 60)
```

```
    print("شروع پردازش ایجاد نقاط متراکم در پلیگون‌ها □")
```

```
    print("=" * 60)
```

```
# □ مرحله ۱: یافتن پلیگون‌های مورد نظر
```

```
print("\n□ جستجوی پلیگون‌های با پیشوند 'o'")
```

```
all_fcs = arcpy.ListFeatureClasses() # لیست تمام Feature Classها
```

```
o_polygons = [] # لیست پلیگون‌های هدف
```

```
for fc in all_fcs:
```

```
    if fc.startswith('o'): # شرط نام: شروع با حرف 'o'
```

```
        try:
```

```
            desc = arcpy.Describe(fc)
```

```
            if desc.shapeType == "Polygon": # فقط پلیگون‌ها
```

```
                o_polygons.append(fc)
```

```
        except:
```

```
            continue # در صورت خطا، ادامه بده
```

```
print(f"\n□ یافت شد 'o' پلیگون با پیشوند {len(o_polygons)} تعداد □")
```

```

# □ مرحله ۲: پردازش هر پلیگون
print("□ مرحله ۲: پردازش پلیگون ها")
print("-" * 40)

successful = 0 # شمارنده موفقیت

for i, polygon_fc in enumerate(o_polygons, 1):
    print(f"\n{i}/{len(o_polygons)} - پردازش: {polygon_fc}")
    print("-" * 30)

    try:
        # □ زیرمرحله ۱-۲: بررسی اندازه پلیگون
        desc = arcpy.Describe(polygon_fc)
        extent = desc.extent

        width = extent.XMax - extent.XMin
        height = extent.YMax - extent.YMin
        area = width * height

        print(f"□ اندازه: {width:.1f} × {height:.1f} متر")
        print(f"□ مساحت: {area:.1f} متر مربع")

        # □ زیرمرحله ۲-۲: تعیین نام خروجی
        output_name = f"{polygon_fc}_DensePoints"

        # □ زیرمرحله ۳-۲: انتخاب روش بر اساس مساحت
        if area < 10: # □ بسیار کوچک

```

```

print(" پلیگون بسیار کوچک - ایجاد نقطه مرکزی □ ")

arcpy.FeatureToPoint_management(
    polygon_fc,
    output_name,
    "CENTROID"
)

count = 1

elif area < 100: # □ کوچک

    print(" پلیگون کوچک - استفاده از روش ساده □ ")

    count = simple_grid_points(polygon_fc, output_name, spacing=0.25)

else: # □ متوسط

    print(" □ استفاده از روش Fishnet ")

    # بر اساس مساحت cell size محاسبه

    cell_size = math.sqrt(area) / 5 # نسبت به ریشه مساحت

    cell_size = max(0.5, min(cell_size, 5)) # محدوده ۰.۵ تا ۵ متر

    count = create_points_using_fishnet(
        polygon_fc,
        output_name,
        cell_size=cell_size
    )

print(f" □ ایجاد شد: {output_name} ( {count} نقطه ) ")

successful += 1

```

except Exception as e:

```
print(f' □ خطا در پردازش {polygon_fc}: {str(e)}')
```

```
# □ تلاش با روش جایگزین (نقطه مرکزی)
```

```
try:
```

```
    output_name = f'{polygon_fc}_DensePoints'
```

```
    arcpy.FeatureToPoint_management(polygon_fc, output_name, "CENTROID")
```

```
    print(f' □ ایجاد نقطه مرکزی به جای آن')
```

```
    successful += 1
```

```
except:
```

```
    print(f' □ شکست کامل')
```

```
# □ مرحله ۳: نمایش نتایج نهایی
```

```
print("\n" + "=" * 60)
```

```
print("□ مرحله ۳: نتایج نهایی")
```

```
print("=" * 60)
```

```
print(f"\n□ پردازش کامل شد!")
```

```
print(f'□ موفق: {successful} از {len(o_polygons)}')
```

```
print(f'□ ناموفق: {len(o_polygons) - successful}')
```

```
# □ های ایجاد شده Feature Class مرحله ۴: نمایش
```

```
print("\n□ Feature Class های ایجاد شده:")
```

```
print("-" * 40)
```

```
dense_points_fcs = [fc for fc in arcpy.ListFeatureClasses("*_DensePoints")]
```



```

if dense_points_fcs:
    for fc in dense_points_fcs:
        try:
            count = arcpy.GetCount_management(fc)[0]
            desc = arcpy.Describe(fc)
            print(f' □ {fc}: {count} نقطه")

        except:
            print(f' □ {fc}: نامعلوم")

    else:
        print(" □ هیچ Feature Class نشد")

print("\n" + "=" * 60)
print("□ برنامه با موفقیت به پایان رسید")
print("=" * 60)

except Exception as e:
    # □ مدیریت خطاهای اصلی
    print(f"\n□ خطای اصلی: {str(e)}")
    print("=" * 60)

    # □ نمایش جزئیات خطا برای عیب‌یابی
    import traceback
    print("\n□ جزئیات خطا:")
    print(traceback.format_exc())

    print("\n□ پیشنهادات عیب‌یابی:")
    print("1. □ بررسی مسیر پایگاه داده")

```

```
print("2. □ بررسی دسترسی به فایل ها")
```

```
print("3. □ بررسی license ArcGIS")
```

```
print("4. □ بررسی فضای دیسک")
```

```
# =====
```

```
# □ بخش ۵: اطلاعات تکمیلی
```

```
# =====
```

```
"""
```

□ نکات مهم برای کاربر:

1. □ سیستم مختصات:

- سیستم مختصات از پلیگون ورودی به ارث می‌رود
- برای نتایج دقیق، پلیگون‌ها باید در سیستم متریک باشند

2. □ مدیریت حافظه:

- استفاده می‌کند (in\_memory) کد از حافظه موقت
- پس از اجرا، حافظه پاکسازی می‌شود

3. □ بهینه‌سازی:

- برای پلیگون‌های بزرگ، ممکن است اجرا زمان‌بر باشد
- می‌توان پارامترهای تراکم را تنظیم کرد

4. □ اجرای مجدد:

- باشد، فایل‌های قدیمی بازنویسی می‌شوند overwriteOutput=True اگر
- کنید False برای حفظ نسخه‌های قدیمی، آن را

5. □ گزارش گیری:

- نتایج در کنسول نمایش داده می شوند -
- می توان خروجی را به فایل لاگ هم ذخیره کرد -

□ پشتیبانی:

- در صورت مشکل با توسعه دهنده تماس بگیرید -
- خطاها را با جزئیات کامل گزارش دهید -

""""

انیمیشن کد قبل

```
import arcpy
import os
import re
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
from PIL import Image

point_gdb_path = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Gosal.gdb"
output_folder = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Animation_Points"
output_gif = os.path.join(output_folder, "Points_Animation.gif")

arcpy.env.workspace = point_gdb_path

all_datasets = arcpy.ListDatasets() + [""]
point_features = []
```

```

for ds in all_datasets:

    arcpy.env.workspace = os.path.join(point_gdb_path, ds) if ds else point_gdb_path

    fcs = arcpy.ListFeatureClasses('o*')

    if fcs:

        for fc in fcs:

            full_path = os.path.join(arcpy.env.workspace, fc) if ds else fc

            point_features.append(full_path)

def extract_number(name):

    matches = re.findall(r'\d+\.\d*', os.path.basename(name))

    return float(matches[0]) if matches else 0

point_features_sorted = sorted(point_features, key=extract_number)

all_layers_points = []
layer_numbers = []

for point_fc in point_features_sorted:

    layer_number = extract_number(point_fc)

    layer_numbers.append(layer_number)

points = []

with arcpy.da.SearchCursor(point_fc, ['SHAPE@X', 'SHAPE@Y']) as cursor:

    for row in cursor:

        points.append((row[0], row[1]))

all_layers_points.append(points)

```

```

all_x = []
all_y = []
for points in all_layers_points:
    if points:
        all_x.extend([p[0] for p in points])
        all_y.extend([p[1] for p in points])

xmin, xmax = min(all_x), max(all_x)
ymin, ymax = min(all_y), max(all_y)

x_range = xmax - xmin
y_range = ymax - ymin
xmin -= x_range * 0.05
xmax += x_range * 0.05
ymin -= y_range * 0.05
ymax += y_range * 0.05

total_points = sum(len(points) for points in all_layers_points)

images = []
os.makedirs(output_folder, exist_ok=True)

color_gradient = plt.cm.plasma(np.linspace(0.2, 0.8, len(point_features_sorted)))
color_hex_list = [colors.rgb2hex(color) for color in color_gradient]

for i, (points, layer_number, frame_color) in enumerate(zip(
    all_layers_points, layer_numbers, color_hex_list)):

```

```
if not points:
```

```
    continue
```

```
fig, ax = plt.subplots(figsize=(12, 10), dpi=150)
```

```
fig.patch.set_facecolor('#2C3E50')
```

```
ax.set_facecolor('#1A1A2E')
```

```
x_coords = [p[0] for p in points]
```

```
y_coords = [p[1] for p in points]
```

```
ax.scatter(x_coords, y_coords,
```

```
          s=25,
```

```
          c=frame_color,
```

```
          edgecolors='#FFFFFF',
```

```
          linewidths=0.3,
```

```
          alpha=0.7,
```

```
          marker='o')
```

```
ax.set_xlim(xmin, xmax)
```

```
ax.set_ylim(ymin, ymax)
```

```
ax.set_aspect('equal')
```

```
ax.set_xlabel('X Coordinate', fontsize=12, color='#ECF0F1')
```

```
ax.set_ylabel('Y Coordinate', fontsize=12, color='#ECF0F1')
```

```
ax.set_title(f'Elevation: {layer_number}', fontsize=16, fontweight='bold', color='#ECF0F1')
```

```
ax.tick_params(colors='#ECF0F1')
```

```
for spine in ax.spines.values():
```

```
    spine.set_color('#ECF0F1')
```

```
ax.grid(True, alpha=0.2, linestyle='--', linewidth=0.5, color='#7F8C8D')
```

```
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: format(int(x), ',')))
```

```
plt.tight_layout()
```

```
fig.canvas.draw()
```

```
img_array = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)
```

```
img_array = img_array.reshape(fig.canvas.get_width_height()[::-1] + (3,))
```

```
images.append(Image.fromarray(img_array))
```

```
plt.close(fig)
```

```
if len(images) > 1:
```

```
    images[0].save(
```

```
        output_gif,
```

```
        save_all=True,
```

```
        append_images=images[1:],
```

```
        duration=800,
```

```
        loop=0,
```

```
        optimize=True,
```

```
        quality=95
```

```
)

print(f'انیمیشن ساخته شد {output_gif}')

print(f'تعداد فریم‌ها: {len(images)}')

print(f'کل پوینت‌ها: {total_points:,}')
```

## کد ایجاد نقاط غیرمعدنی با منطق ثابت

```
import arcpy
import numpy as np
import os
import time
import re

#
=====
=
# بخش ۱: تنظیمات محیطی و مسیرها □
#
=====
=
#
# این بخش شامل تنظیمات اصلی برنامه است □
# - arcpy.env.workspace: مسیر اصلی کار (ها GDB فولدر حاوی)
# - input_gdb: Geodatabase لیتولوژی رسترهای حاوی
# - sample_gdb: Geodatabase (پیشوند 'o') حاوی نقاط معدنی
# - output_gdb: Geodatabase (می‌تواند همان) خروجی (باشد input_gdb)
#
```



```

# □ ها باید موجود باشند GDB نکته: مسیرها باید به درستی تنظیم شوند و

#
=====
=

arcpy.env.overwriteOutput = True # □ اجازه رونویسی فایل‌های موجود

arcpy.env.workspace = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal" # □ مسیر اصلی

# مسیرهای اصلی

input_gdb = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Litho_Rasters.gdb" # □ رسترهای لیتولوژی
sample_gdb = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Gosal.gdb" # □ نقاط معدنی
output_gdb = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Litho_Rasters.gdb" # □ خروجی

#
=====
=

# □ بخش ۲: تنظیمات توسط کاربر (قابل تغییر)

#
=====
=

#
# این بخش شامل گزینه‌هایی است که کاربر می‌تواند تغییر دهد □
# - USE_RECLASSIFY_FILTER: اگر True باشد، فقط رسترهای حاوی متن خاص فیلتر می‌شوند
# - ALTERNATIVE_TEXT: (مثلاً "ColorRaster") متن جایگزین برای فیلتر کردن
# - ZERO_LAST_CLASS: اگر True باشد، کلاس آخر با احتمال صفر تحلیل می‌شود
#
# □ کنید False را USE_RECLASSIFY_FILTER پیشنهاد: برای انتخاب تمام رسترها،

```

#

=====

=

USE\_RECLASSIFY\_FILTER = True # ☐ فیلتر رسترها فعال/غیرفعال

ALTERNATIVE\_TEXT = "ColorRaster" # ☐ متن برای فیلتر رسترها

ZERO\_LAST\_CLASS = False # ☐ تحلیل لایه آخر با احتمال صفر

#

=====

=

# ☐ بخش ۳: تنظیمات ثابت ایجاد نقاط غیرمعدنی

#

=====

=

#

# ☐ این بخش شامل پارامترهای ثابت برای ایجاد نقاط غیرمعدنی است

# - GRID\_SPACING: فاصله بین نقاط شبکه (بر حسب متر)

# - DISTANCE\_ZONES: مناطق فاصله و درصد حذف نقاط

# - REMOVAL\_METHOD: روش حذف نقاط ('systematic' پیشنهادی)

#

# ☐ منطق حذف نقاط:

# حذف ۱۰۰٪ نقاط: (0-10m) نزدیک‌ترین منطقه

# حذف ۸۰٪ نقاط: (10-20m) منطقه دوم

# حذف ۶۰٪ نقاط: (20-35m) منطقه سوم

# حذف ۴۰٪ نقاط: (35-50m) منطقه چهارم

# حذف ۲۰٪ نقاط: (50-70m) منطقه پنجم

# بدون حذف (۱۰۰٪ حفظ): m دورتر از ۷۰

```

#
=====

=

GRID_SPACING = 5.0 # □ فاصله بین نقاط شبکه (متر)

DISTANCE_ZONES = [
    (0, 10, 1.0), # 0-10 نقاط ۱۰۰٪ حذف متر:
    (10, 20, 0.8), # 10-20 نقاط ۸۰٪ حذف متر:
    (20, 35, 0.6), # 20-35 نقاط ۶۰٪ حذف متر:
    (35, 50, 0.4), # 35-50 نقاط ۴۰٪ حذف متر:
    (50, 70, 0.2), # 50-70 نقاط ۲۰٪ حذف متر:
]

REMOVAL_METHOD = 'systematic' # □ روش حذف سیستماتیک (پیشنهادی)

#
=====

=

# □ بخش ۴: نمایش تنظیمات برنامه

#
=====

=

#

این بخش تنظیمات انتخاب شده را نمایش می‌دهد تا کاربر تأیید کند

#
=====

=

print(f"\n{' '*80}")

```

```

print("\n□ تنظیمات کلی برنامه ")

print(f'{'='*80}')

print(f'\n□ تنظیمات تحلیل رسترها ")

print(f' - Reclassify: {'فعال' if USE_RECLASSIFY_FILTER else 'غیرفعال'}")

print(f' - متن جایگزین: '{ALTERNATIVE_TEXT}''")

print(f' - {'غیرفعال' if ZERO_LAST_CLASS else 'فعال'}: لایه آخر با احتمال صفر - ")

print(f'\n□ تنظیمات ایجاد نقاط غیرمعدنی (ثابت) ")

print(f' - {'GRID_SPACING'} متر: فاصله شبکه - ")

print(f' - {'REMOVAL_METHOD'}: روش حذف - ")

print(f' - {'مناطق فاصله' - ")

for min_dist, max_dist, delete_ratio in DISTANCE_ZONES:

    keep_ratio = 1.0 - delete_ratio

    print(f'      {'min_dist'}-{'max_dist'}m: حذف {'delete_ratio:.0%'} (نگهداری {'keep_ratio:.0%'}))

#
=====
=

# □ بخش ۵: یافتن و تطبیق رسترها و نقاط معدنی □

#
=====
=

#

# □ اهداف این بخش:

# input_gdb لیست کردن رسترهای لیتولوژی از 1.

```

```

# sample_gdb از ('o' با پیشوند) لیست کردن نقاط معدنی. 2.
#
# تطبیق رسترها و نقاط بر اساس ارتفاع (عدد در نام فایل). 3.
#
# نحوه تطبیق:
#
# ارتفاع از نام فایل استخراج می شود (مثلاً ۱۲_۵ یا ۱۵) -
#
# رستر و نقطه با ارتفاع یکسان جفت می شوند -
#
# برای هر جفت، نقاط غیرمعدنی ایجاد می شود -
#
=====
=

# خروجی GDB بررسی وجود
if not arcpy.Exists(output_gdb):
    print(f"\n□ ایجاد GDB: {output_gdb}")
    arcpy.CreateFileGDB_management(os.path.dirname(output_gdb), os.path.basename(output_gdb))

# لیست رسترها بر اساس تنظیمات کاربر. 1.
print(f"\n{'='*80}")
print("□ یافتن رسترها")
print(f"{'='*80}")

all_rasters = []
try:
    original_workspace = arcpy.env.workspace
    arcpy.env.workspace = input_gdb
    all_in_gdb = arcpy.ListRasters()

```

```

if USE_RECLASSIFY_FILTER:
    if ALTERNATIVE_TEXT:
        search_patterns = [ALTERNATIVE_TEXT]

        print(f"جستجو برای رسترهای حاوی '{ALTERNATIVE_TEXT}'")

    else:
        search_patterns = ["Reclassify"]

        print("جستجو برای رسترهای حاوی 'Reclassify'")

    for ras in all_in_gdb:
        for pattern in search_patterns:
            if pattern in ras:
                all_rasters.append(ras)

                print(f"   یافت شد: {ras}")

                break

        else:
            for ras in all_in_gdb:
                all_rasters.append(ras)

                print(f"   یافت شد: {ras}")

    arcpy.env.workspace = original_workspace

except Exception as e:
    print(f"خطا در لیست کردن رسترها: {e}")

    arcpy.env.workspace = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal"

print(f"تعداد رسترهای یافت شده: {len(all_rasters)}")

```

```

if len(all_rasters) == 0:

    print("❑ هیچ رستری یافت نشد")

    exit()

# 2. استخراج ارتفاع از نام رسترها

raster_info = []

for raster_name in all_rasters:

    height_match = re.search(r'(?:\f\b)?(\d+(?:_\d+)?)', raster_name)

    if height_match:

        height_str = height_match.group(1)

        height_normalized = height_str.replace('_', '.')

        raster_info.append({

            'name': raster_name,

            'path': os.path.join(input_gdb, raster_name),

            'height': height_normalized,

            'height_str': height_str

        })

        print(f" - {raster_name} → ارتفاع: {height_normalized}")

    else:

        raster_info.append({

            'name': raster_name,

            'path': os.path.join(input_gdb, raster_name),

            'height': raster_name,

            'height_str': raster_name

        })

        print(f" - {raster_name} → ارتفاع: نام نامشخص")

```

# 3. لیست تمام پوینت‌های نمونه

```
print(f'\n{'='*80}')
```

```
print("□ یافتن پوینت‌های نمونه ...")
```

```
print(f{'='*80}')
```

```
all_sample_points = []
```

```
try:
```

```
    original_workspace = arcpy.env.workspace
```

```
    arcpy.env.workspace = sample_gdb
```

```
    point_fcs = arcpy.ListFeatureClasses("o*", "Point")
```

```
    if point_fcs:
```

```
        for fc in point_fcs:
```

```
            height_match = re.search(r'o(\d+(?:_\d+)?)', fc)
```

```
            if height_match:
```

```
                height_str = height_match.group(1)
```

```
                height_normalized = height_str.replace('_', '.')
```

```
                all_sample_points.append({
```

```
                    'name': fc,
```

```
                    'path': os.path.join(sample_gdb, fc),
```

```
                    'height': height_normalized,
```

```
                    'height_str': height_str
```

```
                })
```

```
                print(f" - {fc} → ارتفاع: {height_normalized}")
```

```
            else:
```



```

all_sample_points.append({
    'name': fc,
    'path': os.path.join(sample_gdb, fc),
    'height': fc,
    'height_str': fc
})

print(f" - {fc} → ارتفاع: نام نامشخص")

```

```

arcpy.env.workspace = original_workspace

```

```

except Exception as e:

```

```

    print(f"خطا در لیست کردن پوینت‌ها: {e}")

```

```

arcpy.env.workspace = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal"

```

```

print(f"\n□ جمع‌بندی:")

```

```

print(f"تعداد رسترها: {len(raster_info)}")

```

```

print(f"تعداد پوینت‌های نمونه: {len(all_sample_points)}")

```

```

if len(all_sample_points) == 0:

```

```

    print("□ هیچ پوینت نمونه‌ای یافت نشد")

```

```

    exit()

```

```

# 4. تطبیق رسترها و پوینت‌های هم‌ارتفاع.

```

```

print(f"\n{'='*80}")

```

```

print("□ تطبیق رسترها و پوینت‌های هم‌ارتفاع")

```

```

print(f"{'='*80}")

```

```

matched_pairs = []

for raster in raster_info:
    for point in all_sample_points:
        if raster['height'] == point['height']:
            matched_pairs.append({
                'raster': raster,
                'point': point,
                'height': raster['height']
            })
            print(f"□ تطبیق: {raster['name']} ↔ {point['name']}")

print(f"\nتعداد جفت‌های تطبیق شده: {len(matched_pairs)}")

if len(matched_pairs) == 0:
    print("□ هیچ جفت تطبیق شده‌ای یافت نشد")
    print("...تلاش برای تطبیق بر اساس بخشی از نام")

    for raster in raster_info:
        for point in all_sample_points:
            if raster['height_str'] in point['name'] or point['height_str'] in raster['name']:
                matched_pairs.append({
                    'raster': raster,
                    'point': point,
                    'height': raster['height_str']
                })
                print(f"□ تطبیق (جایگزین): {raster['name']} ↔ {point['name']}")

```

```

if len(matched_pairs) == 0:

    print("❑ !هنوز هیچ جفت تطبیق شده‌ای یافت نشد")

    exit()

#
=====
=
# ❑ بخش ۶: تابع اصلی ایجاد نقاط غیر معدنی
#
=====
=
#
# ❑ create_fixed_nonmineral_points: نام تابع
# هدف: ایجاد نقاط غیر معدنی با منطق ثابت حذف بر اساس فاصله
#
# ❑ ورودی‌ها:
# - raster_path: مسیر رستر لیتولوژی
# - mineral_points_fc: مسیر نقاط معدنی ('o' پیشوند)
# - output_name: نام خروجی ('n' با پیشوند)
#
# ❑ خروجی‌ها:
# - output_fc: مسیر Feature Class شده
# - final_count: تعداد نقاط نهایی
# - initial_count: تعداد نقاط اولیه
#
# ❑ مراحل اجرا در تابع:
# خواندن رستر و استخراج محدوده 1.

```

```
# ایجاد شبکه ۵ متری در کل محدوده 2.
# محاسبه فاصله هر نقطه از نقاط معدنی 3.
# حذف نقاط بر اساس مناطق فاصله 4.
# اضافه کردن فیلدهای اطلاعاتی 5.
# بازگشت نتایج 6.
```

```
#
```

```
=====
=
```

```
def create_fixed_nonmineral_points(raster_path, mineral_points_fc, output_name):
```

```
    """
```

```
    ایجاد نقاط غیرمعدنی با تنظیمات ثابت
```

```
    منطق:
```

```
    1. شبکه ۵ متری در کل محدوده
```

```
    2. حذف بر اساس فاصله از نقاط معدنی:
```

```
        - 0-10m: ۱۰۰٪ حذف
```

```
        - 10-20m: ۸۰٪ حذف
```

```
        - 20-35m: ۶۰٪ حذف
```

```
        - 35-50m: ۴۰٪ حذف
```

```
        - 50-70m: ۲۰٪ حذف
```

```
        - >70m: بدون حذف
```

```
    """
```

```
    print(f"\n□ ایجاد نقاط غیرمعدنی: {output_name}")
```

```
    print(f"منطق: شبکه ۵ متری + حذف متغیر بر اساس فاصله")
```

try:

# خواندن رستر و استخراج محدوده 1.

raster = arcpy.Raster(raster\_path)

desc = arcpy.Describe(raster)

extent = desc.extent

spatial\_ref = desc.spatialReference

print(f"\n محدوده رستر:")

print(f" X: {extent.XMin:.1f} تا {extent.XMax:.1f}")

print(f" Y: {extent.YMin:.1f} تا {extent.YMax:.1f}")

# ایجاد فایل خروجی 2.

output\_fc = os.path.join(sample\_gdb, output\_name)

if arcpy.Exists(output\_fc):

print(f" □ ...لایه موجود حذف می‌شود")

arcpy.Delete\_management(output\_fc)

# ایجاد شبکه نقاط ۵ متری 3.

width = extent.XMax - extent.XMin

height = extent.YMax - extent.YMin

num\_cols = int(width / GRID\_SPACING) + 1

num\_rows = int(height / GRID\_SPACING) + 1

print(f"\n ...متر {GRID\_SPACING} فاصله) {num\_rows} × {num\_cols} ایجاد شبکه")

# ایجاد feature class

```

arcpy.CreateFeatureclass_management(
    os.path.dirname(output_fc),
    os.path.basename(output_fc),
    "POINT",
    spatial_reference=spatial_ref
)

# 4. تولید نقاط
print("تولید نقاط...")

points = []

for row in range(num_rows):
    y = extent.YMin + row * GRID_SPACING
    for col in range(num_cols):
        x = extent.XMin + col * GRID_SPACING
        if x <= extent.XMax and y <= extent.YMax:
            points.append(arcpy.Point(x, y))

# درج نقاط
with arcpy.da.InsertCursor(output_fc, ["SHAPE@"]) as cursor:
    for point in points:
        cursor.insertRow([point])

initial_count = len(points)
print(f"تعداد نقاط اولیه: {initial_count}")

# 5. اگر نقاط معدنی وجود دارد
if arcpy.Exists(mineral_points_fc):

```

```

mineral_count = int(arcpy.GetCount_management(mineral_points_fc).getOutput(0))

print(f"تعداد نقاط معدنی: {mineral_count:,}")

if mineral_count > 0:
    # محاسبه فاصله
    print("\n محاسبه فواصل...")
    arcpy.Near_analysis(output_fc, mineral_points_fc)

    # اضافه کردن فیلد فاصله
    arcpy.AddField_management(output_fc, "DISTANCE", "DOUBLE")
    arcpy.CalculateField_management(output_fc, "DISTANCE", "!NEAR_DIST!", "PYTHON3")

    # حذف فیلدهای موقت
    arcpy.DeleteField_management(output_fc, ["NEAR_FID", "NEAR_DIST"])

    # حذف نقاط بر اساس مناطق فاصله ثابت 6
    print("\n حذف نقاط بر اساس مناطق فاصله ثابت...")

    zone_stats = []

    for zone_index, (min_dist, max_dist, delete_ratio) in enumerate(DISTANCE_ZONES):
        # انتخاب نقاط در این منطقه
        where_clause = f"DISTANCE >= {min_dist} AND DISTANCE < {max_dist}"

        # ایجاد لایه موقت
        layer_name = f"zone_{zone_index}"
        arcpy.MakeFeatureLayer_management(output_fc, layer_name, where_clause)

```

```

zone_count = int(arcpy.GetCount_management(layer_name).getOutput(0))

if zone_count == 0:
    print(f"منطقه {min_dist}-{max_dist}m: 0 نقطه")
    zone_stats.append((min_dist, max_dist, delete_ratio, 0, 0, 0))
    arcpy.Delete_management(layer_name)
    continue

delete_count = int(zone_count * delete_ratio)
keep_count = zone_count - delete_count

print(f"منطقه {min_dist}-{max_dist}m: {zone_count:,} حذف، {delete_count:,} نقطه")
zone_stats.append((min_dist, max_dist, delete_ratio, zone_count, delete_count, keep_count))

if delete_ratio == 1.0: # حذف کامل
    arcpy.DeleteFeatures_management(layer_name)
elif delete_count > 0:
    # (امین نقطه n حذف هر) روش سیستماتیک
    step = zone_count / delete_count if delete_count > 0 else 1

    # علامت گذاری نقاط برای حذف
    arcpy.AddField_management(layer_name, "DELETE_FLAG", "SHORT")

    with arcpy.da.UpdateCursor(layer_name, ["OBJECTID", "DELETE_FLAG"]) as cursor:
        for i, row in enumerate(cursor):
            # حذف هر چندمین نقطه
            if i % step < 1:

```



```

        row[1] = 1
    else:
        row[1] = 0
    cursor.updateRow(row)

    # حذف نقاط علامت‌دار
    delete_layer = f"delete_{zone_index}"
    arcpy.MakeFeatureLayer_management(layer_name, delete_layer, "DELETE_FLAG = 1")
    arcpy.DeleteFeatures_management(delete_layer)
    arcpy.Delete_management(delete_layer)

    # حذف فیلد موقت
    try:
        arcpy.DeleteField_management(layer_name, ["DELETE_FLAG"])
    except:
        pass

    # حذف لایه موقت
    arcpy.Delete_management(layer_name)

    # 7. نقاط با فاصله بیشتر از ۷۰ متر (بدون حذف).
    last_max_dist = DISTANCE_ZONES[-1][1] # آخرین محدوده = ۷۰
    where_clause = f"DISTANCE >= {last_max_dist}"
    layer_name = "zone_over_70"
    arcpy.MakeFeatureLayer_management(output_fc, layer_name, where_clause)
    over_70_count = int(arcpy.GetCount_management(layer_name).getOutput(0))
    zone_stats.append((last_max_dist, float('inf'), 0.0, over_70_count, 0, over_70_count))

```

```

print(f"نقطه (بدون حذف) {last_max_dist}m: {over_70_count:,} منطقه >منطقه")

arcpy.Delete_management(layer_name)

# نمایش آمار مناطق

print(f"\n□ آمار مناطق فاصله:")

print("-" * 80)

print(f"{'حذف!>۱۰'} {'نقاط اولیه!>۱۵'} {'حذف شده!>۱۵'} {'باقی مانده!>۱۵'} {'(m) فاصله!<15'}")

print("-" * 80)

total_zones_initial = 0
total_zones_deleted = 0
total_zones_kept = 0

for min_d, max_d, del_ratio, z_count, del_count, keep_count in zone_stats:
    if max_d == float('inf'):
        range_str = f"{min_d}+"
    else:
        range_str = f"{min_d}-{max_d}"

    print(f"{'range_str:<15'} {'del_ratio:<10.0%'} {'z_count:<15,'} {'del_count:<15,'}
{'keep_count:<15,'}")

    total_zones_initial += z_count
    total_zones_deleted += del_count
    total_zones_kept += keep_count

print("-" * 80)

```

```

print(f'کل!>\5} {"<:10} {total_zones_initial:<15,} {total_zones_deleted:<15,}
{total_zones_kept:<15,}')

# حذف فیلد distance

try:
    arcpy.DeleteField_management(output_fc, ["DISTANCE"])
except:
    pass

# اضافه کردن فیلدهای اطلاعاتی (طول فیلدها افزایش یافته). 8.
arcpy.AddField_management(output_fc, "PointType", "TEXT", field_length=30) # افزایش یافته
arcpy.AddField_management(output_fc, "Height", "TEXT", field_length=20)
arcpy.AddField_management(output_fc, "Source", "TEXT", field_length=200)
arcpy.AddField_management(output_fc, "GridSpacing", "FLOAT")
arcpy.AddField_management(output_fc, "Logic", "TEXT", field_length=150) # افزایش یافته

with arcpy.da.UpdateCursor(output_fc, ["PointType", "Height", "Source", "GridSpacing", "Logic"])
as cursor:
    for row in cursor:
        row[0] = "NonMineral_FixedLogic"
        row[1] = output_name.replace("n", "")
        row[2] = f"5m_grid_fixed_zones"
        row[3] = GRID_SPACING
        # بهروزرسانی منطق بر اساس محدوده‌های جدید
        logic_str = "0-10(100%),10-20(80%),20-35(60%),35-50(40%),50-70(20%)"
        row[4] = logic_str
        cursor.updateRow(row)

```

شمارش نهایی. 9. #

```
final_count = int(arcpy.GetCount_management(output_fc).getOutput(0))
```

```
deleted_count = initial_count - final_count
```

```
print(f"\n□ نتایج نهایی:")
```

```
print(f" نقاط اولیه: {initial_count:,}")
```

```
print(f" نقاط حذف شده: {deleted_count:,}")
```

```
print(f" نقاط نهایی: {final_count:,}")
```

```
if initial_count > 0:
```

```
    print(f" درصد باقی مانده: {(final_count/initial_count*100):.1f}%")
```

```
return output_fc, final_count, initial_count
```

except Exception as e:

```
    print(f"□ خطا در ایجاد نقاط: {str(e)}")
```

```
import traceback
```

```
traceback.print_exc()
```

```
return None, 0, 0
```

```
#
```

```
=====
```

```
=
```

```
# □ بخش ۷: اجرای اصلی برنامه
```

```
#
```

```
=====
```

```
=
```

```
#
```

```
# □ این بخش تابع اصلی را برای هر جفت رستر-نقطه فراخوانی می کند
```

```
# □ آمار اجرا و نتایج را جمع‌آوری و نمایش می‌دهد
```

```
# □ گزارش نهایی در فایل متنی ذخیره می‌شود
```

```
#
```

```
=====
```

```
=
```

```
print(f"\n{' '*80}")
```

```
print("□ شروع پردازش با منطق ثابت")
```

```
print(f"{' '*80}")
```

```
non_mineral_points_created = []
```

```
success_count = 0
```

```
failed_count = 0
```

```
total_initial_points = 0
```

```
total_final_points = 0
```

```
total_deleted_points = 0
```

```
for i, pair in enumerate(matched_pairs, 1):
```

```
    print(f"\n□ پردازش تراز {i} از {len(matched_pairs)}")
```

```
    raster_info = pair['raster']
```

```
    point_info = pair['point']
```

```
    height = pair['height']
```

```
    # نام خروجی
```

```
    height_str = str(height).replace('.', '_')
```

```
    output_name = f"n{height_str}"
```

```

# بررسی وجود قبلی
output_fc = os.path.join(sample_gdb, output_name)
if arcpy.Exists(output_fc):
    print(f"□ لایه موجود حذف می‌شود {output_name}")
    arcpy.Delete_management(output_fc)

# ایجاد نقاط غیرمعدنی با منطق ثابت
result_fc, final_count, initial_count = create_fixed_nonmineral_points(
    raster_path=raster_info['path'],
    mineral_points_fc=point_info['path'],
    output_name=output_name
)

# ثبت نتیجه
if result_fc and final_count > 0:
    non_mineral_points_created.append({
        'height': height,
        'fc_name': output_name,
        'fc_path': result_fc,
        'initial_count': initial_count,
        'final_count': final_count,
        'deleted_count': initial_count - final_count,
        'percentage_kept': (final_count / initial_count * 100) if initial_count > 0 else 0,
        'raster': raster_info['name'],
        'mineral_points': point_info['name']
    })

    success_count += 1

```

```
total_initial_points += initial_count
```

```
total_final_points += final_count
```

```
total_deleted_points += (initial_count - final_count)
```

```
print(f"□ تراز {height}: {final_count:}, {initial_count-final_count:} (نقطه حذف شد {initial_count-final_count:}, نقطه ایجاد شد {final_count:})")
```

```
else:
```

```
failed_count += 1
```

```
print(f"□ تراز {height}: ناموفق")
```

```
#
```

```
=====
```

```
=
```

```
# □ بخش ۸: گزارش نهایی و خلاصه
```

```
#
```

```
=====
```

```
=
```

```
#
```

```
# □ این بخش نتایج نهایی را نمایش می‌دهد و گزارش کاملی ایجاد می‌کند
```

```
# □ ذخیره می‌شود timestamp گزارش در فایل متنی با
```

```
# □ نکات نهایی برای تحلیل بیزین ارائه می‌شود
```

```
#
```

```
=====
```

```
=
```

```
print(f"\n{'='*80}")
```

```
print("□ گزارش نهایی")
```

```
print(f"{'='*80}")
```

```
if non_mineral_points_created:
```

```
overall_percentage = (total_final_points / total_initial_points * 100) if total_initial_points > 0 else 0
```

```
print(f"\n□ آمار کلی:")
```

```
print(f"تعداد ترازهای پردازش شده: {len(matched_pairs)}")
```

```
print(f"تعداد موفق: {success_count}")
```

```
print(f"تعداد ناموفق: {failed_count}")
```

```
print(f"کل نقاط اولیه: {total_initial_points:,}")
```

```
print(f"کل نقاط حذف شده: {total_deleted_points:,}")
```

```
print(f"کل نقاط نهایی: {total_final_points:,}")
```

```
print(f"درصد کلی باقی‌مانده: {overall_percentage:.1f}%")
```

```
print(f"\n□ منطق استفاده شده (ثابت):")
```

```
print(f"متر {GRID_SPACING}: فاصله شبکه -")
```

```
print(f"روش حذف: سیستماتیک -")
```

```
print(f"مناطق فاصله: -")
```

```
for min_d, max_d, ratio in DISTANCE_ZONES:
```

```
    keep_ratio = 1.0 - ratio
```

```
    print(f"حذف {ratio:.0%} (نگهداری {keep_ratio:.0%}) {min_d}-{max_d}m")
```

```
print(f"بدون حذف (نگهداری ۱۰۰٪) >{DISTANCE_ZONES[-1][1]}m")
```

```
print(f"\n□ جزئیات هر تراز:")
```

```
print("-" * 100)
```

```
print(f"تراز: >۱۰ {نام لایه: >۱۵} {اولیه: >۱۲} {نهایی: >۱۲} {حذف شده: >۱۲} {درصد باقی: >۱۲}")
```

```
print("-" * 100)
```



```

for item in non_mineral_points_created:

    print(f'{item["height"]:<10} {item["fc_name"]:<15} "

          f'{item["initial_count"]:<12,} {item["final_count"]:<12,} "

          f'{item["deleted_count"]:<12,} {item["percentage_kept"]:<12.1f}%")

# ذخیره گزارش

timestamp = time.strftime('%Y%m%d_%H%M%S')

report_file = os.path.join(os.path.dirname(sample_gdb),
f'NonMineral_FixedLogic_Report_{timestamp}.txt')

with open(report_file, 'w', encoding='utf-8') as f:

    f.write("=" * 100 + "\n")

    f.write("گزارش ایجاد نقاط غیرمعدنی با منطق ثابت\n")

    f.write("=" * 100 + "\n\n")

    f.write(f'تاریخ ایجاد: {time.strftime('%Y-%m-%d %H:%M:%S')}\n')

    f.write(f'مکان ذخیره سازی: {sample_gdb}\n\n')

    f.write("منطق ثابت استفاده شده:\n")

    f.write(f'{GRID_SPACING} متر: فاصله شبکه -\n')

    f.write(f'روش حذف: سیستماتیک -\n')

    f.write("مناطق فاصله -\n")

    for min_d, max_d, ratio in DISTANCE_ZONES:

        keep_ratio = 1.0 - ratio

        f.write(f'{min_d}-{max_d}m: حذف {ratio:.0%} (نگهداری {keep_ratio:.0%})\n')

    last_max = DISTANCE_ZONES[-1][1]

    f.write(f'بدون حذف (نگهداری ۱۰۰٪): >{last_max}m\n')

```

```

f.write(f"\nآمار کلی:\n")

f.write(f"تعداد ترازهای پردازش شده: {len(matched_pairs)}\n")

f.write(f"تعداد موفق: {success_count}\n")

f.write(f"تعداد ناموفق: {failed_count}\n")

f.write(f"کل نقاط اولیه: {total_initial_points:,}\n")

f.write(f"کل نقاط حذف شده: {total_deleted_points:,}\n")

f.write(f"کل نقاط نهایی: {total_final_points:,}\n")

f.write(f"درصد کلی باقی مانده: {overall_percentage:.1f}%\n\n")


f.write("\nجزئیات هر تراز:\n")

f.write("-" * 100 + "\n")

f.write(f"\nتراز: >۱۰ { 'نام لایه: >۱۵' { 'اولیه: >۱۲' { 'نهایی: >۱۲' { 'حذف شده: >۱۲' { 'درصد باقی: >۱۲' }\n")

f.write("-" * 100 + "\n")


for item in non_mineral_points_created:

    f.write(f'{item["height"]:<10} {item["fc_name"]:<15} '

           f'{item["initial_count"]:<12,} {item["final_count"]:<12,} '

           f'{item["deleted_count"]:<12,} {item["percentage_kept"]:<12.1f}%\n')


print(f"\n□ گزارش کامل ذخیره شد: {report_file}")


else:

    print("\n□ هیچ نقطه غیرمعدنی ایجاد نشد")


print(f"\n{'='*80}")

print("\n□ ایجاد نقاط غیرمعدنی با منطق ثابت کامل شد")

```

```

print(f'{'='*80}')

#
=====

=

# □ راهنمای تحلیل بیزین
#
=====

=

#
# □ این بخش نکات مهم برای تحلیل بیزین را ارائه می دهد
# □ Prior Probability:  $P(A|B) = P(B|A) \times P(A) / P(B)$  فرمول
# □ نقاط معدنی / (نقاط معدنی + نقاط غیرمعدنی) = Prior: در این پروژه
#
=====

=

print(f"\n□ برای تحلیل بیزین")
print(f'1. از {success_count} با پیشوند غیرمعدنی با پیشوند')
print(f'2. از {len(matched_pairs)} با پیشوند معدنی با پیشوند')
print(f'3. برای هر تراز، نسبت نقاط معدنی به غیرمعدنی را محاسبه کنید')
print(f'4. احتمال Prior = (نقاط معدنی + نقاط غیرمعدنی)')

if success_count > 0:
    print(f"\n□ اکنون می توانید تحلیل بیزین واقعی را شروع کنید")
    print(f" {sample_gdb}: نقاط غیرمعدنی در")
    print(f" و منطق ثابت حذف متغیر 'n' با پیشوند")
else:

```

```

print(f'\n❑ هیچ نقطه غیرمعدنی ایجاد نشد. تحلیل بیزین ممکن نیست')

print(f'\n{'='*80}')

print("❑ خلاصه منطق ثابت:")

print(f'{'='*80}')

print(f'1. شبکه ۵ متری در کل محدوده رستر')

print(f'2. حذف کامل نقاط در فاصله ۰-۱۰ متر از نقاط معدنی')

print(f'3. حذف ۸۰٪ نقاط در فاصله ۱۰-۲۰ متر')

print(f'4. حذف ۶۰٪ نقاط در فاصله ۲۰-۳۵ متر')

print(f'5. حذف ۴۰٪ نقاط در فاصله ۳۵-۵۰ متر')

print(f'6. حذف ۲۰٪ نقاط در فاصله ۵۰-۷۰ متر')

print(f'7. بدون حذف برای نقاط با فاصله بیشتر از ۷۰ متر')

```

انیمیشن کد قبل

```

import arcpy
import os
import re
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
from PIL import Image

point_gdb_path = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Gosal.gdb"
output_folder = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Animation"
output_gif = os.path.join(output_folder, "Points_N.gif")

```

```

arcpy.env.workspace = point_gdb_path

all_datasets = arcpy.ListDatasets() + [""]
point_features = []

for ds in all_datasets:
    arcpy.env.workspace = os.path.join(point_gdb_path, ds) if ds else point_gdb_path
    fcs = arcpy.ListFeatureClasses('n*')
    if fcs:
        for fc in fcs:
            full_path = os.path.join(arcpy.env.workspace, fc) if ds else fc
            point_features.append(full_path)

def extract_number(name):
    matches = re.findall(r'\d+\.\d*', os.path.basename(name))
    return float(matches[0]) if matches else 0

point_features_sorted = sorted(point_features, key=extract_number)

all_layers_points = []
layer_numbers = []

for point_fc in point_features_sorted:
    layer_number = extract_number(point_fc)
    layer_numbers.append(layer_number)

points = []

with arcpy.da.SearchCursor(point_fc, ['SHAPE@X', 'SHAPE@Y']) as cursor:

```

```
        for row in cursor:

            points.append((row[0], row[1]))

    all_layers_points.append(points)

all_x = []
all_y = []
for points in all_layers_points:
    if points:
        all_x.extend([p[0] for p in points])
        all_y.extend([p[1] for p in points])

xmin, xmax = min(all_x), max(all_x)
ymin, ymax = min(all_y), max(all_y)

x_range = xmax - xmin
y_range = ymax - ymin
xmin -= x_range * 0.05
xmax += x_range * 0.05
ymin -= y_range * 0.05
ymax += y_range * 0.05

total_points = sum(len(points) for points in all_layers_points)

images = []
os.makedirs(output_folder, exist_ok=True)

color_gradient = plt.cm.viridis(np.linspace(0.2, 0.8, len(point_features_sorted)))
```

