پیوست:

کدهای پایتون روش‌های داده‌محور

ایجاد نقاط معدنی با شبکه‌بندی متراکم و دقیق

```
# ================================
#  بخش ۱: وارد کردن کتابخانه‌ها  #
# ================================
import arcpy  # کتابخانه اصلی ArcGIS برای پردازش داده‌های مکانی 
import os     # برای کار با مسیرها و سیستم فایل 
import math   # برای محاسبات ریاضی 

# ================================
# بخش ۲: تنظیمات محیطی ⚙ 
# ================================
#  پارامتر ورودی ۱: مسیر پایگاه داده Geodatabase
#  اشاره کند .gdb باید به یک فایل یا
#  کاربر باید این مسیر را اصلاح کند یا
arcpy.env.workspace = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Gosal.gdb"

#  پارامتر ورودی ۲: اجازه رونویسی خروجی‌ها
#  اگر یا True باشد، فایل‌های موجود بازنویسی می‌شوند
#  اگر یا False باشد، با خطا مواجه می‌شود
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: 0.5)

      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))

#  ایجاد  مرحله ۸: Feature Class موقت
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
)

#  مرحله ۹: درج نقاط در Feature Class
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: (عدد اعشاری، پیش‌فرض: 2 متر) اندازه سلول شبکه

        خروجی:
        - (عدد صحیح) تعداد نقاط ایجاد شده

        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)")

# مرحله ۵: ایجاد fishnet (شبکه ماهیگیری)
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 ستون‌ها
    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: (عدد اعشاری، پیش‌فرض: 0.5 متر) فاصله بین نقاط

    خروجی:
    - (عدد صحیح) تعداد نقاط ایجاد شده

    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

# □ ایجاد مرحله ۴: Feature Class خروجی
out_fc = arcpy.CreateFeatureclass_management(
    arcpy.env.workspace,  # مسیر جی‌دی‌بی
    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("\nمرحله ۱: 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'):  # شروع با حرف شرط نام: '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"□ تعداد {len(o_polygons)} پلیگون با پیشوند 'o' یافت شد.\n")

#    مرحله ۲: پردازش هر پلیگون
print("مرحله ۲: پردازش پلیگون‌ها")
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 ۲-۱: 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 ۲-۳: انتخاب روش بر اساس مساحت
    if area < 10:  #    بسیار کوچک
        print("    ایجاد نقطه مرکزی - پلیگون بسیار کوچک")
        arcpy.FeatureToPoint_management(
            polygon_fc,
            output_name,
            "CENTROID"
        )
        count = 1

    elif area < 100:  #    کوچک
        print("    استفاده از روش ساده - پلیگون کوچک")
        count = simple_grid_points(polygon_fc, output_name, spacing=0.25)

    else:  #    متوسط
        print("    استفاده از روش Fishnet")
        # محاسبه cell size بر اساس مساحت
```

```
            cell_size = math.sqrt(area) / 5  # نسبت به ریشه مساحت
            cell_size = max(0.5, min(cell_size, 5))  # محدوده 0.5 تا 5 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}")

    # مرحله ۴: نمایش Feature Class های ایجاد شده
    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}: اطلاعات نامعلوم")
    else:
        print("  ⚠ هیچ Feature Class ای ایجاد 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 فایل‌ها ")
    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:
- کد از حافظه موقت (in_memory) استفاده می‌کند
- 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:

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

5. گزارش‌گیری:

- نتایج در کنسول نمایش داده می‌شوند
- می‌توان خروجی را به فایل لاگ هم ذخیره کرد

پشتیبانی:

- در صورت مشکل با توسعه‌دهنده 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 (فولدر حاوی GDBها)
#   - input_gdb: Geodatabase حاوی رسترهای لیتولوژی
#   - sample_gdb: Geodatabase حاوی نقاط معدنی (پیشوند 'o')
#   - output_gdb: Geodatabase خروجی (می‌تواند همان input_gdb باشد)
#
# ⚠ نکته: مسیرها باید به درستی تنظیم شوند و GDBها باید موجود باشند ⚠
#
================================================================================
====


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 می‌شوند فقط رسترهای حاوی متن خاص فیلتر باشد،
#   - ALTERNATIVE_TEXT: متن جایگزین برای فیلتر کردن (مثلاً "ColorRaster")
#   - ZERO_LAST_CLASS: اگر True می‌شود کلاس آخر با احتمال صفر تحلیل باشد،
#
#   برای انتخاب تمام رسترها، پیشنهاد: USE_RECLASSIFY_FILTER را False کنید  #
#
================================================================================
====


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 100% نقاط :(0-10m) نزدیک‌ترین منطقه
#   حذف 80% نقاط :(10-20m) منطقه دوم
#   حذف 60% نقاط :(20-35m) منطقه سوم
#   حذف 40% نقاط :(35-50m) منطقه چهارم
#   حذف 20% نقاط :(50-70m) منطقه پنجم
#   بدون حذف (100% حفظ) :mدورتر از 70
#
================================================================================

GRID_SPACING = 5.0  #    (متر) فاصله بین نقاط شبکه

DISTANCE_ZONES = [
    (0, 10, 1.0),    # 0-10 نقاط 100% حذف متر:
    (10, 20, 0.8),   # 10-20 نقاط 80% حذف متر:
    (20, 35, 0.6),   # 20-35 نقاط 60% حذف متر:
    (35, 50, 0.4),   # 35-50 نقاط 40% حذف متر:
    (50, 70, 0.2),   # 50-70 نقاط 20% حذف متر:
]

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

#
================================================================================

#   بخش ۴: نمایش تنظیمات برنامه
#
================================================================================
```

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

print(f"\n{'='*80}")
print("⚙️ تنظیمات کلی برنامه")
print(f"{'='*80}")

print(f"\n📊 تنظیمات تحلیل رسترها:")
print(f"   - فیلتر Reclassify: {'فعال' if USE_RECLASSIFY_FILTER else 'غیرفعال'}")
print(f"   - متن جایگزین: '{ALTERNATIVE_TEXT}'")
print(f"   - لایه آخر با احتمال صفر: {'فعال' if ZERO_LAST_CLASS else 'غیرفعال'}")

print(f"\n🎯 تنظیمات ایجاد نقاط غیرمعدنی (ثابت):")
print(f"   - فاصله شبکه: {GRID_SPACING} متر")
print(f"   - روش حذف: {REMOVAL_METHOD}")
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:
#   1. لیست کردن رسترهای لیتولوژی از input_gdb
#   2. لیست کردن نقاط معدنی (با پیشوند 'o') از sample_gdb
#   3. تطبیق رسترها و نقاط بر اساس ارتفاع (عدد در نام فایل)
#
#    نحوه تطبیق:
#   - ارتفاع از نام فایل استخراج می‌شود (مثلاً 12_5 یا 15)
#   - رستر و نقطه با ارتفاع یکسان جفت می‌شوند
#   - برای هر جفت، نقاط غیرمعدنی ایجاد می‌شود
#
# ================================================================================


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


# 1. لیست رسترها بر اساس تنظیمات کاربر
print(f"\n{'='*80}")
print("  در حال 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} → نام نامشخص: ارتفاع")

# 3. لیست تمام پوینت‌های نمونه
print(f"\n{'='*80}")
print("  در حال 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} → نام نامشخص: ارتفاع")

    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 تطبیق‌شده: {len(matched_pairs)}")

if len(matched_pairs) == 0:
    print("⚠□ هیچ جفت تطبیق‌شده‌ای 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"  (جایگزین) تطبیق: {raster['name']} ←→ {point['name']}")


if len(matched_pairs) == 0:
    print("  هنوز هیچ جفت تطبیق‌شده‌ای یافت نشد!")
    exit()


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

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

def create_fixed_nonmineral_points(raster_path, mineral_points_fc, output_name):
    """
    ایجاد نقاط غیرمعدنی با تنظیمات ثابت
    منطق:
    1. شبکه 5 متری در کل محدوده
    2. حذف بر اساس فاصله از نقاط معدنی:
       - 0-10m: 100 حذف%
       - 10-20m: 80 حذف%
       - 20-35m: 60 حذف%
       - 35-50m: 40 حذف%
       - 50-70m: 20 حذف%
       - >70m: بدون حذف
    """

    print(f"\n  ایجاد نقاط غیرمعدنی: {output_name}")
    print(f"  منطق: شبکه 5 متری + حذف متغیر بر اساس فاصله")

    try:
        # 1. خواندن رستر و استخراج محدوده
        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}")

# 2. ایجاد فایل خروجی
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)

# 3. ایجاد شبکه نقاط 5 متری
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  ایجاد شبکه {num_rows}×{num_cols} (فاصله {GRID_SPACING} متر)...")

# 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("\nمحاسبه 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"])

        # 6. حذف نقاط بر اساس مناطق فاصله ثابت
        print("\nحذف 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:,} نقطه، 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:
                # روش سیستماتیک (حذف هر nامین نقطه)
                step = zone_count / delete_count if delete_count > 0 else 1

                # علامت‌گذاری 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. (بدون حذف) نقاط با فاصله بیشتر از 70 متر
      last_max_dist = DISTANCE_ZONES[-1][1]  # 70 = آخرین محدوده
      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"  منطقه >{last_max_dist}m: {over_70_count:,} (بدون حذف) نقطه")
      arcpy.Delete_management(layer_name)

      # نمایش آمار مناطق
```

```
        print(f"\n آمار مناطق فاصله:")
        print("-" * 80)
        print(f"{'فاصله (m)':<15} {'باقی مانده':<15} {'حذف شده':<15} {'نقاط اولیه':<15} {'حذف':<10}")
        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':<15} {'':<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


    # 8. (طول فیلدها افزایش یافته) اضافه کردن فیلدهای اطلاعاتی
    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
            # بەروزرسانی 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)


