Imports & Settings

In [1]:
```
from utils_data import *
from utils_gis import *
from utils_images import *
from utils_plotting import *
import os
import glob
import json
import shutil
import re
import numpy as np
import pandas as pd
import geopandas as gpd
import whitebox
import rasterio
from shapely.geometry import box
from rasterio.plot import show
import matplotlib.pyplot as plt
from matplotlib.patches import Patch
from matplotlib.colors import ListedColormap
import seaborn as sns


##### set dataframe display options
# show decimal format (not scientific notation)
pd.options.display.float_format = '{:.2f}'.format

# display all columns
pd.set_option('display.max_columns', None)
```

In [2]:
```
########################
# Initial Folder Setup
########################

# path to data directory
data_dir = r'../data'
if not os.path.isdir(data_dir):
```

```
    os.makedirs(data_dir)


# path to sonora county data directory
sonora_dir = r'../data/sonora'
if not os.path.isdir(sonora_dir):
    os.makedirs(sonora_dir)
```

Surficial Geology

OVERVIEW

- Sonora's surficial geology is shaped by three primary geological processes: in-situ weathering of bedrock, gravity-driven sediment deposition, and fluvial sediment transportation and deposition. These processes result in eight distinct geologic map units, each defined by unique textural, compositional, and topographic characteristics.

    1. **Artificial fill (af1).** Man-made deposits of soil, sand, gravel, concrete, asphalt, or other materials used to alter the landscape, typically associated with construction activities such as road building, land leveling, or embankment creation.

    2. **Alluvium (Qal).** Unconsolidated sediments composed of varying mixtures of silt, sand, clay, and gravel deposited by rivers and streams. These materials are typically found in riverbeds, floodplains, and deltas, representing active or recent fluvial processes.

    3. **Alluvial fans (Qaf).** Cone-shaped deposits of alluvium formed where high-gradient streams emerge from mountainous areas onto flatter plains, causing a decrease in flow velocity and subsequent sediment deposition. They are characterized by a fan-like spread of sediments due to the lateral shifting of stream channels over time. Considered as child unit of alluvium.

    4. **Allvual terraces (Qat).** Step-like landforms along river valleys composed of alluvium, representing former floodplain levels that have been dissected due to river incision or changes in base level. Considered as a child unit of alluvium.

    5. **Paleochannel alluvium (Qapc).** Considered as a child of alluvium or terrace deposits, and represent unconsolidated alluvial sediments residing in conspicuous, inactive, relict river channel landforms, known as paleochannels.

6. **Colluvium (Qc).** Unconsolidated, gravity-driven deposits forming a thin veneer of actively moving soil and rock fragments on steep slopes greater than 12 degrees. This material moves downslope through processes such as creep, sheetwash, and small-scale mass wasting.

7. **Landslide deposits (Qls).** Chaotic accumulations of rock, soil, and debris formed from the rapid downslope movement of materials under the influence of gravity. These deposits are indicative of mass wasting events and can vary greatly in size, composition, and internal structure depending on the nature of the landslide. Considered as a child unit of colluvium.

8. **Residuum (Qr).** Weathered rock material that remains in situ above its parent bedrock, resulting from prolonged chemical and physical weathering processes, and often forming the basis for soil development directly over the source rock.

PURPOSE

- The surficial geologic map is downloaded as an ESRI file geodatabase that includes multiple tables. The MapUnitsPolys feature class in the geodatabase consists of polygons representing the aerial extent of these surficial geologic map units in the map area, which serve as labeled targets.

SPECIAL NOTE

- Map units Qls and Qapc are not in the Warren County area dataset, which is used for training and validation. To mitigate potential errors with these unseen map units, they will be re-assigned to a parent lithology. This is not problematic nor biased because surficial geologic map units are hierarchical in nature. In this particular case, Qls is rare, relatively small, and can be considered a child of colluvium. Qapc is also rare, relatively small, and can be considered a child of alluvium. Qls and Qapc will be re-assigned to it's neighbors value; in the case of multiple touching neighbors, the neighbor sharing the longest border will be used.