```
color_hex_list = [colors.rgb2hex(color) for color in color_gradient]
```

```
for i, (points, layer_number, frame_color) in enumerate(zip(
    all_layers_points, layer_numbers, color_hex_list)):
```

```
    if not points:
```

```
        continue
```

```
    fig, ax = plt.subplots(figsize=(12, 10), dpi=150)
```

```
    fig.patch.set_facecolor('#2C3E50')
```

```
    ax.set_facecolor('#1A1A2E')
```

```
    x_coords = [p[0] for p in points]
```

```
    y_coords = [p[1] for p in points]
```

```
    ax.scatter(x_coords, y_coords,
```

```
               s=25,
```

```
               c=frame_color,
```

```
               edgecolors='FFFFFF',
```

```
               linewidths=0.3,
```

```
               alpha=0.7,
```

```
               marker='o')
```

```
    ax.set_xlim(xmin, xmax)
```

```
    ax.set_ylim(ymin, ymax)
```

```
    ax.set_aspect('equal')
```

```
ax.set_xlabel('X Coordinate', fontsize=12, color='#ECF0F1')
ax.set_ylabel('Y Coordinate', fontsize=12, color='#ECF0F1')

ax.set_title(f'Elevation: {layer_number}', fontsize=16, fontweight='bold', color='#ECF0F1')

ax.tick_params(colors='#ECF0F1')

for spine in ax.spines.values():
    spine.set_color('#ECF0F1')

ax.grid(True, alpha=0.2, linestyle='--', linewidth=0.5, color='#7F8C8D')

ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: format(int(x), ', ')))

plt.tight_layout()

fig.canvas.draw()
img_array = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)
img_array = img_array.reshape(fig.canvas.get_width_height()[::-1] + (3,))

images.append(Image.fromarray(img_array))
plt.close(fig)

if len(images) > 1:
    images[0].save(
        output_gif,
        save_all=True,
        append_images=images[1:],
```



```

        duration=800,
        loop=0,
        optimize=True,
        quality=95
    )

    print(f'انیمیشن ساخته شد {output_gif}')
    print(f'تعداد فریم‌ها: {len(images)}')
    print(f'کل پوینت‌ها: {total_points:,}')

```

بیزین

```

import arcpy
import numpy as np
import os
import time
import re
import math
from collections import defaultdict

#
=====
=
# □ بخش ۱: تنظیمات قابل ویرایش توسط کاربر
#
=====
=
#
# □ در این بخش می‌توانید مسیرها، الگوها و پارامترهای تحلیل را تنظیم کنید

```

# □ هر پارامتری که نیاز به تغییر دارد در این قسمت قرار گرفته است.

#

=====

=

# ArcGIS تنظیمات محیطی

arcpy.env.overwriteOutput = True # □ اجازه رونویسی فایل‌های موجود

arcpy.env.workspace = r"M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal" # □ مسیر اصلی کار

# 1. □ مسیره‌های اصلی (حتمی برای ویرایش)

input\_gdb = r"M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal\Litho\_Rasters.gdb" # □ جی‌دی‌بی حاوی

رسترهای لیتولوژی ورودی

sample\_gdb = r"M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal\Gosal.gdb" # □ جی‌دی‌بی حاوی نقاط

معدنی و غیرمعدنی

output\_gdb = r"M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal\Litho\_Rasters.gdb" # □ جی‌دی‌بی خروجی

(باشد input\_gdb ممکن است همان)

# 2. □ الگوی نام رسترهای ورودی (قابل تنظیم)

# استفاده می‌شود input\_gdb این الگو برای جستجوی رسترها در

# مثال‌ها:

# - "\*ColorRaster\*" همه رسترهای حاوی : "ColorRaster"

# - "\*Litho\*" همه رسترهای حاوی : "Litho"

# - "\*.tif" همه فایل‌های تیف :

# - "\*" همه فایل‌های رستری :

input\_raster\_pattern = "\*ColorRaster\*"

# 3. □ الگوی نام فایل‌های نقاط (قابل تنظیم)

# فرمت: prefix + height + suffix

mineral\_point\_prefix = "o" # پیشوند فایل نقاط معدنی (مانند o875\_Points)

mineral\_point\_suffix = "\_Points" # پسوند فایل نقاط معدنی

non\_mineral\_prefix = "n" # پیشوند فایل نقاط غیرمعدنی (مانند n875)

non\_mineral\_suffix = "" # پسوند فایل نقاط غیرمعدنی (می‌تواند خالی باشد)

# 4. □ ارتفاع‌های مورد نظر برای تحلیل (قابل تنظیم)

# :دو روش برای تعیین ارتفاع‌ها

# روش ۱: لیست دستی ارتفاع‌ها (فعال)

# روش ۲: محدوده ارتفاع با گام (غیرفعال - کامنت شده)

```
heights = [
    875, 887.5, 900, 912.5, 925, 937.5, 950, 962.5, 975, 987.5,
    1000, 1012.5, 1025, 1037.5, 1050, 1062.5, 1075, 1087.5,
    1100, 1112.5, 1125
]
```

# روش ۲: استفاده از محدوده ارتفاع (در صورت نیاز کامنت را بردارید)

# start\_height = 875 # ارتفاع شروع

# end\_height = 1125 # ارتفاع پایان

# step = 12.5 # گام افزایش ارتفاع

# heights = [start\_height + i \* step for i in range(int((end\_height - start\_height) / step) + 1)]

# 5. □ تنظیمات تحلیل بیزین (قابل تنظیم)

threshold = 0.5 # آستانه طبقه‌بندی (۰ تا ۱)

مقادیر بالاتر از این آستانه به عنوان معدنی طبقه‌بندی می‌شوند

```
confidence_levels = { # □ گزارش گیری سطوح اطمینان برای
```

```
'خیلی زیاد': ۰,۸, # احتمال  $\leq ۰,۸$ 
```

```
'زیاد': ۰,۶, # احتمال  $\leq ۰,۶$ 
```

```
'متوسط': ۰,۴, # احتمال  $\leq ۰,۴$ 
```

```
'کم': ۰,۰, # احتمال  $> ۰,۴$ 
```

```
}
```

```
# 6. □ تنظیمات خروجی (قابل تنظیم)
```

```
create_probability_raster = True # □ ایجاد رستر احتمالات (احتمال معدنی بودن)
```

```
create_classification_raster = True # □ ایجاد رستر طبقه بندی (معدنی/غیرمعدنی)
```

```
create_report = True # □ ایجاد گزارش متنی از نتایج
```

```
clean_temp_files = True # □ پاکسازی فایل های موقت بعد از پردازش
```

```
#
```

```
=
```

```
# □ بخش ۲: تابع اصلی تحلیل بیزین
```

```
#
```

```
=
```

```
#
```

```
# □ این تابع تحلیل بیزین را برای یک ارتفاع خاص انجام می دهد.
```

```
# □ ورودی ها:
```

```
# - raster_path: مسیر رستر لیتولوژی
```

```
# - mineral_fc: مسیر نقاط معدنی (Feature Class)
```

```
# - non_mineral_fc: مسیر نقاط غیرمعدنی
```

```
# - output_name: نام خروجی
```

```
# - height: ارتفاع مورد تحلیل
```

```
#
```

```
# □ مراحل تحلیل:
```

```
# خواندن رستر و استخراج مقادیر از نقاط ۱.
```

```
# محاسبه احتمالات شرطی ۲.
```

```
# اعمال قضیه بیز ۳.
```

```
# ایجاد رسترهای خروجی ۴.
```

```
# گزارش گیری ۵.
```

```
#
```

```
=====
```

```
=
```

```
def simple_bayesian_analysis(raster_path, mineral_fc, non_mineral_fc, output_name, height):
```

```
    """تحلیل بیزین ساده و مستقیم برای یک ارتفاع خاص"""
```

```
    print(f"\n{'='*80}")
```

```
    print(f"□ تحلیل بیزین برای ارتفاع {height}")
```

```
    print(f"{'='*80}")
```

```
    start_time = time.time() # □ زمان شروع برای محاسبه مدت پردازش
```

```
    try:
```

```
        # 1. □ خواندن رستر ورودی
```

```
        print("1. □ خواندن رستر...")
```

```
        raster = arcpy.Raster(raster_path)
```

```
        raster_name = os.path.basename(raster_path)
```

```

print(f"رستر: {raster_name}")

# 2. □ استخراج مقادیر از نقاط معدنی

print("\n2. □ استخراج مقادیر نقاط معدنی...")

temp_mineral = os.path.join(output_gdb, f"temp_m_{output_name}")
if arcpy.Exists(temp_mineral):
    arcpy.Delete_management(temp_mineral)

# □ استخراج مقادیر رستر در مکان نقاط معدنی

arcpy.sa.ExtractValuesToPoints(mineral_fc, raster, temp_mineral, "NONE", "VALUE_ONLY")

# □ خواندن مقادیر استخراج شده

mineral_values = []
with arcpy.da.SearchCursor(temp_mineral, ["RASTERVALU"]) as cursor:
    for row in cursor:
        if row[0] is not None:
            try:
                val = int(float(row[0]))
                mineral_values.append(val)
            except:
                continue # در صورت خطا، ادامه بده

mineral_count = len(mineral_values)
print(f"تعداد نقاط معدنی: {mineral_count}")

# 3. □ استخراج مقادیر از نقاط غیرمعدنی

print("\n3. □ استخراج مقادیر نقاط غیرمعدنی...")

```

```

temp_non_mineral = os.path.join(output_gdb, f"temp_nm_{output_name}")

if arcpy.Exists(temp_non_mineral):
    arcpy.Delete_management(temp_non_mineral)

# □ استخراج مقادیر رستر در مکان نقاط غیرمعدنی

arcpy.sa.ExtractValuesToPoints(non_mineral_fc, raster, temp_non_mineral, "NONE",
"VALUE_ONLY")

# □ خواندن مقادیر استخراج شده

non_mineral_values = []

with arcpy.da.SearchCursor(temp_non_mineral, ["RASTERVALU"]) as cursor:
    for row in cursor:
        if row[0] is not None:
            try:
                val = int(float(row[0]))
                non_mineral_values.append(val)
            except:
                continue # در صورت خطا، ادامه بده

non_mineral_count = len(non_mineral_values)

print(f"تعداد نقاط غیرمعدنی: {non_mineral_count}")

# □ بررسی کفایت داده

if mineral_count == 0 or non_mineral_count == 0:
    print("داده کافی نیست! حداقل یک نقطه معدنی و یک نقطه غیرمعدنی نیاز است □")

    return {'success': False, 'height': height}

# 4. □ شناسایی کلاس‌های منحصر به فرد در داده‌ها

```

```

print("\n4. □ شناسایی کلاس‌های واقعی")

# مجموعه کلاس‌های موجود در نقاط معدنی
unique_from_mineral = set(mineral_values)

# مجموعه کلاس‌های موجود در نقاط غیرمعدنی
unique_from_non_mineral = set(non_mineral_values)

# اجتماع کل کلاس‌ها
unique_classes = sorted(list(unique_from_mineral.union(unique_from_non_mineral)))

# □ اگر هیچ کلاسی شناسایی نشد، از رستر نمونه‌گیری می‌کنیم
if not unique_classes:

    print(" □ هیچ کلاسی از نقاط استخراج نشد. نمونه‌گیری از رستر")

    temp_sample = os.path.join(output_gdb, f"sample_{output_name}")

    if arcpy.Exists(temp_sample):

        arcpy.Delete_management(temp_sample)

    # ایجاد ۱۰۰۰ نقطه تصادفی در محدوده رستر

    arcpy.CreateRandomPoints_management(output_gdb, f"sample_{output_name}",

                                         raster.extent, "", 1000)

    # استخراج مقادیر از نقاط تصادفی

    arcpy.sa.ExtractValuesToPoints(temp_sample, raster, temp_sample, "NONE", "VALUE_ONLY")

    # خواندن مقادیر استخراج شده

    with arcpy.da.SearchCursor(temp_sample, ["RASTERVALU"]) as cursor:

        for row in cursor:

            if row[0] is not None:

```



```

try:
    val = int(float(row[0]))
    unique_classes.append(val)
except:
    continue

# پاکسازی فایل موقت نمونه‌گیری
if arcpy.Exists(temp_sample):
    arcpy.Delete_management(temp_sample)

# حذف مقادیر تکراری و مرتب‌سازی
unique_classes = sorted(list(set(unique_classes)))
print(f" {unique_classes} : کلاس‌های شناسایی شده ")

# بررسی تعداد کلاس‌ها □
if len(unique_classes) < 2:
    print(f"فقط یک کلاس وجود دارد - تحلیل بیزین محدود خواهد بود □")

# شمارش تکرار هر کلاس در نقاط معدنی و غیرمعدنی □ 5.
print(f"\n5. □ شمارش واقعی")

# دیکشنری برای شمارش کلاس‌ها در نقاط معدنی
mineral_counts = defaultdict(int)

# دیکشنری برای شمارش کلاس‌ها در نقاط غیرمعدنی
non_mineral_counts = defaultdict(int)

# شمارش نقاط معدنی

```

```

for val in mineral_values:
    mineral_counts[val] += 1

# شمارش نقاط غیر معدنی
for val in non_mineral_values:
    non_mineral_counts[val] += 1

# نمایش نتایج شمارش
print("نتایج شمارش:")

for cls in unique_classes:
    m_count = mineral_counts[cls]
    nm_count = non_mineral_counts[cls]
    m_percent = (m_count / mineral_count * 100) if mineral_count > 0 else 0
    nm_percent = (nm_count / non_mineral_count * 100) if non_mineral_count > 0 else 0
    print(f"کلاس {cls}: {m_count} معدنی ({m_percent:.1f}%) | {nm_count} غیر معدنی ({nm_percent:.1f}%)")

# 6. □ محاسبه احتمالات بیزین
print("\n6. □ محاسبه بیزین واقعی")

# P(Class|Mineral) = تعداد نقاط معدنی در این کلاس / کل نقاط معدنی
p_class_given_mineral = {}

# P(Class|NonMineral) = تعداد نقاط غیر معدنی در این کلاس / کل نقاط غیر معدنی
p_class_given_non_mineral = {}

for cls in unique_classes:
    p_class_given_mineral[cls] = mineral_counts[cls] / mineral_count if mineral_count > 0 else
0.0001

```

```

    p_class_given_non_mineral[cls] = non_mineral_counts[cls] / non_mineral_count if
non_mineral_count > 0 else 0.0001

# محاسبه احتمالات Prior

total_points = mineral_count + non_mineral_count

p_mineral = mineral_count / total_points # P(Mineral)

p_non_mineral = non_mineral_count / total_points # P(NonMineral)

print(f" P(معدنی) = {p_mineral:.4f}, P(غیرمعدنی) = {p_non_mineral:.4f}")

# محاسبه احتمالات Posterior: P(Mineral|Class)

posterior_mineral = {} # احتمال معدنی بودن با شرط کلاس

bayes_factor = {} # ضریب بیزین (نسبت احتمالات)

for cls in unique_classes:

    # قانون احتمال کل: P(Class) = P(Class|Mineral)*P(Mineral) + P(Class|NonMineral)*P(NonMineral)

    p_class = (p_class_given_mineral[cls] * p_mineral) + (p_class_given_non_mineral[cls] *
p_non_mineral)

    # قضیه بیز: P(Mineral|Class) = P(Class|Mineral) * P(Mineral) / P(Class)

    if p_class > 0:

        posterior = (p_class_given_mineral[cls] * p_mineral) / p_class

    else:

        posterior = 0

    # محدود کردن مقدار به بازه [0, 1]

    posterior = max(0.0, min(1.0, posterior))

    posterior_mineral[cls] = posterior

```

```

# محاسبه ضریب بیزین: P(Class|Mineral) / P(Class|NonMineral)

if p_class_given_non_mineral[cls] > 0:
    factor = p_class_given_mineral[cls] / p_class_given_non_mineral[cls]
else:
    factor = 1000 if p_class_given_mineral[cls] > 0 else 0

bayes_factor[cls] = min(factor, 1000) # محدود کردن به حداکثر ۱۰۰۰

# 7. □ ایجاد جدول نتایج در Geodatabase
print("\n7. □ ایجاد جدول نتایج...")

output_table_name = f"Bayesian_Results_{output_name}"
output_table = os.path.join(output_gdb, output_table_name)

# پاکسازی جدول قبلی اگر وجود دارد
if arcpy.Exists(output_table):
    arcpy.Delete_management(output_table)

# ایجاد جدول جدید
arcpy.CreateTable_management(output_gdb, output_table_name)

# تعریف فیلدهای جدول
fields = [
    ("HEIGHT", "TEXT", "۲۰", "(ارتفاع)", # ارتفاع مورد تحلیل
    ("CLASS", "LONG", "کلاس"), # کلاس لیتولوژی
    ("MINERAL_COUNT", "LONG", "تعداد معدنی", # تعداد نقاط معدنی در این کلاس

```

```

("NONMINERAL_COUNT", "LONG", "تعداد غیرمعدنی در این کلاس #"),
("P_CLASS_GIVEN_MINERAL", "DOUBLE", "احتمال_کلاس_با_شرط_معدنی"), # P(Class|Mineral)
("P_CLASS_GIVEN_NONMINERAL", "DOUBLE", "احتمال_کلاس_با_شرط_غیرمعدنی"), #
P(Class|NonMineral)
("P_MINERAL_GIVEN_CLASS", "DOUBLE", "احتمال_معدنی_با_شرط_کلاس"), # P(Mineral|Class)
("BAYES_FACTOR", "DOUBLE", "ضریب_بیزین"), # نسبت احتمالات
("CLASSIFIED", "SHORT", "طبقه‌بندی"), # ۰=غیرمعدنی ۱=معدنی
("CONFIDENCE", "TEXT", "سطح اطمینان # (اعتماد, ۵۰)"),
]

# افزودن فیلدها به جدول

for field_name, field_type, alias, *args in fields:
    if field_type == "TEXT":
        arcpy.AddField_management(output_table, field_name, field_type, "", "", args[0])
    else:
        arcpy.AddField_management(output_table, field_name, field_type)
        arcpy.AlterField_management(output_table, field_name, new_field_alias=alias)

# پر کردن جدول با نتایج

mineral_classes = [] # لیست کلاس‌های معدنی
non_mineral_classes = [] # لیست کلاس‌های غیرمعدنی

with arcpy.da.InsertCursor(output_table,
    ["HEIGHT", "CLASS", "MINERAL_COUNT", "NONMINERAL_COUNT",
    "P_CLASS_GIVEN_MINERAL", "P_CLASS_GIVEN_NONMINERAL",
    "P_MINERAL_GIVEN_CLASS", "BAYES_FACTOR",
    "CLASSIFIED", "CONFIDENCE"]) as cursor:

```

```

for cls in unique_classes:

    # طبقه‌بندی با آستانه threshold

    classified = 1 if posterior_mineral[cls] >= threshold else 0

    # ثبت کلاس‌های معدنی و غیرمعدنی برای گزارش

    if classified == 1:

        mineral_classes.append(cls)

    else:

        non_mineral_classes.append(cls)

    # تعیین سطح اعتماد بر اساس probability

    confidence = "کم"

    for conf_name, conf_threshold in confidence_levels.items():

        if posterior_mineral[cls] >= conf_threshold:

            confidence = conf_name

            break

    # درج ردیف در جدول

    cursor.insertRow([

        str(height), # ارتفاع

        cls, # کلاس

        mineral_counts[cls], # تعداد معدنی

        non_mineral_counts[cls], # تعداد غیرمعدنی

        round(p_class_given_mineral[cls], 6), # P(Class|Mineral)

        round(p_class_given_non_mineral[cls], 6), # P(Class|NonMineral)

        round(posterior_mineral[cls], 6), # P(Mineral|Class)

```

```
round(bayes_factor[cls], 3), # ضریب بیزین  
classified, # طبقه‌بندی  
confidence # سطح اعتماد  
)
```

# 8. □ ایجاد رستر احتمالات (Probability Raster)

```
prob_raster = None
```

```
if create_probability_raster:
```

```
    print("\n8. □ ایجاد رستر احتمالات...")
```

```
    prob_output_name = f"Bayesian_Prob_{output_name}"
```

```
    prob_raster_path = os.path.join(output_gdb, prob_output_name)
```

```
    # پاکسازی رستر قبلی
```

```
    if arcpy.Exists(prob_raster_path):
```

```
        arcpy.Delete_management(prob_raster_path)
```

```
    # ساخت رستر احتمالات با تابع Con
```

```
    prob_raster = None
```

```
    for cls in unique_classes:
```

```
        posterior_val = posterior_mineral[cls]
```

```
        # برای جلوگیری از صفر (مقادیر بسیار کوچک)
```

```
        if posterior_val == 0:
```

```
            posterior_val = 0.001
```

```
        # اگر اولین کلاس است، رستر جدید ایجاد کن
```

```
        if prob_raster is None:
```

```

prob_raster = arcpy.sa.Con(raster == cls, posterior_val, 0)

else:

    # در غیر این صورت، به رستر موجود اضافه کن

    prob_raster = prob_raster + arcpy.sa.Con(raster == cls, posterior_val, 0)

# برای سلول‌هایی که در هیچ کلاسی نیستند، مقدار ۰.۵ قرار بده

prob_raster = arcpy.sa.Con(prob_raster == 0, 0.5, prob_raster)

# ذخیره رستر

prob_raster.save(prob_raster_path)

print(f" □ رستر احتمالات: {prob_output_name}")

else:

    prob_output_name = None

# 9. □ ایجاد رستر طبقه‌بندی (Classification Raster)

class_raster = None

if create_classification_raster:

    print("\n9. □ ایجاد رستر طبقه‌بندی...")

    class_output_name = f"Bayesian_Class_{output_name}"

    class_raster_path = os.path.join(output_gdb, class_output_name)

    # پاکسازی رستر قبلی

    if arcpy.Exists(class_raster_path):

        arcpy.Delete_management(class_raster_path)

    # ساخت رستر طبقه‌بندی

    class_raster = None

```



```

for cls in unique_classes:

    # □ سیستم کدگذاری:

    # کد اصلی کلاس (کمتر از ۱۰۰۰): >= threshold: اگر احتمال معدنی بودن -
    # کد اصلی + ۱۰۰۰ (بیشتر یا مساوی ۱۰۰۰): < threshold: اگر احتمال معدنی بودن -

    if posterior_mineral[cls] >= threshold:

        new_value = cls # کلاس معدنی - کد اصلی

    else:

        new_value = cls + 1000 # کلاس غیرمعدنی - کد اصلی + ۱۰۰۰

    # ایجاد یا به‌روزرسانی رستر

    if class_raster is None:

        class_raster = arcpy.sa.Con(raster == cls, new_value, 0)

    else:

        class_raster = class_raster + arcpy.sa.Con(raster == cls, new_value, 0)

    # برای سلول‌هایی که در هیچ کلاسی نیستند، ۰ قرار بده

    class_raster = arcpy.sa.Con(arcpy.sa.IsNull(class_raster), 0, class_raster)

    # ذخیره رستر

    class_raster.save(class_raster_path)

    print(f" □ رستر طبقه‌بندی: {class_output_name}")

    # □ اعمال رمپ رنگی به رستر (اختیاری)

    try:

        print(" □ اعمال رمپ رنگی به رستر طبقه‌بندی")

```

```
# پاکسازی لایه قبلی
```

```
class_raster_lyr = class_raster_path + ".lyr"
```

```
if arcpy.Exists(class_raster_lyr):
```

```
    arcpy.Delete_management(class_raster_lyr)
```

```
except Exception as e:
```

```
    print(f" □ در اعمال رمپ رنگی خطایی رخ داد: {str(e)}")
```

```
else:
```

```
    class_output_name = None
```

```
# 10. □ محاسبه مساحت معدنی
```

```
mineral_area = 0
```

```
mineral_percentage = 0
```

```
try:
```

```
    if class_raster is not None:
```

```
        cell_size = raster.meanCellWidth
```

```
        cell_area = cell_size * cell_size
```

```
# استخراج مناطق معدنی (کدهای کمتر از ۱۰۰۰)
```

```
mineral_raster = arcpy.sa.Con(class_raster < 1000, 1, 0)
```

```
# شمارش سلول‌های معدنی
```

```
mineral_cells = arcpy.GetRasterProperties_management(mineral_raster, "SUM")
```

```
mineral_cells_count = float(mineral_cells.getOutput(0))
```

```
# شمارش کل سلول‌ها
```

```

total_cells = arcpy.GetRasterProperties_management(class_raster, "COUNT")
total_cells_count = float(total_cells.getOutput(0))

# محاسبه مساحت به کیلومتر مربع

mineral_area = mineral_cells_count * cell_area / 1000000 # کیلومتر مربع

mineral_percentage = (mineral_cells_count / total_cells_count * 100) if total_cells_count > 0
else 0

print(f"    □ مساحت معدنی: {mineral_area:.2f} کیلومتر مربع ({mineral_percentage:.1f}%)")

except Exception as e:
    print(f"    □ محاسبه مساحت با مشکل مواجه شد: {str(e)}")

# 11. □ پاکسازی فایل‌های موقت

if clean_temp_files:
    print("\n10. □ پاکسازی فایل‌های موقت...")

    for temp in [temp_mineral, temp_non_mineral]:
        if arcpy.Exists(temp):
            try:
                arcpy.Delete_management(temp)
            except:
                pass

# 12. □ نمایش نتایج تحلیل

end_time = time.time()

processing_time = end_time - start_time

```

```

print(f"\n□ تحلیل ارتفاع {height} موفقیت انجام شد")

print(f"□ ثانیه {processing_time:.2f}: زمان پردازش")

# نمایش خلاصه نتایج

print(f"\n□ نتایج بیزین برای ارتفاع {height}:")

print("-" * 120)

print(f"{'\>:'} کلاس {'P(C|M)':<10} {'P(C|NM)':<10} {'P(M|C)':<12} {'کد' {'\>:'} طبقه {'\>:'} ضرب {'\>:'} رستری {'\>:'})

print("-" * 120)

# مرتب‌سازی کلاس‌ها بر اساس احتمال معدنی بودن (نزولی)

sorted_classes = sorted(unique_classes, key=lambda x: posterior_mineral[x], reverse=True)

# نمایش ۱۵ کلاس اول

for cls in sorted_classes[:15]:

    classified = "معدنی" if posterior_mineral[cls] >= threshold else "غیرمعدنی"

    raster_code = cls if posterior_mineral[cls] >= threshold else cls + 1000

    print(f"{cls:<8} {p_class_given_mineral[cls]:<10.4f} {p_class_given_non_mineral[cls]:<10.4f} "
          f"{posterior_mineral[cls]:<12.4f} {bayes_factor[cls]:<10.3f} "
          f"{classified:<10} {raster_code:<12}")

# نمایش مساحت معدنی

if mineral_area > 0:

    print(f"\n□ مساحت معدنی کل: {mineral_area:.2f} کیلومتر مربع ({mineral_percentage:.1f}%)")

# نمایش فایل‌های خروجی

```

```
print(f"\n□ رسترهای ایجاد شده:")
```

```
if prob_output_name:
```

```
    print(f"  1. {prob_output_name} - (۰ تا ۱) رستر احتمالات")
```

```
if class_output_name:
```

```
    print(f"  2. {class_output_name} - (کدهای معنادار) رستر طبقه‌بندی")
```

```
# □ بازگشت نتایج
```

```
return {
```

```
    'success': True,
```

```
    'height': height,
```

```
    'unique_classes': unique_classes,
```

```
    'mineral_classes': mineral_classes,
```

```
    'non_mineral_classes': non_mineral_classes,
```

```
    'mineral_area': mineral_area,
```

```
    'mineral_percentage': mineral_percentage,
```

```
    'top_class': sorted_classes[0] if sorted_classes else None,
```

```
    'top_posterior': posterior_mineral[sorted_classes[0]] if sorted_classes else 0,
```

```
    'table_name': output_table_name,
```

```
    'prob_raster': prob_output_name,
```

```
    'class_raster': class_output_name
```

```
}
```

```
except Exception as e:
```

```
    print(f"□ خطا در تحلیل ارتفاع {height}: {str(e)}")
```

```
import traceback
```

```
traceback.print_exc()
```

```
return {'success': False, 'height': height}
```

```
#
=====
=

# □ بخش ۳: اجرای اصلی برنامه

#
=====
=

#

# □ این تابع برنامه اصلی را اجرا می‌کند و برای همه ارتفاع‌ها تحلیل را انجام می‌دهد.

# □ مراحل اجرا:

# ۱. نمایش تنظیمات

# ۲. جستجوی رسترها

# ۳. تطبیق ارتفاع‌ها

# ۴. اجرای تحلیل برای هر ارتفاع

# ۵. ایجاد گزارش نهایی

#
=====
=

def main():

    """تابع اصلی اجرای تحلیل برای همه ارتفاع‌ها"""

    print("شروع تحلیل بیزین برای همه ارتفاع‌ها □")

    print("=" * 80)

    # □ نمایش تنظیمات برنامه

    print("تنظیمات ورودی □")

    print(f"مسیر رسترهای ورودی {input_gdb}")
```

```

print(f" الگوی رسترهای ورودی: {input_raster_pattern}")

print(f" مسیر نقاط نمونه: {sample_gdb}")

print(f" مسیر خروجی: {output_gdb}")

print(f" تعداد ارتفاع ها: {len(heights)}")

print(f" آستانه طبقه‌بندی: {threshold}")

print()

# □ لیست کردن رسترهای موجود

arcpy.env.workspace = input_gdb

all_rasters = arcpy.ListRasters(input_raster_pattern)

arcpy.env.workspace = os.path.dirname(output_gdb)


print(f" '{input_raster_pattern}': {len(all_rasters)}")


if all_rasters:

    print("رسترهای یافت شده:")

    for ras in all_rasters[:10]: # نمایش ۱۰ رستر اول

        print(f" - {ras}")

    if len(all_rasters) > 10:

        print(f"رستر دیگر {len(all_rasters) - 10} و ...")

    else:

        print("□ هیچ رستری یافت نشد")

    return


# □ ایجاد لیست پردازش

processing_list = []

```

```
for height in heights:
```

```
    height_str = str(height)
```

```
    height_file = str(height).replace('.', '_')
```

```
    # نام فایل‌های نقاط
```

```
    mineral_name = f'{mineral_point_prefix}{height_file}{mineral_point_suffix}'
```

```
    non_mineral_name = f'{non_mineral_prefix}{height_file}{non_mineral_suffix}'
```

```
    # مسیرها
```

```
    mineral_path = os.path.join(sample_gdb, mineral_name)
```

```
    non_mineral_path = os.path.join(sample_gdb, non_mineral_name)
```

```
    # بررسی وجود فایل‌های نقاط
```

```
    mineral_exists = arcpy.Exists(mineral_path)
```

```
    non_mineral_exists = arcpy.Exists(non_mineral_path)
```

```
    if not mineral_exists:
```

```
        print(f"□ ارتفاع {height}: نقاط معدنی ({mineral_name}) وجود ندارند")
```

```
        continue
```

```
    if not non_mineral_exists:
```

```
        print(f"□ ارتفاع {height}: نقاط غیرمعدنی ({non_mineral_name}) وجود ندارند")
```

```
        continue
```

```
    # □ یافتن رستر متناظر با ارتفاع
```

```
    found_raster = None
```



# روش ۱: جستجوی مستقیم با ارتفاع در نام فایل

```
height_patterns = [height_file, height_str.replace('.', '_'), f"*{height_str}"]
```

```
for pattern in height_patterns:
```

```
    for ras in all_rasters:
```

```
        if pattern in ras:
```

```
            found_raster = ras
```

```
            break
```

```
if found_raster:
```

```
    break
```

# روش ۲: جستجو با ارتفاع به صورت عددی

```
if not found_raster:
```

```
    for ras in all_rasters:
```

```
        # استخراج اعداد از نام فایل
```

```
        numbers = re.findall(r'\d+\.\d*', ras)
```

```
        for num in numbers:
```

```
            try:
```

```
                if abs(float(num) - height) < 0.1: # اختلاف کمتر از ۰,۱
```

```
                    found_raster = ras
```

```
                    break
```

```
            except:
```

```
                pass
```

```
if found_raster:
```

```
    break
```

# اگر رستر یافت شد، به لیست پردازش اضافه کن

```

if found_raster:

    raster_path = os.path.join(input_gdb, found_raster)

    if arcpy.Exists(raster_path):

        processing_list.append({

            'height': height_str,

            'height_file': height_file,

            'raster_name': found_raster,

            'raster_path': raster_path,

            'mineral_path': mineral_path,

            'non_mineral_path': non_mineral_path,

            'mineral_name': mineral_name,

            'non_mineral_name': non_mineral_name

        })

        print(f"□ ارتفاع {height}: رستر '{found_raster}' یافت شد")

    else:

        print(f"□ ارتفاع {height}: رستر یافت شده وجود ندارد")

    else:

        print(f"□ ارتفاع {height}: رستر متناظر یافت نشد")

print(f"\n□ {len(processing_list)} ارتفاع برای پردازش آماده هستند")

# □ بررسی وجود ارتفاع برای پردازش

if not processing_list:

    print("□ هیچ ارتفاعی برای پردازش وجود ندارد")

    return

# □ اجرای پردازش برای هر ارتفاع

```

```

results = []

success_count = 0

failed_count = 0

for i, item in enumerate(processing_list, 1):

    print(f"\n{'='*60}")

    print(f"□ پردازش {i} از {len(processing_list)} - ارتفاع {item['height']}")

    print(f"رستر: {item['raster_name']}")

    print(f"نقاط معدنی: {item['mineral_name']}")

    print(f"نقاط غیرمعدنی: {item['non_mineral_name']}")

    print(f"{'='*60}")

    # فراخوانی تابع تحلیل بیزین

    result = simple_bayesian_analysis(

        raster_path=item['raster_path'],

        mineral_fc=item['mineral_path'],

        non_mineral_fc=item['non_mineral_path'],

        output_name=item['height_file'],

        height=item['height']

    )

    results.append(result)

    if result['success']:

        success_count += 1

    else:

        failed_count += 1

```

```

# □ گزارش نهایی

print(f"\n{'='*80}")

print("□ گزارش نهایی")

print(f"{'='*80}")


print(f"\n□ نتایج:")

print(f"تعداد کل: {len(processing_list)}")

print(f"موفق: {success_count}")

print(f"ناموفق: {failed_count}")


# □ نمایش جزئیات خروجی‌ها

if success_count > 0:

    print(f"\n□ خروجی‌ها در {output_gdb}:")


    successful = [r for r in results if r['success']]

    for result in successful:

        print(f"\nارتفاع {result['height']}:")

        print(f"کل کلاس‌ها: {result.get('unique_classes', [])}")

        print(f"کلاس‌های معدنی: {result.get('mineral_classes', [])}")

        print(f"کلاس‌های غیرمعدنی: {[c+1000 for c in result.get('non_mineral_classes', [])]}")

        if result.get('top_class'):

            print(f"بهترین کلاس: {result['top_class']} (P={result['top_posterior']:.3f})")

        if result.get('mineral_area', 0) > 0:

            print(f"مساحت معدنی: {result['mineral_area']:.2f} km² ( {result['mineral_percentage']:.1f}% )")

        print(f"جدول نتایج: {result.get('table_name')}")

        if result.get('prob_raster'):

```

```

print(f" رستر احتمالات: {result.get('prob_raster')}")

if result.get('class_raster'):

    print(f" رستر طبقه‌بندی: {result.get('class_raster')}")

# □ ایجاد گزارش متنی

if create_report:

    report_dir = os.path.dirname(output_gdb)

    report_path = os.path.join(report_dir,
f"Bayesian_Analysis_Report_{time.strftime('%Y%m%d_%H%M%S')}.txt")

    try:

        with open(report_path, 'w', encoding='utf-8') as f:

            f.write("="*70 + "\n")

            f.write("گزارش تحلیل بیزین\n")

            f.write("="*70 + "\n\n")

            f.write(f"تاریخ تولید: {time.strftime('%Y-%m-%d %H:%M:%S')}\n")

            f.write(f"تعداد کل ارتفاع‌ها: {len(processing_list)}\n")

            f.write(f"تعداد موفق: {success_count}\n")

            f.write(f"تعداد ناموفق: {failed_count}\n\n")

            f.write("تنظیمات:\n")

            f.write(f"مسیر رسترهای ورودی: {input_gdb}\n")

            f.write(f"الگوی رسترهای ورودی: {input_raster_pattern}\n")

            f.write(f"مسیر نقاط نمونه: {sample_gdb}\n")

            f.write(f"مسیر خروجی: {output_gdb}\n")

```

```

f.write(f"- آستانه طبقه‌بندی: {threshold}\n\n")

f.write("\nسیستم کدگذاری رستر خروجی")

f.write("\nکد ۰: مناطق نامشخص/بدون داده")

f.write(f"- {threshold} (P >= {threshold})\n")

f.write(f"- {threshold} (P < {threshold})\n\n")

f.write("\nارتفاع‌های موفق")

for result in successful:

    f.write(f"- ارتفاع {result['height']}: ")

    if result.get('top_class'):

        f.write(f"- {result['top_class']} (احتمال {result['top_posterior']:.3f})")

        if result.get('mineral_area', 0) > 0:

            f.write(f"- مساحت {result['mineral_area']:.2f} km²")

            f.write(f"- {len(result.get('mineral_classes', []))} کلاس معدنی")

        f.write("\n")

f.write("\n" + "="*70 + "\n")

f.write("\nنکات مهم")

f.write("\n1. مقادیر احتمال معدنی بودن را نشان می‌دهند (Bayesian_Prob_*) رسترهای احتمالات")

f.write("\n2. کلاس‌های معدنی و غیرمعدنی را نشان می‌دهند (Bayesian_Class_*) رسترهای طبقه‌بندی")

f.write("\n3. استفاده کنید 'Classified'، از رمپ رنگی ArcGIS برای مشاهده نتایج در")

f.write("\n4. کلاس‌های معدنی (کد > ۱۰۰۰) با رنگ سبز و غیرمعدنی (کد <= ۱۰۰۰) با رنگ قرمز نمایش داده می‌شوند.\n")

f.write("\n" + "="*70 + "\n")

```

```
print(f"\n□ گزارش ایجاد شد: {report_path}")
```

```
except Exception as e:
```

```
print(f"□ خطا در ایجاد گزارش: {str(e)}")
```

```
print(f"\n□ تحلیل بیزین کامل شد")
```

```
print(f"□ نتایج در {output_gdb}")
```

```
print(f"\n□ نکات مهم:")
```

```
print(f" - مقادیر احتمال معدنی بودن (۰ تا ۱): (Bayesian_Prob_*) رسترهای احتمالات -")
```

```
print(f" - کلاس‌های معدنی و غیرمعدنی: (Bayesian_Class_*) رسترهای طبقه‌بندی -")
```

```
print(f" - کلاس‌های معدنی: کد اصلی کلاس (کمتر از ۱۰۰۰) -")
```

```
print(f" - کلاس‌های غیرمعدنی: کد اصلی + ۱۰۰۰ (بیشتر یا مساوی ۱۰۰۰) -")
```

```
print(f" - مناطق نامشخص: کد ۰ -")
```

```
#
```

```
=
```

```
# □ بخش ۴: نقطه شروع برنامه
```

```
#
```

```
=
```

```
#
```

```
# □ فراخوانی می‌شود main اگر این فایل مستقیماً اجرا شود، تابع.
```

```
# □ کنید import برای استفاده در اسکریپت دیگر، می‌توانید توابع را
```

```
#
```

```
=
```

```
if __name__ == "__main__":
```

```
main()
```

ترکیب ۷ پارامتر بیزین

```
import arcpy
import os
import re
from arcpy.sa import *
#
=====
#
# ArcGIS تنظیمات اولیه و محیط :بخش ۱ □
#
=====
#
# ها و تعریف مسیرهاست سازی اکستنشن، فعال ArcGIS این بخش شامل تنظیمات پایه □
# پارامترهای اصلی در این قسمت قابل تنظیم هستند □
#
=====
#
# های رستری برای تحلیل Spatial Analyst سازی اکستنشن فعال
arcpy.CheckOutExtension("Spatial")
# (مهم برای اجراهای مکرر)های خروجی اجازه رونویسی فایل
arcpy.env.overwriteOutput = True
# تعریف مسیر اصلی پروژه □
main_path = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal"
# خروجی Geodatabase نام و مسیر □
output_gdb_name = "Bayesian.gdb"
```



```

output_gdb_path = os.path.join(main_path, output_gdb_name)

# □ ایجاد Geodatabase وجود در صورت عدم وجود
if not arcpy.Exists(output_gdb_path):
    arcpy.CreateFileGDB_management(main_path, output_gdb_name)
    print(f"□ Geodatabase {output_gdb_name} ایجاد شد.")
else:
    print(f"■ □ Geodatabase {output_gdb_name} از قبل موجود است.")

#
=====
=

# □ ۲ ها و تنظیمات پردازش تعریف ورودی: بخش
#
=====
=

#
# □ های ورودی و تعریف ترازهای ارتفاعی است Geodatabase این بخش شامل لیست
# □ ها بسته به نیاز پروژه قابل ویرایش هستند این لیست
#
=====
=

# □ های ورودی حاوی رسترهای احتمالات بی‌زین Geodatabase لیست
# برای ترازهای مختلف است *Bayesian_Prob_ حاوی رسترهای GDB هر
input_gdbs = [
    "EuclideanBuffers.gdb", # رسترهای حاصل از تحلیل Euclidean Buffers
    "LineDensity.gdb", # رسترهای تراکم خطی
    "Fault_Intersections.gdb", # های گسلی رسترهای تقاطع
    "Intersections_Density.gdb", # هارسترهای تراکم تقاطع
    "Alteration_Rasters.gdb", # رسترهای دگرسانی

```

```

"Litho_Rasters.gdb", # رسترهای لیتولوژی
"Voronoi_Output.gdb" # Voronoi رسترهای حاصل از تحلیل
]

# تولید لیست ترازهای ارتفاعی مورد بررسی □
# فرمول:  $start + n * step \leq end$ 

elevation_levels = []

start = 875 # (متر) ارتفاع شروع

end = 1125.1 # 1125 برای شامل شدن + 0.1 (متر) ارتفاع پایان

step = 12.5 # (متر) گام افزایش ارتفاع

current = start

while current <= end:

    elevation_levels.append(current)

    current += step

print(f"□ تعداد ترازهای ارتفاعی: {len(elevation_levels)}")

print(f"□ محدوده ارتفاعی: {start} تا {end-step} متر")

print(f"□ گام ارتفاعی: {step} متر")

#
=====
=
# □ توابع کمکی: بخش ۳
#
=====
=
#
# □ شونداين بخش شامل توابعی است که در پردازش اصلی استفاده می
# □ دهد. تابع یک وظیفه خاص را انجام می

```

```

#
=====

=

def extract_elevation_from_name(raster_name):
    """
    استخراج مقدار ارتفاع از نام رستر

    □ کنداین تابع ارتفاع را از نام رستر استخراج می
    □ (رشته)نام رستر :ورودی
    □ در صورت عدم یافتن None یا (اعشاری)مقدار ارتفاع :خروجی
    □ الگوهای شناسایی:

    1. underscore: 912_5 یا 875_5 → 912.5
    2. های اعشاری فرمت: 912.5 یا 912.5 → 912.5
    3. های عددی ساده فرمت: 912 یا 912 → 912.0

    """
    # □ الگوهای مختلف برای شناسایی ارتفاع در نام رستر
    patterns = [
        r'o?(\d+)_(\d+)', # □ الگو ۱: 912_5 یا 875_5 هایی مثل برای فرمت:
        r'o?(\d+\. \d+)', # □ الگو ۲: های اعشاری برای فرمت:
        r'o?(\d+)' # □ الگو ۳: های عددی ساده برای فرمت:
    ]
    for pattern in patterns:
        match = re.search(pattern, raster_name)
        if match:
            if len(match.groups()) == 2:
                # □ underscore: های با برای فرمت
                return float(f'{match.group(1)}.{match.group(2)}')
            else:

```

# □ تبدیل کن float مستقیماً به: های عددی برای فرمت

```
return float(match.group(1))
```

# □ اگر هیچ الگویی مطابقت نداشت

```
return None
```

#

```
=====
```

=

# □ پردازش اصلی برای هر تراز ارتفاعی: بخش ۴

#

```
=====
```

=

#

# □ این بخش هسته اصلی برنامه است که برای هر تراز ارتفاعی:

# کندآوری می‌های ورودی جمع GDB رسترهای مربوطه را از تمام ۱.

# کندرسترها را با هم جمع می ۲.

# کندرستر نهایی را ذخیره و آنالیز می ۳.

#

# □ elevation\_levels برای هر تراز ارتفاعی در: حلقه اصلی

#

```
=====
```

=

# □ شمارنده برای گزارش پیشرفت

```
total_elevations = len(elevation_levels)
```

```
processed_count = 0
```

```
for elevation in elevation_levels:
```

```
    processed_count += 1
```

```
    print(f'\n{'='*60}')
```

```
    print(f'□ پردازش تراز ارتفاعی {elevation} ({processed_count} از {total_elevations})')
```

```
    print(f'{'='*60}')
```

```

# □ لیست برای ذخیره مسیرهای کامل رسترهای این تراز
elevation_rasters = []

# □ های ورودی Geodatabase جستجو در تمام
for gdb_name in input_gdbs:
    gdb_path = os.path.join(main_path, gdb_name)

    # □ GDB بررسی وجود
    if arcpy.Exists(gdb_path):
        try:
            # □ فعلی برای جستجو GDB به workspace تنظیم
            arcpy.env.workspace = gdb_path

            # □ Bayesian_Prob_*) (جستجوی رسترهای احتمالات بیزین
            rasters = arcpy.ListRasters("Bayesian_Prob_*")

            # □ بررسی هر رستر یافت شده
            for raster in rasters:
                # □ استخراج ارتفاع از نام رستر
                raster_elevation = extract_elevation_from_name(raster)

                # □ (۰.۱ با خطای ۰) اگر ارتفاع استخراج شد و با تراز فعلی مطابقت دارد
                if raster_elevation is not None and abs(raster_elevation - elevation) < 0.01:
                    # □ ایجاد مسیر کامل رستر
                    raster_full_path = os.path.join(gdb_path, raster)

                    # □ اضافه کردن به لیست
                    elevation_rasters.append(raster_full_path)

                    print(f"□ اضافه شد '{gdb_name}' از '{raster}' رستر")

            except Exception as e:
                print(f"□ خطا در پردازش {gdb_name}: {e}")

        else:

```

```

print(f" □ GDB '{gdb_name}' یافت نشد.")