    # 9. شمارش نهایی
    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 باقی‌مانده: {(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 می‌کند
# 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:,} نقطه ایجاد شد ({initial_count-final_count:,} نقطه حذف شد)")
    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
# گزارش در فایل متنی با timestamp ذخیره می‌شود
# 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"\nآمار 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⚙️ (ثابت) منطق استفاده شده:")
    print(f"   - فاصله شبکه: {GRID_SPACING} متر")
    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"     {min_d}-{max_d}m: حذف {ratio:.0%} (نگهداری {keep_ratio:.0%})")
    print(f"     >{DISTANCE_ZONES[-1][1]}m: بدون حذف (نگهداری) 100%")

    print(f"\n   جزئیات هر تراز:")
    print("-" * 100)
    print(f"{'تراز':<10} {'نام لایه':<15} {'اولیه':<12} {'نهایی':<12} {'حذف شده':<12} {'درصد باقی':<12}")
    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 ذخیره‌سازی: {sample_gdb}\n\n")

    f.write("منطق ثابت استفاده شده:\n")
    f.write(f"  - فاصله شبکه: {GRID_SPACING} متر\n")
    f.write(f"  - سیستماتیک **:** روش حذف\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"      >{last_max}m: **(%100** نگهداری **)** بدون حذف\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 باقی‌مانده: {overall_percentage:.1f}%\n\n")

    f.write("جزئیات هر تراز:\n")
    f.write("-" * 100 + "\n")
    f.write(f"{'تراز':<10} {'نام لایه':<15} {'اولیه':<12} {'نهایی':<12} {'حذف شده':<12} {'درصد باقی':<12}\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("⚠ هیچ نقطه غیرمعدنی ایجاد نشد!")

print(f"\n{'='*80}")
print("✓ ایجاد نقاط غیرمعدنی با منطق ثابت کامل شد!")
print(f"{'='*80}")

# ============================================================================
# بخش ۹: راهنمای تحلیل بیزین
# ============================================================================
#
# این بخش نکات مهم برای تحلیل بیزین را ارائه می‌دهد
# فرمول Prior Probability: P(A|B) = P(B|A) × P(A) / P(B)
# در این پروژه: Prior = (نقاط غیرمعدنی + نقاط معدنی) / نقاط معدنی
# ============================================================================
```

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

if success_count > 0:
    print(f"\n   اکنون می‌توانید تحلیل بیزین واقعی را شروع کنید!")
    print(f"   نقاط غیرمعدنی در: {sample_gdb}")
    print(f"   با پیشوند 'n' و منطق ثابت حذف متغیر")
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 5 متری در کل محدوده رستر")
print(f"2. حذف کامل نقاط در فاصله 0-10 متر از نقاط معدنی")
print(f"3. حذف 80% نقاط در فاصله 10-20 متر")
print(f"4. حذف 60% نقاط در فاصله 20-35 متر")
print(f"5. حذف 40% نقاط در فاصله 35-50 متر")
print(f"6. حذف 20% نقاط در فاصله 50-70 متر")
print(f"7. بدون حذف برای نقاط با فاصله بیشتر از 70 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.
# ============================================================================


# تنظیمات محیطی ArcGIS
arcpy.env.overwriteOutput = True  # اجازه رونویسی فایل‌های موجود
arcpy.env.workspace = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal"  # مسیر اصلی کار


# 1. مسیرهای 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 (ممکن است همان input_gdb باشد)
```

```
# 2.  الگوی نام رسترهای ورودی (قابل تنظیم)
#   این الگو برای جستجوی رسترها در input_gdb استفاده می‌شود.
#   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 = {  #  سطوح اطمینان برای گزارش‌گیری
    'خیلی زیاد': 0.8,  # احتمال ≥ ۸۰٪
    'زیاد': 0.6,       # احتمال ≥ ۶۰٪
    'متوسط': 0.4,      # احتمال ≥ ۴۰٪
    'کم': 0.0          # احتمال < ۴۰٪
}


# 6.  تنظیمات خروجی (قابل تنظیم)
create_probability_raster = True   #  ایجاد رستر احتمالات (احتمال معدنی بودن)
create_classification_raster = True  #  ایجاد رستر طبقه‌بندی (معدنی/غیرمعدنی)
create_report = True               #  ایجاد گزارش متنی از نتایج
clean_temp_files = True            #  پاکسازی فایل‌های موقت بعد از پردازش


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

```
#
#    این تابع تحلیل بیزین را برای یک ارتفاع خاص انجام می‌دهد.
#    ورودی‌ها:
#    - 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:
#    ۱. خواندن رستر و استخراج مقادیر از نقاط
#    ۲. محاسبه احتمالات شرطی
#    ۳. اعمال قضیه بیز
#    ۴. ایجاد رسترهای خروجی
#    ۵. گزارش‌گیری
#
# ============================================================================

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("❌ حداقل یک نقطه معدنی و یک نقطه غیرمعدنی نیاز است! داده کافی نیست.")
        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"  کلاس‌های شناسایی شده: {unique_classes}")

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

# 5. شمارش تکرار هر کلاس در نقاط معدنی و غیرمعدنی
print("\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 غیرمعدنی
    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} غیرمعدنی
({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 غیرمعدنی در این کلاس / کل نقاط غیرمعدنی
    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", "20 ,"ارتفاع),  # ارتفاع مورد تحلیل
    ("CLASS", "LONG", "کلاس"),         # کد کلاس لیتولوژی
```

```
        ("MINERAL_COUNT", "LONG", "تعداد معدنی"),  # تعداد نقاط معدنی در این کلاس
        ("NONMINERAL_COUNT", "LONG", "تعداد غیرمعدنی"),  # تعداد نقاط 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"),  # ۱=غیرمعدنی، ۰=معدنی
        ("CONFIDENCE", "TEXT", "اعتماد", "50")  # سطح اطمینان
    ]

    # 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:
    # - اگر احتمال معدنی بودن >= threshold: (کمتر از ۱۰۰۰) کد اصلی کلاس
    # - اگر احتمال معدنی بودن < threshold: (بیشتر یا مساوی ۱۰۰۰) کد اصلی + ۱۰۰۰
    if posterior_mineral[cls] >= threshold:
        new_value = cls  # کد اصلی - کلاس معدنی
    else:
        new_value = cls + 1000  # کد اصلی + ۱۰۰۰ کلاس غیرمعدنی

    # 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} با موفقیت انجام شد!")
    print(f"  زمان پردازش: {processing_time:.2f} ثانیه")


    # نمایش خلاصه نتایج
    print(f"\n  نتایج بیزین برای ارتفاع {height}:")
    print("-" * 120)
```

```
    print(f"{'کلاس':<8} {'P(C|M)':<10} {'P(C|NM)':<10} {'P(M|C)':<12} {'طبقه':<10} {'ضریب':<10} {'کد رستری':<12}")

    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} - رستر احتمالات (۰ تا ۱)")
    if class_output_name:
        print(f"  2. {class_output_name} - رستر طبقه‌بندی (کدهای معنادار)")
```

```
        # 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 ارتفاع‌ها: {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"تعداد رسترهای یافت شده با الگوی '{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"  ... و {len(all_rasters) - 10} رستر دیگر")
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 ۱.۰
                    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"\nنتایج:")
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"\nخروجی‌ها در {output_gdb}:")
```

```
    successful = [r for r in results if r['success']]
    for result in successful:
        print(f"\n ارتفاع {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("سیستم کدگذاری رستر خروجی:\n")
f.write("- مناطق نامشخص/بدون داده: کد **0**\n")
f.write(f"- کد X (X < 1000): کلاس X (معدنی، P >= {threshold})\n")
f.write(f"- کد X+1000 (X >= 1000): کلاس X (غیرمعدنی، P < {threshold})\n\n")

f.write("ارتفاع‌های موفق:\n")
for result in successful:
    f.write(f"- ارتفاع {result['height']}: ")
    if result.get('top_class'):
        f.write(f"کلاس برتر {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"، کلاس معدنی {len(result.get('mineral_classes', []))}")
    f.write("\n")
```