REFERENCE

- Massey, M.A., Bottoms, A.E., Hammond, M.L., III, Morris, Emily, and McHugh, Michelle, 2021, Surficial geologic map of the Sonora 7.5-minute quadrangle, central Kentucky. Kentucky Geological Survey, Contract Report Series 13, Issue 43, 1:24000 scale. https://ngmdb.usgs.gov/Prodesc/proddesc_111983.htm

Surficial Geologic Map Polygons

In [3]:
```
############################
# Download Map Geodatabase
############################

# direct download URL for Sonora surficial geologic map geodatabase
url = r'https://ngmdb.usgs.gov/ngm-bin/gems_download.pl?id=1630&pid=111983'

# download directory path
output_dir = r'../data/sonora'


##### download zip & extract contents
download_zip(url, output_dir)
```

In [4]:
```
############################
# Save Map Layer as GeoJSON
############################

# path to geodatabase
gdb_path = glob.glob(r'../data/sonora/**/*.gdb', recursive=True)[0]

# name of geodatabase layer containing polygons
map_layer = 'MapUnitPolys'

# path for output GeoJSON
output_path = r'../data/sonora/geology.geojson'


##### verify polygons & save as GeoJSON
if not os.path.isfile(output_path):

    # read feature class layer from geodatabase
    gdf = gpd.read_file(gdb_path, layer=map_layer)

    # fix any broken geometries
    gdf['geometry'] = gdf['geometry'].buffer(0)

    # verify all geometries are valid & write file to GeoJSON
```

```
    if gdf.is_valid.all():
        gdf.to_file(output_path, driver='GeoJSON')
    else:
        print('Something is wrong with polygon geometries...')
else:
    print(f"File - {output_path} - already exists...")
```

In [ ]:

```
##############################
# Re-Assign Selected Map Units
##############################

# path to geology GeoJSON
geo_path = r'../data/sonora/geology.geojson'

# dictionary of map unit symbols and re-assignments
map_units = {'Qapc': 'Qat', 'Qls': 'Qc'}


##### Re-assign map units & save as new GeoJSON
gdf = gpd.read_file(geo_path)

for key, val in map_units.items():
    gdf.loc[gdf['Symbol']==key, 'Symbol'] = val

print(gdf['Symbol'].unique())

gdf.to_file(geo_path, driver='GeoJSON')
```

In [7]:

```
################################################
# Convert GeoJSON to GeoTIFF with Metadata JSON
################################################

# path to GeoJSON to convert to image
input_path = r'../data/sonora/geology.geojson'

# resolution of output image (in spatial units of coordinate reference system)
# NOTE: 5 chosen as resolution to match original DEM dataset
output_resolution = 5

# attribute to use as category in raster (if none, then output image is binary)
attribute = 'Symbol'
```

```
# path to output image (and JSON metadata)
output_path = r'../data/sonora/geology.tif'


##### custom function to convert GeoJSON to GeoTIFF with JSON metadata
gis_to_image(input_path, output_path, output_resolution, attribute)
```

Dataset Boundaries

In [8]:
```
############################
# Save Boundary as GeoJSON
############################

# path to geology GeoJSON file
geology_path = r'../data/sonora/geology.geojson'

# path for output GeoJSON
output_path = r'../data/sonora/boundary.geojson'


##### verify polygon & save as GeoJSON
if not os.path.isfile(output_path):

    # read GeoJSON into geodataframe
    gdf = gpd.read_file(geology_path)

    # perform small buffer to fill any possible slivers/gaps
    gdf['geometry'] = gdf.geometry.buffer(0.1)

    # merge all polygons to single polygon for map area of interest
    aoi = gdf.unary_union

    # create new geodataframe of area of interest
    gdf_aoi = gpd.GeoDataFrame(geometry=[aoi], crs=gdf.crs)

    # write boundary to GeoJSON if it has all valid geometries
    if gdf_aoi.is_valid.all():
        gdf_aoi.to_file(output_path, driver='GeoJSON')
```

```
    else:
        print('Something is wrong with polygon...')

else:
    print(f"File - {output_path} - already exists...")
```