# □ بررسی اینکه آیا رستری برای این تراز یافت شده است
if not elevation_rasters:
    print(f" □ هیچ رستری برای تراز {elevation} یافت نشد.")
    continue

print(f" □ تعداد رسترهای یافت شده: {len(elevation_rasters)}")

# □ جمع پیکسلی رسترها
print(f" □ {len(elevation_rasters)} رستر...")

try:
    # 1. □ بارگذاری اولین رستر به عنوان پایه
    print(f" □ بارگذاری اولین رستر: {os.path.basename(elevation_rasters[0])}")
    sum_raster = Raster(elevation_rasters[0])

    # 2. □ جمع کردن بقیه رسترها
    for i in range(1, len(elevation_rasters)):
        print(f" □ {i+1}: اضافه کردن رستر {os.path.basename(elevation_rasters[i])}")
        sum_raster = sum_raster + Raster(elevation_rasters[i])

    # 3. □ گذاری و ذخیره رستر نهایی نام
    # underscore تبدیل نقطه به نام فایل (مثلاً 912.5 → 912_5)
    output_name = f"Bayesian_Sum_{str(elevation).replace('.', '_')}"
    output_path = os.path.join(output_gdb_path, output_name)
    print(f" □ ذخیره رستر نهایی: {output_name}")
    sum_raster.save(output_path)

    print(f" □ ' {output_name}' رستر با موفقیت ذخیره شد.")

    # 4. □ نمایش آمار رستر نهایی
    print(f" □ محاسبه آمار رستر...")

```

```

# □ محاسبه حداقل، حداکثر و میانگین
min_val = arcpy.GetRasterProperties_management(sum_raster, "MINIMUM")
max_val = arcpy.GetRasterProperties_management(sum_raster, "MAXIMUM")
mean_val = arcpy.GetRasterProperties_management(sum_raster, "MEAN")

# □ float ها به تبدیل خروجی
min_val = float(min_val.getOutput(0))
max_val = float(max_val.getOutput(0))
mean_val = float(mean_val.getOutput(0))

# □ نمایش آمار
print(f" □ محدوده مقادیر: {min_val:.4f} تا {max_val:.4f}")
print(f" □ میانگین: {mean_val:.4f}")

# 5. □ ArcGIS و آمار برای نمایش بهتر در (Pyramids) ها ایجاد هرم
print(f" □ ...ها و آمار ایجاد هرم")
arcpy.BuildPyramidsandStatistics_management(output_path)
print(f" □ ...ها و آمار ایجاد شدندهرم")

except Exception as e:
print(f" □ {e}: خطا در پردازش تراز")

#
=====
=

# □ بخش ۵: بازگردانی تنظیمات و گزارش نهایی
#
=====
=

#
# □ :شود و شامل این بخش پس از اتمام پردازش اجرا می
# ۱. به مسیر اصلی workspace بازگرداندن

```

```

# نمایش گزارش نهایی ۲.

# های ایجاد شده کردن لایه لیست ۳.

#
=====
=

# □ به مسیر اصلی workspace بازگرداندن
arcpy.env.workspace = main_path

# □ نمایش گزارش نهایی
print("\n" + "="*60)
print("□ پردازش کامل شد!")
print(f"□ ذخیره شدند '{output_gdb_path}' ها در خروجی")
print("="*60)

# □ خروجی Geodatabase های ایجاد شده در کردن لایه لیست
print("\n□ Bayesian.gdb: های ایجاد شده در لایه")

# □ کردن خروجی برای لیست GDB به workspace تنظیم
arcpy.env.workspace = output_gdb_path

# □ Bayesian_Sum_* با الگوی (جستجوی رسترهای ایجاد شده)
output_rasters = arcpy.ListRasters("Bayesian_Sum_*")
if output_rasters:
    print(f"□ های ایجاد شده تعداد لایه: {len(output_rasters)}")
    print("-" * 50)
    for raster in output_rasters:
        print(f"□ {raster}")

# □ نمایش آمار هر رستر
try:
    # □ بارگذاری رستر برای خواندن آمار
    raster_obj = Raster(raster)

```



```

# □ محاسبه آمار
min_val = arcpy.GetRasterProperties_management(raster_obj, "MINIMUM")
max_val = arcpy.GetRasterProperties_management(raster_obj, "MAXIMUM")
mean_val = arcpy.GetRasterProperties_management(raster_obj, "MEAN")

# □ تبدیل به float
min_val = float(min_val.getOutput(0))
max_val = float(max_val.getOutput(0))
mean_val = float(mean_val.getOutput(0))

# □ نمایش آمار
print(f" □ محدوده: {min_val:.4f} تا {max_val:.4f}")

print(f" □ میانگین: {mean_val:.4f}")

print()

except Exception as e:
print(f" □ خطا در خواندن آمار رستر: {e}")

print()

else:
print(" □ ای ایجاد نشده هیچ لایه.")

#
=====
=

# □ پاکسازی و خاتمه: بخش ۶

#
=====
=

#

# □ کندرا آزاد می ArcGIS این بخش منابع.

# □ کنید CheckIn ها را، آن همیشه پس از اتمام کار با اکستنشن: مهم.

```

```

#
=====
=
# □ آژادسازی اکستنشن Spatial Analyst
arcpy.CheckInExtension("Spatial")
print("□ آژاد شد Spatial Analyst اکستنشن")
print("\n" + "="*60)
print("□ برنامه با موفقیت به پایان رسید")
print("="*60)
#
=====
=
# □ نکات مهم و راهنمایی: بخش ۷
#
=====
=
#
#
# □ نکات مهم برای استفاده از این اسکریپت:
#
# 1. □ هاساختار پوشه:
# - موجود باشند (main_path) های ورودی در مسیر اصلی GDB مطمئن شوید تمام
# - باشد 'Bayesian_Prob_*' باید حاوی رسترهایی با الگوی GDB هر
#
# 2. □ گذاری رسترها نام:
# - (Bayesian_Prob_912_5 مثلاً) رسترها باید در نام خود ارتفاع را داشته باشند
# - 912_5, 912.5, o912.5, o912_5: های پشتیبانی شده فرمت
#
# 3. □ ترازهای ارتفاعی:

```

```

# شود ۵ متر پردازش می. فرض از ۸۷۵ تا ۱۱۲۵ متر با گام ۱۲ به صورت پیش - #
# را ویرایش کنید step و end, start برای تغییر محدوده، متغیرهای - #
#
# 4. □ :سازي پهنه
# بر باشد برای تعداد زیاد رسترها، ممکن است پردازش زمان - #
# استفاده شده است ArcGIS تر در برای نمایش سریع (Pyramids) ها از هرم - #
#
# 5. □ :یابی عیب
# گذاری آن را بررسی کنید اگر رستری یافت نشد، نام - #
# های ورودی در مسیر صحیح قرار دارند GDB اطمینان حاصل کنید که تمام - #
# دسترسی به مسیرها را بررسی کنید - #
#
# 6. □ :تفسیر نتایج
# ها هستند مجموع احتمالات از تمام لایه 'Bayesian_Sum_' رسترهای خروجی - #
# دهنده پتانسیل معدنی بالاتر است مقادیر بالاتر نشان - #
# های ورودی دارد محدوده مقادیر بستگی به تعداد لایه - #
#
=====
=

```

انیمیشن تلفیق ها

```

import arcpy
import os
import re

```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
from PIL import Image
import traceback

#
=====
=

# □ بخش ۱: تنظیمات اولیه و مسیرها
#
=====
=

# □ مسیرهای اصلی
GDB_PATH = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Voronoi_Output.gdb"
OUTPUT_FOLDER = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Animation"
GDB_NAME = os.path.basename(GDB_PATH).replace(".gdb", "")
OUTPUT_GIF = os.path.join(OUTPUT_FOLDER, f"{GDB_NAME}_Bayesian.gif")

#
=====
=

# □ بخش ۲: تنظیمات قابل تغییر توسط کاربر
#
=====
=

# □ تنظیمات رنگ و طبقه‌بندی
WHITE_FIRST_CLASS = False
NUM_CLASSES = 5
```

```
RASTER_PREFIX = "Bayesian"
```

```
COLOR_SCHEME = "blue_red"
```

```
#
```

```
=====
```

```
=
```

```
# □ :۳ آماده‌سازی محیط و فایل‌ها □
```

```
#
```

```
=====
```

```
=
```

```
def setup_workspace_and_filter_rasters():
```

```
    """و فیلتر رسترها workspace تنظیم"""
```

```
    arcpy.env.workspace = GDB_PATH
```

```
    all_rasters = arcpy.ListRasters()
```

```
    print(f"□ Geodatabase: {GDB_NAME}")
```

```
    print(f"□ GDB: {len(all_rasters)} رسترها در □")
```

```
    if RASTER_PREFIX:
```

```
        rasters = [r for r in all_rasters if r.startswith(RASTER_PREFIX)]
```

```
        print(f"□ '{RASTER_PREFIX}' فیلتر رسترها با پیشوند □")
```

```
        print(f"□ :تعداد رسترهای فیلتر شده {len(rasters)} □")
```

```
        other_rasters = [r for r in all_rasters if not r.startswith(RASTER_PREFIX)]
```

```
        if other_rasters:
```

```
            print(f"□ :رسترهای حذف شده {' '.join(other_rasters[:5])}", end="")
```

```
            if len(other_rasters) > 5:
```

```
                print(f"□ و {len(other_rasters) - 5} مورد دیگر...")
```

```

        else:

            print()

    else:

        rasters = all_rasters

        print("هیچ پیشوندی مشخص نشده، همه رسترها پردازش می‌شوند ")

    if len(rasters) == 0:

        print("هیچ رستری با پیشوند مشخص شده یافت نشد ")

        print("را بررسی کنید RASTER_PREFIX لطفاً پیشوند ")

        exit()

    print(f"تعداد رسترهای پیدا شده: {len(rasters)}")

    return rasters

#
=====
=
# بخش ۴: مرتب‌سازی رسترها
#
=====
=

def extract_number(name):

    """استخراج عدد از نام رستر برای مرتب‌سازی"""

    matches = re.findall(r'\d+\.\d*', name)

    return float(matches[0]) if matches else 0

def sort_rasters(rasters):

    """مرتب‌سازی رسترها بر اساس عدد موجود در نام"""

```

```

rasters_sorted = sorted(rasters, key=extract_number)

print("□ رسترهای مرتب شده بر اساس ارتفاع:")

for i, r in enumerate(rasters_sorted[:10]):
    num = extract_number(r)
    print(f" {i+1:2d}. {r} → {num}")

if len(rasters_sorted) > 10:
    print(f" ... و {len(rasters_sorted) - 10} مورد دیگر")

return rasters_sorted

#
=====
=
# □ کلی Extent بخش ۵: محاسبه
#
=====
=

def calculate_overall_extent(rasters):
    """محاسبه محدوده جغرافیایی کلی همه رسترها"""
    print("\n□ کلی همه رسترها Extent محاسبه")

    extents = []
    for r in rasters:
        desc = arcpy.Describe(r)
        extents.append(desc.extent)

```

```
xmin = min(e.XMin for e in extents)
xmax = max(e.XMax for e in extents)
ymin = min(e.YMin for e in extents)
ymax = max(e.YMax for e in extents)
```

```
print(f"□ Extent محاسبه شد:")
```

```
print(f"□ X: {xmin:.1f} تا {xmax:.1f}")
```

```
print(f"□ Y: {ymin:.1f} تا {ymax:.1f}")
```

```
return xmin, xmax, ymin, ymax
```

```
#
```

```
=====
```

```
=
```

```
# □ بخش ۶: محاسبه محدوده مقادیر رستری
```

```
#
```

```
=====
```

```
=
```

```
def calculate_value_range(rasters):
```

```
    """محاسبه حداقل و حداکثر مقادیر واقعی در تمام رسترها"""
```

```
    print("\n□ محاسبه محدوده مقادیر در تمام رسترها...")
```

```
    all_valid_values = []
```

```
    for r in rasters:
```

```
        try:
```

```
            arr = arcpy.RasterToNumPyArray(r)
```

```
            # ایجاد ماسک برای مقادیر معتبر
```



```
valid_mask = (arr != 4294967295) & (arr >= 0) # مقادیر منفی NoData حذف
```

```
if np.any(valid_mask):
```

```
    all_valid_values.append(arr[valid_mask])
```

```
except Exception as e:
```

```
    print(f"خطا در خواندن رستر {r}: {e}")
```

```
if all_valid_values:
```

```
    all_valid_values = np.concatenate(all_valid_values)
```

```
    vmin, vmax = np.min(all_valid_values), np.max(all_valid_values)
```

```
    print(f"محدوده مقادیر محاسبه شد")
```

```
    print(f"حداقل: {vmin:.2f}")
```

```
    print(f"حداکثر: {vmax:.2f}")
```

```
else:
```

```
    vmin, vmax = 0, 43
```

```
    print(f"هیچ مقدار معتبری یافت نشد. استفاده از مقادیر پیش فرض")
```

```
    print(f"محدوده: {vmin} تا {vmax}")
```

```
return vmin, vmax
```

```
#
```

```
=====
```

```
=
```

```
# بخش ۷: تنظیمات طبقه‌بندی و رنگ‌ها
```

```
#
```

```
=====
```

```
=
```

```

def create_color_palettes():

    """ایجاد دیکشنری پالت‌های رنگی"""

    return {

        'blue_red': {

            3: ['#FFFFFF', '#4393c3', '#d6604d'],

            4: ['#FFFFFF', '#2166ac', '#f4a582', '#b2182b'],

            5: ['#FFFFFF', '#053061', '#4393c3', '#f4a582', '#b2182b'],

            6: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#f4a582', '#d6604d'],

            7: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#f4a582', '#d6604d'],

            8: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#f4a582', '#fddbc7', '#d6604d'],

            9: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#d1e5f0', '#fddbc7', '#f4a582',

            '#d6604d'],

            10: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#d1e5f0', '#fddbc7', '#f4a582',

            '#d6604d', '#b2182b'],

            11: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#d1e5f0', '#f7f7f7', '#fddbc7',

            '#f4a582', '#d6604d', '#b2182b'],

            12: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#d1e5f0', '#f7f7f7', '#fddbc7',

            '#f4a582', '#ef8a62', '#d6604d', '#b2182b']

        },

        'viridis': {

            3: ['#FFFFFF', '#35b779', '#440154'],

            4: ['#FFFFFF', '#35b779', '#21918c', '#440154'],

            5: ['#FFFFFF', '#6ece58', '#21918c', '#31688e', '#440154'],

            6: ['#FFFFFF', '#b5de2b', '#35b779', '#21918c', '#31688e', '#440154'],

            7: ['#FFFFFF', '#fde725', '#6ece58', '#35b779', '#21918c', '#31688e', '#440154'],

            8: ['#FFFFFF', '#fde725', '#b5de2b', '#6ece58', '#35b779', '#21918c', '#31688e', '#440154'],

            9: ['#FFFFFF', '#fde725', '#b5de2b', '#6ece58', '#35b779', '#1f9e89', '#26828e', '#31688e',

            '#440154'],

            10: ['#FFFFFF', '#fde725', '#d8e219', '#b5de2b', '#8fd744', '#6ece58', '#35b779', '#1f9e89',

            '#26828e', '#440154']

```

```

    },
    'plasma': {
        3: ['#FFFFFF', '#f0f921', '#0d0887'],
        4: ['#FFFFFF', '#fdb42f', '#9c179e', '#0d0887'],
        5: ['#FFFFFF', '#f0f921', '#ed7953', '#9c179e', '#0d0887'],
        6: ['#FFFFFF', '#f0f921', '#fdb42f', '#ed7953', '#9c179e', '#0d0887'],
        7: ['#FFFFFF', '#f0f921', '#fdb42f', '#ed7953', '#cc4778', '#9c179e', '#0d0887'],
        8: ['#FFFFFF', '#f0f921', '#fdc328', '#fdb42f', '#ed7953', '#cc4778', '#9c179e', '#0d0887'],
        9: ['#FFFFFF', '#f0f921', '#fdc328', '#fdb42f', '#fb9f3a', '#ed7953', '#cc4778', '#9c179e', '#0d0887']
    },
    'terrain': {
        3: ['#FFFFFF', '#d9f0d3', '#8c510a'],
        4: ['#FFFFFF', '#d9f0d3', '#f6e8c3', '#8c510a'],
        5: ['#FFFFFF', '#d9f0d3', '#f6e8c3', '#dfc27d', '#8c510a'],
        6: ['#FFFFFF', '#d9f0d3', '#c7eae5', '#f6e8c3', '#dfc27d', '#8c510a'],
        7: ['#FFFFFF', '#d9f0d3', '#c7eae5', '#f6e8c3', '#dfc27d', '#bf812d', '#8c510a'],
        8: ['#FFFFFF', '#d9f0d3', '#c7eae5', '#f6e8c3', '#dfc27d', '#bf812d', '#8c510a', '#543005']
    }
}

```

```
def get_color_palette(color_scheme, num_classes, white_first=False):
```

```
    """دریافت پالت رنگی بر اساس تنظیمات"""
```

```
    color_palettes = create_color_palettes()
```

```
    if color_scheme in color_palettes:
```

```
        if num_classes in color_palettes[color_scheme]:
```

```
            colors_list = color_palettes[color_scheme][num_classes]
```

```
    else:
```

```

available_classes = list(color_palettes[color_scheme].keys())

closest = min(available_classes, key=lambda x: abs(x - num_classes))

colors_list = color_palettes[color_scheme][closest]

print(f"تعریف نشده {color_scheme} برای طرح {num_classes} تعداد طبقات")

print(f"طبقه {closest}: استفاده از نزدیک‌ترین")

else:

    print(f"blue_red تعریف نشده. استفاده از طرح پیش‌فرض '{color_scheme}' طرح رنگی")

    if num_classes in color_palettes["blue_red"]:

        colors_list = color_palettes["blue_red"][num_classes]

    else:

        colors_list = color_palettes["blue_red"][5]

# مدیریت سفید بودن اولین طبقه

if not white_first and colors_list and colors_list[0] == '#FFFFFF':

    colors_list = colors_list[1:]

    if len(colors_list) < num_classes:

        colors_list.append('#67001f')

return colors_list

def setup_classification(vmin, vmax):

    """تنظیمات طبقه‌بندی و رنگ‌بندی"""

    num_classes = max(3, min(12, NUM_CLASSES))

    print(f"\nتنظیمات طبقه‌بندی:")

    print(f"Legend: {num_classes} تعداد طبقات")

    # ایجاد مرزهای طبقات

```

```

class_bounds = np.linspace(vmin, vmax, num_classes + 1)

class_bounds = np.round(class_bounds, 1)

print(f"□ مرزهای طبقات: {class_bounds}")

# دریافت پالت رنگی

colors_list = get_color_palette(COLOR_SCHEME, num_classes, WHITE_FIRST_CLASS)

print(f"□ تنظیمات رنگ:")

print(f"□ طرح رنگی: {COLOR_SCHEME}")

print(f"□ تعداد رنگ‌ها: {len(colors_list)}")

# ایجاد Colormap

cmap_custom = colors.ListedColormap(colors_list)

norm_classified = colors.BoundaryNorm(class_bounds, cmap_custom.N)

return num_classes, class_bounds, colors_list, cmap_custom, norm_classified

#
=====
=

# □ بخش ۸: ساخت فریم‌های انیمیشن

#
=====
=

def create_animation_frame(raster_name, extent_limits, class_info, frame_index, total_frames):

    """ایجاد یک فریم انیمیشن"""

    try:

        # □ دریافت اطلاعات رستر

```

```

desc = arcpy.Describe(raster_name)

# □ تبدیل رستر به آرایه NumPy
arr = arcpy.RasterToNumPyArray(raster_name)

# □ ایجاد ماسک برای سلول‌های NoData
nodata_mask = (arr == 4294967295)
display_arr = np.where(nodata_mask, np.nan, arr)

# □ ایجاد Figure
fig, (ax_map, ax_legend) = plt.subplots(
    1, 2,
    figsize=(16, 8),
    gridspec_kw={'width_ratios': [3, 1]},
    dpi=150
)

# □ بخش نقشه
ax_map.imshow(
    display_arr,
    cmap=class_info['cmap'],
    norm=class_info['norm'],
    extent=[desc.extent.XMin, desc.extent.XMax, desc.extent.YMin, desc.extent.YMax],
    origin='upper',
    interpolation='nearest'
)

ax_map.set_xlim(extent_limits['xmin'], extent_limits['xmax'])

```

```

ax_map.set_ylim(extent_limits['ymin'], extent_limits['ymax'])

ax_map.set_aspect('equal')


number = extract_number(raster_name)
ax_map.set_title(f'Level: {number} | Frame: {frame_index+1}/{total_frames}',
                 fontsize=14, fontweight='bold', pad=10)
ax_map.axis('off')


# □ بخش Legend
ax_legend.set_title('Legend', fontsize=14, fontweight='bold', pad=10)


# تنظیمات ارتفاع Legend
if class_info['num_classes'] > 8:
    legend_height = 0.07
    legend_spacing = 0.09
    start_y = 0.9
else:
    legend_height = 0.08
    legend_spacing = 0.1
    start_y = 0.85


# رسم مستطیل های Legend
for j in range(class_info['num_classes']):
    color = class_info['colors'][j]
    lower_bound = class_info['bounds'][j]
    upper_bound = class_info['bounds'][j + 1]
    label = f'{lower_bound} - {upper_bound}'

```

```

rect = plt.Rectangle(
    (0.1, start_y - j * legend_spacing),
    0.3, legend_height,
    facecolor=color,
    edgecolor='black',
    linewidth=0.5
)
ax_legend.add_patch(rect)

ax_legend.text(
    0.45, start_y - j * legend_spacing + legend_height/2,
    label,
    fontsize=10 if class_info['num_classes'] <= 8 else 9,
    va='center', ha='left'
)

# توضیحات تنظیمات
info_text = f'Classes: {class_info['num_classes']}'
if WHITE_FIRST_CLASS:
    info_text += " | First class: White"

ax_legend.text(
    0.05, 0.05, info_text,
    fontsize=9, style='italic', color='red',
    va='bottom', ha='left'
)

ax_legend.set_xlim(0, 1)

```



```

ax_legend.set_ylim(0, 1)
ax_legend.axis('off')

# ذخیره تصویر □
plt.tight_layout(pad=2.0)
fig.canvas.draw()
img_array = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)
img_array = img_array.reshape(fig.canvas.get_width_height()[::-1] + (3,))

plt.close(fig)
return Image.fromarray(img_array)

```

except Exception as e:

```

print(f"خطا در پردازش {raster_name}: {e}")
traceback.print_exc()
plt.close('all')
return None

```

#

=====

=

# انیمیشن GIF بخش ۹: ساخت فایل □

#

=====

=

def create\_gif\_animation(images, output\_path):

"""از فریم‌ها GIF ساخت فایل"""

if len(images) > 1:

```
print(f"\n□ ساخت GIF انیمیشن...")

print(f"□ تعداد فریم‌ها: {len(images)}")

print(f"□ اندازه هر فریم: {images[0].size}")
```

```
images[0].save(
    output_path,
    save_all=True,
    append_images=images[1:],
    duration=600,
    loop=0,
    optimize=True
)
```

```
return True
```

```
else:
```

```
print("\n□ برای ساخت انیمیشن وجود ندارد")

print("□ حداقل ۲ فریم برای ساخت انیمیشن نیاز است")

return False
```

```
#
```

```
=====
```

```
=
```

```
# □ بخش اصلی اجرا
```

```
#
```

```
=====
```

```
=
```

```
def main():
```

```
    """تابع اصلی اجرای اسکریپت"""
```

```
print("=" * 60)

print("□ شروع فرآیند ساخت انیمیشن از رسترهای ArcGIS")

print("=" * 60)

try:

    # 1. آماده‌سازی و فیلتر رسترها
    rasters = setup_workspace_and_filter_rasters()

    # 2. مرتب‌سازی رسترها
    rasters_sorted = sort_rasters(rasters)

    # 3. Extent کلی محاسبه
    xmin, xmax, ymin, ymax = calculate_overall_extent(rasters_sorted)
    extent_limits = {'xmin': xmin, 'xmax': xmax, 'ymin': ymin, 'ymax': ymax}

    # 4. محاسبه محدوده مقادیر
    vmin, vmax = calculate_value_range(rasters_sorted)

    # 5. تنظیمات طبقه‌بندی و رنگ
    num_classes, class_bounds, colors_list, cmap_custom, norm_classified = setup_classification(vmin,
vmax)

    class_info = {
        'num_classes': num_classes,
        'bounds': class_bounds,
        'colors': colors_list,
        'cmap': cmap_custom,
        'norm': norm_classified
```

```
}
```

```
# 6. ایجاد پوشه خروجی
```

```
os.makedirs(OUTPUT_FOLDER, exist_ok=True)
```

```
print(f"\n□ پوشه خروجی: {OUTPUT_FOLDER}")
```

```
# 7. ساخت فریم‌ها
```

```
print(f"\n□ ...شروع ساخت فریم‌های انیمیشن")
```

```
print(f"□ تعداد کل فریم‌ها: {len(rasters_sorted)}")
```

```
images = []
```

```
for i, raster in enumerate(rasters_sorted):
```

```
    print(f"□ پردازش فریم {i+1}/{len(rasters_sorted)}: {raster}")
```

```
    frame = create_animation_frame(raster, extent_limits, class_info, i, len(rasters_sorted))
```

```
    if frame:
```

```
        images.append(frame)
```

```
# 8. ساخت GIF
```

```
if create_gif_animation(images, OUTPUT_GIF):
```

```
    print(f"\n□ !انیمیشن با موفقیت ساخته شد")
```

```
    print(f"□ مسیر فایل: {OUTPUT_GIF}")
```

```
    print(f"□ تعداد فریم‌ها: {len(images)}")
```

```
    print(f"□ اندازه هر فریم: {images[0].size}")
```

```
    print(f"\n□ تنظیمات اعمال شده:")
```

```
    print(f"□ طرح رنگی: {COLOR_SCHEME}")
```

```
print(f"□ تعداد طبقات: {num_classes}")

print(f"□ اولین طبقه: {'سفید' if WHITE_FIRST_CLASS else 'رنگی'}")

print(f"□ پیشوند رسترها: '{RASTER_PREFIX if RASTER_PREFIX else 'همه'}'")

print(f"□ مدت هر فریم: ۶۰۰ms")

print(f"□ حالت تکرار: بی‌نهایت")
```

```
print("\n" + "=" * 60)

print("□ فرآیند به پایان رسید")

print("=" * 60)
```

```
except Exception as e:

    print(f"\n□ خطای غیرمنتظره: {e}")

    traceback.print_exc()

    return 1
```

```
return 0
```

```
#
```

```
=====
```

```
=
```

```
# اجرای اسکریپت
```

```
#
```

```
=====
```

```
=
```

```
if __name__ == "__main__":
```

```
    exit_code = main()
```

```
    exit(exit_code)
```

## ماشین بردار پشتیبان SVM

```
import arcpy
import os
import numpy as np
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.utils import class_weight
import re
import warnings
from datetime import datetime
warnings.filterwarnings('ignore')

#
=====
=

# □ بخش ۱: تنظیمات قابل تغییر توسط کاربر
#
=====
=

#
# □ این بخش شامل پارامترهای اصلی برنامه است که کاربر می تواند تغییر دهد.
# □ هر پارامتر با توضیحات کامل در کنار آن آمده است.
#
=====
=
```

# □ مسیرهای ورودی و خروجی

input\_gdb\_path = r"M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal" # □ مسیر اصلی پروژه

output\_gdb\_name = "SVM\_ALL.gdb" # □ نام Geodatabase خروجی

output\_gdb\_path = os.path.join(input\_gdb\_path, output\_gdb\_name) # □ مسیر کامل GDB خروجی

# □ لیست Geodatabase های حاوی رسترهای ورودی

raster\_gdbs = [

"EuclideanBuffers.gdb", # □ رسترهای بافر اقلیدسی

"LineDensity.gdb", # □ رسترهای تراکم خطی

"Fault\_Intersections.gdb", # □ رسترهای تقاطع‌های گسلی

"Intersections\_Density.gdb", # □ رسترهای تراکم تقاطع‌ها

"Alteration\_Rasters.gdb", # □ رسترهای دگرسانی

"Litho\_Rasters.gdb", # □ رسترهای لیتولوژی

"Voronoi\_Output.gdb" # □ رسترهای خروجی Voronoi

]

# □ لیست ترازهای ارتفاعی مورد تحلیل (از ۸۷۵ تا ۱۱۲۵ متر با گام ۱۲.۵ متر)

elevations = [875 + i\*12.5 for i in range(21)] # □ ۲۱ تراز ارتفاعی

# □ SVM (Support Vector Machine) پارامترهای الگوریتم

C\_value = 1.0 # □ margin و misclassification کنترل تعادل بین C: پارامتر

kernel\_type = 'rbf' # □ پیشنهادی (RBF) 'linear', 'rbf', 'poly', 'sigmoid': نوع هسته

gamma\_value = 'scale' # □ یا عدد مشخص (کنترل تأثیر نقاط) 'scale', 'auto': مقدار گاما

test\_size = 0.3 # □ سهم داده‌های تست از کل (۳۰٪ برای تست)

```
random_state = 42      # □ Seed تصادفی برای تکرارپذیری نتایج
```

```
# □ تنظیمات وزن کلاس‌ها (برای مقابله با عدم تعادل داده‌ها)
```

```
class_weights = {0: 1.0, 1: 1.5} # 0: نقاط معدنی (وزن ۱٫۵ برابر) # 1: نقاط غیرمعدنی
```

```
#
```

```
=====
```

```
=
```

```
# □ بخش ۲: توابع کمکی اصلی
```

```
#
```

```
=====
```

```
=
```

```
#
```

```
# □ این بخش شامل توابعی است که در مراحل مختلف پردازش استفاده می‌شوند.
```

```
# □ هر تابع یک وظیفه مشخص را انجام می‌دهد.
```

```
#
```

```
=====
```

```
=
```

```
def extract_elevation_from_name(name):
```

```
    """
```

```
    استخراج تراز ارتفاعی از نام فایل یا رستر
```

```
    این تابع با استفاده از الگوهای مختلف، عدد ارتفاع را از نام استخراج می‌کند.
```

```
    ورودی: نام فایل (رشته)
```

```
    خروجی: None (تراز ارتفاعی (عدد اعشاری) یا
```

```
    الگوهای شناسایی:
```



- o875, o1000, o1012\_5 (نقاط معدنی)
- n875, n1000, n1012\_5 (نقاط غیرمعدنی)
- f875, f1000, f1012\_5 (رسترهای فاصله)
- a875, a1000, a1012\_5 (رسترهای دگرسانی)
- r875, r1000, r1012\_5 (رسترهای لیتولوژی)
- \_1000\_, \_1012\_5 (اعداد درون نام)
- 1012\_5, 912\_5 (فرمت underscore)
- 1000, 875 (اعداد صحیح)

""

# □ لیست الگوهای مختلف برای شناسایی ارتفاع □

patterns = [

- r'o(\d{3,4}(?:\_5)?)', # □ o875, o1000, o1012\_5 (نقاط معدنی)
- r'n(\d{3,4}(?:\_5)?)', # □ n875, n1000, n1012\_5 (نقاط غیرمعدنی)
- r'f(\d{3,4}(?:\_5)?)', # □ f875, f1000, f1012\_5 (رسترهای فاصله)
- r'a(\d{3,4}(?:\_5)?)', # □ a875, a1000, a1012\_5 (رسترهای دگرسانی)
- r'r(\d{3,4}(?:\_5)?)', # □ r875, r1000, r1012\_5 (رسترهای لیتولوژی)
- r'\_(\d{3,4}(?:\_5)?)\_', # □ \_1000\_, \_1012\_5 (اعداد درون نام)
- r'(\d{4})\_5', # □ 1012\_5 (اعداد چهاررقمی با ۵)
- r'(\d{3})\_5', # □ 912\_5 (اعداد سه‌رقمی با ۵)
- r'(\d{4})', # □ 1000 (اعداد چهاررقمی)
- r'(\d{3})' # □ 875 (اعداد سه‌رقمی)

]

# □ جستجوی هر الگو در نام □

```

for pattern in patterns:

    match = re.search(pattern, name)

    if match:

        elev_str = match.group(1)

        # به نقطه اعشاری underscore تبدیل فرمت

        if '_5' in elev_str:

            elev_str = elev_str.replace('_5', '.5')

        try:

            return float(elev_str) # تبدیل به عدد اعشاری

        except:

            continue # در صورت خطا، الگوی بعدی را امتحان کن

return None # اگر هیچ الگویی مطابقت نداشت

```

```

def find_rasters_for_elevation(elevation):

```

```

    """

```

یافتن تمام رسترهای مربوط به یک تراز ارتفاعی خاص

□ های ورودی جستجو کرده و رسترهایی را که GDB این تابع در تمام

مربوط به تراز ارتفاعی مشخص هستند، پیدا می‌کند.

□ ورودی: تراز ارتفاعی (عدد اعشاری)

□ خروجی: لیست مسیرهای کامل رسترهای یافت شده

```

    """

```

```

rasters = [] # لیست برای ذخیره رسترهای یافت شده

```

□ ایجاد الگوهای جستجو بر اساس نوع تراز (صحیح یا اعشاری)

```
if elevation % 1 == 0: # □ (مثلاً ۱۰۰۰) عدد صحیح
```

```
    elev_str = str(int(elevation))
```

```
    search_patterns = [
```

```
        f"{elev_str}",      # □ 1000
```

```
        f"f{elev_str}_",    # □ f1000_
```

```
        f"a{elev_str}_",    # □ a1000_
```

```
        f"r{elev_str}_",    # □ r1000_
```

```
        f"_{elev_str}_",    # □ _1000_
```

```
    ]
```

```
else: # □ (مثلاً ۱۰۱۲,۵) عدد اعشاری
```

```
    elev_str = f"{int(elevation)}_5" # underscore تبدیل به فرمت
```

```
    search_patterns = [
```

```
        f"{elev_str}",      # □ 1012_5
```

```
        f"f{elev_str}_",    # □ f1012_5_
```

```
        f"a{elev_str}_",    # □ a1012_5_
```

```
        f"r{elev_str}_",    # □ r1012_5_
```

```
        f"_{elev_str}_",    # □ _1012_5_
```

```
    ]
```

```
# □ GDB جستجو در هر
```

```
for gdb_name in raster_gdbs:
```

```
    gdb_path = os.path.join(input_gdb_path, gdb_name)
```

```
# □ GDB بررسی وجود
```

```
if arcpy.Exists(gdb_path):
```

```
    arcpy.env.workspace = gdb_path # □ workspace تنظیم
```

```

# □ لیست کردن تمام رسترهای موجود

for raster in arcpy.ListRasters():

    raster_lower = raster.lower() # □ تبدیل به حروف کوچک

    # □ بررسی پسوندهای مجاز برای رسترها

    valid_suffix = any(suffix in raster_lower for suffix in
                        ['reclassify', 'raster', 'colorraster', 'azimuth'])

    if not valid_suffix:

        continue # □ اگر پسوند مجاز نبود، ادامه بده

    # □ جستجوی الگوهای ارتفاع در نام رستر

    found = False

    for pattern in search_patterns:

        if pattern.lower() in raster_lower:

            full_path = os.path.join(gdb_path, raster)

            if full_path not in rasters: # □ جلوگیری از تکراری

                rasters.append(full_path)

            found = True

            break # □ اگر پیدا شد، جستجو متوقف شود

    # □ اگر با الگو پیدا نشد، با تابع استخراج بررسی کن

    if not found:

        elev_from_name = extract_elevation_from_name(raster)

        if elev_from_name is not None and abs(elev_from_name - elevation) < 0.1:

            full_path = os.path.join(gdb_path, raster)

            if full_path not in rasters:

```

```
rasters.append(full_path)
```

```
return rasters # بازگرداندن لیست رسترها
```

```
def find_points_for_elevation(elevation, point_type='mineral'):
```

```
    """
```

برای تراز ارتفاعی مشخص (Feature Class) یافتن فایل نقاط

این تابع نقاط معدنی یا غیرمعدنی مربوط به یک تراز ارتفاعی را پیدا می‌کند.

ورودی‌ها:

- elevation: تراز ارتفاعی

- point\_type: نوع نقاط ('mineral' یا 'nonmineral')

None یا Feature Class خروجی: مسیر

```
    """
```

```
gosal_gdb = os.path.join(input_gdb_path, "Gosal.gdb") # نقاط GDB مسیر
```

```
# ایجاد الگوهای نام‌گذاری بر اساس نوع تراز
```

```
if elevation % 1 == 0: # عدد صحیح
```

```
    elev_str = str(int(elevation))
```

```
    if point_type == 'mineral':
```

```
        # الگوهای نام برای نقاط معدنی
```

```
        for pattern in [f'o{elev_str}_Points', f'o{elev_str}Points', f'o{elev_str}"]:
```

```
            test_path = os.path.join(gosal_gdb, pattern)
```

```
            if arcpy.Exists(test_path):
```

```

        return test_path

    else: # □ نقاط غیرمعدنی

        for pattern in [f'n{elev_str}', f'n{elev_str}_Points', f'n{elev_str}Points']:

            test_path = os.path.join(gosal_gdb, pattern)

            if arcpy.Exists(test_path):

                return test_path

    else: # □ ۵. عدد اعشاری

        elev_str = f'{int(elevation)}_5' # underscore تبدیل به فرمت

        if point_type == 'mineral':

            for pattern in [f'o{elev_str}_Points', f'o{elev_str}Points', f'o{elev_str}']:

                test_path = os.path.join(gosal_gdb, pattern)

                if arcpy.Exists(test_path):

                    return test_path

    else: # □ نقاط غیرمعدنی

        for pattern in [f'n{elev_str}', f'n{elev_str}_Points', f'n{elev_str}Points']:

            test_path = os.path.join(gosal_gdb, pattern)

            if arcpy.Exists(test_path):

                return test_path

    return None # □ اگر هیچ فایل پیدا نشد

def create_temp_name(base_name):

    """

    ایجاد نام موقت معتبر برای فایل‌های میانی

    ایجاد می‌کند ArcGIS این تابع نام‌های معتبر برای فایل‌های موقت □
    
```

□ ورودی: نام پایه (رشته)

□ خروجی: نام معتبر ArcGIS

"""

# □ حذف کاراکترهای غیرمجاز

```
valid_name = re.sub(r'^a-zA-Z0-9_', '_', base_name)
```

# □ اضافه کن 'temp\_' اگر نام با عدد شروع شد،

```
if valid_name[0].isdigit():
```

```
    valid_name = 'temp_' + valid_name
```

# □ حد اکثر ۳۰ کاراکتر برای) محدود کردن طول نام (ArcGIS

```
if len(valid_name) > 30:
```

```
    valid_name = valid_name[:30]
```

```
return valid_name
```

```
def get_common_extent_and_spatial_ref(rasters_list):
```

"""

محاسبه بزرگترین محدوده مشترک و سیستم مختصات از لیست رسترها

□ این تابع از بین تمام رسترهای ورودی، بزرگترین محدوده جغرافیایی

و سیستم مختصات مشترک را استخراج می کند.

□ ورودی: لیست مسیرهای رستر

□ خروجی: (Extent, SpatialReference, cell\_size) یا (None, None, None)

"""

```
if not rasters_list:
```

```

return None, None, None

# □ لیست‌های برای جمع‌آوری اطلاعات
all_extents = [] # □ لیست محدوده‌ها
spatial_refs = [] # □ لیست سیستم‌های مختصات

# □ جمع‌آوری اطلاعات از هر رستر
for raster_path in rasters_list:
    try:
        desc = arcpy.Describe(raster_path)
        all_extents.append(desc.extent)
        spatial_refs.append(desc.spatialReference)
    except:
        continue # □ در صورت خطا، ادامه بده

if not all_extents:
    return None, None, None # □ اگر هیچ اطلاعاتی نبود

# □ محاسبه بزرگترین محدوده (اجتماع تمام محدوده‌ها)
min_x = min(e.XMin for e in all_extents)
min_y = min(e.YMin for e in all_extents)
max_x = max(e.XMax for e in all_extents)
max_y = max(e.YMax for e in all_extents)

# □ آرک‌جی‌آی‌اس Extent ایجاد
common_extent = arcpy.Extent(min_x, min_y, max_x, max_y)

```



# □ سیستم مختصات از اولین رستر معتبر

```
common_spatial_ref = spatial_refs[0] if spatial_refs else None
```

# □ اندازه پیکسل از اولین رستر

```
first_desc = arcpy.Describe(rasters_list[0])
```

```
cell_size = first_desc.meanCellWidth
```

```
return common_extent, common_spatial_ref, cell_size
```

```
def align_raster_to_reference(raster_path, reference_extent, reference_spatial_ref, output_cell_size, output_path):
```

```
    """
```

همسان‌سازی رستر با محدوده، سیستم مختصات و اندازه پیکسل مرجع

این تابع یک رستر را با پارامترهای مرجع هماهنگ می‌کند تا □

همه رسترها دارای مشخصات یکسان باشند.