```
            f.write("\n" + "="*70 + "\n")
            f.write("نکات مهم:\n")
            f.write("1. رسترهای احتمالات (*_Bayesian_Prob) مقادیر احتمال معدنی بودن را نشان می‌دهند.\n")
            f.write("2. رسترهای طبقه‌بندی (*_Bayesian_Class) کلاس‌های معدنی و غیرمعدنی را نشان می‌دهند.\n")
            f.write("3. برای مشاهده نتایج در ArcGIS، از رمپ رنگی 'Classified' استفاده کنید.\n")
            f.write("4. کلاس‌های معدنی (کد < 1000) با رنگ سبز و غیرمعدنی (کد >= 1000) با رنگ قرمز نمایش داده می‌شوند.\n")
            f.write("="*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" - رسترهای احتمالات (*_Bayesian_Prob): مقادیر احتمال معدنی بودن (۰ تا ۱)")
print(f" - رسترهای طبقه‌بندی (*_Bayesian_Class): کلاس‌های معدنی و غیرمعدنی")
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 ***7*** پارامتر بیزین

```
import arcpy
import os
import re
from arcpy.sa import *
#
================================================================
====
#    ۱ بخش: تنظیمات اولیه و محیط ArcGIS
#
================================================================
====
#
#    ها و تعریف مسیرهاستسازی اکستنشن، فعالArcGISاین بخش شامل تنظیمات پایه.
#    پارامترهای اصلی در این قسمت قابل تنظیم هستند.
#
================================================================
====
```

```
# سازی اکستنشن فعال Spatial Analyst های رستری برای تحلیل
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"
#  نام و مسیر Geodatabase خروجی
output_gdb_name = "Bayesian.gdb"
output_gdb_path = os.path.join(main_path, output_gdb_name)
#  ایجاد Geodatabase خروجی 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"ℹ□ Geodatabase {output_gdb_name} از قبل موجود است.")
#
==========================================================================
====
#  ها و تنظیمات پردازش تعریف ورودی :بخش ۲
#
==========================================================================
====
#
#  این بخش شامل لیست Geodatabase های ورودی و تعریف تراز های ارتفاعی است.
#  ها بسته به نیاز پروژه قابل ویرایش هستنداین لیست.
#
==========================================================================
====
#  لیست Geodatabase های ورودی حاوی رستر های احتمالات بیزین
# هر GDB حاوی رستر های Bayesian_Prob_* برای تراز های مختلف است
```

```
input_gdbs = [
"EuclideanBuffers.gdb", # رسترهای حاصل از تحلیل Euclidean Buffers
"LineDensity.gdb", # رسترهای تراکم خطی
"Fault_Intersections.gdb", # رسترهای تقاطع های گسلی
"Intersections_Density.gdb", # رسترهای تراکم تقاطع هار
"Alteration_Rasters.gdb", # رسترهای دگرسانی
"Litho_Rasters.gdb", # رسترهای لیتولوژی
"Voronoi_Output.gdb" # رسترهای حاصل از تحلیل Voronoi
]
# تولید لیست ترازهای ارتفاعی مورد بررسی
# فرمول: start + n * step ≤ end
elevation_levels = []
start = 875 # ارتفاع شروع (متر)
end = 1125.1 # ارتفاع پایان (متر) + 0.1 برای شامل شدن 1125
step = 12.5 # گام افزایش ارتفاع (متر)
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")
# ============================================================================
# بخش ۳: توابع کمکی
# ============================================================================
```

```
#
#    شوندااین بخش شامل توابعی است که در پردازش اصلی استفاده می.
#    دهدهر 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.
    ورودی: نام رستر (رشته)
    خروجی: مقدار ارتفاع (اعشاری) یا None در صورت عدم یافتن
    الگوهای شناسایی:
1. فرمت‌های با underscore: o875_5 یا 912_5 → 912.5
2. فرمت‌های اعشاری: o912.5 یا 912.5 → 912.5
3. فرمت‌های عددی ساده: o912 یا 912 → 912.0
"""
#    الگوهای مختلف برای شناسایی ارتفاع در نام رستر
patterns = [
r'o?(\d+)_(\d+)', #    الگو ۱: o875_5 یا 912_5 مثل برای فرمت‌هایی
r'o?(\d+\.\d+)', #    الگو ۲: فرمت‌های اعشاری برای
r'o?(\d+)' #    الگو ۳: فرمت‌های عددی ساده برای
]
for pattern in patterns:
match = re.search(pattern, raster_name)
if match:
if len(match.groups()) == 2:
#    قسمت اول و دوم را با نقطه ترکیب کن: فرمت‌های underscore با برای
```

```
        return float(f"{match.group(1)}.{match.group(2)}")
    else:
        # تبدیل کن float مستقیماً به: های عددیبرای فرمت
        return float(match.group(1))
    # ⚠️ اگر هیچ الگویی مطابقت نداشت
    return None

# ============================================================================

# بخش ۴: پردازش اصلی برای هر تراز ارتفاعی

# ============================================================================

#
# این بخش هسته اصلی برنامه است که برای هر تراز ارتفاعی:
# ۱. کندآوری می‌های ورودی جمعGDBرسترهای مربوطه را از تمام
# ۲. کندرسترها را با هم جمع می
# ۳. کندرستر نهایی را ذخیره و آنالیز می
#
# حلقه اصلی: برای هر تراز ارتفاعی درelevation_levels
#
# ============================================================================

# 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 = []
#     جستجو در تمام Geodatabase های ورودی
for gdb_name in input_gdbs:
gdb_path = os.path.join(main_path, gdb_name)
#  بررسی وجود GDB
if arcpy.Exists(gdb_path):
try:
#     تنظیم workspace به GDB فعلی برای جستجو
arcpy.env.workspace = gdb_path
#     جستجوی رسترهای احتمالات بیزین (الگوی Bayesian_Prob_*)
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)
#  اگر ارتفاع استخراج شد و با تراز فعلی مطابقت دارد (با خطای ۰.۰۱)
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"  رستر '{raster}' از '{gdb_name}' اضافه شد.")
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"   آوری در حال جمع {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"  اضافه کردن رستر {i+1}: {os.path.basename(elevation_rasters[i])}")

sum_raster = sum_raster + Raster(elevation_rasters[i])

# 3.  گذاری و ذخیره رستر نهایینام

#  تبدیل نقطه به underscore در 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"  رستر '{output_name}' 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")
#  هاfloat 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.  ها ایجاد هرم (Pyramids) و آمار برای نمایش بهتر در ArcGIS
print(f"  ها و آمارایجاد هرم...")
arcpy.BuildPyramidsandStatistics_management(output_path)
print(f"  ها و آمار ایجاد شدندهرم.")
except Exception as e:
print(f"  خطا در پردازش تراز {elevation}: {e}")
#
================================================================================
#  بازگردانی تنظیمات و گزارش نهایی :بخش ۵
#
================================================================================
#
#  :شود و شاملاین بخش پس از اتمام پردازش اجرا می
# ۱.  بازگرداندن workspace به مسیر اصلی
# ۲. نمایش گزارش نهایی
# ۳. های ایجاد شدهکردن لایهلیست
#
================================================================================
```

```
#    بازگرداندن workspace به مسیر اصلی
arcpy.env.workspace = main_path
#    نمایش گزارش نهایی
print("\n" + "="*60)
print("    پردازش کامل شد!")
print(f"    ها در خروجی '{output_gdb_path}' ذخیره شدند.")
print("="*60)
#    خروجی Geodatabase های ایجاد شده در کردن لایه‌لیست
print("\n    های ایجاد شده در لایه Bayesian.gdb:")
#    تنظیم workspace به GDB کردن‌خروجی برای لیست
arcpy.env.workspace = output_gdb_path
#    جستجوی رسترهای ایجاد شده (با الگوی Bayesian_Sum_*)
output_rasters = arcpy.ListRasters("Bayesian_Sum_*")
if output_rasters:
print(f"    های ایجاد شده‌تعداد 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(" ⚠ هیچ لایه ای ایجاد نشده.")
# ============================================================================
# ۶ بخش: پاکسازی و خاتمه
# ============================================================================
#
#    این بخش منابع ArcGIS را آزاد می کند.
#    مهم: همیشه پس از اتمام کار با اکستنشن ها را CheckIn کنید.
# ============================================================================
#    آزادسازی اکستنشن Spatial Analyst
arcpy.CheckInExtension("Spatial")
print(" اکستنشن Spatial Analyst آزاد شد.")
print("\n" + "="*60)
print("برنامه با موفقیت به پایان رسید ")
```

```
print("="*60)

# ================================================================================

# بخش ۷: نکات مهم و راهنمایی

# ================================================================================

#

# نکات مهم برای استفاده از این اسکریپت:

#

# 1. ساختار پوشه‌ها:

# - مطمئن شوید تمام GDBهای ورودی در مسیر اصلی (main_path) موجود باشند.

# - هر GDB باید حاوی رسترهایی با الگوی 'Bayesian_Prob_*' باشد.

#

# 2. نام‌گذاری رسترها:

# - رسترها باید در نام خود ارتفاع را داشته باشند (مثلاً Bayesian_Prob_912_5).

# - فرمت‌های پشتیبانی شده: 912_5, 912.5, o912.5, o912_5

#

# 3. ترازهای ارتفاعی:

# - پیش‌فرض از ۸۷۵ تا ۱۱۲۵ متر با گام ۱۲ متر پردازش می‌شود.

# - برای تغییر محدوده، متغیرهای start، end و step را ویرایش کنید.

#

# 4. بهینه‌سازی ⚡:

# - برای تعداد زیاد رسترها، ممکن است پردازش زمان‌بر باشد.

# - از هرم‌ها (Pyramids) در ArcGIS برای نمایش سریع‌تر استفاده شده است.

#

# 5. عیب‌یابی:

# - اگر رستری یافت نشد، نام‌گذاری آن را بررسی کنید.
```

```
# -  اطمینان حاصل کنید که تمام GDB های ورودی در مسیر صحیح قرار دارند.

# - دسترسی به مسیرها را بررسی کنید.

#

# 6.  تفسیر نتایج:

# - رسترهای خروجی 'Bayesian_Sum_' ها هستندمجموع احتمالات از تمام لایه.

# - دهنده پتانسیل معدنی بالاتر استمقادیر بالاتر نشان.

# - های ورودی داردمحدوده 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"

# ================================================================================

# بخش ۳: آماده‌سازی محیط و فایل‌ها #
# ================================================================================
```

```
def setup_workspace_and_filter_rasters():
    """تنظیم workspace و فیلتر رسترها"""
    arcpy.env.workspace = GDB_PATH
    all_rasters = arcpy.ListRasters()

    print(f"   Geodatabase: {GDB_NAME}")
    print(f"   تعداد کل رسترها در GDB: {len(all_rasters)}")

    if RASTER_PREFIX:
        rasters = [r for r in all_rasters if r.startswith(RASTER_PREFIX)]
        print(f"   فیلتر رسترها با پیشوند '{RASTER_PREFIX}'")
        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("لطفاً RASTER_PREFIX را بررسی کنید.")
        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
```

```
#
==========================================================================
====
#    محاسبه بخش ۵: کلی Extent
#
==========================================================================
====

def calculate_overall_extent(rasters):
    """محاسبه محدوده جغرافیایی کلی همه رسترها"""
    print("\n محاسبه Extent کلی همه رسترها...")

    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 کلی 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)  # حذف NoData و مقادیر منفی

            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 پیش‌فرض:")
        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"⚠ تعداد طبقات {num_classes} برای طرح {color_scheme} تعریف نشده.")
            print(f"استفاده از نزدیک‌ترین: {closest} طبقه")
    else:
        print(f"⚠ طرح رنگی '{color_scheme}' تعریف نشده. استفاده از طرح پیش‌فرض 'blue_red'")
        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"\n  تنظیمات طبقه‌بندی:")
    print(f"  تعداد طبقات Legend: {num_classes}")

    # 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 مستطیل‌های 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

# ============================================================================
# ۹: ساخت فایل انیمیشن GIF
# ============================================================================

def create_gif_animation(images, output_path):
    """ساخت فایل GIF از فریم‌ها"""
    if len(images) > 1:
        print(f"\n🎬 ساخت انیمیشن GIF...")
        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❌ فریم کافی 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. محاسبه Extent کلی
```

```
    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: {600}ms")
        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)
```

ماشین‌بردار 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 می‌تواند 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

# ⚙️ پارامترهای الگوریتم SVM (Support Vector Machine)
C_value = 1.0               # پارامتر C: کنترل تعادل بین misclassification و margin
kernel_type = 'rbf'         # نوع هسته: 'linear', 'rbf', 'poly', 'sigmoid' (RBF پیشنهادی)
gamma_value = 'scale'       # مقدار گاما: 'scale', 'auto' (کنترل تأثیر نقاط یا عدد مشخص)
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: (وزن ۱.۵ برابر) نقاط معدنی، 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: تراز ارتفاعی (عدد اعشاری) یا None