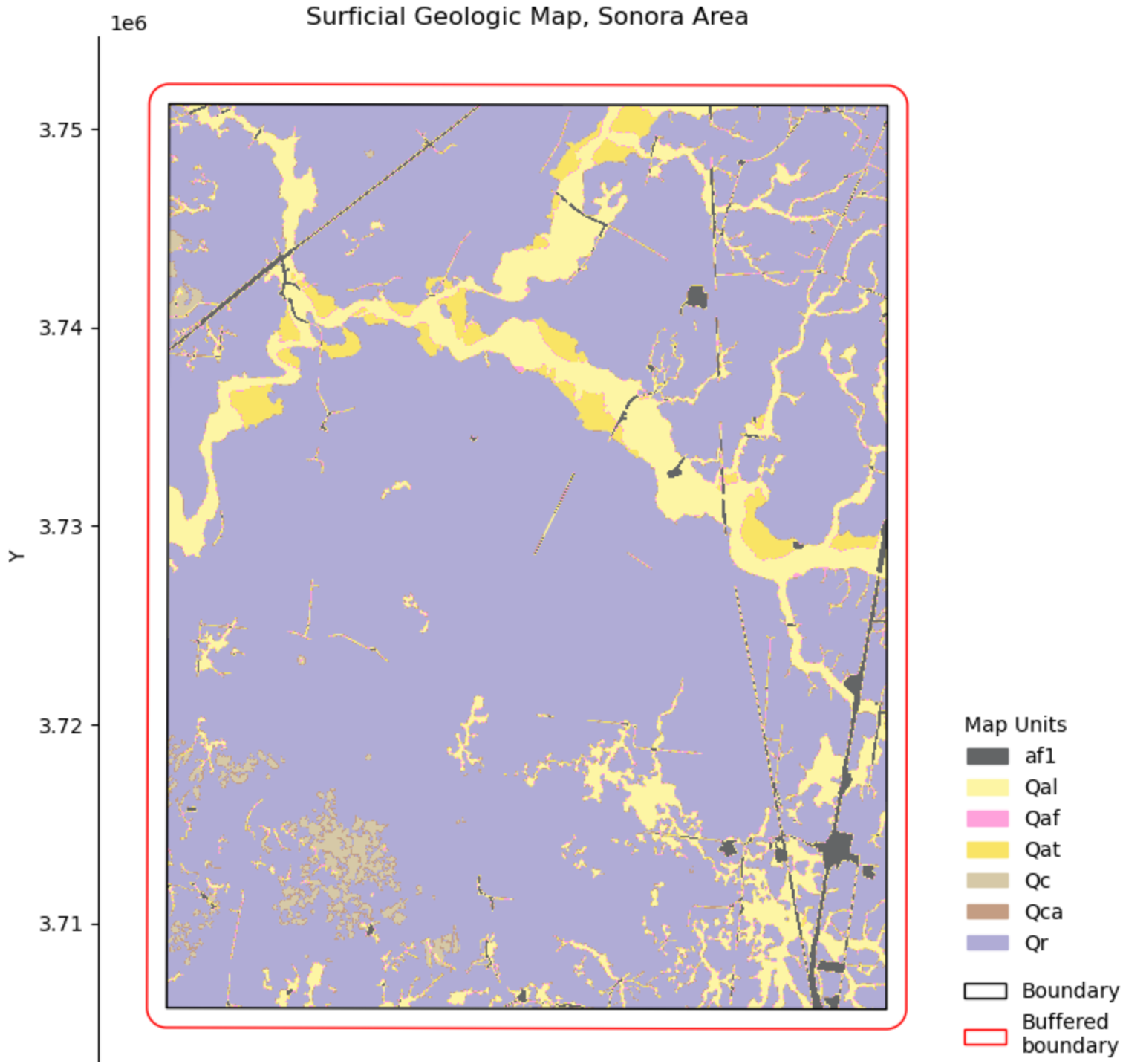
In [9]:
```
##############################################
# Create Buffered Boundary & Save as GeoJSON
##############################################

# path to boundary GeoJSON
input_path = r'../data/sonora/boundary.geojson'

# buffer distance in native CRS spatial units (feet for this project)
# NOTE: buffer distance should be enough to mitigate edge effects for terrain feature calculations (i.e., c
buffer_distance = 1000

# path for output GeoJSON
output_path = f"../data/sonora/boundary_buffer{buffer_distance}.geojson"


##### create buffered boundary & save
if not os.path.isfile(output_path):

    # read dataset boundary as geodataframe
    gdf = gpd.read_file(input_path)

    # buffer (expand) boundary by specified amount
    gdf['geometry'] = gdf['geometry'].buffer(buffer_distance)

    # write new buffered dataset boundary to GeoJSON
    gdf.to_file(output_path, driver='GeoJSON')

else:
    print(f"File - {output_path} - already exists...")
```