□ ورودی‌ها:

- raster\_path: مسیر رستر ورودی

- reference\_extent: محدوده مرجع

- reference\_spatial\_ref: سیستم مختصات مرجع

- output\_cell\_size: اندازه پیکسل خروجی

- output\_path: مسیر خروجی

□ اگر ناموفق False اگر موفق، True: خروجی

```
    """
```

```
try:
```

```

# □ تنظیم ArcGIS پارامترهای محیطی
arcpy.env.extent = reference_extent
arcpy.env.outputCoordinateSystem = reference_spatial_ref
arcpy.env.cellSize = output_cell_size
arcpy.env.snapRaster = raster_path # □ تنظیم Snap برای تراز کردن

# □ ایجاد نام موقت
temp_name = create_temp_name("temp_align")
temp_raster = os.path.join("memory", temp_name)

# □ کپی رستر اولیه
arcpy.CopyRaster_management(raster_path, temp_raster)

# □ پروجکت کردن به سیستم مختصات مرجع
aligned_raster = arcpy.ProjectRaster_management(
    temp_raster,
    output_path,
    reference_spatial_ref,
    "NEAREST", # □ روش درونیایی (نزدیک‌ترین همسایه)
    output_cell_size
)

# □ پاکسازی فایل موقت
if arcpy.Exists(temp_raster):
    arcpy.Delete_management(temp_raster)

return True # □ موفقیت

```

```
except Exception as e:
```

```
    print(f"    □ خطا در همسان سازی: {e}")
```

```
    # □ روش جایگزین ساده تر
```

```
    try:
```

```
        arcpy.Resample_management(
```

```
            raster_path,
```

```
            output_path,
```

```
            output_cell_size,
```

```
            "NEAREST"
```

```
        )
```

```
    # □ تعریف سیستم مختصات
```

```
    arcpy.DefineProjection_management(output_path, reference_spatial_ref)
```

```
    return True # □ موفقیت با روش جایگزین
```

```
except:
```

```
    return False # □ شکست کامل
```

```
def calculate_class_weights(y):
```

```
    """
```

```
    محاسبه خودکار وزن کلاس‌ها بر اساس توزیع داده‌ها
```

```
    □ این تابع وزن کلاس‌ها را بر اساس تعداد نمونه‌های هر کلاس محاسبه می‌کند.
```

```
    □ ورودی: آرایه برچسب‌ها (y)
```

```
    □ خروجی: دیکشنری وزن کلاس‌ها
```

```
    """
```

```
    unique_classes = np.unique(y)
```

```
weights = {}
```

```
for cls in unique_classes:
```

```
    if cls == 1: # □ کلاس معدنی
```

```
        weights[cls] = 1.2 # □ وزن بیشتر (۱,۲ برابر)
```

```
    else: # □ کلاس غیرمعدنی
```

```
        weights[cls] = 1.0 # □ وزن نرمال
```

```
return weights
```

```
#
```

```
=====
```

```
=
```

```
# □ بخش ۳: تابع اصلی پردازش برای هر تراز ارتفاعی
```

```
#
```

```
=====
```

```
=
```

```
#
```

```
# □ این تابع تمام مراحل پردازش را برای یک تراز ارتفاعی انجام می‌دهد.
```

```
# □ مراحل اصلی:
```

```
# ۱. یافتن رسترها و نقاط
```

```
# ۲. آماده‌سازی داده‌ها
```

```
# ۳. SVM آموزش مدل
```

```
# ۴. پیش‌بینی روی منطقه
```

```
# ۵. ایجاد رسترهای خروجی
```

```
#
```

```
=====
```

```
=
```

```

def process_elevation(elevation):
    """پردازش کامل برای یک تراز ارتفاعی خاص"""
    print(f"\n{'='*60}")
    print(f"□ : پردازش تراز ارتفاعی {elevation}")
    print(f"{'='*60}")

    # □ تنظیمات محیط ArcGIS
    arcpy.env.workspace = "memory"
    arcpy.env.overwriteOutput = True

    try:
        # □ مرحله ۱: یافتن رسترهای مربوطه
        print("1. □ جستجوی رسترهای مربوطه...")
        rasters_list = find_rasters_for_elevation(elevation)

        if len(rasters_list) == 0:
            print(f"□ ! یافت نشد {elevation} هیچ رستری برای تراز")
            return None

        print(f"□ {len(rasters_list)} رستر یافت شد")
        for i, r in enumerate(rasters_list[:5], 1):
            print(f"  {i:2d}. {os.path.basename(r)}")
        if len(rasters_list) > 5:
            print(f"  رستر دیگر {len(rasters_list)-5} و ...")

        # □ مرحله ۲: بارگذاری نقاط معدنی و غیرمعدنی

```

```
print("\n2. □ بارگذاری نقاط...")
```

```
mineral_points = find_points_for_elevation(elevation, 'mineral')
```

```
nonmineral_points = find_points_for_elevation(elevation, 'nonmineral')
```

```
if not mineral_points:
```

```
    print(f"□ یافت نشد {elevation} نقاط معدنی برای تراز")
```

```
    return None
```

```
if not nonmineral_points:
```

```
    print(f"□ یافت نشد {elevation} نقاط غیرمعدنی برای تراز")
```

```
    return None
```

```
print(f"□ نقاط معدنی: {os.path.basename(mineral_points)}")
```

```
print(f"□ نقاط غیرمعدنی: {os.path.basename(nonmineral_points)}")
```

```
# □ مرحله ۳: آماده‌سازی داده‌های آموزشی
```

```
print("\n3. □ ...آماده‌سازی داده‌های آموزشی")
```

```
# □ کپی نقاط به حافظه موقت
```

```
mineral_temp = create_temp_name(f"min_{elevation}")
```

```
nonmineral_temp = create_temp_name(f"nonmin_{elevation}")
```

```
arcpy.CopyFeatures_management(mineral_points, os.path.join("memory", mineral_temp))
```

```
arcpy.CopyFeatures_management(nonmineral_points, os.path.join("memory", nonmineral_temp))
```

```
mineral_temp_path = os.path.join("memory", mineral_temp)
```

```

nonmineral_temp_path = os.path.join("memory", nonmineral_temp)

# □ اضافه کردن فیلد کلاس به نقاط

for temp_fc, class_value in [(mineral_temp_path, 1), (nonmineral_temp_path, 0)]:
    if "Class" not in [f.name for f in arcpy.ListFields(temp_fc)]:
        arcpy.AddField_management(temp_fc, "Class", "SHORT")
        arcpy.CalculateField_management(temp_fc, "Class", str(class_value), "PYTHON3")

# □ ترکیب نقاط معدنی و غیرمعدنی

all_points_name = create_temp_name(f'all_{elevation}')
all_points_path = os.path.join("memory", all_points_name)
arcpy.Merge_management([mineral_temp_path, nonmineral_temp_path], all_points_path)

# □ شمارش نقاط و محاسبه وزن کلاس‌ها

try:
    mineral_count = int(arcpy.GetCount_management(mineral_temp_path)[0])
    nonmineral_count = int(arcpy.GetCount_management(nonmineral_temp_path)[0])
    total_count = mineral_count + nonmineral_count

    print(f'□ نقاط معدنی: {mineral_count}')
    print(f'□ نقاط غیرمعدنی: {nonmineral_count}')
    print(f'□ مجموع: {total_count}')

# □ محاسبه وزن کلاس‌ها بر اساس توزیع

actual_class_weights = calculate_class_weights([1] * mineral_count + [0] * nonmineral_count)

print(f'□ وزن کلاس‌ها: معدنی={actual_class_weights.get(1, 1.2):.1f}x,
غیرمعدنی={actual_class_weights.get(0, 1.0):.1f}x")

```

except:

```
print(" □ شمارش نقاط با خطا مواجه شد")
```

```
actual_class_weights = class_weights # □ استفاده از وزن‌های پیش‌فرض
```

```
# □ مرحله ۴: استخراج مقادیر رسترها در مکان نقاط
```

```
print("\n4. □ استخراج مقادیر رسترها...")
```

```
# □ ایجاد لیست رسترها برای استخراج
```

```
extract_list = []
```

```
for i, raster_path in enumerate(rasters_list):
```

```
    short_name = f"V{i:02d}" # □ نام کوتاه برای هر متغیر
```

```
    extract_list.append([raster_path, short_name])
```

```
try:
```

```
    arcpy.sa.ExtractMultiValuesToPoints(all_points_path, extract_list)
```

```
    print(" □ استخراج مقادیر انجام شد")
```

```
except Exception as e:
```

```
    print(f" □ خطا در استخراج مقادیر: {e}")
```

```
    return None
```

```
# □ مرحله ۵: خواندن داده‌های استخراج شده
```

```
print("5. □ خواندن داده‌ها...")
```

```
# □ لیست فیلدها (کلاس + تمام متغیرها)
```

```
field_names = ["Class"] + [f"V{i:02d}" for i in range(len(rasters_list))]
```



```

data = []

try:

    with arcpy.da.SearchCursor(all_points_path, field_names) as cursor:

        for row in cursor:

            if None not in row[1:]: # □ null بررسی عدم وجود مقادیر

                data.append(row)

if len(data) == 0:

    print(" □ هیچ داده معتبری استخراج نشد")

    return None

data_array = np.array(data, dtype=float)

print(f" □ {len(data_array)} نمونه معتبر")

except Exception as e:

    print(f" □ {e} خطا در خواندن داده‌ها")

    return None

# □ مرحله ۶: تقسیم داده‌ها به آموزش و آزمون

print("\n6. □ ...تقسیم داده‌ها")

X = data_array[:, 1:] # □ ویژگی‌ها (ستون‌های دوم به بعد)

y = data_array[:, 0] # □ برچسب‌ها (ستون اول - کلاس)

n_samples = len(X)

if n_samples < 10:

    print(f" □ ({n_samples}) نمونه‌ها بسیار کم است")

```

```

return None

# □ تقسیم تصادفی داده‌ها

try:
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state, stratify=y
    )

    print(f" □ کل نمونه‌ها: {n_samples}")
    print(f" □ آموزش: {len(X_train)}")
    print(f" □ آزمون: {len(X_test)}")

except Exception as e:
    print(f" □ خطا در تقسیم داده‌ها: {e}")
    return None

# □ مرحله ۷: استانداردسازی ویژگی‌ها

print("\n7. □ استانداردسازی...")

try:
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    print(" □ استانداردسازی انجام شد")

except Exception as e:
    print(f" □ خطا در استانداردسازی: {e}")
    return None

```

```

# □ با وزن کلاس SVM مرحله ۸: آموزش مدل
print("\n8. □ SVM (۱,۲ معدنی (x) با وزن کلاس...")

try:
    svm_model = SVC(
        C=C_value,
        kernel=kernel_type,
        gamma=gamma_value,
        random_state=random_state,
        probability=True, # □ فعال‌سازی خروجی احتمالات
        class_weight=actual_class_weights # □ استفاده از وزن کلاس
    )

    svm_model.fit(X_train_scaled, y_train)
    print("□ با وزن کلاس آموزش داده شد SVM مدل □")
except Exception as e:
    print(f"□ SVM خطا در آموزش: {e}")
    return None

# □ مرحله ۹: ارزیابی مدل آموزش‌دیده
print("\n9. □ ارزیابی مدل...")

try:
    train_score = svm_model.score(X_train_scaled, y_train)
    test_score = svm_model.score(X_test_scaled, y_test)

    print(f"□ دقت آموزش: {train_score:.4f}")
    print(f"□ دقت آزمون: {test_score:.4f}")

```

```

y_pred = svm_model.predict(X_test_scaled)

print("\n □ گزارش طبقه‌بندی (۱,۲ با وزن معدنی):")

print(classification_report(y_test, y_pred, target_names=['معدنی', 'غیرمعدنی']))

except Exception as e:

    print(f" □ خطا در ارزیابی مدل: {e}")

    test_score = 0.0

# □ خروجی Geodatabase مرحله ۱۰: آماده‌سازی □
print("\n10. □ ...آماده‌سازی خروجی □")

# □ خروجی اگر وجود ندارد GDB ایجاد □
if not arcpy.Exists(output_gdb_path):

    try:

        arcpy.CreateFileGDB_management(
            os.path.dirname(output_gdb_path),
            os.path.basename(output_gdb_path)
        )

        print(f" □ GDB ایجاد شد: {output_gdb_name}")

    except Exception as e:

        print(f" □ خطا در ایجاد GDB: {e}")

        return None

# □ مرحله ۱۱: پردازش رسترها برای پیش‌بینی روی کل منطقه □
print("11. □ ...پردازش رسترها برای پیش‌بینی □")

```

```
# □ دریافت محدوده و سیستم مختصات مشترک
```

```
common_extent, common_spatial_ref, common_cellsize =  
get_common_extent_and_spatial_ref(rasters_list)
```

```
if common_extent is None or common_spatial_ref is None:
```

```
    print(" □ محدوده یا سیستم مختصات مشترکی یافت نشد")
```

```
    return None
```

```
print(f" □ سیستم مختصات: {common_spatial_ref.name}")
```

```
print(f" □ محدوده مشترک: X[{common_extent.XMin:.1f}-{common_extent.XMax:.1f}], "
```

```
      f"Y[{common_extent.YMin:.1f}-{common_extent.YMax:.1f}]")
```

```
print(f" □ اندازه پیکسل: {common_cellsize} متر")
```

```
# □ همسان‌سازی رسترها با مشخصات مشترک
```

```
aligned_rasters = []
```

```
for i, raster_path in enumerate(rasters_list):
```

```
    print(f" □ رستر {i+1}/{len(rasters_list)}: {os.path.basename(raster_path)}")
```

```
    temp_name = create_temp_name(f"aligned_{elevation}_{i}")
```

```
    temp_path = os.path.join("memory", temp_name)
```

```
try:
```

```
    success = align_raster_to_reference(  
        raster_path,
```

```
        common_extent,
```

```
        common_spatial_ref,
```

```
        1, # □ پیکسل ۱ متر برای خروجی
```

```

temp_path
)

if success and arcpy.Exists(temp_path):
    aligned_rasters.append(temp_path)
    print(f"    □ همسان‌سازی موفق")
else:
    print(f"    □ همسان‌سازی ناموفق")

except Exception as e:
    print(f"    □ خطا: {e}")

if len(aligned_rasters) == 0:
    print(f"    □ هیچ رستر همسان‌سازی شده‌ای ایجاد نشد")
    return None

print(f"    □ {len(aligned_rasters)} رستر همسان‌سازی شدند")

# □ مرحله ۱۲: ایجاد آرایه سه‌بعدی از رسترهای همسان شده
print(f"\n12. □ ...ایجاد آرایه برای پیش‌بینی")

# □ خواندن ابعاد از اولین رستر همسان‌سازی شده
first_aligned = aligned_rasters[0]
try:
    first_array = arcpy.RasterToNumPyArray(first_aligned)
    n_rows, n_cols = first_array.shape
    print(f"    □ ابعاد رستر: {n_rows} × {n_cols}")
    print(f"    □ تعداد پیکسل‌ها: {n_rows * n_cols:,}")

```

except Exception as e:

```
print(f' □ خطا در خواندن ابعاد رستر: {e}')
```

```
return None
```

```
# □ ایجاد آرایه سه بعدی (متغیرها × سطرها × ستون‌ها)
```

```
try:
```

```
combined_array = np.zeros((len(aligned_rasters), n_rows, n_cols))
```

```
for i, raster_path in enumerate(aligned_rasters):
```

```
try:
```

```
raster_array = arcpy.RasterToNumPyArray(raster_path)
```

```
# □ بررسی تطابق ابعاد
```

```
if raster_array.shape == (n_rows, n_cols):
```

```
    combined_array[i] = raster_array
```

```
else:
```

```
# □ تغییر اندازه اگر ابعاد متفاوت بود
```

```
if raster_array.shape[0] > n_rows:
```

```
    raster_array = raster_array[:n_rows, :]
```

```
if raster_array.shape[1] > n_cols:
```

```
    raster_array = raster_array[:, :n_cols]
```

```
if raster_array.shape[0] < n_rows or raster_array.shape[1] < n_cols:
```

```
# □ پد کردن با صفر برای پر کردن فضای خالی
```

```
temp_array = np.zeros((n_rows, n_cols))
```

```
temp_array[:raster_array.shape[0], :raster_array.shape[1]] = raster_array
```

```
combined_array[i] = temp_array
```

```
else:
```

```

combined_array[i] = raster_array

except Exception as e:

    print(f" □ خطا در رستر {i}: {e}")

    combined_array[i] = np.zeros((n_rows, n_cols)) # □ جایگزینی با صفر

print(" □ آرایه ترکیبی ایجاد شد")

except Exception as e:

    print(f" □ خطا در ایجاد آرایه ترکیبی {e}")

    return None

# □ مرحله ۱۳: پیش‌بینی احتمالات روی کل منطقه

print("\n13. □ ...پیش‌بینی روی منطقه")

# □ تغییر شکل آرایه به دو بعدی (پیکسل‌ها × متغیرها)

reshaped = combined_array.reshape(len(aligned_rasters), -1).T

# □ (NaN غیر صفر و غیر) شناسایی پیکسل‌های معتبر □

valid_mask = ~np.isnan(reshaped).any(axis=1) & ~(reshaped == 0).all(axis=1)

valid_data = reshaped[valid_mask]

if len(valid_data) == 0:

    print(" □ هیچ داده معتبری برای پیش‌بینی وجود ندارد")

    return None

print(f" □ {len(valid_data):,} پیکسل معتبر برای پیش‌بینی")

```



```
# □ استانداردسازی و پیش‌بینی
```

```
try:
```

```
    print(" □ استانداردسازی داده‌ها...")
```

```
    valid_data_scaled = scaler.transform(valid_data)
```

```
    print(" □ انجام پیش‌بینی...")
```

```
    predictions = svm_model.predict(valid_data_scaled)
```

```
    probabilities = svm_model.predict_proba(valid_data_scaled)[: , 1] # □ احتمال کلاس معدنی
```

```
    print(" □ پیش‌بینی انجام شد")
```

```
except Exception as e:
```

```
    print(f" □ خطا در پیش‌بینی: {e}")
```

```
    return None
```

```
# □ مرحله ۱۴: ایجاد رسترهای خروجی با سیستم مختصات صحیح
```

```
print("\n14. □ ایجاد رسترهای خروجی با سیستم مختصات صحیح")
```

```
# □ اولیه NoData آرایه‌های خروجی با مقدار
```

```
pred_array_full = np.full((n_rows * n_cols,), -9999.0, dtype=float)
```

```
prob_array_full = np.full((n_rows * n_cols,), -9999.0, dtype=float)
```

```
# □ پر کردن آرایه‌ها با نتایج پیش‌بینی
```

```
pred_array_full[valid_mask] = predictions
```

```
prob_array_full[valid_mask] = probabilities
```

```

# □ بازسازی شکل اصلی

pred_array = pred_array_full.reshape(n_rows, n_cols)
prob_array = prob_array_full.reshape(n_rows, n_cols)


# □ اطلاعات مکانی دقیق برای ذخیره رستر

lower_left = arcpy.Point(common_extent.XMin, common_extent.YMin)
cell_size = 1 # □ پیکسل ۱ متری


# □ ایجاد نام فایل خروجی بر اساس ارتفاع

if elevation % 1 == 0:
    elev_name = str(int(elevation))
else:
    elev_name = f"{int(elevation)}_5"


output_pred = f"SVM_{elev_name}_Prediction"
output_prob = f"SVM_{elev_name}_Probability"


# □ ذخیره رستر پیش‌بینی

try:
    print(f" □ ذخیره رستر پیش‌بینی: {output_pred}")


# □ ArcGIS به رستر NumPy تبدیل آرایه

pred_raster = arcpy.NumPyArrayToRaster(
    pred_array,
    lower_left,
    cell_size,
    cell_size,

```

```

        value_to_nodata=-9999 # □ تعیین مقدار NoData
    )

    # □ ذخیره در حافظه موقت
    temp_pred = os.path.join("memory", "temp_pred")
    pred_raster.save(temp_pred)

    # □ تعریف سیستم مختصات
    arcpy.DefineProjection_management(temp_pred, common_spatial_ref)

    # □ خروجی GDB کپی به
    arcpy.CopyRaster_management(
        temp_pred,
        os.path.join(output_gdb_path, output_pred),
        pixel_type="32_BIT_FLOAT" # □ نوع پیکسل
    )

    print(f" □ {common_spatial_ref.name}: سیستم مختصات) ذخیره شد")

except Exception as e:
    print(f" □ {e}: خطا در ذخیره رستر پیش‌بینی")
    return None

# □ ذخیره رستر احتمالات
try:
    print(f" □ {output_prob}: ذخیره رستر احتمال")

```

```
# □ تبدیل آرایه NumPy به رستر ArcGIS
```

```
prob_raster = arcpy.NumPyArrayToRaster(  
    prob_array,  
    lower_left,  
    cell_size,  
    cell_size,  
    value_to_nodata=-9999 # □ تعیین مقدار NoData  
)
```

```
# □ ذخیره در حافظه موقت
```

```
temp_prob = os.path.join("memory", "temp_prob")  
prob_raster.save(temp_prob)
```

```
# □ تعریف سیستم مختصات
```

```
arcpy.DefineProjection_management(temp_prob, common_spatial_ref)
```

```
# □ خروجی GDB کپی به
```

```
arcpy.CopyRaster_management(  
    temp_prob,  
    os.path.join(output_gdb_path, output_prob),  
    pixel_type="32_BIT_FLOAT" # □ نوع پیکسل  
)
```

```
print(f" □ {common_spatial_ref.name}: سیستم مختصات ذخیره شد")
```

```
except Exception as e:
```

```
print(f" □ {e}: خطا در ذخیره رستر احتمال")
```

```
# □ ادامه می‌دهیم حتی اگر یکی ذخیره نشد
```

```
# □ مرحله ۱۵: پاکسازی فایل‌های موقت
```

```
print("\n15. □ پاکسازی فایل‌های موقت...")
```

```
try:
```

```
    arcpy.env.workspace = "memory"
```

```
    for item in arcpy.ListDatasets() + arcpy.ListFeatureClasses() + arcpy.ListRasters():
```

```
        try:
```

```
            arcpy.Delete_management(item)
```

```
        except:
```

```
            pass
```

```
    print(" □ پاکسازی انجام شد")
```

```
except:
```

```
    print(" □ پاکسازی با خطا مواجه شد")
```

```
# □ نمایش نتایج نهایی
```

```
print(f"\n{'='*60}")
```

```
print(f"□ با موفقیت پردازش شد {elevation} تراز")
```

```
print(f" □ رستر پیش‌بینی: {output_pred}")
```

```
print(f" □ رستر احتمال: {output_prob}")
```

```
print(f" □ دقت مدل: {test_score:.4f}")
```

```
print(f" □ ابعاد خروجی: {n_rows} × {n_cols}")
```

```
print(f" □ سیستم مختصات: {common_spatial_ref.name}")
```

```
print(f" □ وزن کلاس معدنی: {actual_class_weights.get(1, 1.2):.1f}x")
```

```
print(f"{'='*60}")
```

```
# □ بازگشت نتایج پردازش
```

```
return {  
    'elevation': elevation,  
    'n_rasters': len(rasters_list),  
    'n_samples': n_samples,  
    'accuracy': test_score,  
    'prediction': output_pred,  
    'probability': output_prob,  
    'dimensions': f"{n_rows}×{n_cols}",  
    'spatial_ref': common_spatial_ref.name,  
    'class_weight': actual_class_weights.get(1, 1.2)  
}
```

```
except Exception as e:
```

```
# □ مدیریت خطاهای کلی
```

```
print(f"\n{'='*60}")
```

```
print(f"□ خطای کلی در پردازش تراز {elevation}:")
```

```
print(f" {str(e)}")
```

```
print(f"{'='*60}")
```

```
return None
```

```
#
```

```
=
```

```
# □ بخش ۴: اجرای اصلی برنامه برای تمام ترازهای ارتفاعی
```

```
#
```

```
=
```

```
#
```

# □ این بخش نقطه شروع برنامه است و تمام ترازهای ارتفاعی را پردازش می‌کند.

# □ مراحل اصلی:

# 1. تنظیمات اولیه

# 2. پردازش هر تراز

# 3. ایجاد گزارش نهایی

# 4. ذخیره خلاصه نتایج

#

=====

=

if \_\_name\_\_ == "\_\_main\_\_":

# □ ArcGIS تنظیمات اولیه

arcpy.env.overwriteOutput = True

arcpy.CheckOutExtension("Spatial") # □ Spatial فعال‌سازی اکستنشن

print("=" \* 70)

print("□ برای تمام ترازهای ارتفاعی SVM شروع تحلیل")

print("=" \* 70)

print(f"□ تعداد ترازها: {len(elevations)}")

print(f"□ مسیر ورودی: {input\_gdb\_path}")

print(f"□ GDB خروجی: {output\_gdb\_path}")

print(f"□ SVM پارامترهای: C={C\_value}, kernel={kernel\_type}, gamma={gamma\_value}")

print(f"□ وزن کلاس معدنی: {class\_weights[1]}x")

print("=" \* 70)

# □ لیست برای ذخیره نتایج

```
results = []

success_count = 0

failed_count = 0

# □ ثبت زمان شروع

start_time = datetime.now()

print(f"\n□ زمان شروع: {start_time.strftime('%H:%M:%S')}")

# □ پردازش هر تراز ارتفاعی

for i, elevation in enumerate(elevations, 1):

    print(f"\n{'#' * 70}")

    print(f"□ پردازش تراز {i}/{len(elevations)}: {elevation}")

    print(f"{'#' * 70}")

    result = process_elevation(elevation)

    if result:

        results.append(result)

        success_count += 1

        print(f"\n□ تراز {elevation} پردازش شد ({i}/{len(elevations)})")

    else:

        failed_count += 1

        print(f"\n□ تراز {elevation} پردازش نشد ({i}/{len(elevations)})")

# □ نمایش پیشرفت

print(f"\n□ پیشرفت {i}/{len(elevations)}: پردازش")

print(f"□ موفق: {success_count}, □ ناموفق: {failed_count}")
```



```

# □ ثبت زمان پایان
end_time = datetime.now()
duration = end_time - start_time

# □ گزارش نهایی اجرا
print(f"\n{'=' * 70}")
print("□ پردازش تمام ترازها تکمیل شد")
print(f"{'=' * 70}")
print(f"□ زمان شروع: {start_time.strftime('%H:%M:%S')}")
print(f"□ زمان پایان: {end_time.strftime('%H:%M:%S')}")
print(f"□ مدت زمان: {duration}")
print(f"□ تعداد ترازهای موفق: {success_count}/{len(elevations)}")
print(f"□ تعداد ترازهای ناموفق: {failed_count}/{len(elevations)}")

# □ خلاصه نتایج
if results:
    print(f"\n{'=' * 70}")
    print("□ خلاصه نتایج:")
    print(f"{'=' * 70}")
    print(f"{'تراز':>۱۰} {'رسترها':>۸} {'نمونه‌ها':>۱۰} {'دقت':>۸} {'وزن':>۶} {'ابعاد':>۱۲}")
    print("-" * 60)

for res in sorted(results, key=lambda x: x['elevation']):
    print(f"{'res[elevation]':<10} {'res[n_rasters]':<8} "
          f"{'res[n_samples]':<10} {'res[accuracy]':.4f} ")

```

```
f"{res.get('class_weight', 1.0):<6.1f} "
```

```
f"{res.get('dimensions', 'N/A'):<12}")
```

```
# □ محاسبه میانگین دقت
```

```
if success_count > 0:
```

```
    avg_accuracy = sum(r['accuracy'] for r in results) / success_count
```

```
    print(f"\n□ میانگین دقت: {avg_accuracy:.4f}")
```

```
# □ ذخیره خلاصه نتایج در فایل متنی
```

```
summary_file = os.path.join(input_gdb_path, "SVM_Final_Summary_Weighted.txt")
```

```
try:
```

```
    with open(summary_file, 'w', encoding='utf-8') as f:
```

```
        f.write("=" * 70 + "\n")
```

```
        f.write("□ SVM (۱,۲ وزن معدنی x) خلاصه نتایج تحلیل\n")
```

```
        f.write("=" * 70 + "\n")
```

```
        f.write(f"□ تاریخ پردازش: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
```

```
        f.write(f"□ مدت زمان پردازش: {duration}\n")
```

```
        f.write(f"□ تعداد ترازهای پردازش شده: {len(elevations)}\n")
```

```
        f.write(f"□ تعداد موفق: {success_count}\n")
```

```
        f.write(f"□ تعداد ناموفق: {failed_count}\n")
```

```
        f.write("=" * 70 + "\n\n")
```

```
        f.write("□ پارامترهای مدل:\n")
```

```
        f.write(f"□ C: {C_value}\n")
```

```
        f.write(f"□ Kernel: {kernel_type}\n")
```

```
        f.write(f"□ Gamma: {gamma_value}\n")
```

```
        f.write(f"□ Test Size: {test_size}\n")
```

```

f.write(f" □ Random State: {random_state}\n")

f.write(f" □ وزن کلاس معدنی: {class_weights[1]}x\n")

f.write("\n" + "=" * 70 + "\n\n")


f.write("□ نتایج هر تراز:\n")

f.write(f"{'سیستم':>۲۵} {'پیش‌بینی':>۶} {'وزن':>۱۰} {'دقت':>۱۰} {'نمونه‌ها':>۸} {'رسترها':>۱۰} {'تراز':>۳۰}\n")

f.write("-" * 105 + "\n")


for res in sorted(results, key=lambda x: x['elevation']):

    f.write(f"{res['elevation']:<10} {res['n_rasters']:<8} "

           f"{res['n_samples']:<10} {res['accuracy']:.4f}    "

           f"{res.get('class_weight', 1.0):<6.1f} "

           f"{res.get('prediction', 'N/A'):<25} {res.get('spatial_ref', 'N/A'):<30}\n")


if success_count > 0:

    avg_accuracy = sum(r['accuracy'] for r in results) / success_count

    f.write(f"\n□ میانگین دقت: {avg_accuracy:.4f}\n")


f.write("\n" + "=" * 70 + "\n")

f.write("□ ترازهای ناموفق:\n")

failed_elevations = [e for e in elevations if e not in [r['elevation'] for r in results]]

for elev in failed_elevations:

    f.write(f" {elev}\n")


print(f"\n□ خلاصه نتایج در فایل ذخیره شد: {summary_file}")


except Exception as e:

```

```

print(f"❑ ذخیره خلاصه نتایج {e}")

# ❑ پاکسازی نهایی فایل‌های موقت
try:
    arcpy.env.workspace = "memory"

    for item in arcpy.ListDatasets() + arcpy.ListFeatureClasses() + arcpy.ListRasters():
        try:
            arcpy.Delete_management(item)
        except:
            pass
except:
    pass

# ❑ آزادسازی اکستنشن Spatial
arcpy.CheckInExtension("Spatial")

# ❑ پیام پایانی
print(f"\n{'=' * 70}")
print("❑ پردازش کامل شد!")

print(f"❑ ذخیره شده‌اند {output_gdb_path} نتایج در")
print(f"❑ Prediction و Probability: هر تراز شامل ۲ رستر")
print(f"❑ سیستم مختصات به درستی حفظ شده است")
print(f"❑ وزن نقاط معدنی {class_weights[1]}x")
print(f"❑ ذخیره شد {summary_file} خلاصه نتایج در")
print(f"{'=' * 70}")

```

#

=====

=

# □ بخش ۵: نکات مهم و راهنمایی

#

=====

=

#

# □ نکات کلیدی برای استفاده بهینه از این اسکرپت:

#

# 1. □ ساختار داده‌ها:

# های ورودی در مسیر مشخص شده وجود دارند GDB اطمینان حاصل کنید که تمام -

# (f1000\_ColorRaster مثلاً) نام‌گذاری رسترها باید شامل ارتفاع باشد -

# باشند Gosal.gdb نقاط معدنی و غیرمعدنی باید در -

#

# 2. □ SVM تنظیمات:

# (overfit مقادیر کوچک‌تر: کمتر) کنترل تعادل بین حاشیه و خطا: C پارامتر -

# (Xمعدنی: ۱,۵) وزن کلاس: برای مقابله با عدم تعادل داده‌ها -

# مناسب برای داده‌های غیرخطی: RBF هسته -

#

# 3. □ سیستم مختصات:

# اسکرپت به طور خودکار سیستم مختصات را حفظ می‌کند -

# همه رسترهای خروجی دارای سیستم مختصات یکسان هستند -

# اندازه پیکسل خروجی: ۱ متر -

#

# 4. □ بهینه‌سازی اجرا:

# برای تعداد زیاد رسترها، ممکن است پردازش زمان‌بر باشد -

# برای سرعت بیشتر استفاده شده است (memory workspace) از حافظه موقت -

# حافظه overload پردازش بلاک‌بندی برای جلوگیری از -

#

# 5. □ عیب‌یابی:

# اگر رستری پیدا نشد، نام‌گذاری آن را بررسی کنید -

# داشته باشند projection خطاهای سیستم مختصات: بررسی کنید که همه رسترها -

# خطاهای حافظه: تعداد رسترهای ورودی را کاهش دهید -

#

# 6. □ تفسیر نتایج:

# مقادیر ۰ (غیرمعدنی) و ۱ (معدنی): Prediction رستر -

# احتمالات بین ۰ تا ۱ (مثلاً  $0.8 = 80\%$  احتمال معدنی بودن): Probability رستر -

# دقت مدل: هرچه بالاتر باشد، مدل بهتر آموزش دیده است -

# وزن کلاس: نشان‌دهنده اهمیت نسبی کلاس معدنی است -

#

=====

=

خود-توجه عمیق

```
import arcpy
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import os
import sys
```

```

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

import time

import re

from datetime import datetime


# ===== تنظیمات اولیه =====

# □ پارامترهای ورودی - بخش مسیرها و تنظیمات پایه

# base_path (رشته): مسیر اصلی پوشه پروژه
# مثال: "M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal"
# این مسیر باید حاوی تمام پایگاه‌های جغرافیایی زیرمجموعه باشد

# output_folder (رشته): مسیر پوشه خروجی برای ذخیره نتایج
# مثال: "M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Self_Attention"
# این پوشه به طور خودکار ایجاد می‌شود اگر وجود نداشته باشد

base_path = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal"
output_folder = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Self_Attention"

# gdb_list (لیست رشته‌ها): لیست نام پایگاه‌های جغرافیایی
# باید حاوی رسترهای لایه‌های مختلف باشد GDB هر
# موجود باشند base_path ها مهم نیست اما باید در مسیر GDB ترتیب
gdb_list = [
    "EuclideanBuffers.gdb", # حاوی بافرهای اقلیدسی
    "LineDensity.gdb", # حاوی رسترهای تراکم خطی

```

```

"Fault_Intersections.gdb", # حاوی رسترهای تقاطع گسل‌ها
"Intersections_Density.gdb", # حاوی رسترهای تراکم تقاطع‌ها
"Alteration_Rasters.gdb", # حاوی رسترهای دگرسانی
"Litho_Rasters.gdb", # حاوی رسترهای لیتولوژی
"Voronoi_Output.gdb" # Voronoi حاوی خروجی‌های
]

# حاوی پلیگون‌های آموزشی GDB (رشته): مسیر کامل training_gdb
# های آموزشی باشد Feature Class باید حاوی GDB این
training_gdb = os.path.join(base_path, "Gosal.gdb")

# ایجاد پوشه خروجی اگر وجود ندارد
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# پارامترهای تحلیل - بخش تنظیمات عمومی □
# (عدد صحیح): رزولوشن خروجی به متر PIXEL_SIZE
# اندازه پیکسل رسترهای خروجی را تعیین می‌کند
PIXEL_SIZE = 1 # (متر) رزولوشن خروجی

# (لیست رشته‌ها): پسوندهای معتبر برای نام رسترها VALID_RASTER_SUFFIXES
# فقط رسترهایی با این پسوندها پردازش می‌شوند
VALID_RASTER_SUFFIXES = ["Reclassify", "raster", "Raster", "ColorRaster", "Azimuth_raster"]

# (رشته): پیشوند نام پلیگون‌های آموزشی TRAINING_PREFIX

```



# پلیگون‌های آموزشی باید با این پیشوند شروع شوند

TRAINING\_PREFIX = "o"

# ===== Self-Attention پارامترهای مدل =====

# □ Transformer پارامترهای معماری و آموزش مدل

MODEL\_PARAMS = {

'embed\_dim': 32, # بعد فضای ویژگی‌های تبدیل شده - embedding بعد

'num\_heads': 4, # موازی attention برای محاسبه - attention های head تعداد

'ff\_dim': 64, # Transformer درون بلوک - feed-forward بعد لایه

'num\_transformer\_blocks': 2, # عمق مدل - transformer تعداد بلوک‌های

'mlp\_units': [64, 32], # Transformer شبکه عصبی پس از - MLP واحدهای لایه

'dropout\_rate': 0.2, # overfitting برای جلوگیری از - dropout نرخ

'batch\_size': 32, # تعداد نمونه‌ها در هر تکرار آموزش - batch اندازه

'epochs': 50, # ها - تعداد دفعات کامل آموزش روی داده‌ها epoch تعداد

'learning\_rate': 0.001, # نرخ یادگیری - سرعت به‌روزرسانی وزن‌ها

'validation\_split': 0.2, # درصد داده برای اعتبارسنجی - validation نسبت داده

}

# ===== کلاس برای ذخیره گزارش =====

class ReportLogger:

"""

□ ReportLogger کلاس

وظیفه: ذخیره گزارش اجرا هم در فایل و هم نمایش در کنسول

:پارامترهای ورودی

- log\_file\_path (رشته): مسیر کامل فایل گزارش

:متدهای مهم

- start(): شروع ثبت گزارش

- write(text): نوشتن متن در گزارش

- stop(): توقف ثبت گزارش

"""

```
def __init__(self, log_file_path):
```

```
    self.log_file_path = log_file_path
```

```
    self.log_file = None
```

```
    self.original_stdout = sys.stdout
```

```
def start(self):
```

```
    """شروع ذخیره گزارش"""
```

```
    self.log_file = open(self.log_file_path, 'w', encoding='utf-8')
```

```
    # برای ذخیره در فایل و نمایش در کنسول stdout تغییر
```

```
    sys.stdout = self
```

```
def write(self, text):
```

```
    """نوشتن متن در فایل و کنسول"""
```

```
    if text.strip(): # فقط اگر متن غیرخالی است
```

```
        self.original_stdout.write(text)
```

```
        self.log_file.write(text)
```

```
        self.log_file.flush()
```

```
def flush(self):

    """flush کردن هر دو خروجی"""

    self.original_stdout.flush()

    if self.log_file:

        self.log_file.flush()
```

```
def stop(self):

    """توقف ذخیره گزارش"""

    if self.log_file:

        self.log_file.close()

    sys.stdout = self.original_stdout
```

# ===== کلاس‌های کمکی =====

```
class TimeBasedReportCallback(keras.callbacks.Callback):
```

```
    """
```

```
    □ کلاس TimeBasedReportCallback
```

```
    وظیفه: گزارش پیشرفت آموزش هر ۳۰ ثانیه
```

```
    پارامترهای ورودی:
```

```
- report_interval_seconds (عدد): فاصله زمانی گزارش‌دهی (پیش‌فرض: ۳۰ ثانیه)
```

```
    متدهای مهم:
```

```
- on_epoch_begin(): epoch شروع هر
```

```
- on_epoch_end(): epoch پایان هر
```

- print\_progress\_report(): چاپ گزارش پیشرفت

"""

```
def __init__(self, report_interval_seconds=30):
```

```
    super().__init__()
```

```
    self.report_interval = report_interval_seconds
```

```
    self.last_report_time = time.time()
```

```
    self.epoch_times = []
```

```
    self.current_epoch_start = None
```

```
    self.last_logs = {}
```

```
def on_epoch_begin(self, epoch, logs=None):
```

```
    self.current_epoch_start = time.time()
```

```
    print(f"\n{'-' * 60}")
```

```
    print(f"□ شروع اپوک {epoch + 1}/{self.params['epochs']}")
```

```
    print(f"{'-' * 60}")
```

```
    self.last_logs = {}
```

```
def on_epoch_end(self, epoch, logs=None):
```

```
    epoch_end = time.time()
```

```
    epoch_duration = epoch_end - self.current_epoch_start
```

```
    self.epoch_times.append(epoch_duration)
```

```
# محاسبه زمان باقیمانده
```

```
avg_epoch_time = np.mean(self.epoch_times) if self.epoch_times else epoch_duration
```

```
remaining_epochs = self.params['epochs'] - (epoch + 1)
```

```
estimated_remaining = avg_epoch_time * remaining_epochs
```

```

# گزارش نتایج اپوک
print(f"\n□ نتایج اپوک {epoch + 1}:")

print(f"    زمان اپوک : {epoch_duration:.1f} ثانیه")

# ذخیره لاگ‌ها برای گزارش بعدی
self.last_logs = logs.copy() if logs else {}

# validation نمایش متریک‌های
if logs:
    for key, value in logs.items():
        if key.startswith('val_'):
            print(f"    {key}: {value:.4f}")

print(f"    زمان میانگین هر اپوک : {avg_epoch_time:.1f} ثانیه")
print(f"    زمان تخمینی باقی‌مانده : {estimated_remaining/60:.1f} دقیقه")

# گزارش هر ۳۰ ثانیه
current_time = time.time()
if current_time - self.last_report_time >= self.report_interval:
    self.last_report_time = current_time
    self.print_progress_report(epoch, logs, estimated_remaining)

def print_progress_report(self, epoch, logs, estimated_remaining):
    """گزارش پیشرفت"""
    print(f"\n{'=' * 60}")
    print(f"□ گزارش پیشرفت - {time.strftime('%H:%M:%S')}")
    print(f"{'=' * 60}")

```

```

print(f"اپوک فعلی: {epoch + 1}/{self.params['epochs']}")

print(f"پیشرفت: {((epoch + 1) / self.params['epochs'] * 100):.1f}%")

print(f"دقیقه {estimated_remaining/60:.1f}: زمان تخمینی باقی مانده")

if logs:

    print("\n□ متریک‌های فعلی:")

    display_logs = logs if logs else self.last_logs

    # تعریف متریک‌های مورد نظر

    metrics = ['loss', 'accuracy', 'auc', 'precision', 'recall']

    for key in metrics:

        if key in display_logs:

            val_key = f'val_{key}'

            if val_key in display_logs:

                try:

                    train_val = float(display_logs[key])

                    val_val = float(display_logs[val_key])

                    print(f" {key}: {train_val:.4f} | {val_key}: {val_val:.4f}")

                except (ValueError, TypeError):

                    print(f" {key}: {display_logs[key]} | {val_key}: {display_logs[val_key]}")

    print(f"{'=' * 60}")

def on_train_end(self, logs=None):

    print(f"\n{'=' * 60}")

    print("□ آموزش مدل تکمیل شد")

    print(f"{'=' * 60}")

```

```

if self.epoch_times:

    total_time = sum(self.epoch_times)

    print(f"زمان کل آموزش: {total_time/60:.1f} دقیقه")

    print(f"میانگین زمان هر اپوک: {np.mean(self.epoch_times):.1f} ثانیه")

    print(f"تعداد اپوک‌های اجرا شده: {len(self.epoch_times)}")

```

```

class EpochTimerCallback(keras.callbacks.Callback):

```

```

    """

```

```

    □ کلاس EpochTimerCallback

```

```

    ها batch وظیفه: زمان‌سنجی دقیق اپوک‌ها و

```

```

    """

```

```

def __init__(self):

```

```

    super().__init__()

```

```

    self.epoch_times = []

```

```

    self.batch_times = []

```

```

def on_epoch_begin(self, epoch, logs=None):

```

```

    self.epoch_start_time = time.time()

```

```

    self.batch_start_time = time.time()

```

```

    self.batch_count = 0

```

```

    self.batch_times = []

```

```

def on_batch_end(self, batch, logs=None):

```

```

    batch_end = time.time()

```

```

    batch_duration = batch_end - self.batch_start_time

```

```

    self.batch_times.append(batch_duration)

```

```
self.batch_start_time = batch_end
```

```
self.batch_count += 1
```

```
# گزارش هر ۵۰ batch
```

```
if self.batch_count % 50 == 0:
```

```
    if len(self.batch_times) >= 10:
```

```
        avg_batch_time = np.mean(self.batch_times[-10:])
```

```
        print(f"↳ Batch {self.batch_count}: {avg_batch_time:.3f}s/batch", end='\r')
```

```
def on_epoch_end(self, epoch, logs=None):
```

```
    epoch_duration = time.time() - self.epoch_start_time
```

```
    self.epoch_times.append(epoch_duration)
```

```
    if self.batch_times and self.batch_count > 0:
```

```
        start_idx = max(0, len(self.batch_times) - self.batch_count)
```

```
        relevant_batch_times = self.batch_times[start_idx:]
```

```
    if relevant_batch_times:
```

```
        avg_batch_time = np.mean(relevant_batch_times)
```

```
        print(f"\n □ زمان اپوک : {epoch_duration:.1f}s | میانگین batch: {avg_batch_time:.3f}s")
```

```
# ===== توابع کمکی =====
```

```
def get_elevation_from_name(name):
```

```
    """
```

```
    □ تابع get_elevation_from_name
```

```
    وظیفه: استخراج تراز ارتفاعی از نام فایل یا رستر
```