    :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 (5. اعداد چهاررقمی با)
        r'(\d{3})_5',          #    912_5 (5. اعداد سه‌رقمی با)
        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)
            #    تبدیل فرمت underscore به نقطه اعشاری
            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:  #    عدد صحیح (مثلاً ۱۰۰۰)
    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:  #    عدد اعشاری 5. (مثلاً ۱۰۱۲.۵)
    elev_str = f"{int(elevation)}_5"  # تبدیل به فرمت underscore
    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_
    ]
```

```
#    جستجو در هر GDB
for gdb_name in raster_gdbs:
    gdb_path = os.path.join(input_gdb_path, gdb_name)

    # □ بررسی وجود GDB
    if arcpy.Exists(gdb_path):
        arcpy.env.workspace = gdb_path  #    تنظیم workspace

        #    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'):
    """
    یافتن فایل نقاط (Feature Class) برای تراز ارتفاعی مشخص

    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')

    خروجی: مسیر Feature Class یا None
    """
    gosal_gdb = os.path.join(input_gdb_path, "Gosal.gdb")  # مسیر GDB نقاط

    # 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:  #    5. عدد اعشاری
        elev_str = f"{int(elevation)}_5"  # تبدیل به فرمت underscore
        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 فایلی پیدا نشد

def create_temp_name(base_name):
```

```
    """
    ایجاد نام موقت معتبر برای فایل‌های میانی

        این تابع نام‌های معتبر برای فایل‌های موقت ArcGIS ایجاد می‌کند.

        ورودی: نام پایه (رشته)
        خروجی: نام معتبر ArcGIS
    """
    # حذف کاراکترهای غیرمجاز
    valid_name = re.sub(r'[^a-zA-Z0-9_]', '_', base_name)

    # اگر نام با عدد شروع شد، اضافه کن 'temp_'
    if valid_name[0].isdigit():
        valid_name = 'temp_' + valid_name

    # محدود کردن طول نام (حداکثر ۳۰ کاراکتر برای 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 می‌کند.
```

```
    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)
```

```
    # □ ایجاد Extent آرک‌جی‌آی‌اس
    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

    خروجی: True اگر موفق، False اگر ناموفق
```

```
"""
try:
    # ⚙ تنظیم پارامترهای محیطی ArcGIS
    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",  #  (نزدیک‌ترین همسایه) روش درونیابی
        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 همسان‌سازی: {e}")
        # روش جایگزین ساده‌تر
        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: y) آرایه برچسب‌ها
    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 (۲.۱ 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. آموزش مدل SVM
# 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"  ⚠ هیچ رستری برای تراز {elevation} یافت نشد!")
            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"    ... و {len(rasters_list)-5} رستر دیگر")

        # 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"  نقاط معدنی برای تراز {elevation} یافت نشد!")
        return None

    if not nonmineral_points:
        print(f"  نقاط غیرمعدنی برای تراز {elevation} یافت نشد!")
        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:]:  #    بررسی عدم وجود مقادیر null
                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"  خطا در خواندن داده‌ها: {e}")
    return None

#    مرحله ۶: تقسیم داده‌ها به آموزش و آزمون
print("\n۶.  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"  نمونه‌ها بسیار کم است ({n_samples})")
    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

    # مرحله ۸: آموزش مدل SVM با وزن کلاس
    print("\n8. آموزش SVM (1.2x با وزن کلاس معدنی)...")
```

```
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("  مدل SVM با وزن کلاس آموزش داده شد")
except Exception as e:
    print(f"  خطا در آموزش SVM: {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 (با وزن معدنی 1.2x):")
    print(classification_report(y_test, y_pred, target_names=['غیرمعدنی','معدنی']))
```

```
    except Exception as e:
        print(f"  ⚠ خطا در ارزیابی مدل: {e}")
        test_score = 0.0

    # مرحله ۱۰: آماده‌سازی خروجی Geodatabase
    print("\n10. آماده‌سازی خروجی...")

    # ایجاد GDB خروجی اگر وجود ندارد
    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 خروجی ایجاد شد: {output_gdb_name}")
        except Exception as e:
            print(f"  ✗ خطا در ایجاد GDB خروجی: {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 1 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("  ✗ 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("\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 سه‌بعدی (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

# شناسایی پیکسل‌های معتبر (غیر صفر و غیر NaN)
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...")

    #   آرایه‌های خروجی با مقدار NoData اولیه
    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 1 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}")

    #    تبدیل آرایه NumPy به رستر ArcGIS
    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"  ذخیره شد (سیستم مختصات: {common_spatial_ref.name})")

except Exception as e:
    print(f"  خطا در ذخیره رستر پیش‌بینی: {e}")
    return None

# ذخیره رستر احتمالات
try:
    print(f"  ذخیره رستر احتمال: {output_prob}")

    # 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)

    # کپی به GDB خروجی
    arcpy.CopyRaster_management(
        temp_prob,
        os.path.join(output_gdb_path, output_prob),
        pixel_type="32_BIT_FLOAT"  # نوع پیکسل
    )

    print(f"  ذخیره شد (سیستم مختصات: {common_spatial_ref.name})")

except Exception as e:
    print(f"  خطا در ذخیره رستر احتمال: {e}")
    # ⚠ 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"✅ ترازیابی با موفقیت پردازش شد {elevation}!")
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__":
```

```
# ⚙ تنظیمات اولیه ArcGIS
arcpy.env.overwriteOutput = True
arcpy.CheckOutExtension("Spatial")  # فعال‌سازی اکستنشن Spatial

print("=" * 70)
print("شروع تحلیل SVM برای تمام ترازهای ارتفاعی")
print("=" * 70)
print(f"تعداد ترازها: {len(elevations)}")
print(f"مسیر ورودی: {input_gdb_path}")
print(f" خروجی GDB: {output_gdb_path}")
print(f"⚙ پارامترهای SVM: 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 پردازش شد ({i}/{len(elevations)}) تراز {elevation}")
    else:
        failed_count += 1
        print(f"\n پردازش نشد ({i}/{len(elevations)}) تراز {elevation}")

    # نمایش پیشرفت
    print(f"\n پیشرفت: {i}/{len(elevations)} تراز")
    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':<10} {'نمونه‌ها':<8} {'رسترها':<8} {'دقت':<10} {'وزن':<6} {'ابعاد':<12}")
    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("  خلاصه نتایج تحلیل SVM (1.2 وزن معدنی x)\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"{'تراز':<10} {'رسترها':<8} {'نمونه‌ها':<10} {'دقت':<10} {'وزن':<6} {'پیش‌بینی':<25} {'سیستم مختصات':<30}\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"⚠□ خطا در 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"   نتایج در {output_gdb_path} ذخیره شده‌اند")
    print(f"   ✅ رستر 2 هر تراز شامل Probability و Prediction:")
    print(f"   سیستم مختصات به درستی حفظ شده است")
    print(f"   ⚖️ وزن نقاط معدنی: {class_weights[1]}x")
    print(f"   خلاصه نتایج در {summary_file} ذخیره شد")
    print(f"{'=' * 70}")


# ============================================================================
# بخش ۵: نکات مهم و راهنمایی
# ============================================================================
#
# نکات کلیدی برای استفاده بهینه از این اسکریپت:
#
# 1. ✅ ساختار داده‌ها:
#    - اطمینان حاصل کنید که تمام GDBهای ورودی در مسیر مشخص شده وجود دارند.
#    - نام‌گذاری رسترها باید شامل ارتفاع باشد (مثلاً f1000_ColorRaster).
#    - نقاط معدنی و غیرمعدنی باید در Gosal.gdb باشند.
#
# 2. ⚙️ تنظیمات SVM:
#    - پارامتر C: کنترل تعادل بین حاشیه و خطا (کمتر: مقادیر کوچک‌تر overfit).
#    - وزن کلاس: برای مقابله با عدم تعادل داده‌ها (1.5: معدنی x).
#    - هسته RBF: مناسب برای داده‌های غیرخطی.
```

```
#
# 3. □ سیستم مختصات:
#    - اسکریپت به طور خودکار سیستم مختصات را حفظ می‌کند.
#    - همه رسترهای خروجی دارای سیستم مختصات یکسان هستند.
#    - اندازه پیکسل خروجی: 1 متر.
#
# 4. ⚡ بهینه‌سازی اجرا:
#    - برای تعداد زیاد رسترها، ممکن است پردازش زمان‌بر باشد.
#    - برای سرعت بیشتر استفاده شده است (memory workspace) از حافظه موقت.
#    - پردازش بلاک‌بندی برای جلوگیری از overload حافظه.
#
# 5. عیب‌یابی:
#    - اگر رستری پیدا نشد، نام‌گذاری آن را بررسی کنید.
#    - خطاهای سیستم مختصات: بررسی کنید که همه رسترها projection داشته باشند.
#    - خطاهای حافظه: تعداد رسترهای ورودی را کاهش دهید.
#
# 6. تفسیر نتایج:
#    - رستر Prediction: (معدنی) 1 و (غیرمعدنی) 0 مقادیر.
#    - رستر Probability: (مثلاً 0.8 = 80٪ احتمال معدنی بودن) 1 تا 0 احتمالات بین.
#    - 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 مسیر اصلی پوشه پروژه (رشته):
# مثال: "M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal"
# این مسیر باید حاوی تمام پایگاه‌های جغرافیایی زیرمجموعه باشد #

# output_folder مسیر پوشه خروجی برای ذخیره نتایج (رشته):
# 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 حاوی رسترها (GDB) (لیست رشته‌ها): لیست نام پایگاه‌های جغرافیایی
```

```
# هر GDB باید حاوی رسترهای لایه‌های مختلف باشد
# ترتیب GDBها مهم نیست اما باید در مسیر base_path موجود باشند
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"         # حاوی خروجی‌های Voronoi
]

# training_gdb مسیر کامل GDB حاوی پلیگون‌های آموزشی (رشته): 
# این GDB باید حاوی Feature Classهای آموزشی باشد
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    #