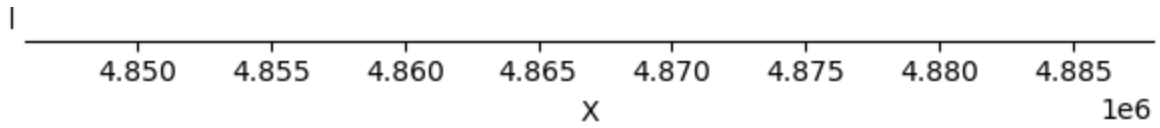
Visualize Target & Boundaries

In [10]:
```
##############################################
# Visualize Target GeoTIFF & Boundary GeoJSONs
```

```
################################################

# path to GeoTIFF
geo_path = r'../data/sonora/geology.tif'

# path to JSON metadata
geo_meta_path = r'../data/sonora/geology.json'

# path to dataset boundary
boundary_path = r'../data/sonora/boundary.geojson'

# path to buffered dataset boundary
buffered_path = glob.glob(r'../data/sonora/*buffer*.geojson')[0]


##### create custom color map from GeoTIFF metadata JSON
# NOTE: these are standard colors from Kentucky Geological Survey
with open(geo_meta_path, 'r') as meta:
    geo_meta = json.load(meta)

colors = {'af1': '#636566',
          'Qal': '#fdf5a4',
          'Qaf': '#ffa1db',
          'Qat': '#f9e465',
          'Qc': '#d6c9a7',
          'Qca': '#c49d83',
          'Qr': '#b0acd6'}

cmap = ListedColormap([colors[symbol] for symbol in geo_meta.keys()])


##### visualize GeoTIFF with custom colormap & dataset boundaries
fig, ax = plt.subplots(figsize=(8,8))

# plot GeoTIFF with custom color map
with rasterio.open(geo_path) as src:
    show(src, ax=ax, cmap=cmap)

# custom colors and legend
handles = [Patch(color=colors[symbol], label=symbol) for symbol in geo_meta.keys()]
legend1 = plt.legend(handles=handles, bbox_to_anchor=(1, 0.1), loc='lower left', frameon=False, title='Map
```

```python
# plot dataset boundary
boundary = gpd.read_file(boundary_path)
boundary.plot(ax=ax, facecolor='none', edgecolor='k', linewidth=1)
boundary_patch = Patch(facecolor='none', edgecolor='k', linewidth=1, label='Boundary')
handles2 = [boundary_patch]

# # plot buffered boundary
buffered = gpd.read_file(buffered_path)
buffered.plot(ax=ax, facecolor='none', edgecolor='red', linewidth=1)
buffered_patch = Patch(facecolor='none', edgecolor='red', linewidth=1, label='Buffered\nboundary')
handles2.append(buffered_patch)
legend2 = plt.legend(handles=handles2, bbox_to_anchor=(1, 0), loc='lower left', frameon=False)

# plot legend for boundaries
ax.add_artist(legend1)
# ax.add_artist(legend2)

# adjust figure styling
ax.set_title('Surficial Geologic Map, Sonora Area')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_position(('outward', 5))  # offset bottom axis
ax.spines['left'].set_position(('outward', 5))    # offset left axis

plt.tight_layout()
plt.show()
```