پارامترهای ورودی:

- name: نام فایل یا رستر

خروجی:

None رشته: تراز ارتفاعی استخراج شده یا -

"""

```
patterns = [
```

```
    r'f?(\d+)_(\d+)',
```

```
    r'f?(\d+)',
```

```
    r'a(\d+)',
```

```
    r'r(\d+)',
```

```
    r'o(\d+)_(\d+)',
```

```
    r'o(\d+)'
```

```
]
```

```
for pattern in patterns:
```

```
    match = re.search(pattern, name)
```

```
    if match:
```

```
        if len(match.groups()) == 2:
```

```
            return f'{match.group(1)}_{match.group(2)}'
```

```
        else:
```

```
            return match.group(1)
```

```
    return None
```

```
def find_rasters_for_elevation(elevation):
```

```
    """
```

```
    □ تابع find_rasters_for_elevation
```

های مختلف GDB وظیفه: پیدا کردن تمام رسترهای مربوط به یک تراز ارتفاعی در

پارامترهای ورودی:

(رشته): تراز ارتفاعی مورد جستجو elevation -

خروجی:

لیست: مسیرهای کامل رسترهای یافت شده -

"""

```
all_rasters = []
```

```
for gdb_name in gdb_list:
```

```
    gdb_path = os.path.join(base_path, gdb_name)
```

```
    if arcpy.Exists(gdb_path):
```

```
        arcpy.env.workspace = gdb_path
```

```
        rasters = arcpy.ListRasters()
```

```
        for raster_name in rasters:
```

```
            has_valid_suffix = any(raster_name.endswith(suffix) for suffix in  
VALID_RASTER_SUFFIXES)
```

```
            elev_in_name = get_elevation_from_name(raster_name)
```

```
            if has_valid_suffix and elev_in_name == elevation:
```

```
                raster_path = os.path.join(gdb_path, raster_name)
```

```
                if raster_path not in all_rasters:
```

```
                    all_rasters.append(raster_path)
```

```
                    print(f" ✓ رستر یافت شد: {os.path.basename(raster_path)}")
```

```
return all_rasters
```

```
def find_training_polygon(elevation):
```

```
    """
```

```
    □ تابع find_training_polygon
```

```
    وظیفه: پیدا کردن پلیگون آموزشی برای تراز ارتفاعی مشخص
```

```
    پارامترهای ورودی:
```

```
    - elevation (رشته): تراز ارتفاعی مورد جستجو
```

```
    خروجی:
```

```
    None رشته: مسیر کامل پلیگون آموزشی یا -
```

```
    """
```

```
    arcpy.env.workspace = training_gdb
```

```
    polygons = arcpy.ListFeatureClasses(f'{TRAINING_PREFIX}{elevation}*')
```

```
    if polygons:
```

```
        return os.path.join(training_gdb, polygons[0])
```

```
    else:
```

```
        all_fcs = arcpy.ListFeatureClasses()
```

```
        for fc in all_fcs:
```

```
            elev_in_name = get_elevation_from_name(fc)
```

```
            if elev_in_name == elevation and fc.startswith(TRAINING_PREFIX):
```

```
                return os.path.join(training_gdb, fc)
```

```
    return None
```

```
def create_mask_from_polygon(polygon_path, template_raster):
```

```
"""
```

□ تابع `create_mask_from_polygon`

وظیفه: ایجاد ماسک باینری از پلیگون آموزشی

پارامترهای ورودی:

- `polygon_path` (رشته): مسیر پلیگون آموزشی

- `template_raster` (یا `arcpy.Raster` رشته یا): رستر الگو برای ابعاد

خروجی:

ماسک باینری (۱ برای داخل پلیگون، ۰ برای خارج): `numpy` آرایه -

```
"""
```

```
print("...ایجاد ماسک از پلیگون آموزشی")
```

```
try:
```

```
    mask_raster = arcpy.sa.ExtractByMask(template_raster, polygon_path)
```

```
    binary_raster = arcpy.sa.Con(arcpy.sa.IsNull(mask_raster), 0, 1)
```

```
    mask_array = arcpy.RasterToNumPyArray(binary_raster, nodata_to_value=0)
```

```
    print(f" ✓ ماسک ایجاد شد. شکل {mask_array.shape}")
```

```
    return mask_array.astype(np.float32)
```

```
except Exception as e:
```

```
    print(f" ✗ خطا در ایجاد ماسک: {e}")
```

```
    template_desc = arcpy.Describe(template_raster)
```

```
    cell_size = template_desc.meanCellWidth
```

```
    extent = template_desc.extent
```

```

height = int((extent.YMax - extent.YMin) / cell_size)
width = int((extent.XMax - extent.XMin) / cell_size)

mask_array = np.zeros((height, width), dtype=np.float32)
print(f" ✓ ماسک خالی ایجاد شد. شکل: {mask_array.shape}")

return mask_array

```

```
def read_raster_to_array(raster_path, template_shape=None):
```

```
    """
```

□ تابع read\_raster\_to\_array

NoData با مدیریت numpy وظیفه: خواندن رستر و تبدیل به آرایه

پارامترهای ورودی:

- raster\_path (رشته): مسیر رستر ورودی

- template\_shape (تاپل): شکل الگو برای تطبیق ابعاد (اختیاری)

خروجی:

در صورت خطا None داده‌های رستر به صورت آرایه یا numpy آرایه -

```
    """
```

```
try:
```

```
    raster = arcpy.Raster(raster_path)
```

```
    no_data_value = raster.noDataValue
```

```
    if no_data_value is not None:
```

```
        raster_array = arcpy.RasterToNumPyArray(raster, nodata_to_value=no_data_value)
```

else:

```
raster_array = arcpy.RasterToNumPyArray(raster)
```

```
raster_array = raster_array.astype(np.float32)
```

if no\_data\_value is not None:

```
raster_array[raster_array == no_data_value] = np.nan
```

if template\_shape is not None and raster\_array.shape != template\_shape:

```
print(f" {template_shape} به {raster_array.shape} تطبیق اندازه از جا ")
```

```
new_array = np.zeros(template_shape, dtype=np.float32)
```

```
new_array[:] = np.nan
```

```
min_rows = min(raster_array.shape[0], template_shape[0])
```

```
min_cols = min(raster_array.shape[1], template_shape[1])
```

```
new_array[:min_rows, :min_cols] = raster_array[:min_rows, :min_cols]
```

```
raster_array = new_array
```

```
return raster_array
```

except Exception as e:

```
print(f" ✖ خطا در خواندن رستر: {e}")
```

```
return None
```

```
def prepare_training_data(rasters, training_polygon):
```

```
"""
```

```
□ تابع prepare_training_data
```

وظیفه: آماده‌سازی داده‌های آموزشی از رسترها و پلیگون آموزشی

پارامترهای ورودی:

- rasters: (لیست): مسیرهای رسترهای ورودی

- training\_polygon: مسیر پلیگون آموزشی

خروجی:

در صورت خطا None یا (X\_train, y\_train, X\_all, y\_all, original\_shape, valid\_mask): تاپل -

```
"""
```

```
print("\n□ آماده‌سازی داده‌های آموزشی...")
```

```
if not rasters:
```

```
    print("❌ هیچ رستری یافت نشد")
```

```
    return None, None, None, None, None, None
```

```
template_raster = arcpy.Raster(rasters[0])
```

```
template_desc = arcpy.Describe(template_raster)
```

```
cell_size = template_desc.meanCellWidth
```

```
extent = template_desc.extent
```

```
template_height = int((extent.YMax - extent.YMin) / cell_size)
```

```
template_width = int((extent.XMax - extent.XMin) / cell_size)
```

```
template_shape = (template_height, template_width)
```

```
print(f"ابعاد template: {template_shape}")
```

```

print(f"سل سایز: {cell_size}m")

mask_array = create_mask_from_polygon(training_polygon, template_raster)

if mask_array.shape != template_shape:
    print(f"تطبیق سایز ماسک از {mask_array.shape} به {template_shape}")
    new_mask = np.zeros(template_shape, dtype=np.float32)
    min_rows = min(mask_array.shape[0], template_shape[0])
    min_cols = min(mask_array.shape[1], template_shape[1])
    new_mask[:min_rows, :min_cols] = mask_array[:min_rows, :min_cols]
    mask_array = new_mask

mask_array = (mask_array > 0).astype(np.float32)

raster_arrays = []
valid_rasters = []

print(f"\nپرستر {len(rasters)} پردازش")

for i, raster_path in enumerate(rasters):
    raster_name = os.path.basename(raster_path)
    print(f" [{i+1}/{len(rasters)}] پردازش: {raster_name}")

    raster_array = read_raster_to_array(raster_path, template_shape)

    if raster_array is not None:
        valid_values = raster_array[~np.isnan(raster_array)]

```



```

if valid_values.size > 0:

    min_val = np.min(valid_values)

    max_val = np.max(valid_values)

    if max_val > min_val:

        raster_norm = (raster_array - min_val) / (max_val - min_val)

    else:

        raster_norm = raster_array.copy()

raster_norm = np.nan_to_num(raster_norm, nan=0.0)

raster_arrays.append(raster_norm)
valid_rasters.append(raster_path)

print(f"    ✓ پردازش شد | شکل: {raster_norm.shape}")

print(f"    محدوده: [{np.min(raster_norm):.3f}, {np.max(raster_norm):.3f}]")

print(f"    میانگین: {np.mean(raster_norm):.3f} | انحراف معیار: {np.std(raster_norm):.3f}")

else:

    print(f"    ✗ هیچ مقدار معتبری در رستر وجود ندارد")

else:

    print(f"    ✗ خطا در پردازش رستر")

if not raster_arrays:

    print(f"    ✗ هیچ آرایه رستری پردازش نشد")

    return None, None, None, None, None

print(f"\n    ✓ {len(raster_arrays)} رستر پردازش شد")

```

```

X = np.stack(raster_arrays, axis=-1)
y = mask_array

print(f"\n □ ابعاد داده‌های ترکیبی:")

print(f" X: {X.shape} (ارتفاع × عرض × کانال)")
print(f" y: {y.shape}")
print(f" تعداد ویژگی‌ها: {X.shape[2]}")

valid_mask = np.ones(X.shape[:2], dtype=bool)

for i in range(X.shape[2]):
    channel_valid = ~np.isnan(X[:, :, i])
    valid_mask = valid_mask & channel_valid

valid_mask = valid_mask & (~np.isnan(y))

valid_indices = np.where(valid_mask)
num_samples = len(valid_indices[0])
print(f"\n □ شناسایی پیکسل‌های معتبر:")
print(f" تعداد پیکسل‌های معتبر: {num_samples:,}")

if num_samples == 0:
    print(" ✗ هیچ پیکسل معتبری یافت نشد")
    return None, None, None, None, None, None

X_all = X[valid_mask]

```

```

y_all = y[valid_mask]

print(f"\n □ داده‌های کامل برای پیش‌بینی:")

print(f" X_all: {X_all.shape} (تمام پیکسل‌های معتبر)")

print(f" y_all: {y_all.shape}")

y_all_int = y_all.astype(np.int32)

class_1_count = np.sum(y_all_int == 1)
class_0_count = np.sum(y_all_int == 0)

print(f" کلاس ۱ (معدن): {class_1_count}")

print(f" کلاس ۰ (غیرمعدن): {class_0_count}")

class_1_indices = np.where(y_all_int == 1)[0]
class_0_indices = np.where(y_all_int == 0)[0]

if class_1_count > 0 and class_0_count > 0:
    imbalance_ratio = max(class_1_count, class_0_count) / min(class_1_count, class_0_count)
    print(f" نسبت عدم تعادل: {imbalance_ratio:.1f}")

    min_class_size = min(class_1_count, class_0_count)
    max_train_samples = min(50000, min_class_size * 2)

    print(f"\n □ ...ایجاد مجموعه آموزشی متعادل")

    if class_1_count > min_class_size:
        selected_class_1 = np.random.choice(class_1_indices, min(min_class_size,
max_train_samples//2), replace=False)

```

```

        selected_class_0 = np.random.choice(class_0_indices, min(min_class_size,
max_train_samples//2), replace=False)

    else:

        selected_class_0 = np.random.choice(class_0_indices, min(min_class_size,
max_train_samples//2), replace=False)

        selected_class_1 = np.random.choice(class_1_indices, min(min_class_size,
max_train_samples//2), replace=False)

    selected_indices = np.concatenate([selected_class_1, selected_class_0])
    np.random.shuffle(selected_indices)

    X_train = X_all[selected_indices]
    y_train = y_all[selected_indices]

    print(f" ✓ مجموعه آموزشی ایجاد شد.")

    print(f"   تعداد نمونه‌های آموزشی: {len(X_train):,}")

    print(f"   کلاس ۱ در آموزش: {np.sum(y_train == 1):,}")

    print(f"   کلاس ۰ در آموزش: {np.sum(y_train == 0):,}")

    else:

        max_train_samples = 50000

        if len(X_all) > max_train_samples:

            selected_indices = np.random.choice(len(X_all), max_train_samples, replace=False)

            X_train = X_all[selected_indices]
            y_train = y_all[selected_indices]

        else:

            X_train = X_all
            y_train = y_all

    print(f" ✓ استفاده از همه نمونه‌ها برای آموزش.")

```

```
print(f"تعداد نمونه‌های آموزشی: {len(X_train):,}")
```

```
original_shape = X.shape[:2]
```

```
return X_train, y_train, X_all, y_all, original_shape, valid_mask
```

```
class TransformerBlock(layers.Layer):
```

```
    """
```

□ TransformerBlock کلاس

Self-Attention برای Transformer وظیفه: پیاده‌سازی یک بلوک

پارامترهای ورودی:

- embedding (عدد): بعد embed\_dim

- attention های head (عدد): تعداد num\_heads

- feed-forward (عدد): بعد لایه ff\_dim

- dropout (عدد): نرخ rate

```
    """
```

```
def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
```

```
    super(TransformerBlock, self).__init__()
```

```
    self.att = layers.MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
```

```
    self.ffn = keras.Sequential([
```

```
        layers.Dense(ff_dim, activation="relu"),
```

```
        layers.Dense(embed_dim),
```

```
    ])
```

```
    self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
```

```
    self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)
```

```
self.dropout1 = layers.Dropout(rate)
```

```
self.dropout2 = layers.Dropout(rate)
```

```
def call(self, inputs, training):
```

```
    attn_output = self.att(inputs, inputs)
```

```
    attn_output = self.dropout1(attn_output, training=training)
```

```
    out1 = self.layernorm1(inputs + attn_output)
```

```
    ffn_output = self.ffn(out1)
```

```
    ffn_output = self.dropout2(ffn_output, training=training)
```

```
    return self.layernorm2(out1 + ffn_output)
```

```
def create_transformer_model(num_features):
```

```
    """
```

```
    □ تابع create_transformer_model
```

```
    Self-Attention کامل برای Transformer وظیفه: ایجاد مدل
```

```
    پارامترهای ورودی:
```

```
    - num_features (عدد): تعداد ویژگی‌های ورودی
```

```
    خروجی:
```

```
    - Keras مدل: Transformer مدل
```

```
    """
```

```
    print("\n□ ایجاد مدل Self-Attention...")
```

```
    inputs = layers.Input(shape=(num_features,))
```

```

x = layers.Dense(MODEL_PARAMS['embed_dim'])(inputs)
x = layers.Reshape((1, MODEL_PARAMS['embed_dim']))(x)

for i in range(MODEL_PARAMS['num_transformer_blocks']):
    print(f" اضافه کردن بلوک Transformer {i+1} ...")

    x = TransformerBlock(
        MODEL_PARAMS['embed_dim'],
        MODEL_PARAMS['num_heads'],
        MODEL_PARAMS['ff_dim'],
        MODEL_PARAMS['dropout_rate']
    )(x)

x = layers.GlobalAveragePooling1D()(x)

for units in MODEL_PARAMS['mlp_units']:
    x = layers.Dense(units, activation="relu")(x)
    x = layers.Dropout(MODEL_PARAMS['dropout_rate'])(x)

outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=MODEL_PARAMS['learning_rate']),
    loss="binary_crossentropy",
    metrics=[
        "accuracy",
        keras.metrics.AUC(name='auc'),

```

```
keras.metrics.Precision(name='precision'),  
keras.metrics.Recall(name='recall')  
]  
)
```

```
print(" مدل ایجاد و کامپایل شد ✓ ")
```

```
return model
```

```
def train_and_predict(X_train, y_train, X_full, model_params=MODEL_PARAMS):
```

```
    """
```

```
    □ تابع train_and_predict
```

وظیفه: آموزش مدل و انجام پیش‌بینی

پارامترهای ورودی:

- X\_train (آرایه numpy): داده‌های آموزشی

- y\_train (آرایه numpy): برچسب‌های آموزشی

- X\_full (آرایه numpy): تمام داده‌ها برای پیش‌بینی

- model\_params (MODEL\_PARAMS): پیش‌فرض (دیکشنری): پارامترهای مدل

خروجی:

- (predictions, model, history, scaler): تاپل

```
    """
```

```
print("\n□ شروع آموزش مدل Self-Attention")
```

```
print("=" * 60)
```

```
print("validation... تقسیم داده‌ها به آموزش و")
```



```

X_train_split, X_val, y_train_split, y_val = train_test_split(
    X_train, y_train,
    test_size=model_params['validation_split'],
    random_state=42,
    stratify=y_train
)

print(f" ✓ نمونه {len(X_train_split):,} داده‌های آموزش")
print(f" ✓ نمونه {len(X_val):,} validation داده‌های")

print("...استانداردسازی داده‌ها")

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_split)
X_val_scaled = scaler.transform(X_val)
X_full_scaled = scaler.transform(X_full)

print(" ✓ داده‌ها استاندارد شدند")

model = create_transformer_model(num_features=X_train_scaled.shape[1])

print("\n□ مدل خلاصه Self-Attention:")
model.summary()

callbacks = [
    TimeBasedReportCallback(report_interval_seconds=30),
    EpochTimerCallback(),
    keras.callbacks.EarlyStopping(

```

```

        monitor='val_auc',
        patience=10,
        restore_best_weights=True,
        mode='max',
        verbose=1
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=5,
        min_lr=1e-6,
        verbose=1
    )
]

print(f"\n□ شروع آموزش با پارامترهای زیر:")
print(f"تعداد اپوک: {model_params['epochs']}")
print(f"batch اندازه: {model_params['batch_size']}")
print(f"نرخ یادگیری: {model_params['learning_rate']}")
print(f"زمان شروع: {time.strftime('%H:%M:%S')}")
print("=" * 60)

start_train_time = time.time()

try:
    history = model.fit(
        X_train_scaled, y_train_split,
        validation_data=(X_val_scaled, y_val),

```

```

        epochs=model_params['epochs'],
        batch_size=model_params['batch_size'],
        callbacks=callbacks,
        verbose=0
    )
except Exception as e:
    print(f"❑ خطا در آموزش مدل: {e}")
    print("تلاش مجدد با تنظیمات ساده‌تر")

    history = model.fit(
        X_train_scaled, y_train_split,
        validation_data=(X_val_scaled, y_val),
        epochs=min(20, model_params['epochs']),
        batch_size=model_params['batch_size'],
        callbacks=[TimeBasedReportCallback(report_interval_seconds=30)],
        verbose=0
    )

train_duration = time.time() - start_train_time
print(f"\n❑ زمان کل آموزش: {train_duration/60:.1f} دقیقه")

print("\n❑ ارزیابی نهایی مدل:")
try:
    val_results = model.evaluate(X_val_scaled, y_val, verbose=0)

    print("validation: متریک‌های")

    if val_results is not None:
        if isinstance(val_results, list) and len(val_results) > 0:

```

```

for metric_name, metric_value in zip(model.metrics_names, val_results):

    try:

        metric_value_float = float(metric_value)

        print(f"    {metric_name}: {metric_value_float:.4f}")

    except (ValueError, TypeError):

        print(f"    {metric_name}: {metric_value}")

    else:

        print(f"    نتایج: {val_results}")

except Exception as e:

    print(f"❑ خطا در ارزیابی مدل: {e}")

print("\n❑ انجام پیش‌بینی روی تمام نمونه‌ها")

start_predict_time = time.time()

try:

    batch_size_predict = 4096

    num_batches = int(np.ceil(len(X_full_scaled) / batch_size_predict))

    print(f"❑ پیش‌بینی {len(X_full_scaled):,} نمونه در {num_batches} batch...")

    all_predictions = []

    for i in range(num_batches):

        start_idx = i * batch_size_predict

        end_idx = min((i + 1) * batch_size_predict, len(X_full_scaled))

        batch_predictions = model.predict(

            X_full_scaled[start_idx:end_idx],

            batch_size=batch_size_predict,

```

```

        verbose=0

    )

    all_predictions.append(batch_predictions)

    if (i + 1) % 10 == 0 or i == num_batches - 1:
        print(f"    Batch {i+1}/{num_batches} - {end_idx:},} نمونه")

    predictions = np.concatenate(all_predictions, axis=0)

    predict_duration = time.time() - start_predict_time
    print(f"    □ زمان پیش‌بینی: {predict_duration:.1f} ثانیه")
    print(f"    □ تعداد پیش‌بینی‌ها: {len(predictions):,}")

    predictions_flat = predictions.flatten()

except Exception as e:
    print(f"    □ خطا در پیش‌بینی: {e}")
    predictions_flat = np.random.uniform(0, 1, size=len(X_full_scaled))
    print(f"    □ استفاده از پیش‌بینی‌های تصادفی به عنوان جایگزین")

return predictions_flat, model, history, scaler

def save_prediction_to_folder(predictions, valid_mask, original_shape, elevation, template_raster_path):
    """
    □ تابع save_prediction_to_folder

    وظیفه: ذخیره نتایج پیش‌بینی به صورت رستر در پوشه خروجی

```

پارامترهای ورودی:

- predictions (numpy آرایه): مقادیر پیش‌بینی شده
- valid\_mask (numpy آرایه): ماسک پیکسل‌های معتبر
- original\_shape (تاپل): شکل اصلی رستر
- elevation (رشته): تراز ارتفاعی
- template\_raster\_path (رشته): مسیر رستر الگو برای کپی کردن مشخصات

خروجی:

رشته: مسیر فایل ذخیره شده یا فایل اضطراری -

"""

```
print(f"\n□ ذخیره رستر در پوشه خروجی")
```

try:

```
template_raster = arcpy.Raster(template_raster_path)
```

```
template_desc = arcpy.Describe(template_raster)
```

```
output_array = np.zeros(original_shape, dtype=np.float32)
```

```
output_array[valid_mask] = predictions
```

```
# نام فایل خروجی
```

```
raster_name = f"Mineral_Probability_{elevation}"
```

```
output_tif_path = os.path.join(output_folder, f"{raster_name}.tif")
```

```
# حذف اگر از قبل وجود دارد
```

```
if arcpy.Exists(output_tif_path):
```

```

print(f" حذف رستر موجود: {raster_name}.tif")

arcpy.management.Delete(output_tif_path)

# نقطه شروع
lower_left = arcpy.Point(template_desc.extent.XMin, template_desc.extent.YMin)

# ایجاد رستر
print(f" ...ایجاد رستر جدید")

new_raster = arcpy.NumPyArrayToRaster(
    output_array,
    lower_left,
    template_desc.meanCellWidth,
    template_desc.meanCellHeight
)

# TIFF ذخیره به صورت
new_raster.save(output_tif_path)

# اعمال سیستم مختصات
print(f" ...اعمال سیستم مختصات")

arcpy.management.DefineProjection(output_tif_path, template_desc.spatialReference)

# تنظیم NoData Value
print(f" ...تنظیم NoData Value")

try:
    arcpy.management.SetRasterProperties(
        output_tif_path,

```

```

        nodata="1 0",

        stats="STATISTICS 1"

    )
except:

    print(" □ NoData Value تنظیم نشود (ممکن است پشتیبانی نشود) ")

print(f"□ رستر با موفقیت ذخیره شد ")

print(f" □ مسیر: {output_tif_path}")

print(f" □ ابعاد: {original_shape}")

print(f" □ سیستم مختصات: {template_desc.spatialReference.name}")


# هم (برای سازگاری بیشتر) ASCII ذخیره به صورت
asc_path = os.path.join(output_folder, f'{raster_name}.asc')

print(f"\n□ ASCII ذخیره نسخه ...")


ncols = original_shape[1]
nrows = original_shape[0]
xllcorner = template_desc.extent.XMin
yllcorner = template_desc.extent.YMin
cellsize = template_desc.meanCellWidth
nodata_value = -9999


with open(asc_path, 'w') as f:

    f.write(f"NCOLS {ncols}\n")

    f.write(f"NROWS {nrows}\n")

    f.write(f"XLLCORNER {xllcorner}\n")

    f.write(f"YLLCORNER {yllcorner}\n")

```



```

f.write(f'CELLSIZE {cellsize}\n')

f.write(f'NODATA_VALUE {nodata_value}\n')

for i in range(nrows):
    row_str = ''.join([f'{val:.6f}' for val in output_array[i, :]])
    f.write(row_str + '\n')

print(f" ✓ فایل ASCII ذخیره شد: {asc_path}")

# گزارش آماری
valid_predictions = predictions[np.isfinite(predictions)]
if len(valid_predictions) > 0:
    print(f"\n□ آمارهای پیش‌بینی:")
    print(f"   تعداد پیکسل‌های معتبر: {len(valid_predictions):,}")
    print(f"   حداقل: {np.min(valid_predictions):.6f}")
    print(f"   حداکثر: {np.max(valid_predictions):.6f}")
    print(f"   میانگین: {np.mean(valid_predictions):.6f}")
    print(f"   میانه: {np.median(valid_predictions):.6f}")

    print(f"\n□ توزیع احتمالات:")
    bins = np.linspace(0, 1, 11)
    hist, _ = np.histogram(valid_predictions, bins=bins)

    for i in range(len(bins)-1):
        percentage = (hist[i] / len(valid_predictions)) * 100
        if percentage > 0.1:
            print(f"   {bins[i]:.1f}-{bins[i+1]:.1f}: {hist[i]:,} پیکسل ({percentage:.1f}%)")

```

```
# ذخیره داده‌های خام برای استفاده‌های بعدی
```

```
npz_path = os.path.join(output_folder, f"{raster_name}_raw_data.npz")
```

```
np.savez(npz_path,
```

```
    predictions=predictions,
```

```
    valid_mask=valid_mask,
```

```
    original_shape=original_shape,
```

```
    raster_array=output_array)
```

```
print(f"\n □ داده‌های خام برای استفاده‌های بعدی ذخیره شد: {npz_path}")
```

```
return output_tif_path
```

```
except Exception as e:
```

```
print(f" □ خطا در ذخیره رستر: {e}")
```

```
# numpy ذخیره اضطراری به صورت
```

```
npz_path = os.path.join(output_folder, f"Mineral_Probability_{elevation}_emergency.npz")
```

```
np.savez(npz_path,
```

```
    predictions=predictions,
```

```
    valid_mask=valid_mask,
```

```
    original_shape=original_shape,
```

```
    error=str(e))
```

```
print(f" □ اضطراری ذخیره شدند numpy داده‌ها به صورت فایل: {npz_path}")
```

```
return npz_path
```

```
# ===== تابع اصلی برای یک تراز =====
```

```
def main(elevation_level):
```

```
    """
```

```
    □ تابع main
```

```
    وظیفه: اجرای کامل تحلیل برای یک تراز ارتفاعی
```

```
    پارامترهای ورودی:
```

```
    - elevation_level (رشته): تراز ارتفاعی مورد پردازش
```

```
    خروجی:
```

```
    در صورت خطا None رشته: مسیر فایل خروجی یا -
```

```
    """
```

```
    print("\n" + "=" * 70)
```

```
    print(f"□ Self-Attention Deep Learning شروع تحلیل {elevation_level}")
```

```
    print("=" * 70)
```

```
    print(f"□ تاریخ: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
```

```
    print(f"□ تراز ارتفاعی: {elevation_level}")
```

```
    print(f"□ پوشه خروجی: {output_folder}")
```

```
    print("=" * 70)
```

```
    start_total_time = time.time()
```

```
    # یافتن رسترهای مربوط به تراز
```

```
    print("\n□ جستجوی رسترهای مربوط به تراز ...")
```

```
    rasters = find_rasters_for_elevation(elevation_level)
```

```
if not rasters:
```

```
    print(f"❌ یافت نشد {elevation_level} هیچ رستری برای تراز")
```

```
    return None
```

```
print(f"❌ تعداد رسترهای یافت شده: {len(rasters)}")
```

```
# یافتن پلیگون آموزشی
```

```
print("\n❌ جستجوی پلیگون آموزشی...")
```

```
training_polygon = find_training_polygon(elevation_level)
```

```
if not training_polygon:
```

```
    print(f"❌ یافت نشد {elevation_level} هیچ پلیگون آموزشی برای تراز")
```

```
    return None
```

```
print(f"❌ پلیگون آموزشی یافت شد: {os.path.basename(training_polygon)}")
```

```
# آماده‌سازی داده‌های آموزشی
```

```
result = prepare_training_data(rasters, training_polygon)
```

```
if result[0] is None:
```

```
    print(f"❌ داده آموزشی کافی یافت نشد")
```

```
    return None
```

```
X_train, y_train, X_all, y_all, original_shape, valid_mask = result
```

```
print(f"\n❌ داده‌های آموزشی آماده شدند")
```

```

print(f"ابعاد X_train: {X_train.shape} (برای آموزش)")

print(f"ابعاد X_all: {X_all.shape} (برای پیش‌بینی روی تمام پیکسل‌ها)")

print(f"ابعاد original_shape: {original_shape}")


# آموزش مدل و پیش‌بینی روی تمام پیکسل‌ها
predictions, model, history, scaler = train_and_predict(
    X_train, y_train, X_all
)


# ذخیره نتایج در پوشه
output_path = save_prediction_to_folder(
    predictions, valid_mask, original_shape,
    elevation_level, rasters[0]
)


# نمایش خلاصه نتایج
total_duration = time.time() - start_total_time


print("\n" + "=" * 70)
print(f"!با موفقیت تکمیل شد {elevation_level} برای تراز Self-Attention تحلیل")
print("=" * 70)
print(f"خلاصه نتایج:")
print(f"{elevation_level}: تراز ارتفاعی")
print(f"{len(X_train):,}: تعداد نمونه‌های آموزشی")
print(f"{len(X_all):,}: تعداد پیکسل‌های پیش‌بینی شده")
print(f"{X_train.shape[1]}: تعداد ویژگی‌ها")

```

```

print(f"زمان کل تحلیل: {total_duration/60:.1f} دقیقه")

if output_path:
    if output_path.endswith('.npz'):
        print(f" {output_path}: ذخیره شد numpy خروجی به صورت فایل ")
    else:
        print(f" {output_path}: رستر در پوشه ذخیره شد ")
    else:
        print(f" !هیچ رستری ذخیره نشد ")

# ذخیره اطلاعات مدل

try:
    model_save_path = os.path.join(output_folder, f"SelfAttention_Model_{elevation_level}.keras")
    model.save(model_save_path)
    print(f"\n {model_save_path}: مدل ذخیره شد ")
except Exception as e:
    print(f" {e}: خطا در ذخیره مدل ")

# ذخیره scaler

try:
    import joblib
    scaler_save_path = os.path.join(output_folder, f"Scaler_{elevation_level}.pkl")
    joblib.dump(scaler, scaler_save_path)
    print(f" {scaler_save_path}: ذخیره شد Scaler ")
except ImportError:
    print(f" (نصب نیست joblib) وجود ندارد Scaler امکان ذخیره ")
except Exception as e:

```

```
print(f"□ خطا در ذخیره Scaler: {e}")
```

```
# ذخیره اطلاعات آموزش
```

```
if history and hasattr(history, 'history'):
```

```
    history_path = os.path.join(output_folder, f"Training_History_{elevation_level}.csv")
```

```
    try:
```

```
        import pandas as pd
```

```
        hist_df = pd.DataFrame(history.history)
```

```
        hist_df.to_csv(history_path, index=False)
```

```
        print(f"□ تاریخچه آموزش ذخیره شد : {history_path}")
```

```
    except Exception as e:
```

```
        print(f"□ خطا در ذخیره تاریخچه آموزش : {e}")
```

```
# با دستورالعمل‌ها readme ایجاد فایل
```

```
readme_path = os.path.join(output_folder, f"README_{elevation_level}.txt")
```

```
with open(readme_path, 'w', encoding='utf-8') as f:
```

```
    f.write("=" * 70 + "\n")
```

```
    f.write(f"□ Self-Attention برای تراز {elevation_level}\n" استفاده از نتایج تحلیل
```

```
    f.write("=" * 70 + "\n\n")
```

```
    f.write(f"تاریخ تحلیل: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
```

```
    f.write(f"تراز ارتفاعی: {elevation_level}\n")
```

```
    f.write(f"پوشه خروجی: {output_folder}\n\n")
```

```
    f.write("□ فایل‌های تولید شده :\n")
```

```
    f.write("-" * 40 + "\n")
```

```
    f.write(f"1. Mineral_Probability_{elevation_level}.tif : رستر احتمالات معدنی\n")
```

f.write(f"2. Mineral\_Probability\_{elevation\_level}.asc : رستر احتمالات معدنی (فرمت ASCII)\n")

f.write(f"3. SelfAttention\_Model\_{elevation\_level}.keras: مدل آموزش دیده\n")

f.write(f"4. Scaler\_{elevation\_level}.pkl : استانداردساز داده ها\n")

f.write(f"5. Training\_History\_{elevation\_level}.csv : تاریخچه آموزش مدل\n")

f.write(f"6. Mineral\_Probability\_{elevation\_level}\_raw\_data.npz: داده های خام\n")

f.write(f"7. Execution\_Report\_{elevation\_level}.txt : گزارش کامل اجرا\n\n")

f.write("□ دستورالعمل نمایش در ArcGIS:\n")

f.write("-" \* 40 + "\n")

f.write("1. اضافه کنید ArcGIS را به TIFF فایل\n")

f.write("2. Properties → روی لایه راست کلیک\n")

f.write("3. Symbolology → Classified تب\n")

f.write("4. Classes: 10\n")

f.write("5. Method: Natural Breaks (Jenks)\n")

f.write("6. Color Ramp: (کم) به سبز (زیاد)\n")

f.write("7. Apply → OK\n\n")

f.write("\n:تفسیر مقادیر احتمال □\n")

f.write("-" \* 40 + "\n")

f.write("0.0 - 0.2: احتمال بسیار کم معدنی\n")

f.write("0.2 - 0.4: احتمال کم معدنی\n")

f.write("0.4 - 0.6: احتمال متوسط معدنی\n")

f.write("0.6 - 0.8: احتمال بالا معدنی\n")

f.write("0.8 - 1.0: احتمال بسیار بالا معدنی\n\n")



```

f.write("□ پارامترهای مدل:\n")

f.write("-" * 40 + "\n")

for key, value in MODEL_PARAMS.items():

    f.write(f"{key}: {value}\n")


print(f"\n□ فایل راهنما ایجاد شد: {readme_path}")


print("\n" + "=" * 70)

print(f"□ زمان پایان تحلیل برای تراز {elevation_level}: {datetime.now().strftime('%H:%M:%S')}")

print("=" * 70)


return output_path


# ===== تولید لیست ترازهای ارتفاعی =====

def generate_elevation_levels():

    """

    □ تابع generate_elevation_levels

    وظیفه: ایجاد لیست ترازهای ارتفاعی از ۸۷۵ تا ۱۱۲۵ با فاصله ۱۲,۵ متر

    خروجی:

    لیست: ترازهای ارتفاعی به صورت رشته -

    """

    elevations = []

    current = 875.0

    while current <= 1125.0:

        if current % 1 == 0:

```

```

# اعداد صحیح
elevations.append(str(int(current)))
else:
    # underline اعداد اعشاری - تبدیل نقطه به
    elev_str = str(current)
    if '.' in elev_str:
        elev_str = elev_str.replace('.', '_')
    elevations.append(elev_str)
    current += 12.5

# نمایش لیست ترازها
print("\n□ لیست ترازهای ارتفاعی برای تحلیل:")
print("=" * 40)
for i, elev in enumerate(elevations, 1):
    print(f"{i:2d}. {elev} متر")
print(f"تراز: {len(elevations)} جمع:")
print("=" * 40)

return elevations

# ===== اجرای اصلی برای همه ترازها =====

if __name__ == "__main__":
    # تنظیم محیط ArcGIS
    arcpy.env.overwriteOutput = True
    arcpy.env.cellSize = PIXEL_SIZE
    arcpy.env.extent = "MAXOF"

```

```
arcpy.env.snapRaster = None

# ایجاد پوشه خروجی اگر وجود ندارد
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# تولید لیست ترازهای ارتفاعی
ELEVATION_LEVELS = generate_elevation_levels()

# ایجاد گزارش اصلی
master_log_file = os.path.join(output_folder, "MASTER_EXECUTION_REPORT.txt")
master_logger = ReportLogger(master_log_file)
master_logger.start()

try:
    # زمان شروع کلی
    total_start_time = time.time()

    print("\n" + "=" * 80)
    print("□ برای تمام ترازهای ارتفاعی Self-Attention شروع تحلیل □")
    print("=" * 80)
    print(f"□ تاریخ شروع: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')} □")
    print(f"□ تعداد ترازها: {len(ELEVATION_LEVELS)} □")
    print(f"□ پوشه خروجی: {output_folder} □")
    print(f"□ فایل گزارش اصلی: {master_log_file} □")
    print("=" * 80)
```

```
# ذخیره لیست ترازها در فایل

with open(os.path.join(output_folder, "elevation_list.txt"), 'w', encoding='utf-8') as f:
    for elev in ELEVATION_LEVELS:
        f.write(f'{elev}\n')


# اجرای تحلیل برای هر تراز

results = {}
successful_elevations = []
failed_elevations = []


for i, elevation in enumerate(ELEVATION_LEVELS, 1):
    print(f'\n{'#' * 80}')
    print(f'□ پردازش تراز {i}/{len(ELEVATION_LEVELS)}: {elevation}')
    print(f{'#' * 80}')

    try:
        # جداگانه برای این تراز logger ایجاد

        level_log_file = os.path.join(output_folder, f'Execution_Report_{elevation}.txt')
        level_logger = ReportLogger(level_log_file)
        level_logger.start()

        # زمان شروع این تراز

        level_start_time = time.time()

        # اجرای تحلیل برای این تراز

        result = main(elevation)
```

```

# زمان پایان این تراز
level_end_time = time.time()

level_duration = level_end_time - level_start_time

if result:

    results[elevation] = result

    successful_elevations.append(elevation)

    print(f"\n□ تراز {elevation} شد با موفقیت پردازش شد")

    print(f"□ {level_duration/60:.1f} دقیقه: زمان پردازش")

else:

    failed_elevations.append(elevation)

    print(f"\n□ تراز {elevation} با مشکل مواجه شد")

    print(f"□ {level_duration/60:.1f} دقیقه: زمان صرف شده")

# این تراز logger توقف

level_logger.stop()

except Exception as e:

    failed_elevations.append(elevation)

    print(f"\n□ خطا در پردازش تراز {elevation}: {str(e)}")

    import traceback

    traceback.print_exc()

# ذخیره خطا در فایل

error_file = os.path.join(output_folder, f'ERROR_{elevation}.txt')

with open(error_file, 'w', encoding='utf-8') as f:

    f.write(f'خطا در پردازش تراز {elevation}:\n')

```

```

f.write(f"زمان: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")

f.write(f"پیام خطا: {str(e)}\n\n")

traceback.print_exc(file=f)

# محاسبه زمان کلی

total_end_time = time.time()

total_duration = total_end_time - total_start_time

# نمایش خلاصه نتایج

print(f"\n{'=' * 80}")

print("□ خلاصه نتایج تمام ترازها")

print(f"{'=' * 80}")

print(f"□ زمان کل اجرا: {total_duration/60:.1f} دقیقه")

print(f"□ تعداد کل ترازها: {len(ELEVATION_LEVELS)}")

print(f"□ ترازهای موفق: {len(successful_elevations)}")

print(f"□ ترازهای ناموفق: {len(failed_elevations)}")

if successful_elevations:

    print(f"\n□ لیست ترازهای موفق:")

    for i, elev in enumerate(successful_elevations, 1):

        print(f" {i:2d}. {elev}")

if failed_elevations:

    print(f"\n□ لیست ترازهای ناموفق:")

    for i, elev in enumerate(failed_elevations, 1):

        print(f" {i:2d}. {elev}")

```

# ذخیره خلاصه در فایل

summary\_file = os.path.join(output\_folder, "SUMMARY\_REPORT.txt")

with open(summary\_file, 'w', encoding='utf-8') as f:

f.write("=" \* 80 + "\n")

f.write("\n□ گزارش خلاصه تحلیل Self-Attention ارتفاعی تمام ترازهای ارتفاعی\n")

f.write("=" \* 80 + "\n\n")

f.write(f"تاریخ تحلیل: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")

f.write(f"پوشه خروجی: {output\_folder}\n")

f.write(f"زمان کل تحلیل: {total\_duration/60:.1f} دقیقه\n")

f.write(f"تعداد کل ترازها: {len(ELEVATION\_LEVELS)}\n")

f.write(f"تعداد ترازهای موفق: {len(successful\_elevations)}\n")

f.write(f"تعداد ترازهای ناموفق: {len(failed\_elevations)}\n\n")

f.write("\n□ ترازهای موفق:\n")

f.write("-" \* 40 + "\n")

for elev in successful\_elevations:

f.write(f"✓ {elev}\n")

f.write("\n□ ترازهای ناموفق:\n")

f.write("-" \* 40 + "\n")

for elev in failed\_elevations:

f.write(f"✗ {elev}\n")

f.write("\n□ فایل‌های خروجی برای هر تراز موفق:\n")

f.write("-" \* 40 + "\n")

```

f.write("1. Mineral_Probability_[تراز].tif - رستر احتمالات معدنی\n")

f.write("2. Mineral_Probability_[تراز].asc - رستر احتمالات معدنی (ASCII)\n")

f.write("3. SelfAttention_Model_[تراز].keras - مدل آموزش دیده\n")

f.write("4. Scaler_[تراز].pkl - استانداردساز داده ها\n")

f.write("5. Training_History_[تراز].csv - تاریخچه آموزش\n")

f.write("6. Mineral_Probability_[تراز]_raw_data.npz - داده های خام\n")

f.write("7. Execution_Report_[تراز].txt - گزارش اجرا\n")

f.write("8. README_[تراز].txt - راهنمای استفاده\n\n")


f.write("\n□: پارامترهای مدل استفاده شده\n")

f.write("-" * 40 + "\n")

for key, value in MODEL_PARAMS.items():

    f.write(f'{key}: {value}\n')


print(f"\n□ خلاصه نتایج ذخیره شد {summary_file}")


# برای مشاهده سریع نتایج bat ایجاد فایل

bat_file = os.path.join(output_folder, "VIEW_RESULTS.bat")

with open(bat_file, 'w', encoding='utf-8') as f:

    f.write("@echo off\n")

    f.write("echo =====\n")

    f.write("echo Self-Attention مشاهده نتایج تحلیل\n")

    f.write("echo =====\n")

    f.write("echo.\n")

    f.write(f'echo پوشه نتایج: {output_folder}\n')

    f.write("echo.\n")