# PIXEL_SIZE رزولوشن خروجی به متر (عدد صحیح): 
# اندازه پیکسل رسترهای خروجی را تعیین می‌کند
PIXEL_SIZE = 1  # رزولوشن خروجی (متر)

# VALID_RASTER_SUFFIXES پسوندهای معتبر برای نام رسترها (لیست رشته‌ها): 
# فقط رسترهایی با این پسوندها پردازش می‌شوند
```

```
VALID_RASTER_SUFFIXES = ["Reclassify", "raster", "Raster", "ColorRaster",
"Azimuth_raster"]


# TRAINING_PREFIX پیشوند نام پلیگون‌های آموزشی (رشته):
پلیگون‌های آموزشی باید با این پیشوند شروع شوند #
TRAINING_PREFIX = "o"


# ==================== پارامترهای مدل Self-Attention ====================
# ⚙ پارامترهای معماری و آموزش مدل Transformer


MODEL_PARAMS = {
    'embed_dim': 32,          # بعد embedding - بعد فضای ویژگی‌های تبدیل شده
    'num_heads': 4,           # تعداد headهای attention - موازی attention برای محاسبه
    'ff_dim': 64,             # بعد لایه feed-forward - درون بلوک Transformer
    'num_transformer_blocks': 2,  # تعداد بلوک‌های transformer - عمق مدل
    'mlp_units': [64, 32],    # واحدهای لایه MLP - شبکه عصبی پس از Transformer
    'dropout_rate': 0.2,      # نرخ dropout - برای جلوگیری از overfitting
    'batch_size': 32,         # اندازه batch - تعداد نمونه‌ها در هر تکرار آموزش
    'epochs': 50,             # تعداد epochها - تعداد دفعات کامل آموزش روی داده‌ها
    'learning_rate': 0.001,   # سرعت به‌روزرسانی وزن‌ها - نرخ یادگیری
    'validation_split': 0.2,  # نسبت داده validation - درصد داده برای اعتبارسنجی
}


# ==================== کلاس برای ذخیره گزارش ====================


class ReportLogger:
    """
        کلاس ReportLogger
    وظیفه: ذخیره گزارش اجرا هم در فایل و هم نمایش در کنسول
```

پارامترهای ورودی:

- *log_file_path* مسیر کامل فایل گزارش **(***رشته***):**

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')
    # تغییر stdout برای ذخیره در فایل و نمایش در کنسول
    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 30 ثانیه

    پارامترهای ورودی:
    - report_interval_seconds (پیش‌فرض: 30 ثانیه): فاصله زمانی گزارش‌دهی (عدد)

    متدهای مهم:
    - on_epoch_begin(): شروع هر epoch
    - on_epoch_end(): پایان هر epoch
    - 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 {}

        # نمایش متریک‌های validation
        if logs:
            for key, value in logs.items():
                if key.startswith('val_'):
                    print(f"   {key}: {value:.4f}")

        print(f"   میانگین زمان هر اپوک: {avg_epoch_time:.1f} ثانیه")
        print(f"   زمان تخمینی باقی‌مانده: {estimated_remaining/60:.1f} دقیقه")

        # گزارش هر 30 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"زمان تخمینی باقی‌مانده: {estimated_remaining/60:.1f} دقیقه")
```

```
        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
    وظیفه: زمان‌سنجی دقیق اپوک‌ها و batchها
    """

    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 50 batch
        if self.batch_count % 50 == 0:
```

```
        if len(self.batch_times) >= 10:
            avg_batch_time = np.mean(self.batch_times[-10:])
            print(f"  ا 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 (رشته):

    خروجی:
    - رشته: تراز ارتفاعی استخراج شده یا None
```

```
    """
    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
    وظیفه: پیدا کردن تمام رسترهای مربوط به یک تراز ارتفاعی در GDBهای مختلف

    پارامترهای ورودی:
    - elevation (رشته): تراز ارتفاعی مورد جستجو

    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 تراز ارتفاعی مورد جستجو :(رشته)

    خروجی:
    - رشته: مسیر کامل پلیگون آموزشی یا None
    """
    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 مسیر پلیگون آموزشی :(رشته)
    - template_raster (رشته یا arcpy.Raster): رستر الگو برای ابعاد
```

خروجی:

- آرایه *numpy:* ماسک باینری ***1)*** برای داخل پلیگون، ***0*** برای خارج***)***

"""

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
    وظیفه: خواندن رستر و تبدیل به آرایه numpy با مدیریت NoData

    پارامترهای ورودی:
    - raster_path مسیر رستر ورودی (رشته):
    - template_shape (اختیاری) شکل الگو برای تطبیق ابعاد (تاپل):

    خروجی:
    - آرایه numpy: داده‌های رستر به صورت آرایه یا None در صورت خطا
    """
    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"  تطبیق اندازه از {raster_array.shape} به {template_shape}")
```

```
            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"  X 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 مسیرهای رسترهای ورودی (لیست)
    - training_polygon مسیر پلیگون آموزشی (رشته)

    خروجی:
    - تاپل: (X_train, y_train, X_all, y_all, original_shape, valid_mask) یا None در صورت خطا
```

```
"""
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"\n پردازش {len(rasters)} رستر...")

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"    ✓ شکل | پردازش شد: {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("✗ هیچ آرایه رستری پردازش نشد!")
    return None, 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("X 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"    (معدن) 1 کلاس: {class_1_count:,}")
print(f"    (غیرمعدن) 0 کلاس: {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 1 در آموزش: {np.sum(y_train == 1):,}")
        print(f"    کلاس 0 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):
    """
        کلاس TransformerBlock
    پیاده‌سازی یک بلوک Self-Attention برای Transformer وظیفه:

    پارامترهای ورودی:
```

```
    - embed_dim بعد :(عدد) embedding
    - num_heads تعداد :(عدد) headهای attention
    - ff_dim بعد لایه :(عدد) feed-forward
    - rate نرخ :(عدد) dropout
    """

    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*

وظیفه**:** ایجاد مدل *Self-Attention* کامل برای *Transformer*

پارامترهای ورودی:

- *num_features* تعداد ویژگی‌های ورودی **(**عدد**):**

خروجی:

- مدل *Keras:* مدل *Transformer* کامپایل شده

"""

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 پارامترهای مدل **(**دیکشنری**):** (پیش‌فرض: MODEL_PARAMS)

خروجی:

- تاپل: (predictions, model, history, scaler)

"""

print("\n شروع آموزش مدل Self-Attention")

print("=" * 60)

print("تقسیم داده‌ها به آموزش و validation...")

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" ✓ داده‌های آموزش: {len(X_train_split):,} نمونه")

print(f" ✓ داده‌های validation: {len(X_val):,} نمونه")

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 خلاصه مدل 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"  اندازه batch: {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(" متریک‌های validation:")
    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("⚠ 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 (آرایه numpy): مقادیر پیش‌بینی شده
    - valid_mask (آرایه numpy): 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
)

# ذخیره به صورت TIFF
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("  ⚠ تنظیم NoData Value (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}")


    # ذخیره به صورت ASCII (برای سازگاری بیشتر) هم
    asc_path = os.path.join(output_folder, f"{raster_name}.asc")
    print(f"\n  ذخیره نسخه ASCII برای سازگاری بیشتر...")


    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  آماره‌های 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}")

    # ذخیره اضطراری به صورت numpy
    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:
    - رشته: مسیر فایل خروجی یا در صورت خطا None
    """

    print("\n" + "=" * 70)
    print(f"  شروع تحلیل Self-Attention Deep Learning برای تراز {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"❌ هیچ رستری برای تراز {elevation_level} یافت نشد!")
    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"❌ هیچ پلیگون آموزشی برای تراز {elevation_level} یافت نشد!")
    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("❌ 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"   تحلیل Self-Attention برای تراز {elevation_level} با موفقیت تکمیل شد!")
print("=" * 70)
print(f"   خلاصه نتایج:")
print(f"   تراز ارتفاعی: {elevation_level}")
print(f"   تعداد نمونه‌های آموزشی: {len(X_train):,}")
print(f"   تعداد پیکسل‌های پیش‌بینی شده: {len(X_all):,}")
```

```
    print(f"   تعداد ویژگی‌ها: {X_train.shape[1]}")
    print(f"   زمان کل تحلیل: {total_duration/60:.1f} دقیقه")

    if output_path:
        if output_path.endswith('.npz'):
            print(f"   ⚠️ خروجی به صورت فایل numpy ذخیره شد: {output_path}")
        else:
            print(f"   ✅ رستر در پوشه ذخیره شد: {output_path}")
    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   ✅ مدل ذخیره شد: {model_save_path}")
    except Exception as e:
        print(f"⚠️ خطا در ذخیره مدل: {e}")

    # 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 ذخیره شد: {scaler_save_path}")
    except ImportError:
        print("⚠️ امکان ذخیره Scaler وجود ندارد (joblib نصب نیست)")
```

```
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}")

# ایجاد فایل readme با دستورالعمل‌ها
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"   دستورالعمل استفاده از نتایج تحلیل Self-Attention برای تراز {elevation_level}\n")
    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 آموزش‌دیده\n")
    f.write(f"4. Scaler_{elevation_level}.pkl : استانداردساز داده‌ها\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: داده‌های 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. فایل TIFF را به ArcGIS اضافه کنید\n")
    f.write("2. روی لایه راست‌کلیک → Properties\n")
    f.write("3. تب Symbology → Classified\n")
    f.write("4. Classes: 10\n")
    f.write("5. Method: Natural Breaks (Jenks)\n")
    f.write("6. Color Ramp: از قرمز (زیاد) به سبز (کم)\n")
    f.write("7. Apply → OK\n\n")

    f.write("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 875 تا 1125 با فاصله 12.5 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:
```

```
            # تبدیل نقطه به - اعداد اعشاری underline
            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"جمع: {len(elevations)} تراز")
    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("  شروع تحلیل Self-Attention برای تمام ترازهای ارتفاعی")
    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:
        # ایجاد logger جداگانه برای این تراز
        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} با موفقیت پردازش شد")
            print(f"  زمان پردازش: {level_duration/60:.1f} دقیقه")
        else:
            failed_elevations.append(elevation)
            print(f"\n⚠ تراز {elevation} با مشکل مواجه شد")
            print(f"  زمان صرف شده: {level_duration/60:.1f} دقیقه")