Surficial Geologic Map, Sonora Area
1e6
3.75
3.74
3.73
3.72
3.71
Y
Map Units
af1
Qal
Qaf
Qat
Qc
Qca
Qr
Boundary
Buffered boundary



In [11]:
```
############################
# Remove Target Geodatabase
############################

# path to directory
dir_path = r'../data/sonora'

# path to subdirectory
subdir_path = [x[0] for x in os.walk(dir_path)][1]


##### delete geodatabase
if os.path.isdir(subdir_path):
    shutil.rmtree(subdir_path)
```

KyFromAbove Tile Index

OVERVIEW

- The KyFromAbove 5K tile index is available as a geodatabase containing three polygon feature classes for accessing digital elevation models (DEMs), aerial imagery, and LiDAR point cloud datasets. These feature classes represent grids of individual tiles, each corresponding to a specific area. The tiles can be downloaded as GeoTIFF images, with each tile featuring various attributes, including download URLs. More information can be found at the KyFromAbove website - https://kyfromabove.ky.gov/

PURPOSE

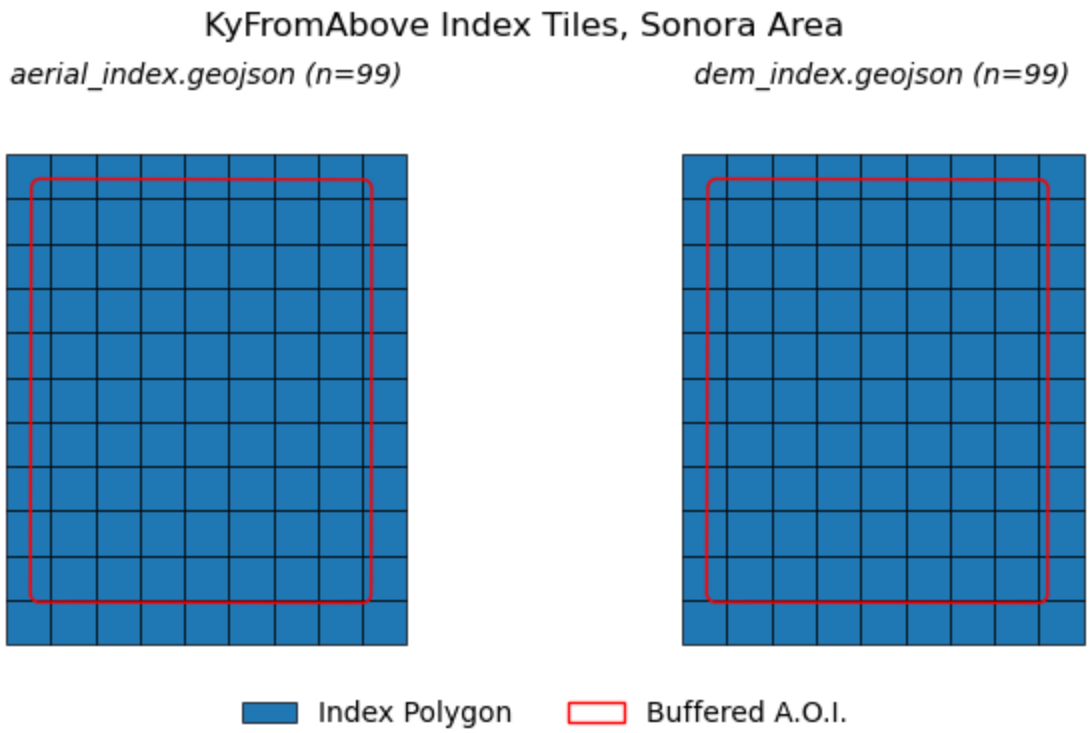
- The tile index is essential for identifying and accessing the specific DEM and aerial imagery GeoTIFF images that correspond to the area of interest covered by the target map datasets. Although the tile index itself is not directly used as a feature, it facilitates the retrieval of the necessary data for further analysis.