```



```
f.write("echo \n فایل های مهم :\n")

f.write("echo 1. SUMMARY_REPORT.txt - خلاصه \n گزارش")

f.write("echo 2. MASTER_EXECUTION_REPORT.txt - کامل \n گزارش")

f.write("echo 3. elevation_list.txt - لیست ترازها \n")

f.write("echo.\n")

f.write("pause\n")
```

```
print(f"\n □ برای مشاهده نتایج ایجاد شد bat فایل {bat_file}")
```

except Exception as e:

```
print(f"\n □ خطای اصلی در اجرا : {str(e)}")
```

```
import traceback
```

```
traceback.print_exc()
```

finally:

```
# اصلی logger توقف
```

```
master_logger.stop()
```

```
# پیام نهایی
```

```
print(f"\n □ تمام خروجی ها در پوشه زیر ذخیره شدند")
```

```
print(f" □ {output_folder}")
```

```
print(f"\n □ گزارش جامع اجرا در فایل زیر ذخیره شد")
```

```
print(f" □ {master_log_file}")
```

if successful\_elevations:

```
print(f"\n □ {len(ELEVATION_LEVELS)} تراز از {len(successful_elevations)} تحلیل با موفقیت برای  
تکمیل شد")
```

else:

```
print(f"\n❑ !تحلیل برای هیچ ترازی تکمیل نشد")
```

انیمیشن این کد

```
import arcpy
import os
import re
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors, cm
from PIL import Image

#
=====
=
# ❑ بخش ۱: تنظیمات اولیه و مسیرها
#
=====
=
#
# ❑ این بخش شامل تعریف مسیرهای ورودی و خروجی و تنظیمات پایه است.
# ❑ تمام پارامترهای مهم در این قسمت قابل تنظیم هستند.
#
=====
=
# ❑ مسیرهای اصلی
input_folder = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Self_Attention" # ❑ پوشه حاوی فایل‌های
asc
```

```
output_folder = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Animation" # □ پوشه خروجی برای  
انیمیشن
```

```
folder_name = os.path.basename(os.path.normpath(input_folder)) # □ استخراج نام پوشه
```

```
output_gif = os.path.join(output_folder, f'{folder_name}.gif') # □ مسیر فایل GIF خروجی
```

```
#
```

```
=
```

```
# □ بخش ۲: تنظیمات قابل تغییر توسط کاربر
```

```
#
```

```
=
```

```
#
```

```
# □ این بخش شامل پارامترهایی است که کاربر می‌تواند بر اساس نیاز خود تغییر دهد
```

```
# □ هر پارامتر با توضیحات کامل در کنار آن قرار دارد
```

```
#
```

```
=
```

```
# □ تنظیمات رنگ و طبقه‌بندی
```

```
WHITE_FIRST_CLASS = False # □ کنید True سفید باشد Legend اگر می‌خواهید اولین طبقه در □
```

```
# □ False: مقدار همه طبقات رنگی هستند
```

```
# □ True: مقدار (یا مقادیر پایین NoData برای نمایش) اولین طبقه سفید
```

```
NUM_CLASSES = 5 # □ Legend تعداد طبقات در (بین ۳ تا ۱۲)
```

```
# □ مقادیر کمتر: تفکیک‌پذیری کمتر اما خوانایی بیشتر
```

```
# □ Legend مقادیر بیشتر: تفکیک‌پذیری بیشتر اما شلوغی
```

FILE\_PREFIX = "Mineral\_Probability\_" # □ می‌شوند فایل‌های asc مورد نظر

پردازش می‌شوند asc خالی گذاشتن: همه فایل‌های □ #

□ # :طرح‌های رنگی پیش‌فرض (انتخاب یکی)

□ # blue\_red: آبی به قرمز

□ # viridis: (سبز-آبی-بنفش) طرح علمی مدرن

□ # plasma: (زرد-نارنجی-بنفش) طرح درخشان

□ # terrain: (سبز-قهوه‌ای) رنگ‌های طبیعی

□ # ocean: آبی‌های عمیق

□ # rainbow: رنگین کمان

□ # sunset: غروب آفتاب

□ # coolwarm: سرد به گرم

COLOR\_SCHEME = "ocean" # □ انتخاب طرح رنگی

#

=

□ # بخش ۳: آماده‌سازی محیط و فایل‌ها

#

=

#

□ # .را آماده می‌کند asc این بخش فایل‌های ورودی

□ # .شامل لیست کردن فایل‌ها و فیلتر کردن بر اساس پیشوند می‌شود

#

=

```

print(f"□ پوشه ورودی: {input_folder}")

# □ موجود در پوشه asc لیست کردن تمام فایل‌های
all_files = [f for f in os.listdir(input_folder) if f.endswith('.asc')]
print(f"□ پوشه asc تعداد کل فایل‌های {len(all_files)}")

if len(all_files) == 0:
    print("□ پوشه یافت نشد asc هیچ فایل")
    print(f"□ لطفاً مسیر ورودی را بررسی کنید {input_folder}")
    exit()

# □ فیلتر فایل‌ها بر اساس پیشوند تعریف شده
if FILE_PREFIX:
    # □ فیلتر فایل‌هایی که با پیشوند مشخص شده شروع می‌شوند
    asc_files = [f for f in all_files if f.startswith(FILE_PREFIX)]
    print(f"□ فیلتر فایل‌ها با پیشوند '{FILE_PREFIX}'")
    print(f"□ تعداد فایل‌های فیلتر شده {len(asc_files)}")

    # □ نمایش فایل‌های حذف شده (برای عیب‌یابی)
    other_files = [f for f in all_files if not f.startswith(FILE_PREFIX)]
    if other_files:
        print(f"□ فایل‌های حذف شده: {' '.join(other_files[:5])}", end="")
        if len(other_files) > 5:
            print(f" و {len(other_files) - 5} مورد دیگر...")
        else:
            print()

```

else:

# □ پردازش می‌شوند asc اگر پیشوند مشخص نشده باشد، همه فایل‌های □

asc\_files = all\_files

print("□ پردازش می‌شوند asc هیچ پیشوندی مشخص نشده، همه فایل‌های □")

# □ برای پردازش asc بررسی وجود فایل

if len(asc\_files) == 0:

print("□ !با پیشوند مشخص شده یافت نشد asc هیچ فایل □")

print("□ را بررسی کنید FILE\_PREFIX لطفاً پیشوند □")

exit()

print(f"□ پیدا شده asc تعداد فایل‌های □: {len(asc\_files)}")

#

=

# □ asc بخش ۴: مرتب‌سازی فایل‌های □

#

=

#

# □ این بخش فایل‌ها را بر اساس عدد موجود در نامشان مرتب می‌کند.

# □ این برای ایجاد انیمیشن ترتیبی مهم است (مثلاً از پایین به بالا).

#

=

def extract\_number(name):

"""

استخراج عدد از نام فایل برای مرتب‌سازی

این تابع اولین عدد (اعم از اعشاری یا صحیح) را از نام فایل استخراج می‌کند

ورودی: نام فایل (رشته)

خروجی: عدد استخراج شده (اعشاری)

مثال‌ها:

"Mineral\_Probability\_875.5.asc" → 875.5

"Mineral\_Probability\_912\_5.asc" → 912.5

"Mineral\_Probability\_100.asc" → 100.0

"""

```
matches = re.findall(r'\d+\.\d*', name)
```

```
return float(matches[0]) if matches else 0
```

# مرتب‌سازی فایل‌ها بر اساس عدد استخراج شده

```
asc_files_sorted = sorted(asc_files, key=extract_number)
```

```
print("مرتب شده بر اساس ارتفاع asc فایل‌های")
```

```
for i, f in enumerate(asc_files_sorted[:10]): # نمایش ۱۰ مورد اول
```

```
    num = extract_number(f)
```

```
    print(f" {i+1:2d}. {f} → {num}")
```

```
if len(asc_files_sorted) > 10:
```

```
    print(f" ... و {len(asc_files_sorted) - 10} مورد دیگر")
```

```
#
```

```
=====
```

```
=
```

# کلی Extent بخش ۵: محاسبه

```

#
=====
=
#
# □ همه رسترها را محاسبه می کند (Extent) این بخش محدوده جغرافیایی کلی
# □ این برای ثابت نگه داشتن محدوده نمایش در تمام فریم‌های انیمیشن ضروری است
#
=====
=

print("\n□ محاسبه Extent همه رسترها")
extents = []
for f in asc_files_sorted:
    asc_path = os.path.join(input_folder, f)
    try:
        # □ به عنوان رستر asc بارگذاری فایل
        raster = arcpy.Raster(asc_path)
        desc = arcpy.Describe(raster)
        extents.append(desc.extent)
    except Exception as e:
        print(f"□ {f}: {e}")

if not extents:
    print("□ هیچ رستری به درستی خوانده نشد")
    exit()

# □ X و Y یافتن حداقل و حداکثر مختصات
xmin = min(e.XMin for e in extents)
xmax = max(e.XMax for e in extents)

```



```

ymin = min(e.YMin for e in extents)
ymax = max(e.YMax for e in extents)

print(f"□ Extent محاسبه شد.")

print(f"□ X: {xmin:.1f} تا {xmax:.1f}")

print(f"□ Y: {ymin:.1f} تا {ymax:.1f}")

#
=====
=

# □ بخش ۶: محاسبه محدوده مقادیر رستری
#
=====
=

#
# □ را در تمام رسترها محاسبه می‌کند (NoData بدون) این بخش حداقل و حداکثر مقادیر واقعی
# □ یکسان در تمام فریم‌ها استفاده می‌شوند Legend این مقادیر برای ایجاد
#
=====
=

print("\n□ محاسبه محدوده مقادیر در تمام رسترها...")

all_valid_values = []

for f in asc_files_sorted:

    asc_path = os.path.join(input_folder, f)

    try:

        # □ به عنوان رستر asc بارگذاری فایل

        raster = arcpy.Raster(asc_path)

```

```

# □ NumPy تبدیل رستر به آرایه
arr = arcpy.RasterToNumPyArray(raster)

# □ ایجاد ماسک برای مقادیر معتبر
# با -۹۹۹۹ مشخص می‌شوند NoData ، مقادیر معمولاً در فایل‌های
valid_mask = (arr != -9999) & (arr > -9998) # NoData حذف

if np.any(valid_mask):
    all_valid_values.append(arr[valid_mask])
except Exception as e:
    print(f"□ {e}: خطا در پردازش فایل □")

# □ محاسبه حداقل و حداکثر کل
if all_valid_values:
    all_valid_values = np.concatenate(all_valid_values)
    vmin, vmax = np.min(all_valid_values), np.max(all_valid_values)
    print(f"□ محدوده مقادیر محاسبه شد:")
    print(f"□ حداقل: {vmin:.2f}")
    print(f"□ حداکثر: {vmax:.2f}")
else:
    # در صورت عدم وجود مقادیر معتبر، از مقادیر پیش‌فرض استفاده می‌شود
    vmin, vmax = 0, 100 # مناسب است ۰-۱۰۰
    print(f"□ هیچ مقدار معتبری یافت نشد. استفاده از مقادیر پیش‌فرض")
    print(f"□ محدوده: {vmin} تا {vmax}")

```

```
#
=====
=

# بخش ۷: تنظیمات طبقه‌بندی و رنگ‌ها
#
=====
=
#
# این بخش شامل:
# Legend تنظیم تعداد طبقات ۱.
# ایجاد مرزهای طبقات ۲.
# تعریف پالت‌های رنگی مختلف ۳.
# انتخاب طرح رنگی بر اساس تنظیمات کاربر ۴.
#
=====
=

# تنظیم تعداد طبقات (محدود شده بین ۳ تا ۱۲)
num_classes = max(3, min(12, NUM_CLASSES))
print(f"\nتنظیمات طبقه‌بندی:")
print(f"    Legend: {num_classes} طبقات")

# (Boundaries) ایجاد مرزهای طبقات
class_bounds = np.linspace(vmin, vmax, num_classes + 1)
class_bounds = np.round(class_bounds, 1) # گرد کردن به یک رقم اعشار
print(f"    مرزهای طبقات: {class_bounds}")
```

```

#
=====
=

# □ تابع: ایجاد پالت‌های رنگی حرفه‌ای

#
=====
=

def get_color_palette(color_scheme, num_classes, white_first=False):
    """
    ایجاد پالت رنگی حرفه‌ای بر اساس طرح انتخابی

    این تابع پالت رنگی متناسب با تعداد طبقات و طرح انتخابی ایجاد می‌کند.
    ورودی‌ها:
    - color_scheme: نام طرح رنگی
    - num_classes: تعداد طبقات
    - white_first: آیا اولین طبقه سفید باشد؟

    HEX خروجی: لیست رنگ‌ها در فرمت
    """

    # □ تعریف پالت‌های رنگی مختلف برای تعداد طبقات متفاوت

    color_palettes = {
        # □ آبی به قرمز
        'blue_red': {
            3: ['#FFFFFF', '#4393c3', '#d6604d'],
            4: ['#FFFFFF', '#2166ac', '#f4a582', '#b2182b'],
            5: ['#FFFFFF', '#053061', '#4393c3', '#f4a582', '#b2182b'],

```

```

6: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#f4a582', '#d6604d'],
7: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#f4a582', '#d6604d'],
8: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#f4a582', '#fddbc7', '#d6604d'],
9: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#d1e5f0', '#fddbc7', '#f4a582',
'#d6604d'],
10: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#d1e5f0', '#fddbc7', '#f4a582',
'#d6604d', '#b2182b'],
11: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#d1e5f0', '#f7f7f7', '#fddbc7',
'#f4a582', '#d6604d', '#b2182b'],
12: ['#FFFFFF', '#053061', '#2166ac', '#4393c3', '#92c5de', '#d1e5f0', '#f7f7f7', '#fddbc7',
'#f4a582', '#ef8a62', '#d6604d', '#b2182b']
},

```

# □ viridis - پالت علمی مدرن

```

'viridis': {
3: ['#FFFFFF', '#35b779', '#440154'],
4: ['#FFFFFF', '#35b779', '#21918c', '#440154'],
5: ['#FFFFFF', '#6ece58', '#21918c', '#31688e', '#440154'],
6: ['#FFFFFF', '#b5de2b', '#35b779', '#21918c', '#31688e', '#440154'],
7: ['#FFFFFF', '#fde725', '#6ece58', '#35b779', '#21918c', '#31688e', '#440154'],
8: ['#FFFFFF', '#fde725', '#b5de2b', '#6ece58', '#35b779', '#21918c', '#31688e', '#440154'],
9: ['#FFFFFF', '#fde725', '#b5de2b', '#6ece58', '#35b779', '#1f9e89', '#26828e', '#31688e',
'#440154'],
10: ['#FFFFFF', '#fde725', '#d8e219', '#b5de2b', '#8fd744', '#6ece58', '#35b779', '#1f9e89',
'#26828e', '#440154']
},

```

# □ plasma - پالت درخشان

```

'plasma': {
3: ['#FFFFFF', '#f0f921', '#0d0887'],

```

```
4: ['#FFFFFF', '#fdb42f', '#9c179e', '#0d0887'],
5: ['#FFFFFF', '#f0f921', '#ed7953', '#9c179e', '#0d0887'],
6: ['#FFFFFF', '#f0f921', '#fdb42f', '#ed7953', '#9c179e', '#0d0887'],
7: ['#FFFFFF', '#f0f921', '#fdb42f', '#ed7953', '#cc4778', '#9c179e', '#0d0887'],
8: ['#FFFFFF', '#f0f921', '#fdc328', '#fdb42f', '#ed7953', '#cc4778', '#9c179e', '#0d0887'],
9: ['#FFFFFF', '#f0f921', '#fdc328', '#fdb42f', '#fb9f3a', '#ed7953', '#cc4778', '#9c179e', '#0d0887']
},
```

# □ terrain - رنگ‌های طبیعی

```
'terrain': {
  3: ['#FFFFFF', '#d9f0d3', '#8c510a'],
  4: ['#FFFFFF', '#d9f0d3', '#f6e8c3', '#8c510a'],
  5: ['#FFFFFF', '#d9f0d3', '#f6e8c3', '#dfc27d', '#8c510a'],
  6: ['#FFFFFF', '#d9f0d3', '#c7eae5', '#f6e8c3', '#dfc27d', '#8c510a'],
  7: ['#FFFFFF', '#d9f0d3', '#c7eae5', '#f6e8c3', '#dfc27d', '#bf812d', '#8c510a'],
  8: ['#FFFFFF', '#d9f0d3', '#c7eae5', '#f6e8c3', '#dfc27d', '#bf812d', '#8c510a', '#543005']
},
```

# □ ocean - آبی‌های عمیق

```
'ocean': {
  3: ['#FFFFFF', '#6baed6', '#08306b'],
  4: ['#FFFFFF', '#6baed6', '#2171b5', '#08306b'],
  5: ['#FFFFFF', '#9ecae1', '#4292c6', '#2171b5', '#08306b'],
  6: ['#FFFFFF', '#c6dbef', '#6baed6', '#4292c6', '#2171b5', '#08306b'],
  7: ['#FFFFFF', '#deebf7', '#9ecae1', '#6baed6', '#4292c6', '#2171b5', '#08306b'],
  8: ['#FFFFFF', '#f7fbff', '#c6dbef', '#9ecae1', '#6baed6', '#4292c6', '#2171b5', '#08306b']
},
```

# □ rainbow - رنگین کمان

'rainbow': {

```
3: ['#FFFFFF', '#00FF7F', '#FF0000'],
4: ['#FFFFFF', '#00FF7F', '#FFFF00', '#FF0000'],
5: ['#FFFFFF', '#0000FF', '#00FF7F', '#FFFF00', '#FF0000'],
6: ['#FFFFFF', '#0000FF', '#00FFFF', '#00FF7F', '#FFFF00', '#FF0000'],
7: ['#FFFFFF', '#8000FF', '#0000FF', '#00FFFF', '#00FF7F', '#FFFF00', '#FF0000'],
8: ['#FFFFFF', '#8000FF', '#0000FF', '#00FFFF', '#00FF7F', '#80FF00', '#FFFF00', '#FF0000']
},
```

# □ sunset - غروب آفتاب

'sunset': {

```
3: ['#FFFFFF', '#FFA500', '#8B0000'],
4: ['#FFFFFF', '#FFD700', '#FF8C00', '#8B0000'],
5: ['#FFFFFF', '#FFD700', '#FFA500', '#FF4500', '#8B0000'],
6: ['#FFFFFF', '#FFFFE0', '#FFD700', '#FFA500', '#FF4500', '#8B0000'],
7: ['#FFFFFF', '#FFFFE0', '#FFD700', '#FFA500', '#FF8C00', '#FF4500', '#8B0000'],
8: ['#FFFFFF', '#FFFFFF0', '#FFFFE0', '#FFD700', '#FFA500', '#FF8C00', '#FF4500', '#8B0000']
},
```

# □ coolwarm - سرد به گرم

'coolwarm': {

```
3: ['#FFFFFF', '#67a9cf', '#ef8a62'],
4: ['#FFFFFF', '#67a9cf', '#fddbc7', '#ef8a62'],
5: ['#FFFFFF', '#2166ac', '#67a9cf', '#fddbc7', '#d6604d'],
6: ['#FFFFFF', '#2166ac', '#67a9cf', '#d1e5f0', '#fddbc7', '#d6604d'],
7: ['#FFFFFF', '#053061', '#2166ac', '#67a9cf', '#fddbc7', '#ef8a62', '#b2182b'],
8: ['#FFFFFF', '#053061', '#2166ac', '#67a9cf', '#d1e5f0', '#fddbc7', '#ef8a62', '#b2182b']
```

```

    }

}

# □ بررسی وجود طرح انتخابی در دیکشنری

if color_scheme in color_palettes:

    if num_classes in color_palettes[color_scheme]:

        colors_list = color_palettes[color_scheme][num_classes]

    else:

        # □ اگر تعداد طبقات دقیق وجود نداشت، نزدیک‌ترین را پیدا کن

        available_classes = list(color_palettes[color_scheme].keys())

        closest = min(available_classes, key=lambda x: abs(x - num_classes))

        colors_list = color_palettes[color_scheme][closest]

        print(f"□ {color_scheme} برای طرح {num_classes} تعداد طبقات")

        print(f"    {closest} طبقه: استفاده از نزدیک‌ترین")

    else:

        # □ اگر طرح انتخابی موجود نبود، از طرح پیش‌فرض استفاده کن

        print(f"□ 'blue_red' تعریف نشده. استفاده از طرح پیش‌فرض '{color_scheme}' طرح رنگی")

        if num_classes in color_palettes["blue_red"]:

            colors_list = color_palettes["blue_red"][num_classes]

        else:

            colors_list = color_palettes["blue_red"][5]

# □ مدیریت سفید بودن اولین طبقه

if not white_first and colors_list[0] == '#FFFFFF':

    colors_list = colors_list[1:] # □ حذف رنگ سفید اول

    # □ اضافه کردن یک رنگ جدید برای حفظ تعداد طبقات

    if color_scheme == 'blue_red':

```



```

        colors_list.append('#67001f') # قرمز تیره

    elif color_scheme == 'viridis':

        colors_list.append('#fde725') # زرد

    elif color_scheme == 'plasma':

        colors_list.append('#f0f921') # زرد روشن

    elif color_scheme == 'terrain':

        colors_list.append('#543005') # قهوه‌ای تیره

    elif color_scheme == 'ocean':

        colors_list.append('#00204d') # آبی بسیار تیره

    elif color_scheme == 'rainbow':

        colors_list.append('#800080') # بنفش

    elif color_scheme == 'sunset':

        colors_list.append('#4b0082') # نیلی

    elif color_scheme == 'coolwarm':

        colors_list.append('#67001f') # قرمز تیره

    return colors_list

#
=====

=

# □ دریافت پالت رنگی انتخابی

#
=====

=

colors_list = get_color_palette(COLOR_SCHEME, num_classes, WHITE_FIRST_CLASS)

print(f"□ تنظیمات رنگ:")

```

```

print(f" □ طرح رنگی: {COLOR_SCHEME}")

print(f" □ تعداد رنگ‌ها: {len(colors_list)}")

# □ بررسی تطابق تعداد رنگ‌ها با تعداد طبقات
if len(colors_list) != num_classes:
    print(f"□ ( {len(colors_list)} ) با تعداد طبقات ( {num_classes} ) تطابق ندارد.")

    # □ استفاده کن matplotlib اگر تعداد رنگ‌ها کمتر است، از رنگ‌های پیش‌فرض
    if len(colors_list) < num_classes:
        try:
            colors_list = plt.cm.get_cmap(COLOR_SCHEME)(np.linspace(0, 1, num_classes))

            colors_list = [colors.rgb2hex(color) for color in colors_list]

            if not WHITE_FIRST_CLASS:
                colors_list = colors_list[1:]

        except:
            # □ استفاده کن viridis هم نبود، از matplotlib اگر طرح رنگی در
            colors_list = plt.cm.get_cmap('viridis')(np.linspace(0, 1, num_classes))

            colors_list = [colors.rgb2hex(color) for color in colors_list]

            if not WHITE_FIRST_CLASS:
                colors_list = colors_list[1:]

# □ برای نمایش رستر Norm و Colormap ایجاد
cmap_custom = colors.ListedColormap(colors_list)
norm_classified = colors.BoundaryNorm(class_bounds, cmap_custom.N)

#
=====
=

# □ بخش ۸: ساخت فریم‌های انیمیشن

```

```

#
=====
=
#
# □ یک فریم (تصویر) ایجاد می کند asc این بخش برای هر فایل
# □ :هر فریم شامل
# نقشه رستر با رنگ بندی طبقه ای ۱.
# در سمت راست Legend ۲.
# عنوان با ارتفاع ۳.
#
=====
=

# □ ایجاد پوشه خروجی در صورت عدم وجود
os.makedirs(output_folder, exist_ok=True)
print(f'\n□ پوشه خروجی: {output_folder}')

# □ لیست برای ذخیره تصاویر فریم ها
images = []

print(f'\n□ شروع ساخت فریم های انیمیشن')
print(f'□ تعداد کل فریم ها: {len(asc_files_sorted)}')

# □ و ساخت فریم asc پردازش هر فایل
for i, f in enumerate(asc_files_sorted):
    print(f'□ پردازش فریم {i+1}/{len(asc_files_sorted)}: {f}')

try:

```

```

# □ ایجاد مسیر کامل فایل asc
asc_path = os.path.join(input_folder, f)

# □ به عنوان رستر asc بارگذاری فایل
raster = arcpy.Raster(asc_path)
desc = arcpy.Describe(raster)

# □ NumPy تبدیل رستر به آرایه
arr = arcpy.RasterToNumPyArray(raster)

# □ NoData ایجاد ماسک برای سلول‌های
# استفاده می‌شود NoData ، معمولاً -۹۹۹۹ برای asc در فایل‌های
nodata_value = -9999
nodata_mask = (arr == nodata_value)

# □ برای نمایش NaN به NoData تبدیل مقادیر
display_arr = np.where(nodata_mask, np.nan, arr)

# □ Legend نقشه و Subplot با دو Figure ایجاد
fig, (ax_map, ax_legend) = plt.subplots(
    1, 2,
    figsize=(16, 8), # □ Figure اندازه کلی
    gridspec_kw={'width_ratios': [3, 1]}, # □ Legend نسبت عرض نقشه به
    dpi=150 # □ رزولوشن بالا
)

# =====

```

```

# □ الف: نمایش نقشه رستر
# =====

# □ سفارشی Colormap نمایش رستر با
im = ax_map.imshow(
    display_arr,
    cmap=cmap_custom,
    norm=norm_classified,
    extent=[desc.extent.XMin, desc.extent.XMax,
            desc.extent.YMin, desc.extent.YMax],
    origin='upper', # □ مبدأ مختصات از بالا
    interpolation='nearest' # □ بدون اینترپولیشن (حفظ وضوح)
)

# □ تنظیم محدوده ثابت برای همه فریم‌ها
ax_map.set_xlim(xmin, xmax)
ax_map.set_ylim(ymin, ymax)
ax_map.set_aspect('equal') # □ حفظ نسبت ابعاد

# □ عنوان نقشه (با استخراج ارتفاع از نام)
number = extract_number(f)
ax_map.set_title(
    f'Mineral Probability - Level: {number} m',
    fontsize=14,
    fontweight='bold',
    pad=10
)

```

```
# □ غیرفعال کردن محورها
ax_map.axis('off')

# =====

# □ در سمت راست Legend بخش ب: ساخت
# =====

ax_legend.set_title(
    'Legend',
    fontsize=14,
    fontweight='bold',
    pad=10
)

# □ بر اساس تعداد طبقات Legend تنظیمات ارتفاع و فاصله
if num_classes > 8:
    legend_height = 0.07
    legend_spacing = 0.09
    start_y = 0.9
else:
    legend_height = 0.08
    legend_spacing = 0.1
    start_y = 0.85

# □ Legend رسم مستطیل‌های رنگی
for j in range(num_classes):
    color = colors_list[j]
```

```

lower_bound = class_bounds[j]
upper_bound = class_bounds[j + 1]
label = f'{lower_bound:.1f} - {upper_bound:.1f}'

# □ رسم مستطیل رنگی
rect = plt.Rectangle(
    (0.1, start_y - j * legend_spacing),
    0.3,
    legend_height,
    facecolor=color,
    edgecolor='black',
    linewidth=0.5
)
ax_legend.add_patch(rect)

# □ اضافه کردن متن محدوده
ax_legend.text(
    0.45,
    start_y - j * legend_spacing + legend_height/2,
    label,
    fontsize=10 if num_classes <= 8 else 9,
    va='center',
    ha='left'
)

# □ اضافه کردن توضیحات تنظیمات
info_text = f'Classes: {num_classes}'
if WHITE_FIRST_CLASS:

```

```
info_text += " | First class: White"

ax_legend.text(
    0.05, 0.05,
    info_text,
    fontsize=9,
    style='italic',
    color='red',
    va='bottom',
    ha='left'
)

# □ تنظیمات ظاهری Legend
ax_legend.set_xlim(0, 1)
ax_legend.set_ylim(0, 1)
ax_legend.axis('off')

# □ برای کاهش حاشیه‌ها Layout تنظیم
plt.tight_layout(pad=2.0)

# =====

# □ به تصویر Figure بخش ج: تبدیل
# =====

# □ Figure به آرایه RGB رندر
fig.canvas.draw()

img_array = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)
img_array = img_array.reshape(fig.canvas.get_width_height()[::-1] + (3,))
```



```
# □ ذخیره تصویر در لیست
```

```
images.append(Image.fromarray(img_array))
```

```
# □ بستن Figure حافظه برای آزادسازی
```

```
plt.close(fig)
```

```
except Exception as e:
```

```
    print(f"□ خطا در پردازش {f}: {e}")
```

```
import traceback
```

```
traceback.print_exc()
```

```
plt.close('all') # □ ها در صورت خطا Figure بستن همه
```

```
#
```

```
=
```

```
# □ انیمیشن GIF بخش ۹: ساخت فایل
```

```
#
```

```
=
```

```
#
```

```
# □ تبدیل می کند GIF این بخش تمام فریم های ایجاد شده را به یک فایل.
```

```
# □ تنظیمات زمان بندی و کیفیت در این بخش کنترل می شوند.
```

```
#
```

```
=
```

```
if len(images) > 1:
```

```
    print(f"\n□ ساخت GIF انیمیشن...")
```

```

print(f"□ تعداد فریم‌ها: {len(images)}")

print(f"□ اندازه هر فریم: {images[0].size}")

# □ GIF ذخیره تصاویر به صورت
images[0].save(
    output_gif,          # □ مسیر فایل خروجی
    save_all=True,       # □ ذخیره همه فریم‌ها
    append_images=images[1:], # □ اضافه کردن فریم‌های بعدی
    duration=600,        # □ مدت نمایش هر فریم (میلی‌ثانیه)
    loop=0,              # □ تکرار بی‌نهایت (۰ = بی‌نهایت)
    optimize=True        # □ بهینه‌سازی فایل
)

# □ گزارش نهایی
print(f"\n□ انیمیشن با موفقیت ساخته شد")

print(f"□ مسیر فایل: {output_gif}")

print(f"□ تعداد فریم‌ها: {len(images)}")

print(f"□ اندازه هر فریم: {images[0].size}")

print(f"\n□ تنظیمات اعمال شده:")

print(f"□ طرح رنگی: {COLOR_SCHEME}")

print(f"□ تعداد طبقات: {num_classes}")

print(f"□ {WHITE_FIRST_CLASS if 'سفید' in COLOR_SCHEME else 'رنگی'} اولین طبقه")

print(f"□ {FILE_PREFIX if 'همه' in FILE_PREFIX else 'پیشوند'} فایل‌ها")

print(f"□ مدت هر فریم: {600}ms")

print(f"□ {0 if 'بی‌نهایت' in loop else 'تکرار'} حالت تکرار")

```

else:

```
print("\n□ وجود ندارد")
```

```
print("□ حداقل ۲ فریم برای ساخت انیمیشن نیاز است")
```

```
#
```

```
=====
```

```
=
```

```
# □ بخش ۱۰: نکات نهایی و راهنمایی
```

```
#
```

```
=====
```

```
=
```

```
#
```

```
# □ نکات مهم برای استفاده بهینه از این اسکریپت:
```

```
#
```

```
# 1. □ آماده‌سازی داده‌ها:
```

```
# در پوشه قرار دارند asc مطمئن شوید تمام فایل‌های -
```

```
# نام فایل‌ها باید شامل عدد (ارتفاع) باشند -
```

```
# فایل‌ها باید هم‌پوشانی مکانی داشته باشند -
```

```
#
```

```
# 2. □ تنظیمات رنگ:
```

```
# برای داده‌های رایج توصیه می‌شود 'blue_red' طرح -
```

```
# استفاده کنید 'coolwarm' برای داده‌های دمایی از -
```

```
# استفاده کنید 'terrain' برای داده‌های ارتفاعی از -
```

```
#
```

```
# 3. □ بهینه‌سازی:
```

```
# - را کاهش دهید DPI برای تعداد زیاد فایل‌ها،
```

```

# می‌شود Legend تعداد طبقات کمتر باعث خوانایی بهتر -
# مناسب برای نمایش آموزشی است duration=600 -
#
# 4. □ عیب‌یابی:
# را بررسی کنید NoData اگر فایلی نمایش داده نمی‌شود، مقدار -
# نادرست است، محدوده مقادیر را بررسی کنید Legend اگر -
# کلی را بررسی کنید Extent اگر اندازه فریم‌ها متفاوت است، -
#
# 5. □ تفسیر نتایج:
# رنگ‌های آبی معمولاً نشان‌دهنده مقادیر پایین‌تر هستند -
# رنگ‌های قرمز نشان‌دهنده مقادیر بالاتر هستند -
# هستند NoData مناطق سفید یا شفاف معمولاً -
#
# 6. □ asc نکات فنی فایل‌های:
# استاندارد داشته باشند Header باید asc فایل‌های -
# با -۹۹۹۹ مشخص می‌شود NoData معمولاً -
# باشد arcpy فایل باید قابل خواندن توسط -

```

تئوری دمپستر شیفر

```

import arcpy
import numpy as np
import os
import glob
import re

```

```
from arcpy.sa import *
```

```
# --- تنظیمات اولیه ---
```

```
"""
```

□ پارامترهای ورودی - بخش مسیرها و تنظیمات پایه

self\_attention\_path (رشته):

Self-Attention خروجی از تحلیل ASC مسیر پوشه حاوی فایل‌های

مثال: "M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal\Self\_Attention"

باشد Mineral\_Probability\_XXX.asc با نام‌های مانند asc. این پوشه باید حاوی فایل‌های

gdb\_paths (لیست رشته‌ها):

های حاوی نتایج روش‌های دیگر Geodatabase لیست مسیرهای

مثال: [

"M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal\SVM.gdb",

"M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal\Bayesian.gdb"

]

باید حاوی رسترهای احتمالی معدنی برای ترازهای مختلف باشد GDB هر

elevations (لیست اعداد):

لیست ترازهای ارتفاعی برای پردازش

تراز از ۸۷۵ تا ۱۱۲۵ با گام ۱۲.۵ متر 21 - [875 + i\*12.5 for i in range(21)] مثال:

output\_gdb (رشته):

خروجی برای ذخیره نتایج ترکیب دمپستر-شیفر Geodatabase مسیر

مثال: "M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal\DempsterShafer\_Results.gdb"

به طور خودکار ایجاد می‌شود اگر وجود نداشته باشد GDB این

"""

self\_attention\_path = r"M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal\Self\_Attention" # پوشه با فایل‌های  
ASC

# برای دیگر روش‌ها GDB مسیرهای

```
gdb_paths = [  
    r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\SVM.gdb",  
    r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Bayesian.gdb"  
]
```

# ترازهای ارتفاعی (از ۸۷۵ تا ۱۱۲۵ با گام ۱۲,۵ متر)

elevations = [875 + i\*12.5 for i in range(21)] # تراز 21

# مسیر خروجی

output\_gdb = r"M:\Mahmood\Survey\WGIS\P24\_GOSAL\Gosal\DempsterShafer\_Results.gdb"

# خروجی اگر وجود ندارد Geodatabase ایجاد

if not arcpy.Exists(output\_gdb):

arcpy.CreateFileGDB\_management(os.path.dirname(output\_gdb), os.path.basename(output\_gdb))

# تنظیم محیط پردازش

arcpy.env.workspace = output\_gdb

arcpy.env.overwriteOutput = True

arcpy.env.cellSize = 1 # رزولوشن ۱ متر

arcpy.env.extent = "MAXOF"

```
arcpy.env.snapRaster = None
```

```
# --- پارامترهای دمپستر-شیفر (قابل تنظیم) ---
```

```
"""
```

□ پارامترهای ترکیب دمپستر-شیفر:

class\_masses (دیکشنری):

توابع جرم اولیه برای هر کلاس:

- "mineral": (پیش‌فرض: ۰,۸۵): تابع جرم برای کلاس معدنی

- "non\_mineral": (پیش‌فرض: ۰,۱۰): تابع جرم برای کلاس غیرمعدنی

- "uncertainty": (پیش‌فرض: ۰,۰۵): تابع جرم برای عدم قطعیت

مجموع باید ۱,۰ باشد

probability\_threshold (عدد):

آستانه احتمالی برای طبقه‌بندی نهایی (پیش‌فرض: ۰,۵)

مقادیر بالاتر از این آستانه به عنوان معدنی طبقه‌بندی می‌شوند

```
"""
```

```
class_masses = {
```

```
    "mineral": 0.85,
```

```
    "non_mineral": 0.10,
```

```
    "uncertainty": 0.05
```

```
}
```

```
probability_threshold = 0.5
```

# --- توابع کمکی ---

```
def extract_elevation_from_filename(filename):
```

```
    """
```

```
    □ extract_elevation_from_filename تابع
```

```
    وظیفه: استخراج تراز ارتفاعی از نام فایل
```

```
    پارامترهای ورودی:
```

```
    - filename (رشته): نام فایل (با یا بدون مسیر)
```

```
    خروجی:
```

```
    در صورت عدم موفقیت None عدد: تراز ارتفاعی استخراج شده یا -
```

```
    """
```

```
    حذف پسوند
```

```
    name_without_ext = os.path.splitext(filename)[0]
```

```
    الگوهای مختلف برای استخراج عدد
```

```
    patterns = [
```

```
        r'(\d+_\d+)', # 912_5
```

```
        r'(\d+.\d+)', # 912.5
```

```
        r'(\d+)' # 912
```

```
    ]
```

```
    for pattern in patterns:
```

```
        matches = re.findall(pattern, name_without_ext)
```

```
        if matches:
```

```
            # استفاده از آخرین عدد پیدا شده
```



```
elevation_str = matches[-1]

# تبدیل به عدد

try:

    # اگر فرمت با زیرخط است
    if '_' in elevation_str:
        elevation = float(elevation_str.replace('_', '.'))
    else:
        elevation = float(elevation_str)

    # بررسی که تراز در محدوده مورد نظر باشد
    if 870 <= elevation <= 1130:
        return elevation

except:

    continue

return None
```

```
def find_files_for_elevation(elevation):
```

```
    """
```

```
    □ find_files_for_elevation تابع
```

وظیفه: پیدا کردن فایل‌های مربوط به یک تراز در همه منابع ورودی

پارامترهای ورودی:

(عدد): تراز ارتفاعی مورد جستجو - elevation

خروجی:

دیکشنری: کلید=نام روش، مقدار=مسیر فایل -

"""

```
files_dict = {}
```

# 1. فایل‌های (ASC Self\_Attention جستجو در پوشه 1.

```
asc_files = glob.glob(os.path.join(self_attention_path, "*.asc"))
```

```
for asc_file in asc_files:
```

```
    filename = os.path.basename(asc_file)
```

```
    file_elevation = extract_elevation_from_filename(filename)
```

```
    if file_elevation and abs(file_elevation - elevation) < 0.1: # اختلاف کمتر از ۰,۱
```

```
        files_dict["Self_Attention"] = asc_file
```

```
        print(f" ✓ Self_Attention: {filename}")
```

```
        break
```

# 2. های دیگر GDB جستجو در 2.

```
for gdb_path in gdb_paths:
```

```
    if not arcpy.Exists(gdb_path):
```

```
        print(f" ✗ {os.path.basename(gdb_path)}: GDB وجود ندارد")
```

```
        continue
```

```
gdb_name = os.path.basename(gdb_path).replace('.gdb', '')
```

```
try:
```

```
    # GDB به workspace تغییر
```

```
    arcpy.env.workspace = gdb_path
```

# لیست تمام رسترها در GDB

```
rasters = arcpy.ListRasters()
```

```
found_raster = None
```

```
for raster_name in rasters:
```

```
    # استخراج تراز از نام رستر
```

```
    raster_elevation = extract_elevation_from_filename(raster_name)
```

```
    if raster_elevation and abs(raster_elevation - elevation) < 0.1:
```

```
        found_raster = os.path.join(gdb_path, raster_name)
```

```
        print(f" ✓ {gdb_name}: {raster_name}")
```

```
        break
```

```
if found_raster:
```

```
    files_dict[gdb_name] = found_raster
```

```
else:
```

```
    print(f" ✗ {gdb_name}: رستری برای تراز {elevation} یافت نشد")
```

```
except Exception as e:
```

```
    print(f" ✗ {gdb_name}: خطا در خواندن {str(e)}")
```

```
return files_dict
```

```
def read_and_prepare_raster(raster_path, target_cellsize=1.0):
```

```
    """
```

```
    □ تابع read_and_prepare_raster
```

وظیفه: خواندن رستر و آماده‌سازی آن برای پردازش

پارامترهای ورودی:

- raster\_path: رشته: مسیر رستر ورودی

- target\_cellsize: (عدد): رزولوشن هدف (پیش‌فرض: ۱,۰ متر)

خروجی:

در صورت خطا None دیکشنری: اطلاعات رستر یا -

"""

try:

```
raster = arcpy.Raster(raster_path)
```

```
# اطلاعات پایه
```

```
arr = arcpy.RasterToNumPyArray(raster, nodata_to_value=-9999)
```

```
lower_left = arcpy.Point(raster.extent.XMin, raster.extent.YMin)
```

```
spatial_ref = raster.spatialReference
```

```
# اگر رزولوشن متفاوت است، ری‌سمپل
```

```
if abs(raster.meanCellWidth - target_cellsize) > 0.01:
```

```
    print(f" {target_cellsize} متر به {raster.meanCellWidth:.2f} ری‌سمپل از ")
```

```
# ذخیره موقت
```

```
temp_raster = arcpy.NumPyArrayToRaster(
```

```
    arr, lower_left, raster.meanCellWidth, raster.meanCellHeight,
```

```
    value_to_nodata=-9999
```

```
)
```

```
# ری‌سمپل
```

```

resampled_path = "in_memory/resampled"
resampled = arcpy.Resample_management(
    temp_raster, resampled_path, str(target_cellsize), "BILINEAR"
)

# خواندن مجدد
arr = arcpy.RasterToNumPyArray(resampled, nodata_to_value=-9999)
cell_width = target_cellsize
cell_height = target_cellsize

# پاک کردن رستر موقت
arcpy.Delete_management(resampled_path)
else:
    cell_width = raster.meanCellWidth
    cell_height = raster.meanCellHeight

metadata = {
    'array': arr,
    'lower_left': lower_left,
    'cell_width': cell_width,
    'cell_height': cell_height,
    'spatial_ref': spatial_ref,
    'extent': raster.extent
}

return metadata

except Exception as e:

```

```
print(f"خطا در خواندن {raster_path}: {str(e)}")
```

```
return None
```

```
def normalize_raster_values(arr, nodata_value=-9999):
```

```
    """
```

```
    □ normalize_raster_values تابع
```

وظیفه: نرمالایز کردن مقادیر رستر به محدوده ۰ تا ۱

پارامترهای ورودی:

- arr (آرایه numpy): آرایه ورودی

- nodata\_value (پیش فرض: -۹۹۹۹) مقدار NoData (عدد): مقدار

خروجی:

آرایه نرمالایز شده: numpy آرایه -

```
    """
```

```
if arr is None:
```

```
    return None
```

```
valid_mask = arr != nodata_value
```

```
if not np.any(valid_mask):
```

```
    return arr
```

```
valid_values = arr[valid_mask]
```

```
arr_min = np.min(valid_values)
```

```
arr_max = np.max(valid_values)
```

```
arr_normalized = np.full_like(arr, nodata_value, dtype=np.float32)
```

```

if arr_max - arr_min > 0:
    arr_normalized[valid_mask] = (valid_values - arr_min) / (arr_max - arr_min)
else:
    arr_normalized[valid_mask] = 0.5

return arr_normalized

```

```
def align_rasters(rasters_metadata):
```

```
    """
```

```
    □ align_rasters تابع
```

وظیفه: هم‌راستا کردن رسترها از نظر موقعیت مکانی و ابعاد

پارامترهای ورودی:

- rasters\_metadata (لیست): لیست دیکشنری‌های اطلاعات رسترها

خروجی:

در صورت خطا (None, None) یا (aligned\_arrays, reference\_metadata) :تاپل -

```
    """
```

```
if not rasters_metadata:
```

```
    return None, None
```

```
# پیدا کردن بیشترین گستره
```

```
all_extents = [meta['extent'] for meta in rasters_metadata if meta]
```

```
if not all_extents:
```

```
    return None, None
```

```
# مختصات max و min پیدا کردن
```

```
xmin = min(extent.XMin for extent in all_extents)
```

```
ymin = min(extent.YMin for extent in all_extents)
```

```
xmax = max(extent.XMax for extent in all_extents)
```

```
ymax = max(extent.YMax for extent in all_extents)
```

```
# سلول سائز (از اولین رستر)
```

```
cell_size = rasters_metadata[0]['cell_width']
```

```
# محاسبه ابعاد جدید
```

```
cols = int(round((xmax - xmin) / cell_size))
```

```
rows = int(round((ymax - ymin) / cell_size))
```

```
print(f" گستره ترکیبی: {xmin:.1f}, {ymin:.1f} تا {xmax:.1f}, {ymax:.1f}")
```

```
print(f" ابعاد جدید: {rows} × {cols}")
```

```
# ایجاد آرایه‌های جدید
```

```
aligned_arrays = []
```

```
reference_metadata = None
```

```
for i, meta in enumerate(rasters_metadata):
```

```
    if meta is None:
```

```
        continue
```

```
# ایجاد آرایه خالی
```

```
new_arr = np.full((rows, cols), -9999, dtype=np.float32)
```



```

# برای این رستر offset محاسبه
x_offset = int(round((meta['extent'].XMin - xmin) / cell_size))
y_offset = int(round((meta['extent'].YMin - ymin) / cell_size))

# کپی داده‌ها
src_rows, src_cols = meta['array'].shape

# محدوده‌های قابل کپی
dest_row_start = max(0, y_offset)
dest_row_end = min(rows, y_offset + src_rows)
dest_col_start = max(0, x_offset)
dest_col_end = min(cols, x_offset + src_cols)

src_row_start = max(0, -y_offset)
src_row_end = src_rows - max(0, (y_offset + src_rows) - rows)
src_col_start = max(0, -x_offset)
src_col_end = src_cols - max(0, (x_offset + src_cols) - cols)

# کپی داده‌ها
new_arr[dest_row_start:dest_row_end, dest_col_start:dest_col_end] = \
    meta['array'][src_row_start:src_row_end, src_col_start:src_col_end]

aligned_arrays.append(new_arr)

if reference_metadata is None:
    reference_metadata = {
        'lower_left': arcpy.Point(xmin, ymin),
        'cell_width': cell_size,

```

```
'cell_height': cell_size,  
'spatial_ref': meta['spatial_ref'],  
'rows': rows,  
'cols': cols  
}
```

```
return aligned_arrays, reference_metadata
```

```
def dempster_shafer_combination(rasters_arrays, class_masses):
```

```
    """
```

```
    □ dempster_shafer_combination تابع
```

وظیفه: ترکیب چند رستر با استفاده از تئوری دمپستر-شیفر

پارامترهای ورودی:

- rasters\_arrays: لیست آرایه‌های نرمالایز شده رسترها

- class\_masses: توابع جرم برای کلاس‌ها

خروجی:

در صورت خطا None احتمال نهایی کلاس معدنی یا numpy آرایه -

```
    """
```

```
if not rasters_arrays:
```

```
    return None
```

```
num_rasters = len(rasters_arrays)
```

```
shape = rasters_arrays[0].shape
```

```
# ماتریس‌های توابع جرم
```

```

m_mineral = np.zeros(shape, dtype=np.float32)
m_non_mineral = np.zeros(shape, dtype=np.float32)
m_uncertainty = np.zeros(shape, dtype=np.float32)

# اختصاص توابع جرم بر اساس مقادیر رسترها
for i, raster_arr in enumerate(rasters_arrays):
    weight = 1.0 / num_rasters

    # تابع جرم برای معدنی: متناسب با مقدار رستر
    m_mineral += raster_arr * class_masses["mineral"] * weight

    # تابع جرم برای غیرمعدنی: عکس مقدار رستر
    m_non_mineral += (1 - raster_arr) * class_masses["non_mineral"] * weight

    # عدم قطعیت ثابت
    m_uncertainty += class_masses["uncertainty"] * weight

# نرمالایز کردن توابع جرم
total_mass = m_mineral + m_non_mineral + m_uncertainty
total_mass[total_mass == 0] = 1e-10

m_mineral_norm = m_mineral / total_mass
m_non_mineral_norm = m_non_mineral / total_mass
m_uncertainty_norm = m_uncertainty / total_mass

# قاعده ترکیب دمپستر
combined_belief = m_mineral_norm

```

```

combined_disbelief = m_non_mineral_norm
combined_uncertainty = m_uncertainty_norm

for i in range(1, num_rasters):

    K = combined_belief * m_non_mineral_norm + combined_disbelief * m_mineral_norm

    new_belief = (combined_belief * m_mineral_norm + combined_belief * m_uncertainty_norm +
                  combined_uncertainty * m_mineral_norm) / (1 - K + 1e-10)

    new_disbelief = (combined_disbelief * m_non_mineral_norm + combined_disbelief *
m_uncertainty_norm +
                    combined_uncertainty * m_non_mineral_norm) / (1 - K + 1e-10)

    new_uncertainty = (combined_uncertainty * m_uncertainty_norm) / (1 - K + 1e-10)

    combined_belief = new_belief
    combined_disbelief = new_disbelief
    combined_uncertainty = new_uncertainty

# احتمال نهایی برای کلاس معدنی

final_probability = combined_belief / (combined_belief + combined_disbelief + 1e-10)

return final_probability

# --- تابع اصلی پردازش ---

def process_elevation_levels():
    """

```

□ تابع process\_elevation\_levels

وظیفه: پردازش اصلی تمام ترازهای ارتفاعی

خروجی:

لیست: نتایج پردازش هر تراز -

"""

```
results_log = []
```

```
# به خروجی workspace برگرداندن
```

```
arcpy.env.workspace = output_gdb
```

```
for elevation in elevations:
```

```
    print(f'\n{'='*60}')
```

```
    print(f'پردازش تراز ارتفاعی {elevation} متر')
```

```
    print(f{'='*60}')
```

```
# پیدا کردن فایل‌های مربوط به این تراز
```

```
    print("...جستجوی فایل‌ها")
```

```
    files_dict = find_files_for_elevation(elevation)
```

```
    if len(files_dict) < 2:
```

```
        print(f'({len(files_dict)} فایل) فایل‌های کافی یافت نشد: {elevation} رد شدن تراز')
```

```
        results_log.append({
```

```
            'elevation': elevation,
```

```
            'status': 'رد شده',
```

```
            'found_files': len(files_dict),
```

```

'methods': list(files_dict.keys())

})

continue

try:

    # خواندن و آماده‌سازی رسترها

    print("خواندن و آماده‌سازی رسترها...")

    rasters_metadata = []
    method_names = []

    for method_name, file_path in files_dict.items():

        print(f"پردازش {method_name}...")

        metadata = read_and_prepare_raster(file_path, target_cellsize=1.0)

        if metadata:

            # نرمالایز کردن

            metadata['array'] = normalize_raster_values(metadata['array'])

            rasters_metadata.append(metadata)

            method_names.append(method_name)

        else:

            print(f"خطا در پردازش {method_name}")

    if len(rasters_metadata) < 2:

        print(f"رد شدن: رسترهای کافی آماده نشدند")

        continue

    # هم‌راستا کردن رسترها

```

```

print("هم‌راستا کردن رسترها...")

aligned_arrays, reference_metadata = align_rasters(rasters_metadata)

if aligned_arrays is None:

    print("خطا در هم‌راستا کردن رسترها ")

    continue

# ترکیب دمپستر-شیفر
print("...ترکیب با روش دمپستر-شیفر")

combined_probability = dempster_shafer_combination(aligned_arrays, class_masses)

if combined_probability is None:

    print("خطا در ترکیب رسترها ")

    continue

# ذخیره رستر خروجی
print("...ذخیره رستر خروجی")

elevation_str = str(elevation).replace('.', '_')
output_name = f"DS_Combined_{elevation_str}"
output_path = os.path.join(output_gdb, output_name)

# حذف اگر از قبل وجود دارد

if arcpy.Exists(output_path):

    arcpy.Delete_management(output_path)

# ایجاد رستر از آرایه
output_raster = arcpy.NumPyArrayToRaster(

```

```

combined_probability,
reference_metadata['lower_left'],
reference_metadata['cell_width'],
reference_metadata['cell_height'],
value_to_nodata=-9999
)

output_raster.save(output_path)

# تعیین سیستم مختصات
if reference_metadata['spatial_ref']:
    arcpy.DefineProjection_management(output_path, reference_metadata['spatial_ref'])

# محاسبه آمارها
valid_mask = combined_probability != -9999
if np.any(valid_mask):
    valid_values = combined_probability[valid_mask]
    mean_val = np.mean(valid_values)
    max_val = np.max(valid_values)
    min_val = np.min(valid_values)
    std_val = np.std(valid_values)
    high_prob_pixels = np.sum(valid_values > probability_threshold)
    high_prob_percent = (high_prob_pixels / len(valid_values)) * 100

    print(f"\nآمارهای خروجی:")
    print(f"میانگین احتمال: {mean_val:.4f}")
    print(f"حد اقل/حداکثر: {min_val:.4f} / {max_val:.4f}")

```



```

print(f"انحراف معیار: {std_val:.4f}")

print(f"> {probability_threshold}: {high_prob_percent:.1f}%")

print(f"تعداد پیکسل‌های معتبر: {len(valid_values):,}")

else:

    mean_val = max_val = min_val = std_val = 0

    high_prob_percent = 0

    print("هشدار: هیچ مقدار معتبری در خروجی وجود ندارد")

# ثبت در لاگ

results_log.append({

    'elevation': elevation,

    'output_name': output_name,

    'mean_probability': float(mean_val),

    'min_probability': float(min_val),

    'max_probability': float(max_val),

    'std_probability': float(std_val),

    'high_prob_percent': float(high_prob_percent),

    'pixel_count': int(np.sum(valid_mask)),

    'status': 'موفق',

    'methods': method_names,

    'input_files': [os.path.basename(files_dict[m]) for m in method_names]

})

print(f"\n✓ ذخیره شد: {output_name}")

except Exception as e:

    error_msg = str(e)

```

```
print(f" ✖ خطا در پردازش تراز {elevation}: {error_msg}")
```

```
import traceback
```

```
traceback.print_exc()
```

```
results_log.append({  
    'elevation': elevation,  
    'status': 'خطا',  
    'error': error_msg,  
    'methods': list(files_dict.keys())  
})
```

```
return results_log
```

```
def create_summary_outputs():
```

```
    """
```

```
    □ تابع create_summary_outputs
```

```
    وظیفه: ایجاد خروجی‌های خلاصه از تمام ترازهای پردازش شده
```

```
    """
```

```
print(f"\n{'='*60}")
```

```
print("ایجاد خروجی‌های خلاصه")
```

```
print(f"{'='*60}")
```

```
try:
```

```
    arcpy.env.workspace = output_gdb
```

```
    ds_rasters = arcpy.ListRasters("DS_Combined_*")
```

```
    if not ds_rasters or len(ds_rasters) < 2:
```

```
print(f"تعداد رسترهای ترکیبی کافی نیست ({len(ds_rasters) if ds_rasters else 0})")

return

print(f"تعداد رسترهای ترکیبی: {len(ds_rasters)}")

# ایجاد لیست رسترها
raster_list = []

for raster_name in ds_rasters:
    raster_path = os.path.join(output_gdb, raster_name)
    raster_list.append(raster_path)

# میانگین کلی
print("محاسبه میانگین کلی")

overall_mean = CellStatistics(raster_list, "MEAN", "DATA")
overall_mean_path = os.path.join(output_gdb, "DS_Overall_Mean")
overall_mean.save(overall_mean_path)
print(f"✓ میانگین کلی: DS_Overall_Mean")

# حداکثر کلی
print("محاسبه حداکثر کلی")

overall_max = CellStatistics(raster_list, "MAXIMUM", "DATA")
overall_max_path = os.path.join(output_gdb, "DS_Overall_Max")
overall_max.save(overall_max_path)
print(f"✓ حداکثر کلی: DS_Overall_Max")

# طبقه‌بندی نهایی
```

```
print("ایجاد طبقه‌بندی نهایی...")
```

```
final_classification = Con(overall_mean >= probability_threshold, 1, 0)
```

```
final_class_path = os.path.join(output_gdb, "DS_Final_Classification")
```

```
final_classification.save(final_class_path)
```

```
print(f"✓ طبقه‌بندی نهایی: DS_Final_Classification")
```

```
# عدم قطعیت
```

```
print("محاسبه عدم قطعیت...")
```

```
overall_min = CellStatistics(raster_list, "MINIMUM", "DATA")
```

```
uncertainty = overall_max - overall_min
```

```
uncertainty_path = os.path.join(output_gdb, "DS_Uncertainty")
```

```
uncertainty.save(uncertainty_path)
```

```
print(f"✓ عدم قطعیت: DS_Uncertainty")
```

```
except Exception as e:
```

```
print(f"خطا در ایجاد خروجی‌های خلاصه: {str(e)}")
```

```
# --- اجرای اصلی ---
```

```
if __name__ == "__main__":
```

```
    """
```

```
    □ نقطه شروع برنامه
```

```
    وظیفه: اجرای کامل پردازش دمپستر-شیفر برای ترکیب خروجی‌های مختلف
```

```
    """
```

```
print("شروع پردازش دمپستر-شیفر")
```

```
print(f"منابع ورودی:")
```

```

print(f" 1. Self_Attention (پوشه): {self_attention_path}")

for i, gdb_path in enumerate(gdb_paths, 2):
    print(f" {i}. {os.path.basename(gdb_path)}: {gdb_path}")

print(f"\nترازهای ارتفاعی: {len(elevations)} (۸۷۵ تا ۱۱۲۵ متر)")

print(f"خروجی: {output_gdb}")

print(f"رزولوشن: {arcpy.env.cellSize} متر")

print(f"{'='*60}")

# ذخیره تنظیمات اولیه

original_workspace = arcpy.env.workspace

try:
    # پردازش اصلی

    log_results = process_elevation_levels()

    # ایجاد خروجی‌های خلاصه

    successful = sum(1 for log in log_results if log['status'] == 'موفق')

    if successful >= 2:
        create_summary_outputs()

    # نمایش نتایج

    print(f"\n{'='*60}")

    print("خلاصه نتایج")

    print(f"{'='*60}")

    successful = sum(1 for log in log_results if log['status'] == 'موفق')

```

```

failed = sum(1 for log in log_results if log['status'] == 'خطا')

skipped = sum(1 for log in log_results if log['status'] == 'رد شده')

print(f"□ موفق: {successful}")

print(f"□ رد شده: {skipped}")

print(f"□ خطا: {failed}")

if successful > 0:
    print(f"\nترازهای پردازش شده موفق:")

    for log in log_results:
        if log['status'] == 'موفق':
            print(f"    تراز {log['elevation']}: {log['output_name']}")

            print(f"    روش‌ها: {' '.join(log['methods'])}")

            print(f"    میانگین احتمال: {log['mean_probability']:.4f}")

# ذخیره لاگ

import csv

log_file = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\dempster_shafer_log.csv"

with open(log_file, 'w', newline="", encoding='utf-8-sig') as f:
    if log_results:
        all_keys = set()

        for log in log_results:
            all_keys.update(log.keys())

        fieldnames = sorted(all_keys)

```

```

writer = csv.DictWriter(f, fieldnames=fieldnames)

writer.writeheader()

writer.writerows(log_results)

print(f"\n□ فایل لاگ ذخیره شد: {log_file}")

print(f"□ خروجی‌ها در {output_gdb}")

finally:

    # بازگردانی workspace

    arcpy.env.workspace = original_workspace

print(f"\n{'='*60}")

print("!پردازش کامل شد")

print(f"{'='*60}")

```

تحلیل جامع سیستم مدل‌سازی سه‌بعدی پتانسیل معدنی

```

import os

import arcpy

from arcpy import env

from arcpy.sa import *

import numpy as np

import pandas as pd

from scipy import interpolate

import re

import warnings

import hashlib

import shutil

```

```

import matplotlib.pyplot as plt

from matplotlib.colors import Normalize, LinearSegmentedColormap

from mpl_toolkits.mplot3d.art3d import Poly3DCollection

from matplotlib.ticker import FuncFormatter, ScalarFormatter

warnings.filterwarnings('ignore')

#
=====

# تنظیمات دستی - این بخش را ویرایش کنید □
#
=====

# تنظیمات ورودی - یکی از این دو را انتخاب کنید 1.

INPUT_TYPE = "GDB" # یا "FOLDER" - "GDB" برای Geodatabase، "FOLDER" برای پوشه رسترها

# اگر INPUT_TYPE = "GDB" باشد:

input_gdb = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\DempsterShafer_Final.gdb"

# اگر INPUT_TYPE = "FOLDER" باشد:

input_folder = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\RasterFiles" # مسیر پوشه حاوی فایل‌های
رستر

RASTER_EXTENSIONS = ['.tif', '.tiff', '.asc', '.img', '.bil'] # پسوندهای قابل خواندن

# 2. تنظیمات خروجی

output_folder = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\3D\DempsterShafer"

output_name = "DempsterShafer_3D"

```



# 3. فیلتر رسترها

FILTER\_CHAR = "Belief" # فقط رسترهای حاوی این کلمه (اختیاری - برای حذف فیلتر "" بگذارید)

# 4. تنظیمات مدل بلوکی - ابعاد دقیق ۱۲,۵ متر

CELL\_SIZE\_XY = 12.5 # (متر) - دقیقاً ۱۲,۵ X و Y اندازه سلول در جهت

CELL\_SIZE\_Z = 12.5 # (متر) - دقیقاً ۱۲,۵ Z اندازه سلول در جهت

# 5. تنظیمات ارتفاع

ELEVATION\_LEVELS = [

875.0, 887.5, 900.0, 912.5, 925.0, 937.5, 950.0, 962.5, 975.0, 987.5,

1000.0, 1012.5, 1025.0, 1037.5, 1050.0, 1062.5, 1075.0, 1087.5, 1100.0, 1112.5, 1125.0

]

# 6. تنظیمات برش (Clip)

CLIP\_ENABLED = True # باشد، برش اعمال می‌شود True اگر

CLIP\_X\_MIN = 306300.0 # برای برش (متر) X (Easting) حداقل مختصات

CLIP\_X\_MAX = 307857.0 # برای برش (متر) X (Easting) حداکثر مختصات

CLIP\_Y\_MIN = 3928820.0 # برای برش (متر) Y (Northing) حداقل مختصات

CLIP\_Y\_MAX = 3929549.0 # برای برش (متر) Y (Northing) حداکثر مختصات

# 7. تنظیمات پیشرفته

FLIP\_RASTER\_VERTICAL = True # برای اصلاح جهت رسترها

INTERPOLATION\_METHOD = 'linear' # روش درونیابی

# 8. تنظیمات سیستم مختصات

```
COORDINATE_SYSTEM = "WGS_1984_UTM_Zone_40N" # UTM Zone 40N سیستم مختصات
```

# 9. تنظیمات کلاس بندی - رنج های مورد نظر

```
CLASS_RANGES = [  
    ('Class_0_to_1', 0.0, 1.0, 'Values 0.0 to 1.0'),  
    ('Class_0_2_to_1', 0.2, 1.0, 'Values 0.2 to 1.0'),  
    ('Class_0_4_to_1', 0.4, 1.0, 'Values 0.4 to 1.0'),  
    ('Class_0_6_to_1', 0.6, 1.0, 'Values 0.6 to 1.0'),  
    ('Class_0_8_to_1', 0.8, 1.0, 'Values 0.8 to 1.0'),  
]
```

# 10. تنظیمات تصویر سازی

```
IMAGE_DPI = 300 # کیفیت تصویر
```

# 11. تنظیمات بلوک های سه بعدی

```
BLOCK_ALPHA = 0.9 # شفافیت بلوک ها
```

```
BLOCK_EDGE_COLOR = 'black' # رنگ لبه بلوک ها
```

```
BLOCK_EDGE_WIDTH = 0.1 # ضخامت لبه
```

```
#
```

```
=
```

```
# □ بدون سفیدی blue_red - پالت رنگی سفارشی
```

```
#
```

```
=
```

```
def create_custom_colormaps():
```

```
"""بدون سفیدی blue_red ایجاد و ثبت پالت رنگی سفارشی"""
```

```
# رنگ‌های شروع (آبی) و پایان (قرمز) - بدون سفیدی در وسط
```

```
simple_blue_red_colors = [
```

```
    '#053061', # آبی تیره شروع
```

```
    '#2166ac', # آبی
```

```
    '#4393c3', # آبی روشن
```

```
    '#92c5de', # آبی خیلی روشن
```

```
    '#f4a582', # قرمز روشن
```

```
    '#d6604d', # قرمز متوسط
```

```
    '#b2182b' # قرمز تیره پایان
```

```
]
```

```
# استفاده از نام فایل برای جلوگیری از تداخل colormap نام
```

```
cmap_name = 'blue_red_direct_no_white'
```

```
# colormap بررسی وجود
```

```
try:
```

```
    # قبلاً وجود دارد، آن را بازگردان colormap اگر
```

```
    if cmap_name in plt.colormaps():
```

```
        print(f" Using existing colormap: {cmap_name}")
```

```
        return {
```

```
            'blue_red': plt.get_cmap(cmap_name),
```

```
            'start_color': '#053061',
```

```
            'end_color': '#b2182b',
```

```
            'colors': simple_blue_red_colors,
```

```

        'description': 'گرادیان مستقیم آبی-قرمز بدون سفیدی'
    }
except:
    pass

# جدید colormap ایجاد
try:
    blue_red_cmap = LinearSegmentedColormap.from_list(
        cmap_name,
        simple_blue_red_colors,
        N=256
    )

    # colormap در matplotlib ثبت
    plt.cm.register_cmap(name=cmap_name, cmap=blue_red_cmap)
    print(f" Created new colormap: {cmap_name}")

except ValueError as e:
    # قبلاً ثبت شده، از همان استفاده کن اگر
    print(f" Colormap already exists, using existing: {cmap_name}")
    if cmap_name in plt.colormaps():
        blue_red_cmap = plt.get_cmap(cmap_name)
    else:
        # اگر ثبت نشده، ایجاد کن
        blue_red_cmap = LinearSegmentedColormap.from_list(
            'temp_blue_red',
            simple_blue_red_colors,
            N=256

```

)

```
return {  
    'blue_red': blue_red_cmap,  
    'start_color': '#053061',  
    'end_color': '#b2182b',  
    'colors': simple_blue_red_colors,  
    'description': 'گرادیان مستقیم آبی-قرمز بدون سفیدی'  
}
```

```
#  
=====
```

```
=
```

```
# □ کلاس اصلی مدل ساز
```

```
#  
=====
```

```
=
```

```
class GDB3DModeler:
```

```
    def __init__(self):
```

```
        """
```

یا پوشه GDB کلاس برای ساخت مدل بلوکی سه بعدی از رسترهای موجود در

```
        """
```

```
        self.input_type = INPUT_TYPE
```

```
        self.output_folder = output_folder
```

```
        self.output_name = output_name
```

```
        self.class_ranges = CLASS_RANGES
```

```
        # تنظیمات برش
```

```

self.clip_enabled = CLIP_ENABLED

self.clip_x_min = CLIP_X_MIN

self.clip_x_max = CLIP_X_MAX

self.clip_y_min = CLIP_Y_MIN

self.clip_y_max = CLIP_Y_MAX


# تنظیم ورودی بر اساس نوع

if self.input_type == "GDB":

    self.input_path = input_gdb

    env.workspace = self.input_path

elif self.input_type == "FOLDER":

    self.input_path = input_folder

    # را به پوشه خروجی موقت تنظیم می‌کنیم workspace برای پوشه،
    temp_workspace = os.path.join(output_folder, "temp_workspace")

    if not os.path.exists(temp_workspace):

        os.makedirs(temp_workspace)

        env.workspace = temp_workspace

else:

    raise ValueError(f"نوع ورودی نامعتبر '{self.input_type}' یا 'GDB' یا 'FOLDER' باشد.")


env.overwriteOutput = True


# UTM Zone 40N تنظیم سیستم مختصات

self.spatial_ref = arcpy.SpatialReference(32640) # WGS84 UTM Zone 40N


# ایجاد پوشه خروجی اگر وجود نداشته باشد

if not os.path.exists(output_folder):

    os.makedirs(output_folder)

```

```
# ایجاد پالت‌های رنگی سفارشی

print("Creating custom color map...")

self.color_maps = create_custom_colormaps()
self.custom_cmap = self.color_maps['blue_red']


# لیست برای ذخیره اطلاعات رسترها

self.rasters_info = []
self.filtered_rasters = []


# اطلاعات ابعاد مدل

self.model_extent = None
self.x_range = None
self.y_range = None
self.z_range = None
self.aspect_ratio = 1.0


# اطلاعات برش

self.clip_extent = None
self.clip_geometry = None


# چاپ تنظیمات

self.print_settings()


# برش اگر فعال باشد geometry ایجاد

if self.clip_enabled:

    self.create_clip_geometry()
```

```

def create_clip_geometry(self):
    """
    برای محدوده برش geometry ایجاد
    """
    try:
        # ایجاد نقطه‌های گوشه‌های محدوده برش
        points = [
            arcpy.Point(self.clip_x_min, self.clip_y_min), # گوشه پایین چپ
            arcpy.Point(self.clip_x_max, self.clip_y_min), # گوشه پایین راست
            arcpy.Point(self.clip_x_max, self.clip_y_max), # گوشه بالا راست
            arcpy.Point(self.clip_x_min, self.clip_y_max), # گوشه بالا چپ
            arcpy.Point(self.clip_x_min, self.clip_y_min) # بستن پلیگون
        ]

        # ایجاد آرایه از نقاط
        array = arcpy.Array(points)

        # ایجاد پلیگون
        self.clip_geometry = arcpy.Polygon(array, self.spatial_ref)

        # برای برش extent ایجاد
        self.clip_extent = arcpy.Extent(
            self.clip_x_min, self.clip_y_min,
            self.clip_x_max, self.clip_y_max
        )

```



```

print(f"□ Clip geometry created successfully")

print(f"  Clip Extent: X[{self.clip_x_min:.1f} - {self.clip_x_max:.1f}], "
      f"Y[{self.clip_y_min:.1f} - {self.clip_y_max:.1f}]")

print(f"  Clip Area: {(self.clip_x_max - self.clip_x_min):.1f}m × "
      f"{(self.clip_y_max - self.clip_y_min):.1f}m = "
      f"{((self.clip_x_max - self.clip_x_min) * (self.clip_y_max - self.clip_y_min)):.0f} m²")

```

except Exception as e:

```

print(f"□ Error creating clip geometry: {str(e)}")

self.clip_enabled = False

```

def print\_settings(self):

```

"""

```

نمایش تنظیمات جاری

```

"""

```

```

print("=" * 70)

```

```

print(f"3D VOXEL MODEL GENERATOR - {self.output_name}")

```

```

print("=" * 70)

```

```

print(f"Input Type: {self.input_type}")

```

```

print(f"Input Path: {self.input_path}")

```

```

print(f"Output: {self.output_folder}")

```

```

print(f"Model Name: {self.output_name}")

```

# نمایش تنظیمات برش

```

if self.clip_enabled:

```

```

    print(f"\n□ CLIP SETTINGS (ENABLED):")

```

```

    print(f"  X (Easting): {self.clip_x_min:.1f}m to {self.clip_x_max:.1f}m")

```

```

    print(f"  Y (Northing): {self.clip_y_min:.1f}m to {self.clip_y_max:.1f}m")

```

```

print(f" Area: {((self.clip_x_max - self.clip_x_min) * (self.clip_y_max - self.clip_y_min)):.0f}
m²")

print(f" Width: {(self.clip_x_max - self.clip_x_min):.1f} m")
print(f" Height: {(self.clip_y_max - self.clip_y_min):.1f} m")
else:
    print(f"\n❑ CLIP SETTINGS: DISABLED")

print(f"\nClassification Ranges:")
for class_name, min_val, max_val, description in self.class_ranges:
    print(f" {class_name}: {min_val:.1f} to {max_val:.1f}")

print(f"\nVoxel Size: {CELL_SIZE_XY}m × {CELL_SIZE_XY}m × {CELL_SIZE_Z}m")
print(f"Color Map: DIRECT BLUE-RED (NO WHITE)")
print(f"Start (0.0): {self.color_maps['start_color']} (Dark Blue)")
print(f"End (1.0): {self.color_maps['end_color']} (Dark Red)")
print(f"ALL POINTS WILL BE DISPLAYED - NO SAMPLING")
print("=" * 70)
print()

def list_rasters_from_gdb(self):
    """
    با فیلتر GDB لیست کردن رسترهای موجود در
    """
    print("Searching for rasters in GDB...")

    try:
        # دریافت لیست تمام رسترها
        all_rasters = []

```



```

        all_rasters.append({
            'name': file,
            'full_path': full_path,
            'relative_path': os.path.relpath(full_path, self.input_path)
        })

    print(f"Total raster files found: {len(all_rasters)}")

    # نمایش ۱۰ فایل اول
    if all_rasters:
        print("Sample raster files found:")
        for i, r in enumerate(all_rasters[:10]):
            print(f" {i+1:2d}. {r['name']} ({r['relative_path']})")
        if len(all_rasters) > 10:
            print(f" ... and {len(all_rasters) - 10} more")

    return all_rasters

except Exception as e:
    print(f"Error reading folder: {str(e)}")
    return []

def list_rasters(self):
    """
    لیست کردن رسترها بر اساس نوع ورودی
    """
    if self.input_type == "GDB":
        all_rasters = self.list_rasters_from_gdb()

```

```

        # تبدیل به فرمت مشابه برای سازگاری

        all_rasters_formatted = [{'name': r, 'full_path': os.path.join(self.input_path, r)}
                                  for r in all_rasters]

    elif self.input_type == "FOLDER":
        all_rasters_formatted = self.list_rasters_from_folder()
    else:
        return []

    # فیلتر کردن بر اساس حرف خاص (اگر تعریف شده باشد)

    if FILTER_CHAR and FILTER_CHAR.strip():
        self.filtered_rasters = [r for r in all_rasters_formatted if FILTER_CHAR in r['name']]
        print(f'Filtered rasters with '{FILTER_CHAR}': {len(self.filtered_rasters)}')
    else:
        self.filtered_rasters = all_rasters_formatted
        print(f'No filter applied, using all {len(self.filtered_rasters)} rasters')

    return self.filtered_rasters

def extract_elevation_from_name(self, raster_name):
    """
    استخراج تراز ارتفاعی از نام رستر
    """
    try:
        # حذف پسوندها

        name_without_ext = os.path.splitext(raster_name)[0]

        # جستجوی مستقیم در لیست ارتفاعها

        for elev in ELEVATION_LEVELS:

```

```

# تبدیل ارتفاع به رشته‌های مختلف

str_variations = [
    str(elev),
    str(elev).replace('.', '_'), # 887.5 -> 887_5
    str(int(elev)) if elev.is_integer() else str(elev), # 875.0 -> 875
]

for str_var in str_variations:
    if str_var in name_without_ext:
        return float(elev)

# آخرین تلاش: استخراج همه اعداد

all_numbers = re.findall(r'\d+\.\d*', name_without_ext)

for num_str in all_numbers:
    try:
        num = float(num_str)

        if num in ELEVATION_LEVELS:
            return num

        elif 800 <= num <= 1200:
            # پیدا کردن نزدیک‌ترین ارتفاع در لیست

            closest = min(ELEVATION_LEVELS, key=lambda x: abs(x - num))

            if abs(closest - num) < 1.0: # اگر اختلاف کمتر از ۱ متر باشد
                return closest

    except:
        continue

print(f"❑ Could not extract elevation from '{raster_name}' - will try to use minimum value")

return None

```

```
except Exception as e:
```

```
    print(f'Error extracting elevation from {raster_name}: {e}')
```

```
    return None
```

```
def match_rasters_to_elevations(self):
```

```
    """
```

```
    مطابقت دادن رسترها با سطوح ارتفاعی
```

```
    """
```

```
    print("\nMatching rasters to elevation levels...")
```

```
    matched_count = 0
```

```
    unmatched_count = 0
```

```
    for raster_info in self.filtered_rasters:
```

```
        raster_name = raster_info['name']
```

```
        raster_path = raster_info['full_path']
```

```
        elevation = self.extract_elevation_from_name(raster_name)
```

```
        if elevation is not None:
```

```
            self.rasters_info.append({
```

```
                'name': raster_name,
```

```
                'elevation': elevation,
```

```
                'path': raster_path,
```

```
                'matched': True if elevation in ELEVATION_LEVELS else False,
```

```
                'source_type': self.input_type
```

```
            })
```

```

        matched_count += 1

    else:

        # اگر ارتفاع استخراج نشد، از ارتفاع پیش فرض استفاده می کنیم

        default_elevation = ELEVATION_LEVELS[unmatched_count % len(ELEVATION_LEVELS)]
    if ELEVATION_LEVELS else 0

    self.rasters_info.append({

        'name': raster_name,

        'elevation': default_elevation,

        'path': raster_path,

        'matched': False,

        'source_type': self.input_type,

        'note': 'Used default elevation'

    })

    unmatched_count += 1

    print(f"   □ Using default elevation {default_elevation}m for '{raster_name}'")


# مرتب سازی بر اساس ارتفاع

self.rasters_info.sort(key=lambda x: x['elevation'])


print(f"\nMatching results:")

print(f"   Matched rasters: {matched_count}")

print(f"   Unmatched rasters (using default): {unmatched_count}")

print(f"   Total organized rasters: {len(self.rasters_info)}")


# نمایش لیست مرتب شده

print(f"\nSorted raster list by elevation:")

for i, info in enumerate(self.rasters_info):

    match_status = "✓" if info.get('matched', False) else "□"

```



```
note = f' ({info['note']})' if 'note' in info else ""  
  
print(f" {i+1:3d}. {match_status} {info['elevation']:6.1f}m: {info['name']} {note}")
```

```
return self.rasters_info
```

```
def clip_raster(self, raster_path, temp_folder):
```

```
    """
```

```
    برش رستر با استفاده از محدوده مشخص شده
```

```
    """
```

```
    try:
```

```
        # ArcGIS نام فایل موقت - کوتاه کردن نام برای جلوگیری از خطای
```

```
        raster_name = os.path.basename(raster_path)
```

```
        # استخراج ارتفاع از نام فایل برای نام کوتاه
```

```
        elevation_match = re.search(r'\d+\.\d*', raster_name)
```

```
        if elevation_match:
```

```
            elev_str = elevation_match.group().replace('.', '_')
```

```
            clipped_name = f"clip_{elev_str[:8]}"
```

```
        else:
```

```
            # اگر ارتفاع پیدا نشد، از شماره استفاده می‌کنیم
```

```
            import hashlib
```

```
            name_hash = hashlib.md5(raster_name.encode()).hexdigest()[:8]
```

```
            clipped_name = f"clip_{name_hash}"
```

```
        clipped_path = os.path.join(temp_folder, clipped_name)
```

```
        # بررسی اینکه آیا رستر در محدوده برش قرار دارد
```

```
        raster_desc = arcpy.Describe(raster_path)
```

```

raster_extent = raster_desc.extent

# بررسی تقاطع با محدوده برش
if (raster_extent.XMin > self.clip_x_max or
    raster_extent.XMax < self.clip_x_min or
    raster_extent.YMin > self.clip_y_max or
    raster_extent.YMax < self.clip_y_min):
    print(f"   □ Raster '{raster_name}' is outside clip extent, skipping")
    return None

# اعمال برش
print(f"   ✂ □ Clipping raster {raster_name}...")

# برای extent تعیین
clip_extent_str = f"{self.clip_x_min} {self.clip_y_min} {self.clip_x_max} {self.clip_y_max}"

# برای برش با نام کوتاه Clip_management استفاده از
arcpy.Clip_management(
    in_raster=raster_path,
    rectangle=clip_extent_str,
    out_raster=clipped_path,
    in_template_dataset=None,
    nodata_value="nan",
    clipping_geometry="NONE",
    maintain_clipping_extent="NO_MAINTAIN_EXTENT"
)

print(f"   □ Clipped to: {clipped_name}")

```

```

        return clipped_path

    except Exception as e:

        print(f"    ❑ Error clipping raster: {str(e)}")

        return None

def validate_rasters(self):

    """
    بررسی صحت رسترها
    """

    print("\nValidating rasters...")

    valid_rasters = []
    invalid_rasters = []

    # ایجاد پوشه موقت برای رسترهای برش خورده
    temp_clip_folder = None

    if self.clip_enabled:

        temp_clip_folder = os.path.join(self.output_folder, "temp_clipped_rasters")

        if not os.path.exists(temp_clip_folder):

            os.makedirs(temp_clip_folder)

            print(f"    Temporary clip folder: {temp_clip_folder}")

    for info in self.rasters_info:

        try:

            # بررسی وجود فایل

            if arcpy.Exists(info['path']):

                # اگر برش فعال است، ابتدا رستر را برش بزن

```

```
if self.clip_enabled:

    clipped_path = self.clip_raster(info['path'], temp_clip_folder)

    if clipped_path:

        # به روزرسانی مسیر به رستر برش خورده
        info['clipped_path'] = clipped_path

        info['original_path'] = info['path']

        info['path'] = clipped_path

        info['clipped'] = True

    else:

        info['clipped'] = False

        info['clipped_path'] = None

else:

    info['clipped'] = False

    info['clipped_path'] = None

# خواندن رستر برای بررسی (رستر اصلی یا برش خورده)
if 'clipped_path' in info and info['clipped_path']:

    raster_to_check = Raster(info['clipped_path'])

    info['is_clipped_version'] = True

else:

    raster_to_check = Raster(info['path'])

    info['is_clipped_version'] = False

desc = arcpy.Describe(raster_to_check)

# بررسی سیستم مختصات
raster_spatial_ref = desc.spatialReference

if raster_spatial_ref.name == "Unknown":
```

```

print(f"   □ {info['name']}: Unknown spatial reference")

# اگر سیستم مختصات نامشخص است، سیستم پیش فرض را اعمال می کنیم

info['spatial_ref'] = self.spatial_ref
else:
    info['spatial_ref'] = raster_spatial_ref

info.update({
    'width': desc.width,
    'height': desc.height,
    'cell_size_x': desc.meanCellWidth,
    'cell_size_y': desc.meanCellHeight,
    'extent': desc.extent,
    'valid': True,
    'data_type': desc.dataType
})

# اگر برش اعمال شده، ابعاد جدید را نمایش بده

if self.clip_enabled and info.get('clipped', False):
    clipped_name = os.path.basename(info['clipped_path'])
    print(f"   ✓ {info['name']}: CLIPPED as {clipped_name} ({desc.width}x{desc.height}
cells), "

        f"elevation: {info['elevation']}m, "
        f"extent: X[{desc.extent.XMin:.1f}-{desc.extent.XMax:.1f}], "
        f"Y[{desc.extent.YMin:.1f}-{desc.extent.YMax:.1f}]")
else:
    print(f"   ✓ {info['name']}: {desc.width}x{desc.height} cells, "

        f"elevation: {info['elevation']}m, "
        f"cellsize: {desc.meanCellWidth:.1f}m")

```

```

        valid_rasters.append(info)

    else:

        info['valid'] = False
        invalid_rasters.append(info)

        print(f" ✖ {info['name']}: File not found at {info['path']}")

    except Exception as e:

        info['valid'] = False
        invalid_rasters.append(info)

        print(f" ✖ {info['name']}: Error - {str(e)}")

self.rasters_info = valid_rasters

print(f"\nValidation results:")
print(f" Valid rasters: {len(self.rasters_info)}")
print(f" Invalid rasters: {len(invalid_rasters)}")

return len(self.rasters_info) > 0

def calculate_model_aspect_ratio(self):
    """
    محاسبه نسبت ابعاد واقعی مدل
    """

    if not self.model_extent:

        return 1.0

```

```

# محاسبه نسبت عرض به طول

x_range = self.x_range[1] - self.x_range[0]
y_range = self.y_range[1] - self.y_range[0]

if y_range == 0:
    return 1.0

self.aspect_ratio = x_range / y_range

print(f"\nModel Aspect Ratio Calculation:")
print(f" X Range: {x_range:.1f} meters")
print(f" Y Range: {y_range:.1f} meters")
print(f" Aspect Ratio (X/Y): {self.aspect_ratio:.3f}")

return self.aspect_ratio

def create_3d_block_model(self):
    """
    ایجاد مدل بلوکی سه بعدی
    """
    if not self.rasters_info:
        print("No valid rasters found!")
        return None

    print("\nBuilding 3D block model...")

    try:
        # خواندن اولین رستر برای استخراج اطلاعات هندسی

```

```

first_raster = Raster(self.rasters_info[0]['path'])

desc = arcpy.Describe(first_raster)

extent = desc.extent

# اگر برش فعال است، از محدوده برش استفاده می‌کنیم
if self.clip_enabled:

    # اعمال محدوده برش

    x_min = max(extent.XMin, self.clip_x_min)
    x_max = min(extent.XMax, self.clip_x_max)
    y_min = max(extent.YMin, self.clip_y_min)
    y_max = min(extent.YMax, self.clip_y_max)

    # اطمینان از صحت محدوده

    if x_min >= x_max or y_min >= y_max:

        print(f"□ Clip extent is invalid after applying to rasters")
        print(f" Using original raster extent instead")

        x_min, x_max = extent.XMin, extent.XMax
        y_min, y_max = extent.YMin, extent.YMax
    else:

        x_min, y_min = extent.XMin, extent.YMin
        x_max, y_max = extent.XMax, extent.YMax

    # ذخیره اطلاعات ابعاد برای استفاده در تصاویر

    self.model_extent = arcpy.Extent(x_min, y_min, x_max, y_max)

    self.x_range = [x_min, x_max]
    self.y_range = [y_min, y_max]

    print(f"Model extent {'(CLIPPED)' if self.clip_enabled else ':'})

```



```
print(f" X (Easting): {x_min:.1f} to {x_max:.1f} m")
print(f" Y (Northing): {y_min:.1f} to {y_max:.1f} m")
print(f" Width: {(x_max - x_min):.1f} m")
print(f" Height: {(y_max - y_min):.1f} m")
print(f" Area: (((x_max - x_min) * (y_max - y_min)):.0f} m2)")
```

```
# تعیین محدوده ارتفاعی
```

```
elevations = [info['elevation'] for info in self.rasters_info]
```

```
z_min = min(elevations)
```

```
z_max = max(elevations)
```

```
self.z_range = [z_min, z_max]
```

```
print(f" Z (Elevation): {z_min:.1f} to {z_max:.1f} m")
```

```
print(f" Height (Z range): {(z_max - z_min):.1f} m")
```

```
# محاسبه نسبت ابعاد
```

```
aspect_ratio = self.calculate_model_aspect_ratio()
```

```
# محاسبه تعداد سلول‌ها بر اساس ابعاد دقیق
```

```
x_cells = max(1, int(round((x_max - x_min) / CELL_SIZE_XY)))
```

```
y_cells = max(1, int(round((y_max - y_min) / CELL_SIZE_XY)))
```

```
z_cells = max(1, int(round((z_max - z_min) / CELL_SIZE_Z)))
```

```
# محاسبه مجدد ابعاد دقیق بر اساس تعداد سلول‌ها
```

```
x_max_adj = x_min + (x_cells * CELL_SIZE_XY)
```

```
y_max_adj = y_min + (y_cells * CELL_SIZE_XY)
```

```
z_max_adj = z_min + (z_cells * CELL_SIZE_Z)
```

```

print(f"\nBlock model dimensions:")

print(f"  X: {x_cells} cells (EXACT {CELL_SIZE_XY}m) = {x_min:.1f} to {x_max_adj:.1f} m")

print(f"  Y: {y_cells} cells (EXACT {CELL_SIZE_XY}m) = {y_min:.1f} to {y_max_adj:.1f} m")

print(f"  Z: {z_cells} cells (EXACT {CELL_SIZE_Z}m) = {z_min:.1f} to {z_max_adj:.1f} m")

print(f"  Total blocks: {x_cells * y_cells * z_cells:,}")

print(f"  Block volume: {CELL_SIZE_XY * CELL_SIZE_XY * CELL_SIZE_Z:.1f} m³")

print(f"  Model volume: {(x_cells * y_cells * z_cells * CELL_SIZE_XY * CELL_SIZE_XY * CELL_SIZE_Z):.0f} m³")

# ایجاد آرایه سه بعدی برای ذخیره مقادیر
block_model = np.zeros((z_cells, y_cells, x_cells), dtype=np.float32)
block_model[:] = np.nan

# ایجاد آرایه برای مختصات
x_coords = np.linspace(x_min, x_max_adj, x_cells, endpoint=False) + (CELL_SIZE_XY / 2.0)
y_coords = np.linspace(y_min, y_max_adj, y_cells, endpoint=False) + (CELL_SIZE_XY / 2.0)
z_coords = np.linspace(z_min, z_max_adj, z_cells, endpoint=False) + (CELL_SIZE_Z / 2.0)

# پردازش هر لایه ارتفاعی
print("\nProcessing elevation layers...")

for layer_idx, raster_info in enumerate(self.rasters_info):
    raster_name = raster_info['name']
    elevation = raster_info['elevation']

    print(f" [{layer_idx + 1:2d}/{len(self.rasters_info)}] Elevation {elevation:6.1f}m: {raster_name}")