        # توقف logger این تراز
        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"\nلیست 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("گزارش خلاصه تحلیل Self-Attention برای 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("  □ 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"X {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 آموزش‌دیده\n")
f.write("4. Scaler_[تراز].pkl - استانداردساز داده‌ها\n")
f.write("5. Training_History_[تراز].csv - تاریخچه آموزش\n")
f.write("6. Mineral_Probability_[تراز]_raw_data.npz - داده‌های خام\n")
f.write("7. Execution_Report_[تراز].txt - گزارش اجرا\n")
f.write("8. README_[تراز].txt - راهنمای استفاده\n\n")

f.write("  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}")

    # ایجاد فایل bat برای مشاهده سریع نتایج
    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 مشاهده نتایج تحلیل Self-Attention\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 فایل‌های مهم:\n")
        f.write("echo 1. SUMMARY_REPORT.txt - گزارش خلاصه\n")
        f.write("echo 2. MASTER_EXECUTION_REPORT.txt - گزارش کامل\n")
        f.write("echo 3. elevation_list.txt - لیست ترازها\n")
        f.write("echo.\n")
        f.write("pause\n")

    print(f"\n✅ فایل bat برای مشاهده نتایج ایجاد شد: {bat_file}")

except Exception as e:
    print(f"\n❌ خطای اصلی در اجرا: {str(e)}")
    import traceback
    traceback.print_exc()
```

```
    finally:
        # توقف logger اصلی
        master_logger.stop()

    # پیام نهایی
    print(f"\n تمام خروجی‌ها 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 تحلیل با موفقیت برای {len(successful_elevations)} تراز از {len(ELEVATION_LEVELS)} تراز تکمیل شد!")
    else:
        print(f"\n⚠ تحلیل برای هیچ ترازی تکمیل نشد!")
```

انیمیشن این کد چون با بقیه متفاوت میباشد

```
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 خروجی GIF


# ================================================================================
# 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  # ○ اگر می‌خواهید اولین طبقه در Legend سفید باشد True کنید

                #   همه طبقات رنگی هستند :False مقدار

                #   (یا مقادیر پایین NoData برای نمایش) اولین طبقه سفید :True مقدار


NUM_CLASSES = 5         #   تعداد طبقات در Legend (بین ۳ تا ۱۲)

                #   مقادیر کمتر: تفکیک‌پذیری کمتر اما خوانایی بیشتر

                #   Legend مقادیر بیشتر: تفکیک‌پذیری بیشتر اما شلوغی


FILE_PREFIX = "Mineral_Probability_"  #   پیشوند 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 #
# ================================================================================
#
# این بخش فایل‌های ورودی asc را آماده می‌کند. #
# شامل لیست کردن فایل‌ها و فیلتر کردن بر اساس پیشوند می‌شود. #
# ================================================================================


print(f"  پوشه ورودی: {input_folder}")


# لیست کردن تمام فایل‌های asc موجود در پوشه
all_files = [f for f in os.listdir(input_folder) if f.endswith('.asc')]
print(f"  تعداد کل فایل‌های asc در پوشه: {len(all_files)}")


if len(all_files) == 0:
    print("□ هیچ فایل asc در پوشه یافت نشد!")
    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:
    # ⚠️ اگر پیشوند مشخص نشده باشد، همه فایل‌های asc پردازش می‌شوند
    asc_files = all_files
    print("⚠️  هیچ پیشوندی مشخص نشده، همه فایل‌های asc پردازش می‌شوند.")


# ⚠️ بررسی وجود فایل asc برای پردازش
if len(asc_files) == 0:
    print("❌ هیچ فایل asc با پیشوند مشخص شده یافت نشد!")
    print("   لطفاً پیشوند FILE_PREFIX را بررسی کنید.")
    exit()


print(f"✅ تعداد فایل‌های asc پیدا شده: {len(asc_files)}")


# ============================================================================
# بخش ۴: مرتب‌سازی فایل‌های asc
```

```
#
================================================================================
====
#
#    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("    فایل‌های asc مرتب شده بر اساس ارتفاع:")
```

```
for i, f in enumerate(asc_files_sorted[:10]):  #    نمایش 10 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")


#
==========================================================================
====
#    محاسبه بخش ۵: کلی Extent
#
==========================================================================
====
#
#    این بخش محدوده جغرافیایی کلی (Extent) همه رسترها را محاسبه می‌کند.
#    این برای ثابت نگه داشتن محدوده نمایش در تمام فریم‌های انیمیشن ضروری است.
#
==========================================================================
====


print("\n   محاسبه Extent کلی همه رسترها...")
extents = []
for f in asc_files_sorted:
    asc_path = os.path.join(input_folder, f)
    try:
        #    بارگذاری فایل asc به عنوان رستر
        raster = arcpy.Raster(asc_path)
        desc = arcpy.Describe(raster)
        extents.append(desc.extent)
    except Exception as e:
```

```
        print(f"⚠ خطا در خواندن فایل {f}: {e}")

if not extents:
    print("❌ هیچ رستری به درستی خوانده نشد!")
    exit()

#  یافتن حداقل و حداکثر مختصات X و Y
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 کلی محاسبه شد:")
print(f"   X: {xmin:.1f} تا {xmax:.1f}")
print(f"   Y: {ymin:.1f} تا {ymax:.1f}")

# ============================================================================
# بخش ۶: محاسبه محدوده مقادیر رستری
# ============================================================================
#
# این بخش حداقل و حداکثر مقادیر واقعی (بدون NoData) را در تمام رسترها محاسبه می‌کند.
# این مقادیر برای ایجاد Legend یکسان در تمام فریم‌ها استفاده می‌شوند.
# ============================================================================
```

```
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:
        # بارگذاری فایل asc به عنوان رستر
        raster = arcpy.Raster(asc_path)

        # تبدیل رستر به آرایه NumPy
        arr = arcpy.RasterToNumPyArray(raster)

        # ایجاد ماسک برای مقادیر معتبر
        # معمولاً در فایل‌های asc، مقادیر NoData با -9999 مشخص می‌شوند
        valid_mask = (arr != -9999) & (arr > -9998)  # حذف NoData

        if np.any(valid_mask):
            all_valid_values.append(arr[valid_mask])
    except Exception as e:
        print(f"⚠ خطا در پردازش فایل {f}: {e}")

# 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  # برای احتمال معدنی، محدوده 0-100 مناسب است
    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:
# ۱. تنظیم تعداد طبقات Legend
# ۲. ایجاد مرزهای طبقات
# ۳. تعریف پالت‌های رنگی مختلف
# ۴. انتخاب طرح رنگی بر اساس تنظیمات کاربر
# ============================================================================

# تنظیم تعداد طبقات (محدود شده بین ۳ تا ۱۲)
num_classes = max(3, min(12, NUM_CLASSES))
print(f"\n⚙ تنظیمات طبقه‌بندی:")
print(f"   تعداد طبقات Legend: {num_classes}")

# ایجاد مرزهای طبقات (Boundaries)
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؟


        خروجی: لیست رنگ‌ها در فرمت HEX
    """


    #    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']

},

پالت علمی مدرن - *viridis*

'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', '#FFFFF0', '#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"⚠ تعداد طبقات {num_classes} برای طرح {color_scheme} تعریف نشده.")
            print(f"  استفاده از نزدیک‌ترین: {closest} طبقه")
    else:
        # اگر طرح انتخابی موجود نبود، از طرح پیش‌فرض استفاده کن ⚠
        print(f"⚠ طرح رنگی '{color_scheme}' تعریف نشده. استفاده از طرح پیش‌فرض 'blue_red'")
        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"⚠ هشدار: تعداد رنگ‌ها ({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:
            # ⚠ اگر طرح رنگی در matplotlib هم نبود، از viridis استفاده کن
            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:]

# ایجاد Colormap و Norm برای نمایش رستر
cmap_custom = colors.ListedColormap(colors_list)
```

```
norm_classified = colors.BoundaryNorm(class_bounds, cmap_custom.N)


# ================================================================================
#    بخش ۸: ساخت فریم‌های انیمیشن
# ================================================================================
#
#    این بخش برای هر فایل asc ایجاد می‌کند یک فریم (تصویر).
#    هر فریم شامل:
#  ۱. نقشه رستر با رنگ‌بندی طبقه‌ای
#  ۲. Legend در سمت راست
#  ۳. عنوان با ارتفاع
# ================================================================================


#    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 فریم‌ها
images = []


print(f"\n  شروع ساخت فریم‌های انیمیشن...")
print(f"  تعداد کل فریم‌ها: {len(asc_files_sorted)}")


#    پردازش هر فایل asc و ساخت فریم
```

```
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)

        #  بارگذاری فایل asc به عنوان رستر
        raster = arcpy.Raster(asc_path)
        desc = arcpy.Describe(raster)

        #  تبدیل رستر به آرایه NumPy
        arr = arcpy.RasterToNumPyArray(raster)

        #  ایجاد ماسک برای سلول‌های NoData
        #  در فایل‌های asc برای NoData ، معمولاً استفاده می‌شود -9999
        nodata_value = -9999
        nodata_mask = (arr == nodata_value)

        #  تبدیل مقادیر NoData به NaN برای نمایش
        display_arr = np.where(nodata_mask, np.nan, arr)

        #  ایجاد Figure با دو Subplot (نقشه و Legend)
        fig, (ax_map, ax_legend) = plt.subplots(
            1, 2,
            figsize=(16, 8),  #  اندازه کلی Figure
            gridspec_kw={'width_ratios': [3, 1]},  #  نسبت عرض نقشه به Legend
```

```
    dpi=150  # رزولوشن بالا
)

# ==========================================
# بخش الف: نمایش نقشه رستر □ #
# ==========================================

# نمایش رستر با Colormap سفارشی
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')

# ==========================================
# بخش ب: ساخت Legend در سمت راست
# ==========================================

ax_legend.set_title(
    'Legend',
    fontsize=14,
    fontweight='bold',
    pad=10
)

# تنظیمات ارتفاع و فاصله Legend بر اساس تعداد طبقات
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
```

```
    # رسم مستطیل‌های رنگی Legend
    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')