```

```
try:

    # خواندن رستر
    raster = Raster(raster_info['path'])

    # در مدل Z پیدا کردن نزدیکترین سطح
    z_idx = np.argmin(np.abs(z_coords - elevation))

    # numpy تبدیل رستر به آرایه
    raster_array = arcpy.RasterToNumPyArray(raster, nodata_to_value=np.nan)

    # معکوس کردن جهت عمودی اگر نیاز باشد
    if FLIP_RASTER_VERTICAL:
        raster_array = np.flipud(raster_array)

    # بررسی اندازه رستر
    if raster_array.shape[0] != y_cells or raster_array.shape[1] != x_cells:
        print(f"   ❌ Raster size mismatch: {raster_array.shape} vs ({y_cells}, {x_cells})")

        # نمونه‌برداری ساده انجام می‌دهیم
        y_indices = np.linspace(0, raster_array.shape[0]-1, y_cells).astype(int)
        x_indices = np.linspace(0, raster_array.shape[1]-1, x_cells).astype(int)
        raster_array_resized = raster_array[np.ix_(y_indices, x_indices)]

        print(f"   ✅ Resized to: {raster_array_resized.shape}")
    else:
        raster_array_resized = raster_array
```

```

        ذخیره در مدل بلوکی
        block_model[z_idx, :, :] = raster_array_resized

    except Exception as e:
        print(f"    □ Processing error: {str(e)}")
        import traceback
        traceback.print_exc()

    درونیایی برای پر کردن مقادیر خالی
    print("\nInterpolating missing values...")
    filled_count = self.interpolate_missing_values(block_model)
    print(f"    □ {filled_count:,} NaN values interpolated")

    محاسبه آمار کلی مدل
    valid_values = block_model[~np.isnan(block_model)]
    if len(valid_values) > 0:
        print(f"\nModel statistics:")
        print(f"    Min value: {valid_values.min():.4f}")
        print(f"    Max value: {valid_values.max():.4f}")
        print(f"    Mean value: {valid_values.mean():.4f}")
        print(f"    Std deviation: {valid_values.std():.4f}")
        print(f"    Non-NaN cells: {len(valid_values):,}, {len(valid_values)/(x_cells*y_cells*z_cells)*100:.1f}%")

    پاکسازی رسترهای موقت برش خورده
    if self.clip_enabled:
        print(f"\nCleaning up temporary clipped rasters...")
        temp_clip_folder = os.path.join(self.output_folder, "temp_clipped_rasters")

```

```

if os.path.exists(temp_clip_folder):

    try:

        # حذف کامل پوشه و محتویات آن
        shutil.rmtree(temp_clip_folder)

        print(f" □ Temporary files cleaned up")

    except Exception as e:

        print(f" □ Could not clean up temporary files: {e}")

return {
    'block_model': block_model,
    'x_coords': x_coords,
    'y_coords': y_coords,
    'z_coords': z_coords,
    'cell_size_xy': CELL_SIZE_XY,
    'cell_size_z': CELL_SIZE_Z,
    'extent': self.model_extent,
    'spatial_ref': self.spatial_ref,
    'elevations': elevations,
    'num_layers': len(self.rasters_info),
    'aspect_ratio': aspect_ratio,
    'x_range': self.x_range,
    'y_range': self.y_range,
    'z_range': self.z_range,
    'x_cells': x_cells,
    'y_cells': y_cells,
    'z_cells': z_cells,
    'clip_enabled': self.clip_enabled,
    'clip_extent': self.clip_extent if self.clip_enabled else None,

```

```
        'model_x_min': x_min,  
        'model_x_max': x_max_adj,  
        'model_y_min': y_min,  
        'model_y_max': y_max_adj,  
        'model_z_min': z_min,  
        'model_z_max': z_max_adj  
    }
```

```
except Exception as e:
```

```
    print(f"Error creating model: {str(e)}")
```

```
    import traceback
```

```
    traceback.print_exc()
```

```
    return None
```

```
def interpolate_missing_values(self, block_model):
```

```
    """
```

```
    در مدل بلوکی NaN درونیابی مقادیر
```

```
    """
```

```
    z, y, x = block_model.shape
```

```
    filled_count = 0
```

```
    for i in range(y):
```

```
        for j in range(x):
```

```
            column = block_model[:, i, j]
```

```
            # NaN شمارش مقادیر
```

```
            nan_count = np.sum(np.isnan(column))
```

```
            if nan_count == 0:
```

```

        continue

        # باشند، صفر می کنیم NaN اگر تمام مقادیر
        if nan_count == z:
            block_model[:, i, j] = 0
            filled_count += z

        # باشند، درون یابی می کنیم NaN اگر برخی مقادیر
        else:
            valid_idx = ~np.isnan(column)

            if np.sum(valid_idx) > 1: # حداقل دو نقطه برای درون یابی
                z_idx = np.arange(z)

                if INTERPOLATION_METHOD == 'linear':
                    f = interpolate.interp1d(z_idx[valid_idx], column[valid_idx],
                                              kind='linear', bounds_error=False,
                                              fill_value=(column[valid_idx][0],
                                                         column[valid_idx][-1]))

                    block_model[:, i, j] = f(z_idx)

                elif INTERPOLATION_METHOD == 'nearest':
                    f = interpolate.interp1d(z_idx[valid_idx], column[valid_idx],
                                              kind='nearest', bounds_error=False,
                                              fill_value=(column[valid_idx][0],
                                                         column[valid_idx][-1]))

                    block_model[:, i, j] = f(z_idx)

            filled_count += nan_count

    return filled_count

```

```

def get_points_for_class(self, block_model, min_val, max_val):

```

```

"""
پیدا کردن نقاط در محدوده مشخص
"""

# NaNها برای مقایسه صحیح با
valid_mask = ~np.isnan(block_model)

# تعریف محدوده
points_mask = (block_model >= min_val) & (block_model <= max_val) & valid_mask

# پیدا کردن اندیس‌ها
points_idx = np.where(points_mask)

return points_idx

def create_voxel_cube(self, center_x, center_y, center_z, value, cmap, norm, original_min,
original_max):
    """
    کامل با تمام وجه‌های قابل مشاهده voxel ایجاد یک مکعب
    """
    # ابعاد دقیق مکعب - دقیقاً ۱۲,۵ متر
    dx = CELL_SIZE_XY / 2.0
    dy = CELL_SIZE_XY / 2.0
    dz = CELL_SIZE_Z / 2.0

    # تعریف ۸ رأس مکعب
    vertices = np.array([
        # رئوس پایین

```



```
[center_x - dx, center_y - dy, center_z - dz], # 0
[center_x + dx, center_y - dy, center_z - dz], # 1
[center_x + dx, center_y + dy, center_z - dz], # 2
[center_x - dx, center_y + dy, center_z - dz], # 3
# رئوس بالا
[center_x - dx, center_y - dy, center_z + dz], # 4
[center_x + dx, center_y - dy, center_z + dz], # 5
[center_x + dx, center_y + dy, center_z + dz], # 6
[center_x - dx, center_y + dy, center_z + dz], # 7
])
```

# تعریف ۶ وجه مکعب

```
faces = [
    # منفی (z) وجه پایین
    [vertices[0], vertices[1], vertices[2], vertices[3]],
    # مثبت (z) وجه بالا
    [vertices[4], vertices[5], vertices[6], vertices[7]],
    # منفی (y) وجه جلو
    [vertices[0], vertices[1], vertices[5], vertices[4]],
    # مثبت (y) وجه عقب
    [vertices[3], vertices[2], vertices[6], vertices[7]],
    # منفی (x) وجه چپ
    [vertices[0], vertices[3], vertices[7], vertices[4]],
    # مثبت (x) وجه راست
    [vertices[1], vertices[2], vertices[6], vertices[5]]
]
```

```
# نرمالایز کردن مقدار برای نمایش
```

```
if original_max > 1.0 or original_min < 0.0:
```

```
    # اگر داده خارج از محدوده ۰-۱ است، نرمالایز می‌کنیم
```

```
    normalized_value = (value - original_min) / (original_max - original_min)
```

```
    normalized_value = max(0.0, min(1.0, normalized_value)) # محدود کردن به ۰-۱
```

```
else:
```

```
    normalized_value = value
```

```
# رنگ‌بازی بر اساس مقدار نرمالایز شده
```

```
color = cmap(norm(normalized_value))
```

```
return faces, color, normalized_value
```

```
def create_3d_voxel_visualization(self, class_info, points_idx, values, model_data):
```

```
    """
```

```
    سه‌بعدی واقعی با مکعب‌های به هم چسبیده - تمام نقاط نمایش داده می‌شوند voxel ایجاد یک مدل
```

```
    """
```

```
    if len(points_idx[0]) == 0:
```

```
        print(f" □ No points found for visualization")
```

```
        return
```

```
    class_name, min_val, max_val, description = class_info
```

```
    print(f" Creating 3D voxel visualization for {class_name}...")
```

```
    x_coords = model_data['x_coords']
```

```
    y_coords = model_data['y_coords']
```

```
    z_coords = model_data['z_coords']
```

```

z_idx, y_idx, x_idx = points_idx

try:

    # ایجاد پوشه تصاویر
    image_folder = os.path.join(self.output_folder, "3D_Visualizations")
    if not os.path.exists(image_folder):
        os.makedirs(image_folder)

    # تعداد کل نقاط - تمام نقاط نمایش داده می شوند
    total_points = len(z_idx)

    # محاسبه محدوده واقعی داده ها
    actual_min = values.min()
    actual_max = values.max()

    # محاسبه محدوده کلی مدل (نه فقط این کلاس)
    block_model = model_data['block_model']
    all_values = block_model[~np.isnan(block_model)]
    global_min = all_values.min()
    global_max = all_values.max()

    # clip اطلاعات
    clip_info = ""
    if model_data.get('clip_enabled', False):
        clip_info = f"\nCLIPPED AREA: X[{model_data['model_x_min']:.1f}-{model_data['model_x_max']:.1f}], " \
            f"Y[{model_data['model_y_min']:.1f}-{model_data['model_y_max']:.1f}]"

```

```

# نمایش اطلاعات محدوده

print(f" Total blocks: {total_points:,}")
print(f" Requested range: {min_val:.1f} to {max_val:.1f}")
print(f" Actual class range: {actual_min:.3f} to {actual_max:.3f}")
print(f" Global model range: {global_min:.3f} to {global_max:.3f}")
print(f" ALL POINTS WILL BE DISPLAYED")
print(f" USING DIRECT BLUE-RED COLOR MAP (NO WHITE)")
print(f" COLOR BASED ON GLOBAL RANGE: {global_min:.3f} (Blue) to {global_max:.3f} (Red)")

if model_data.get('clip_enabled', False):
    print(f" MODEL IS CLIPPED TO SPECIFIED EXTENT")
print(f" Z-AXIS: 850m to 1200m with 50m intervals from 875m to 1125m")

# ایجاد figure

fig = plt.figure(figsize=(22, 16), dpi=150)
ax = fig.add_subplot(111, projection='3d')

# سفارشی - بر اساس محدوده کلی colormap استفاده از
cmap = self.custom_cmap

# همیشه از محدوده ۰-۱ برای نرمالایز کردن استفاده می کنیم
norm = Normalize(vmin=0.0, vmax=1.0)

# استفاده می کنیم batch برای نمایش بهتر تعداد زیاد مکعب ها، از روش
print(f" Creating {total_points:,} voxel cubes...")

# سیستم crash اگر تعداد نقاط خیلی زیاد است، برای جلوگیری از
if total_points > 20000:

```

```

print(f"    Large dataset - creating cubes in batches...")

# های کوچکتر batch تقسیم به
batch_size = 5000
voxels_created = 0

for batch_start in range(0, total_points, batch_size):
    batch_end = min(batch_start + batch_size, total_points)
    batch_indices = np.arange(batch_start, batch_end)

    for idx in batch_indices:
        # نرمالایز کردن بر اساس محدوده کلی (۰ تا ۱)
        # ابتدا نرمالایز کردن به محدوده اصلی داده‌ها
        normalized_value = (values[idx] - global_min) / (global_max - global_min)
        normalized_value = max(0.0, min(1.0, normalized_value)) # محدود کردن به ۰-۱

        # کامل با نرمالایز کردن بر اساس محدوده کلی voxel ایجاد یک مکعب
        faces, color, norm_val = self.create_voxel_cube(
            x_coords[x_idx[idx]],
            y_coords[y_idx[idx]],
            z_coords[z_idx[idx]],
            normalized_value,
            cmap,
            norm,
            0.0, # حداقل همیشه ۰
            1.0 # حداکثر همیشه ۱
        )

```

```

# ایجاد مکعب
cube = Poly3DCollection(faces,
                        facecolors=color,
                        edgecolors=BLOCK_EDGE_COLOR,
                        linewidths=BLOCK_EDGE_WIDTH,
                        alpha=BLOCK_ALPHA)
ax.add_collection3d(cube)

voxels_created += (batch_end - batch_start)
print(f"    Created {voxels_created:,}/{total_points:,} voxels")
else:
    # برای تعداد کمتر، همه را یکجا ایجاد می‌کنیم
    for i in range(total_points):
        # نرمالایز کردن بر اساس محدوده کلی (۰ تا ۱)
        normalized_value = (values[i] - global_min) / (global_max - global_min)
        normalized_value = max(0.0, min(1.0, normalized_value)) # محدود کردن به ۰-۱

        faces, color, norm_val = self.create_voxel_cube(
            x_coords[x_idx[i]],
            y_coords[y_idx[i]],
            z_coords[z_idx[i]],
            normalized_value,
            cmap,
            norm,
            0.0, # حداقل همیشه ۰
            1.0 # حداکثر همیشه ۱

```

```

    )

    cube = Poly3DCollection(faces,
                            facecolors=color,
                            edgecolors=BLOCK_EDGE_COLOR,
                            linewidths=BLOCK_EDGE_WIDTH,
                            alpha=BLOCK_ALPHA)

    ax.add_collection3d(cube)

    print(f"    Created all {total_points:,} voxel cubes")

    # تنظیم برچسب‌های محورها
    ax.set_xlabel('Easting (m)', fontsize=14, labelpad=15)
    ax.set_ylabel('Northing (m)', fontsize=14, labelpad=15)
    ax.set_zlabel('Elevation (m)', fontsize=14, labelpad=15)

    # فرمت اعداد صحیح برای محور X و Y
    def integer_formatter(x, pos):
        return f'{int(x):,}'

    ax.xaxis.set_major_formatter(FuncFormatter(integer_formatter))
    ax.yaxis.set_major_formatter(FuncFormatter(integer_formatter))

    # تنظیم محدوده محورها
    x_min, x_max = x_coords[x_idx].min(), x_coords[x_idx].max()
    y_min, y_max = y_coords[y_idx].min(), y_coords[y_idx].max()
    z_min, z_max = z_coords[z_idx].min(), z_coords[z_idx].max()

```

```

x_padding = (x_max - x_min) * 0.1
y_padding = (y_max - y_min) * 0.1

ax.set_xlim(x_min - x_padding, x_max + x_padding)
ax.set_ylim(y_min - y_padding, y_max + y_padding)

# ===== Z تنظیمات جدید برای محور =====

# از ۸۵۰ تا ۱۲۰۰ متر Z تنظیم محدوده محور
ax.set_zlim(850, 1200)

# ایجاد تقسیم‌بندی ۵۰ متری از ۸۷۵ تا ۱۱۲۵

z_ticks = []
z_tick_labels = []

# از ۸۷۵ شروع می‌کنیم و ۵۰ متر ۵۰ متر تا ۱۱۲۵ می‌رویم

start_z = 875
end_z = 1125
step = 50

current_z = start_z
while current_z <= end_z:
    z_ticks.append(current_z)
    z_tick_labels.append(f'{current_z}')
    current_z += step

# اضافه کردن مقادیر حداقل و حداکثر اگر در لیست نیستند

if 850 not in z_ticks:
    z_ticks.insert(0, 850)

```



```

z_tick_labels.insert(0, '850')

if 1200 not in z_ticks:
    z_ticks.append(1200)
    z_tick_labels.append('1200')

# اعمال تقسیم‌بندی به محور Z
ax.set_zticks(z_ticks)
ax.set_zticklabels(z_tick_labels, fontsize=11)

# تنظیم fontsize برای tick labels
ax.tick_params(axis='x', labelsz=11)
ax.tick_params(axis='y', labelsz=11)
ax.tick_params(axis='z', labelsz=11)

# برای خوانایی بهتر Z های محور label تنظیم چرخش
for t in ax.get_zticklabels():
    t.set_rotation(0)

# تنظیم نسبت ابعاد
x_range = ax.get_xlim()[1] - ax.get_xlim()[0]
y_range = ax.get_ylim()[1] - ax.get_ylim()[0]
z_range = ax.get_zlim()[1] - ax.get_zlim()[0]

max_range = max(x_range, y_range, z_range)
if max_range > 0:
    ax.set_box_aspect([x_range/max_range, y_range/max_range, z_range/max_range])

# تنظیم دید

```

```

ax.view_init(elev=25, azim=45)

# با برچسب مناسب ایجاد colorbar

sm = plt.cm.ScalarMappable(cmap=self.custom_cmap, norm=norm)
sm.set_array([])

cbar = plt.colorbar(sm, ax=ax, shrink=0.6, pad=0.1)

cbar.set_label(f'Normalized Value [0-1]\nBlue={global_min:.3f}, Red={global_max:.3f}',
               fontsize=13, rotation=270, labelpad=25)

cbar.ax.tick_params(labelsize=11)

# اضافه کردن عنوان

clip_status = "CLIPPED" if model_data.get('clip_enabled', False) else "FULL"
main_title = f'3D Voxel Model - {self.output_name} ({clip_status})\n' \
             f'Class: {class_name} (Requested: {min_val:.1f} to {max_val:.1f})\n' \
             f'Total Blocks: {total_points:,} | Class Range: {actual_min:.3f} to {actual_max:.3f}\n' \
             f'Color Range (Global): {global_min:.3f} (Blue) to {global_max:.3f} (Red)\n' \
             f'Voxel Size: {CELL_SIZE_XY}m × {CELL_SIZE_XY}m × {CELL_SIZE_Z}m | Color: '
Direct Blue-Red (No White)\n' \
             f'Z-axis: 850m to 1200m with 50m intervals (875, 925, 975, 1025, 1075, 1125)'

if model_data.get('clip_enabled', False):
    main_title += f'\nCLIPPED AREA: X[ {model_data['model_x_min']:.1f}- '
{model_data['model_x_max']:.1f}], " \
                f'Y[ {model_data['model_y_min']:.1f}- {model_data['model_y_max']:.1f}] "'

plt.title(main_title, fontsize=16, fontweight='bold', pad=20)

# اضافه کردن اطلاعات

info_text = f'Project: {self.output_name}\n' \

```

```

f'Model Type: {clip_status}\n' \
f'Class: {class_name}\n' \
f'Blocks: {total_points:,}\n' \
f'Requested: {min_val:.1f} to {max_val:.1f}\n' \
f'Actual Class: {actual_min:.3f} to {actual_max:.3f}\n' \
f'Global Range: {global_min:.3f} to {global_max:.3f}\n' \
f'Color Based on Global Range\n' \
f'Color Map: Direct Blue-Red\n' \
f'Blue = {global_min:.3f}\n' \
f'Red = {global_max:.3f}\n' \
f'Voxel Size: {CELL_SIZE_XY}m³\n' \
f'ALL POINTS DISPLAYED\n' \
f'Z-axis: 850m to 1200m\n' \
f'Z-ticks: 50m intervals\n' \
f'(875, 925, 975, 1025, 1075, 1125)'

if model_data.get('clip_enabled', False):
    info_text += f"\n\nCLIP EXTENT:\n" \
        f'X: {model_data['model_x_min']:.1f}-{model_data['model_x_max']:.1f}\n' \
        f'Y: {model_data['model_y_min']:.1f}-{model_data['model_y_max']:.1f}\n' \
        f'Area: (((model_data['model_x_max']-
model_data['model_x_min'])*(model_data['model_y_max']-model_data['model_y_min'])):.0f} m²"

    ax.text2D(0.02, 0.98, info_text, transform=ax.transAxes,
        fontsize=10, verticalalignment='top',
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.9))

# اضافه کردن نمونه رنگ‌ها

color_samples_text = f"Color Gradient (No White):\n" \

```

```

        f"• {self.color_maps['start_color']} (Dark Blue) = {global_min:.3f}\n" \
        f"• {self.color_maps['end_color']} (Dark Red) = {global_max:.3f}"

ax.text2D(0.75, 0.02, color_samples_text, transform=ax.transAxes,
          fontsize=9, verticalalignment='bottom',
          bbox=dict(boxstyle='round', facecolor='white', alpha=0.7))

# تنظيم layout
plt.tight_layout()

# ذخيره تصوير
safe_class_name = class_name

clip_suffix = "_CLIPPED" if model_data.get('clip_enabled', False) else ""

image_path = os.path.join(image_folder,
f"{self.output_name}_{safe_class_name}_3D_VOXELS{clip_suffix}.png")

plt.savefig(image_path, dpi=IMAGE_DPI, bbox_inches='tight',
            facecolor='white', edgecolor='none')

plt.close()

print(f"   □ 3D voxel visualization saved: {image_path}")
print(f"   All {total_points:,} points displayed")
print(f"   Color based on GLOBAL range: {global_min:.3f} to {global_max:.3f}")
print(f"   Values normalized to [0-1] range based on global range")
print(f"   Direct Blue-Red color map (NO WHITE)")
print(f"   Blue ({global_min:.3f}): {self.color_maps['start_color']}")
print(f"   Red ({global_max:.3f}): {self.color_maps['end_color']}")
print(f"   Class range: {actual_min:.3f} to {actual_max:.3f}")
print(f"   Voxel size: {CELL_SIZE_XY}m × {CELL_SIZE_XY}m × {CELL_SIZE_Z}m")

```

```

print(f"    Z-axis range: 850m to 1200m (Fixed)")

print(f"    Z-axis ticks: 50m intervals (875, 925, 975, 1025, 1075, 1125)")

if model_data.get('clip_enabled', False):

    print(f"    Model clipped to specified extent")

# به تابع global_max و global_min ارسال - ID ایجاد نمودار ۲

self.create_2d_plot(class_info, points_idx, values, model_data, image_folder, global_min,
global_max)

except Exception as e:

    print(f"    ❑ Error creating 3D voxel visualization: {str(e)}")

    import traceback

    traceback.print_exc()

def create_2d_plot(self, class_info, points_idx, values, model_data, image_folder, global_min,
global_max):

    """

    برای مرجع - با رنگ بر اساس محدوده کلی ID ایجاد یک نمودار ۲

    """

    try:

        class_name, min_val, max_val, description = class_info

        x_coords = model_data['x_coords']

        y_coords = model_data['y_coords']

        z_idx, y_idx, x_idx = points_idx

        # محاسبه محدوده واقعی این کلاس

        actual_min = values.min()

        actual_max = values.max()

```

```

# اگر تعداد نقاط زیاد است D نمونه برداری برای نمودار ۲
if len(x_idx) > 50000:
    sample_size = 50000
    indices = np.random.choice(len(x_idx), sample_size, replace=False)
    x_sample = x_coords[x_idx[indices]]
    y_sample = y_coords[y_idx[indices]]
    values_sample = values[indices]
    sample_info = f" (Sampled: {sample_size:}, points)"
else:
    x_sample = x_coords[x_idx]
    y_sample = y_coords[y_idx]
    values_sample = values
    sample_info = f" ({len(x_idx):}, points)"

# ایجاد نمودار ۲
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 8))

# نمودار ۱: نقشه پهنه‌ای با رنگ‌بندی بر اساس محدوده کلی
scatter1 = ax1.scatter(x_sample, y_sample, c=values_sample,
                      cmap=self.custom_cmap, s=1, alpha=0.6,
                      vmin=global_min, vmax=global_max) # استفاده از global range

# ===== تنظیمات جدید برای محورها =====
ax1.set_xlabel('Easting (m)', fontsize=12)
ax1.set_ylabel('Northing (m)', fontsize=12)

```

```

# (اعداد صحيح) X و Y فرمت اعداد برای محور
def integer_formatter(x, pos):
    return f'{int(x):,}'

ax1.xaxis.set_major_formatter(FuncFormatter(integer_formatter))
ax1.yaxis.set_major_formatter(FuncFormatter(integer_formatter))

# تنظیم fontsize برای tick labels
ax1.tick_params(axis='both', labelsize=11)

clip_status = "CLIPPED" if model_data.get('clip_enabled', False) else "FULL"
title1 = f'Plan View - {class_name} ({clip_status}){sample_info}\n' \
    f'Class Range: {actual_min:.3f} to {actual_max:.3f}\n' \
    f'Color Range (Global): {global_min:.3f} to {global_max:.3f}\n' \
    f'Color: Direct Blue-Red (No White)\n' \
    f'Axes: Easting and Northing in meters (integers)'

if model_data.get('clip_enabled', False):
    title1 += f'\nClip Extent: X[{model_data['model_x_min']:.1f}-\
{model_data['model_x_max']:.1f}], " \
        f"Y[{model_data['model_y_min']:.1f}-{model_data['model_y_max']:.1f}]"

ax1.set_title(title1, fontsize=14, fontweight='bold')
ax1.set_aspect('equal')
cbar1 = plt.colorbar(scatter1, ax=ax1)
cbar1.set_label(f'Value\nBlue={global_min:.3f}, Red={global_max:.3f}', fontsize=11)
ax1.grid(True, alpha=0.3)

```

```

# اضافه کردن مستطیل محدوده برش اگر فعال باشد

if model_data.get('clip_enabled', False):

    from matplotlib.patches import Rectangle

    rect = Rectangle((model_data['model_x_min'], model_data['model_y_min']),
                     model_data['model_x_max'] - model_data['model_x_min'],
                     model_data['model_y_max'] - model_data['model_y_min'],
                     linewidth=2, edgecolor='red', facecolor='none',
                     linestyle='--', alpha=0.7, label='Clip Extent')

    ax1.add_patch(rect)

    ax1.legend(loc='upper right', fontsize=9)

نمودار ۲: هیستوگرام با رنگ‌بندی آبی-قرمز سفارشی بر اساس محدوده کلی

# هیستوگرام رنگی

n, bins, patches = ax2.hist(values, bins=50, edgecolor='black', alpha=0.7)

# رنگ‌آمیزی هر میله هیستوگرام بر اساس مقدار وسط آن نرمالایز شده نسبت به محدوده کلی

bin_centers = 0.5 * (bins[:-1] + bins[1:])

# نرمالایز کردن نسبت به محدوده کلی

col = (bin_centers - global_min) / (global_max - global_min)

col = np.clip(col, 0, 1) # محدود کردن به بازه ۰-۱

for c, p in zip(col, patches):

    plt.setp(p, 'facecolor', self.custom_cmap(c))

ax2.set_xlabel('Value', fontsize=12)

ax2.set_ylabel('Frequency', fontsize=12)

ax2.set_title(f'Value Distribution - {class_name}\n')

```



```

f'Total: {len(values):,} blocks\n'\
f'Class Range: {actual_min:.3f} to {actual_max:.3f}\n'\
f'Color Range (Global): {global_min:.3f} to {global_max:.3f}\n'\
f'Color: Direct Blue-Red (No White)',
    fontsize=14, fontweight='bold')
ax2.grid(True, alpha=0.3)

# خطوط برای محدوده درخواستی
ax2.axvline(min_val, color='green', linestyle='-', linewidth=3, alpha=0.7,
            label=f'Requested Min: {min_val:.1f}')
ax2.axvline(max_val, color='red', linestyle='-', linewidth=3, alpha=0.7,
            label=f'Requested Max: {max_val:.1f}')

# خطوط برای محدوده واقعی کلاس
ax2.axvline(actual_min, color='blue', linestyle='--', linewidth=2, alpha=0.5,
            label=f'Class Min: {actual_min:.3f}')
ax2.axvline(actual_max, color='orange', linestyle='--', linewidth=2, alpha=0.5,
            label=f'Class Max: {actual_max:.3f}')

# خطوط برای محدوده کلی
ax2.axvline(global_min, color='navy', linestyle='-', linewidth=2, alpha=0.3,
            label=f'Global Min: {global_min:.3f}')
ax2.axvline(global_max, color='darkred', linestyle='-', linewidth=2, alpha=0.3,
            label=f'Global Max: {global_max:.3f}')

ax2.legend(fontsize=8, loc='upper right')

clip_status = "CLIPPED" if model_data.get('clip_enabled', False) else "FULL"

```

```

plt.suptitle(f'2D Reference Plots - {self.output_name}: {class_name} ({clip_status})\n' \
            f'Class Range: {actual_min:.3f} to {actual_max:.3f} | Requested: {min_val:.1f} to {max_val:.1f}\n' \
            f'Color Range (Global): {global_min:.3f} (Blue) to {global_max:.3f} (Red)\n' \
            f'Color Map: Direct Blue-Red (No White)\n' \
            f'Axes: Easting and Northing in meters (integers)',
            fontsize=16, fontweight='bold')

plt.tight_layout()

# ذخیره نمودار ۲D
safe_class_name = class_name

clip_suffix = "_CLIPPED" if model_data.get('clip_enabled', False) else ""

image_path = os.path.join(image_folder,
f"{self.output_name}_{safe_class_name}_2D_PLOT{clip_suffix}.png")

plt.savefig(image_path, dpi=300, bbox_inches='tight')

plt.close()

print(f"   □ 2D reference plot saved: {image_path}")
print(f"   Color based on GLOBAL range: {global_min:.3f} to {global_max:.3f}")
print(f"   Direct Blue-Red color map (NO WHITE)")
print(f"   X and Y axes displayed as integers (meters)")

except Exception as e:

    print(f"   □ Error creating 2D reference plot: {str(e)}")

def create_single_shapefile(self, class_info, points_idx, values, model_data):
    """
    برای یک کلاس خاص - با نام فیلدهای کوتاه Shapefile ایجاد یک
    """

```

```

x_coords = model_data['x_coords']
y_coords = model_data['y_coords']
z_coords = model_data['z_coords']
spatial_ref = model_data['spatial_ref']

class_name, min_val, max_val, description = class_info
z_idx, y_idx, x_idx = points_idx

# ایجاد پوشه Shapefile
shapefile_folder = os.path.join(self.output_folder, "Shapefiles")
if not os.path.exists(shapefile_folder):
    os.makedirs(shapefile_folder)

# clip با توجه به وضعیت Shapefile نام
clip_suffix = "_CLIPPED" if model_data.get('clip_enabled', False) else ""
shapefile_name = f"{self.output_name}_{class_name}{clip_suffix}.shp"
shapefile_path = os.path.join(shapefile_folder, shapefile_name)

try:
    # موجود اگر وجود دارد shapefile حذف
    if arcpy.Exists(shapefile_path):
        arcpy.Delete_management(shapefile_path)

    # ایجاد Feature Class
    arcpy.CreateFeatureclass_management(
        shapefile_folder,
        shapefile_name,
        "POINT",

```

```
spatial_reference=spatial_ref,  
has_z="ENABLED",  
has_m="DISABLED"  
)
```

# اضافه کردن فیلدها با نام‌های کوتاه (حداکثر ۱۰ کاراکتر)

```
arcpy.AddField_management(shapefile_path, "Value", "FLOAT") # 4 کاراکتر
```

```
arcpy.AddField_management(shapefile_path, "Elev", "FLOAT") # 4 کاراکتر
```

```
arcpy.AddField_management(shapefile_path, "Class", "TEXT", field_length=20) # 5 کاراکتر
```

```
arcpy.AddField_management(shapefile_path, "X_Coord", "FLOAT") # 6 کاراکتر
```

```
arcpy.AddField_management(shapefile_path, "Y_Coord", "FLOAT") # 6 کاراکتر
```

```
arcpy.AddField_management(shapefile_path, "Z_Coord", "FLOAT") # 6 کاراکتر
```

```
arcpy.AddField_management(shapefile_path, "Vol_m3", "FLOAT") # 5 کاراکتر
```

# فعال باشد - با نام کوتاه clip اگر clip status اضافه کردن فیلد

```
if model_data.get('clip_enabled', False):
```

```
    arcpy.AddField_management(shapefile_path, "ClipStat", "TEXT", field_length=10) # 8 کاراکتر
```

```
    arcpy.AddField_management(shapefile_path, "ClipXMin", "FLOAT") # 7 کاراکتر
```

```
    arcpy.AddField_management(shapefile_path, "ClipXMax", "FLOAT") # 7 کاراکتر
```

```
    arcpy.AddField_management(shapefile_path, "ClipYMin", "FLOAT") # 7 کاراکتر
```

```
    arcpy.AddField_management(shapefile_path, "ClipYMax", "FLOAT") # 7 کاراکتر
```

# Feature Class پر کردن

```
if model_data.get('clip_enabled', False):
```

```
    fields = ["SHAPE@", "Value", "Elev", "Class",
```

```
              "X_Coord", "Y_Coord", "Z_Coord", "Vol_m3",
```

```

        "ClipStat", "ClipXMin", "ClipXMax", "ClipYMin", "ClipYMax"]

    else:

        fields = ["SHAPE@", "Value", "Elev", "Class",
                  "X_Coord", "Y_Coord", "Z_Coord", "Vol_m3"]

    print(f' Adding {len(z_idx):,} blocks to Shapefile...')

    batch_size = 10000
    total_points = len(z_idx)
    block_volume = CELL_SIZE_XY * CELL_SIZE_XY * CELL_SIZE_Z

    with arcpy.da.InsertCursor(shapefile_path, fields) as cursor:
        for start_idx in range(0, total_points, batch_size):
            end_idx = min(start_idx + batch_size, total_points)

            for idx in range(start_idx, end_idx):
                x = x_coords[x_idx[idx]]
                y = y_coords[y_idx[idx]]
                z = z_coords[z_idx[idx]]
                value = values[idx]

                point = arcpy.Point(x, y, z)
                point_geometry = arcpy.PointGeometry(point, spatial_ref)

                if model_data.get('clip_enabled', False):
                    cursor.insertRow([point_geometry, value, z, class_name,
                                     x, y, z, block_volume,
                                     "CLIPPED",

```

```

        model_data.get('model_x_min', 0),
        model_data.get('model_x_max', 0),
        model_data.get('model_y_min', 0),
        model_data.get('model_y_max', 0)])

    else:

        cursor.insertRow([point_geometry, value, z, class_name,
                           x, y, z, block_volume])

    if end_idx % 50000 == 0:
        print(f"    Processed {end_idx:,}/{total_points:,} blocks")

print(f"    □ Shapefile created: {shapefile_path}")

# ایجاد فایل پروجکشن
prj_path = shapefile_path.replace(".shp", ".prj")
try:
    spatial_ref.saveAsXMLFile(prj_path)
except:
    pass

return True

except Exception as e:
    print(f"    □ Error creating Shapefile: {str(e)}")
    return False

def create_visualizations(self, model_data):
    """

```

برای هر کلاس D ایجاد تصاویر ۳

"""

```
print("\n" + "=" * 70)
```

```
clip_status = "CLIPPED" if model_data.get('clip_enabled', False) else "FULL"
```

```
print(f"□ CREATING 3D VOXEL VISUALIZATIONS FOR {self.output_name} ({clip_status})")
```

```
print("=" * 70)
```

```
block_model = model_data['block_model']
```

```
for class_info in self.class_ranges:
```

```
    class_name, min_val, max_val, description = class_info
```

```
    print(f"\n□ Processing: {class_name}")
```

```
    # پیدا کردن نقاط برای این کلاس
```

```
    points_idx = self.get_points_for_class(block_model, min_val, max_val)
```

```
    if len(points_idx[0]) == 0:
```

```
        print(f"□ No points found for {class_name}")
```

```
        continue
```

```
    z_idx, y_idx, x_idx = points_idx
```

```
    values = block_model[points_idx]
```

```
    print(f" Found {len(z_idx):,} blocks in requested range {min_val:.1f} to {max_val:.1f}")
```

```
    print(f" Actual data range: {values.min():.3f} to {values.max():.3f}")
```

```
    # محاسبه محدوده کلی
```

```
    all_values = block_model[~np.isnan(block_model)]
```

```
global_min = all_values.min()
```

```
global_max = all_values.max()
```

```
print(f" Color based on GLOBAL range: {global_min:.3f} to {global_max:.3f}")
```

```
print(f" All points will be displayed as 3D voxels (normalized based on global range)")
```

```
print(f" Color map: DIRECT BLUE-RED (NO WHITE)")
```

```
print(f" Z-axis range: 850m to 1200m (Fixed)")
```

```
print(f" Z-axis ticks: 50m intervals (875, 925, 975, 1025, 1075, 1125)")
```

```
print(f" X and Y axes displayed as integers (meters)")
```

```
# ایجاد تصویر ۳D Voxel
```

```
self.create_3d_voxel_visualization(class_info, points_idx, values, model_data)
```

```
def process_all_classes(self, model_data):
```

```
    """
```

```
    پردازش همه کلاس‌ها و ایجاد خروجی‌ها
```

```
    """
```

```
    print("\n" + "=" * 70)
```

```
    clip_status = "CLIPPED" if model_data.get('clip_enabled', False) else "FULL"
```

```
    print(f"□ PROCESSING ALL CLASSES FOR {self.output_name} ({clip_status})")
```

```
    print("=" * 70)
```

```
    block_model = model_data['block_model']
```

```
    # محاسبه محدوده کلی برای نمایش
```

```
    all_values = block_model[~np.isnan(block_model)]
```

```
    global_min = all_values.min()
```

```
    global_max = all_values.max()
```



```

# نمایش اطلاعات کلی مدل

print(f"GLOBAL MODEL RANGE: {global_min:.3f} to {global_max:.3f}")

print(f"COLOR RANGE SAME FOR ALL CLASSES: Blue={global_min:.3f},
Red={global_max:.3f}")

print(f"Z-AXIS RANGE: 850m to 1200m (Fixed)")

print(f"Z-AXIS TICKS: 50m intervals (875, 925, 975, 1025, 1075, 1125)")

print(f"X AND Y AXES: Displayed as integers (meters)")

if model_data.get('clip_enabled', False):

    print(f"\n□ CLIP EXTENT INFORMATION:")

    print(f" X (Easting): {model_data['model_x_min']:.1f}m to
{model_data['model_x_max']:.1f}m")

    print(f" Y (Northing): {model_data['model_y_min']:.1f}m to
{model_data['model_y_max']:.1f}m")

    print(f" Width: {(model_data['model_x_max'] - model_data['model_x_min']):.1f} m")

    print(f" Height: {(model_data['model_y_max'] - model_data['model_y_min']):.1f} m")

    print(f" Area: {((model_data['model_x_max'] - model_data['model_x_min']) *
(model_data['model_y_max'] - model_data['model_y_min'])):.0f} m²")

    print(f" Z (Elevation): {model_data['model_z_min']:.1f}m to
{model_data['model_z_max']:.1f}m")

    print(f" Model Volume: {(model_data['x_cells'] * model_data['y_cells'] * model_data['z_cells'] *
CELL_SIZE_XY * CELL_SIZE_XY * CELL_SIZE_Z):.0f} m³")

print()

# ایجاد Shapefile ها

print("\n□ Creating Shapefiles...")

for class_info in self.class_ranges:

    class_name, min_val, max_val, description = class_info

```

```

print(f"\n□ Processing Shapefile: {class_name}")

points_idx = self.get_points_for_class(block_model, min_val, max_val)

if len(points_idx[0]) == 0:
    print(f" □ No points found for {class_name}")
    continue

z_idx, y_idx, x_idx = points_idx
values = block_model[points_idx]

print(f" Found {len(z_idx):,} blocks")

self.create_single_shapefile(class_info, points_idx, values, model_data)

# ایجاد تصاویر ۳D
self.create_visualizations(model_data)

clip_status = "CLIPPED" if model_data.get('clip_enabled', False) else "FULL"
print(f"\n□ PROCESSING COMPLETED FOR '{self.output_name}' ({clip_status})!")
print(f"□ Outputs created:")
print(f" □ Shapefiles: {len(self.class_ranges)} shapefiles")
print(f" □ 3D Visualizations: {len(self.class_ranges)} 3D voxel images (DIRECT BLUE-RED)")
print(f" □ 2D Plots: {len(self.class_ranges)} 2D reference plots (DIRECT BLUE-RED)")
print(f"\n□ COLOR RANGE INFORMATION:")
print(f" • Color based on GLOBAL MODEL RANGE: {global_min:.3f} to {global_max:.3f}")
print(f" • SAME COLOR RANGE FOR ALL CLASSES")
print(f" • Start (Blue): {global_min:.3f} = {self.color_maps['start_color']}")
print(f" • End (Red): {global_max:.3f} = {self.color_maps['end_color']}")

```

```

print(f" • All points normalized to [0-1] based on global range")

print(f"\n□ AXES SETTINGS:")

print(f" • Z-axis range: 850m to 1200m (Fixed)")

print(f" • Z-axis ticks: 50m intervals (875, 925, 975, 1025, 1075, 1125)")

print(f" • X and Y axes displayed as integers (meters)")


if model_data.get('clip_enabled', False):

    print(f"\n□ CLIP INFORMATION:")

    print(f" • Clip Extent: X[ {model_data['model_x_min']:.1f} - {model_data['model_x_max']:.1f} ],
"

        f"Y[ {model_data['model_y_min']:.1f} - {model_data['model_y_max']:.1f} ]")

    print(f" • Clip Area: {((model_data['model_x_max'] - model_data['model_x_min']) *
(model_data['model_y_max'] - model_data['model_y_min'])):.0f} m²")

    print(f" • Model Dimensions:
{model_data['x_cells']} × {model_data['y_cells']} × {model_data['z_cells']} voxels")


print(f"\n□ GENERAL SETTINGS:")

print(f" • Input Type: {self.input_type}")

print(f" • ALL POINTS DISPLAYED - NO SAMPLING")

print(f" • Color Map: DIRECT BLUE-RED (NO WHITE)")

print(f" • Voxel size: {CELL_SIZE_XY}m × {CELL_SIZE_XY}m × {CELL_SIZE_Z}m")


#
=====
=
# □ تابع اصلی اجرا
#
=====
=

def main():

```

```
"""
```

```
تابع اصلی اجرای مدلسازی
```

```
"""
```

```
print("\n" + "=" * 70)
```

```
clip_status = "CLIPPED" if CLIP_ENABLED else "FULL"
```

```
print(f'3D VOXEL MODEL GENERATOR - {output_name} ({clip_status})')
```

```
print("=" * 70)
```

```
try:
```

```
    # ایجاد شیء مدل ساز 1.
```

```
    print(f'Starting processing for '{output_name}'...')
```

```
    modeler = GDB3DModeler()
```

```
    # لیست کردن رسترها 2.
```

```
    rasters = modeler.list_rasters()
```

```
    if not rasters:
```

```
        print("No rasters found. Exiting.")
```

```
        return
```

```
    # مطابقت با سطوح ارتفاعی 3.
```

```
    modeler.match_rasters_to_elevations()
```

```
    if not modeler.rasters_info:
```

```
        print("No valid rasters found. Exiting.")
```

```
        return
```

```
    # بررسی صحت رسترها 4.
```

```
if not modeler.validate_rasters():

    print("Not enough valid rasters. Exiting.")

    return


# 5. ایجاد مدل بلوکی سه بعدی

model_data = modeler.create_3d_block_model()


if model_data is None:

    print("Error creating block model. Exiting.")

    return


# 6. پردازش همه کلاس‌ها و ایجاد خروجی‌ها

modeler.process_all_classes(model_data)


except KeyboardInterrupt:

    print("\n❑ Processing stopped by user.")

except Exception as e:

    print(f"\n❑ Error: {str(e)}")

    import traceback

    traceback.print_exc()

finally:

    print("\n❑ Program finished.")


#
=====
=

# اجرای برنامه
```

```
#
=====

=

if __name__ == "__main__":
    # بررسی نصب arcpy
    try:
        import arcpy
        print("□ ArcPy loaded")

        # اجرای اصلی
        main()

    except ImportError:
        print("□ ArcPy not found!")
        print("Please install ArcGIS Pro or ArcMap.")
```