    #    تنظیم Layout برای کاهش حاشیه‌ها
    plt.tight_layout(pad=2.0)

    # ==========================================
    #    بخش ج: تبدیل Figure به تصویر
    # ==========================================

    #    رندر Figure به آرایه RGB
```

```
        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 برای آزادسازی حافظه
        plt.close(fig)

    except Exception as e:
        print(f"  خطا در پردازش {f}: {e}")
        import traceback
        traceback.print_exc()
        plt.close('all')  #    بستن همه Figure ها در صورت خطا

#
================================================================================
====
#    انیمیشن GIF بخش ۹: ساخت فایل
#
================================================================================
====
#
#    این بخش تمام فریم‌های ایجاد شده را به یک فایل GIF تبدیل می‌کند.
#  تنظیمات زمان‌بندی و کیفیت در این بخش کنترل می‌شوند  
#
================================================================================
====
```

```
if len(images) > 1:
    print(f"\n  ساخت GIF انیمیشن...")
    print(f"  تعداد فریم‌ها: {len(images)}")
    print(f"  اندازه هر فریم: {images[0].size}")

    #  ذخیره تصاویر به صورت GIF
    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 (0 = 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"    اولین طبقه: {'سفید' if WHITE_FIRST_CLASS else 'رنگی'}")
    print(f"    پیشوند فایل‌ها: '{FILE_PREFIX if FILE_PREFIX else 'همه'}'")
    print(f"  مدت هر فریم: {600}ms")
```

```
    print(f"حالت تکرار: {'بی‌نهایت'}")

else:
    print("\n⚠ فریم کافی برای ساخت انیمیشن 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:
#    - مطمئن شوید تمام فایل‌های asc در پوشه قرار دارند.
#    - نام فایل‌ها باید شامل عدد (ارتفاع) باشند.
#    - فایل‌ها باید هم‌پوشانی مکانی داشته باشند.
#
# 2. تنظیمات رنگ:
#    - طرح 'blue_red' برای داده‌های رایج توصیه می‌شود.
#    - برای داده‌های دمایی از 'coolwarm' استفاده کنید.
#    - برای داده‌های ارتفاعی از 'terrain' استفاده کنید.
#
# 3. ⚙ بهینه‌سازی:
#    - برای تعداد زیاد فایل‌ها، DPI را کاهش دهید.
#    - تعداد طبقات Legend کمتر باعث خوانایی بهتر می‌شود.
```

```
#    - duration=600 مناسب برای نمایش آموزشی است.
#
# 4. عیب‌یابی:
#    - اگر فایلی نمایش داده نمی‌شود، مقدار NoData را بررسی کنید.
#    - اگر Legend نادرست است، محدوده مقادیر را بررسی کنید.
#    - اگر اندازه فریم‌ها متفاوت است، Extent کلی را بررسی کنید.
#
# 5. تفسیر نتایج:
#    - رنگ‌های آبی معمولاً نشان‌دهنده مقادیر پایین‌تر هستند.
#    - رنگ‌های قرمز نشان‌دهنده مقادیر بالاتر هستند.
#    - مناطق سفید یا شفاف معمولاً NoData هستند.
#
# 6. نکات فنی فایل‌های asc:
#    - فایل‌های asc باید Header استاندارد داشته باشند.
#    - معمولاً NoData با -9999 مشخص می‌شود.
#    - فایل باید قابل خواندن توسط arcpy باشد.
```

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):
  مسیر پوشه حاوی فایل‌های ASC خروجی از تحلیل Self-Attention
  مثال: "M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Self_Attention"
  این پوشه باید حاوی فایل‌های Mineral_Probability_XXX.asc باشد. با نام‌های مانند asc.

gdb_paths (لیست رشته‌ها):
  لیست مسیرهای Geodatabase های حاوی نتایج روش‌های دیگر
  مثال: [
    "M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\SVM.gdb",
    "M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Bayesian.gdb"
  ]
  هر GDB باید حاوی رسترهای احتمالی معدنی برای ترازهای مختلف باشد

elevations (لیست اعداد):
  لیست ترازهای ارتفاعی برای پردازش
  مثال: [875 + i*12.5 for i in range(21)] - 21 متر 12.5 با گام 1125 تا 875 تراز از

output_gdb (رشته):
  مسیر Geodatabase خروجی برای ذخیره نتایج ترکیب دمپستر-شیفر
  مثال: "M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\DempsterShafer_Results.gdb"
  این GDB به طور خودکار ایجاد می‌شود اگر وجود نداشته باشد
"""

self_attention_path = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\Self_Attention"  # پوشه با فایل‌های ASC
```

```
# مسیرهای GDB برای دیگر روش‌ها
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 875 تا 1125 با گام 12.5 متر)
elevations = [875 + i*12.5 for i in range(21)]  # 21 تراز

# مسیر خروجی
output_gdb = r"M:\Mahmood\Survey\WGIS\P24_GOSAL\Gosal\DempsterShafer_Results.gdb"

# ایجاد Geodatabase خروجی اگر وجود ندارد
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 1 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":* ***0.85 :*** تابع جرم برای کلاس معدنی ***(***پیش‌فرض

- *"non_mineral":* ***0.10 :*** تابع جرم برای کلاس غیرمعدنی ***(***پیش‌فرض

- *"uncertainty":* ***0.05 :*** تابع جرم برای عدم قطعیت ***(***پیش‌فرض

مجموع باید ***1.0*** باشد

probability_threshold (عدد):

0.5 : آستانه احتمالی برای طبقه‌بندی نهایی ***(***پیش‌فرض

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):
    """
     تابع extract_elevation_from_filename
    وظیفه: استخراج تراز ارتفاعی از نام فایل
```

پارامترهای ورودی:

- *filename* **(**نام فایل **(**با یا بدون مسیر**)** **:)**رشته**(**

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):
    """
    تابع find_files_for_elevation
    وظیفه: پیدا کردن فایل‌های مربوط به یک تراز در همه منابع ورودی

    پارامترهای ورودی:
    - elevation (عدد): تراز ارتفاعی مورد جستجو

    خروجی:
    - دیکشنری: کلید=نام روش، مقدار=مسیر فایل
    """
    files_dict = {}

    # 1. جستجو در پوشه Self_Attention (فایل‌های ASC)
    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:  # 0.1 اختلاف کمتر از
        files_dict["Self_Attention"] = asc_file
        print(f"  ✓ Self_Attention: {filename}")
        break

# 2. جستجو در GDB های دیگر
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:
        # تغییر workspace به GDB
        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"  ✗ خطا در خواندن {gdb_name}: {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 مسیر رستر ورودی (رشته)
    - target_cellsize رزولوشن هدف (پیش‌فرض: 1.0 متر) (عدد)

    خروجی:
    - اطلاعات رستر یا None در صورت خطا دیکشنری
    """
    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"    ری‌سمپل از {raster.meanCellWidth:.2f} به {target_cellsize} متر")

    # 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):
    """
      تابع normalize_raster_values
    وظیفه: نرمالایز کردن مقادیر رستر به محدوده 0 تا 1

    پارامترهای ورودی:
    - arr (آرایه numpy): آرایه ورودی
    - nodata_value مقدار (عدد): NoData (پیش‌فرض: -9999)
```

```
    خروجی:
    - آرایه numpy: آرایه نرمالایز شده
    """
    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):
    """
        تابع align_rasters
    وظیفه: هم‌راستا کردن رسترها از نظر موقعیت مکانی و ابعاد

    پارامترهای ورودی:
```

```
- rasters_metadata لیست دیکشنری‌های اطلاعات رسترها (لیست):

خروجی:
- تاپل: (aligned_arrays, reference_metadata) یا (None, None) در صورت خطا
"""
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

# پیدا کردن min و max مختصات
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 سایز (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)

    # محاسبه offset برای این رستر
    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):
    """
        تابع dempster_shafer_combination
    وظیفه: ترکیب چند رستر با استفاده از تئوری دمپستر-شیفر

    پارامترهای ورودی:
    - rasters_arrays (لیست): لیست آرایه‌های نرمالایز شده رسترها
```

```
    - class_masses توابع جرم برای کلاس‌ها (دیکشنری):

    خروجی:
    - آرایه numpy: احتمال نهایی کلاس معدنی یا None در صورت خطا
    """
    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 تراز های ارتفاعی

    خروجی:
    - لیست: نتایج پردازش هر تراز
    """

    results_log = []

    # برگرداندن workspace به خروجی
    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"  رد شدن تراز {elevation}: فایل‌های کافی یافت نشد ({len(files_dict)} فایل)")
        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 آماره‌ها
        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"\nآمارههای خروجی:")
        print(f"  میانگین احتمال: {mean_val:.4f}")
        print(f"  حداقل/حداکثر: {min_val:.4f} / {max_val:.4f}")
        print(f"  انحراف معیار: {std_val:.4f}")
        print(f"  پیکسلهای با احتمال > {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"\nتراز های ارتفاعی: {len(elevations)} (متر **1125** تا **875**)تراز")
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
warnings.filterwarnings('ignore')

#
==========================================================================
====
#    تنظیمات دستی - این بخش را ویرایش کنید
#
==========================================================================
====

# 1. تنظیمات ورودی - یکی از این دو را انتخاب کنید
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"  # (اختیاری - برای حذف فیلتر "" بگذارید) فقط رسترهای حاوی این کلمه

# 4. تنظیمات مدل بلوکی - ابعاد دقیق 12.5 متر
```

```
CELL_SIZE_XY = 12.5      # اندازه سلول در جهت X و Y دقیقاً 12.5 - (متر)
CELL_SIZE_Z = 12.5       # اندازه سلول در جهت Z دقیقاً 12.5 - (متر)

# 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  # اگر True باشد، برش اعمال می‌شود
CLIP_X_MIN = 306300.0  # حداقل مختصات X (Easting) (متر) برای برش
CLIP_X_MAX = 307857.0  # حداکثر مختصات X (Easting) (متر) برای برش
CLIP_Y_MIN = 3928820.0  # حداقل مختصات Y (Northing) (متر) برای برش
CLIP_Y_MAX = 3929549.0  # حداکثر مختصات Y (Northing) (متر) برای برش

# 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"  # سیستم مختصات UTM Zone 40N

# 9. تنظیمات کلاس‌بندی - رنج‌های 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 تصویرسازی
IMAGE_DPI = 300  # کیفیت تصویر

# 11. تنظیمات بلوک‌های سه‌بعدی
BLOCK_ALPHA = 0.9  # شفافیت بلوک‌ها
BLOCK_EDGE_COLOR = 'black'  # رنگ لبه بلوک‌ها
BLOCK_EDGE_WIDTH = 0.1  # ضخامت لبه

#
================================================================================
====
#    پالت رنگی سفارشی - blue_red بدون سفیدی
#
================================================================================
====

def create_custom_colormaps():
    """ایجاد و ثبت پالت رنگی سفارشی blue_red بدون سفیدی"""

    # بدون سفیدی در وسط - (قرمز) و پایان (آبی) رنگ‌های شروع
    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
]

# نام colormap با استفاده از نام فایل برای جلوگیری از تداخل
cmap_name = 'blue_red_direct_no_white'

# بررسی وجود colormap
try:
    # اگر colormap قبلاً وجود دارد، آن را بازگردان
    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

# ایجاد colormap جدید
try:
    blue_red_cmap = LinearSegmentedColormap.from_list(
        cmap_name,
        simple_blue_red_colors,
        N=256
```

```
    )

    # ثبت colormap در matplotlib
    plt.cm.register_cmap(name=cmap_name, cmap=blue_red_cmap)
    print(f"   Created new colormap: {cmap_name}")

except ValueError as e:
    # اگر colormap قبلاً ثبت شده، از همان استفاده کن
    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 مدلساز 
#
==========================================================================
====


class GDB3DModeler:
    def __init__(self):
        """
        کلاس برای ساخت مدل بلوکی سه‌بعدی از رسترهای موجود در GDB یا پوشه
        """
        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
        # برای پوشه، workspace را به پوشه خروجی موقت تنظیم می‌کنیم
        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}. باید 'FOLDER' یا 'GDB' باشد.")

    env.overwriteOutput = True

    # تنظیم سیستم مختصات UTM Zone 40N
    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)

        # ایجاد extent برای برش
        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):
        """
        لیست کردن رسترهای موجود در GDB با فیلتر
        """
        print("Searching for rasters in GDB...")

        try:
            # دریافت لیست تمام رسترها
            all_rasters = []
            for dirpath, dirnames, filenames in arcpy.da.Walk(self.input_path,
datatype="RasterDataset"):
                for filename in filenames:
                    all_rasters.append(filename)

            print(f"Total rasters in GDB: {len(all_rasters)}")
            return all_rasters

        except Exception as e:
            print(f"Error reading GDB: {str(e)}")
            return []

    def list_rasters_from_folder(self):
        """
        لیست کردن رسترهای موجود در پوشه با پسوندهای تعریف شده
```

```
    """
    print(f"Searching for raster files in folder: {self.input_path}")

    try:
        all_rasters = []

        # جستجوی بازگشتی در پوشه
        for root, dirs, files in os.walk(self.input_path):
            for file in files:
                # بررسی پسوند فایل
                file_ext = os.path.splitext(file)[1].lower()
                if file_ext in RASTER_EXTENSIONS:
                    # اضافه کردن مسیر کامل
                    full_path = os.path.join(root, file)
                    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 10 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 نزدیکـترین ارتفاع در لیست
                    closest = min(ELEVATION_LEVELS, key=lambda x: abs(x - num))
                    if abs(closest - num) < 1.0:  # اگر اختلاف کمتر از 1 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 پیش‌فرض استفاده می‌کنیم
                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}'")
```

```
    # مرتب‌سازی 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:
        # نام فایل موقت ArcGIS - 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}...")

        # تعیین extent برای برش
        clip_extent_str = f"{self.clip_x_min} {self.clip_y_min} {self.clip_x_max}
{self.clip_y_max}"
```

```
        # استفاده از Clip_management برای برش با نام کوتاه
        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:
                            # به‌روزرسانی مسیر به رستر برش‌خورده
                            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 برش‌خورده)
        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 پیش‌فرض را اعمال می‌کنیم
            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 سه‌بعدی
    """
    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 سه‌بعدی 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))


                # تبدیل رستر به آرایه numpy

                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()

    # درون‌یابی 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 برش‌خورده
        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):
```

```
"""
درون‌یابی مقادیر NaN در مدل بلوکی
"""
z, y, x = block_model.shape
filled_count = 0

for i in range(y):
    for j in range(x):
        column = block_model[:, i, j]

        # شمارش مقادیر NaN
        nan_count = np.sum(np.isnan(column))
        if nan_count == 0:
            continue

        # اگر تمام مقادیر NaN باشند، صفر می‌کنیم
        if nan_count == z:
            block_model[:, i, j] = 0
            filled_count += z
        # اگر برخی مقادیر NaN باشند، درون‌یابی می‌کنیم
        else:
            valid_idx = ~np.isnan(column)
            if np.sum(valid_idx) > 1:  # حداقل دو نقطه برای درون‌یابی
                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
        """
        # برای مقایسه صحیح با NaNها
        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):
```

```
"""
ایجاد یک مکعب voxel کامل با تمام وجه‌های قابل مشاهده
"""
ابعاد دقیق مکعب - دقیقاً 12.5 متر #
dx = CELL_SIZE_XY / 2.0
dy = CELL_SIZE_XY / 2.0
dz = CELL_SIZE_Z / 2.0

تعریف 8 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 6 وجه مکعب #
faces = [
    وجه پایین # (منفی z)
    [vertices[0], vertices[1], vertices[2], vertices[3]],
    وجه بالا # (مثبت z)
    [vertices[4], vertices[5], vertices[6], vertices[7]],
```

```
        # وجه جلو (منفی y)
        [vertices[0], vertices[1], vertices[5], vertices[4]],
        # وجه عقب (مثبت y)
        [vertices[3], vertices[2], vertices[6], vertices[7]],
        # وجه چپ (منفی x)
        [vertices[0], vertices[3], vertices[7], vertices[4]],
        # وجه راست (مثبت x)
        [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 1-0 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 1-0
    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):
    """
    ایجاد یک مدل voxel سه‌بعدی واقعی با مکعب‌های به هم چسبیده - تمام نقاط نمایش داده می‌شوند
    """
    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 می‌شوند
    total_points = len(z_idx)

    # محاسبه محدوده واقعی داده‌ها
    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()


        # اطلاعات clip
        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")


        # ایجاد figure
        fig = plt.figure(figsize=(22, 16), dpi=150)
        ax = fig.add_subplot(111, projection='3d')


        # استفاده از colormap سفارشی - بر اساس محدوده کلی
        cmap = self.custom_cmap
```

```
# همیشه از محدوده 0-1 برای نرمالایز کردن استفاده می‌کنیم
norm = Normalize(vmin=0.0, vmax=1.0)

# استفاده می‌کنیم batch برای نمایش بهتر تعداد زیاد مکعب‌ها، 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 (0 تا 1) #
            # 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 0-1

            # 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,  # 0 حداقل همیشه
                1.0   # 1 حداکثر همیشه
            )

            # 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 0 تا 1
            normalized_value = (values[i] - global_min) / (global_max - global_min)
            normalized_value = max(0.0, min(1.0, normalized_value))  # محدود کردن به 0-1

            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,  # 0 حداقل همیشه
            1.0   # 1 حداکثر همیشه
        )

        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 روی محورها
from matplotlib.ticker import ScalarFormatter

class PlainFormatter(ScalarFormatter):
    def __call__(self, x, pos=None):
        if x == 0:
```

```
            return '0'
        if abs(x) >= 1000:
            return f'{x:,.0f}'
        else:
            return f'{x:.0f}'

for axis in [ax.xaxis, ax.yaxis, ax.zaxis]:
    axis.set_major_formatter(PlainFormatter())
    axis.set_tick_params(labelsize=12)

# 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
z_padding = (z_max - z_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)
ax.set_zlim(z_min - z_padding, z_max + z_padding)

# 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)

        # ایجاد colorbar با برچسب مناسب
        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)'

        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'


        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))

        # تنظیم layout
        plt.tight_layout()

        # ذخیره تصویر
        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")
        if model_data.get('clip_enabled', False):
            print(f"      Model clipped to specified extent")

        # 2 ایجاد نمودار 2D - ارسال global_min و global_max به تابع
        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):
    """
    ایجاد یک نمودار 2D کلی برای مرجع - با رنگ بر اساس محدوده 2
    """
    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 2D نمونه‌برداری برای نمودار
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 2D
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 8))

# نمودار 1: 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
ax1.set_xlabel('Easting (m)', fontsize=12)
ax1.set_ylabel('Northing (m)', fontsize=12)
```

```
        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)'

        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 2: 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 1-0

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)',
                     fontsize=16, fontweight='bold')
        plt.tight_layout()

        # ذخیره نمودار 2D
        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)")

    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):
    """
    ایجاد یک Shapefile برای یک کلاس خاص - با نام فیلدهای کوتاه
    """
    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)

    # نام Shapefile با توجه به وضعیت clip
```

```
        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:
            # حذف shapefile موجود اگر وجود دارد
            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 10 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
```

```
        # اضافه کردن فیلد clip status اگر clip با نام کوتاه - فعال باشد
        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

        # پر کردن Feature Class
        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):
    """
    ایجاد تصاویر 3D برای هر کلاس
    """
    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)")

        # 3 ایجاد تصویرD 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}")

    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 Shapefileها
    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 3D
    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")

        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:
        # 1. ایجاد شیء مدلساز
        print(f"Starting processing for '{output_name}'...")
        modeler = GDB3DModeler()

        # 2. لیست کردن رسترها
        rasters = modeler.list_rasters()

        if not rasters:
            print("No rasters found. Exiting.")
            return

        # 3. مطابقت با سطوح ارتفاعی
```

```
    modeler.match_rasters_to_elevations()

    if not modeler.rasters_info:
        print("No valid rasters found. Exiting.")
        return

    # 4. بررسی صحت رسترها
    if not modeler.validate_rasters():
        print("Not enough valid rasters. Exiting.")
        return

    # 5. ایجاد مدل بلوکی سه‌بعدی
    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.")
```