In [12]:
```
################################################
# Download & Save Index Tile Layers as GeoJSONs
################################################

# directory path to save files
output_dir = r'../data/sonora/index_tiles'
if not os.path.isdir(output_dir):
    os.makedirs(output_dir)

# URL for KyFromAbove Tile Index geodatabase download
url = r'https://ky.app.box.com/index.php?rm=box_download_shared_file&vanity_name=kymartian-kyaped-5k-tile-g


##### download & extract index tile geodatabase
# NOTE: this is for entire state of Kentucky
download_zip(url, output_dir)
```

In [13]:
```
################################################
# Extract Tiles that Intersect Sonora
################################################

# list of paths to statewide tile index GeoJSONs (for DEM & Aerial Imagery)
gdb_path = glob.glob(r'../data/sonora/index_tiles/*.gdb')[0]

# path to buffered dataset boundary GeoJSON
buffered_path = glob.glob(r'../data/sonora/*buffer*.geojson')[0]

# directory path for output
output_dir = r'../data/sonora'


##### extract dem & aerial index tiles for area of interest
get_aoi_index_polygons(gdb_path, buffered_path, output_dir)


##### delete index subdirectory with geodatabase
shutil.rmtree(r'../data/sonora/index_tiles')
```

In [14]:
```
#####################################################
# Visualize Index Tiles & Buffered Dataset Boundary
#####################################################
```

```
# list of paths to index tiles
index_paths = glob.glob(r'../data/sonora/*index*.geojson')

# path to buffered dataset boundary
buffered_path = glob.glob(r'../data/sonora/*buffer*.geojson')[0]


##### read GeoJSONs as geodataframes & visualize coverage
fig, ax = plt.subplots(ncols=2, figsize=(8,3.5))


buffered = gpd.read_file(buffered_path)

# plot tile index and buffered boundary for reference
for idx, path in enumerate(index_paths):
    index = gpd.read_file(path)
    name = os.path.basename(path)
    name = name + f" (n={len(index)})"
    index.plot(ax=ax[idx], edgecolor='k', linewidth=0.5)
    buffered.plot(ax=ax[idx], facecolor='none', edgecolor='red', linewidth=1)
    ax[idx].set_title(name, style='italic', fontsize=10)
    ax[idx].set_axis_off()

# create custom patches for legend
buffered_patch = Patch(facecolor='none', edgecolor='red', linewidth=1, label='Buffered A.O.I.')
index_patch = Patch(edgecolor='k', linewidth=0.5, label='Index Polygon')
handles = [index_patch, buffered_patch]
plt.figlegend(handles=handles, loc='lower center', frameon=False, ncols=2)

plt.suptitle('KyFromAbove Index Tiles, Sonora Area', y=1.05)
plt.show()
```



Digital Elevation Model (DEM)

OVERVIEW

- The Kentucky statewide digital elevation model (DEM) is generated from airborne LiDAR data collected since 2010. The data collection was divided into phases: Phase 1 produced a DEM with a 5-foot resolution, while Phase 2 resulted in a more detailed 2-foot resolution DEM. These datasets are publicly accessible in small tiles available for download in either .img (with associated metadata files) or .tif (GeoTIFF) formats. This project will utilize the Phase 1 5 foot DEM.

PURPOSE

- The DEM serves as a fundamental base layer for interpreting and delineating surficial geologic map units in this project. The DEM also enables the derivation of multiple terrain features and indices, which provide additional insights and are calculated in a subsequent section.

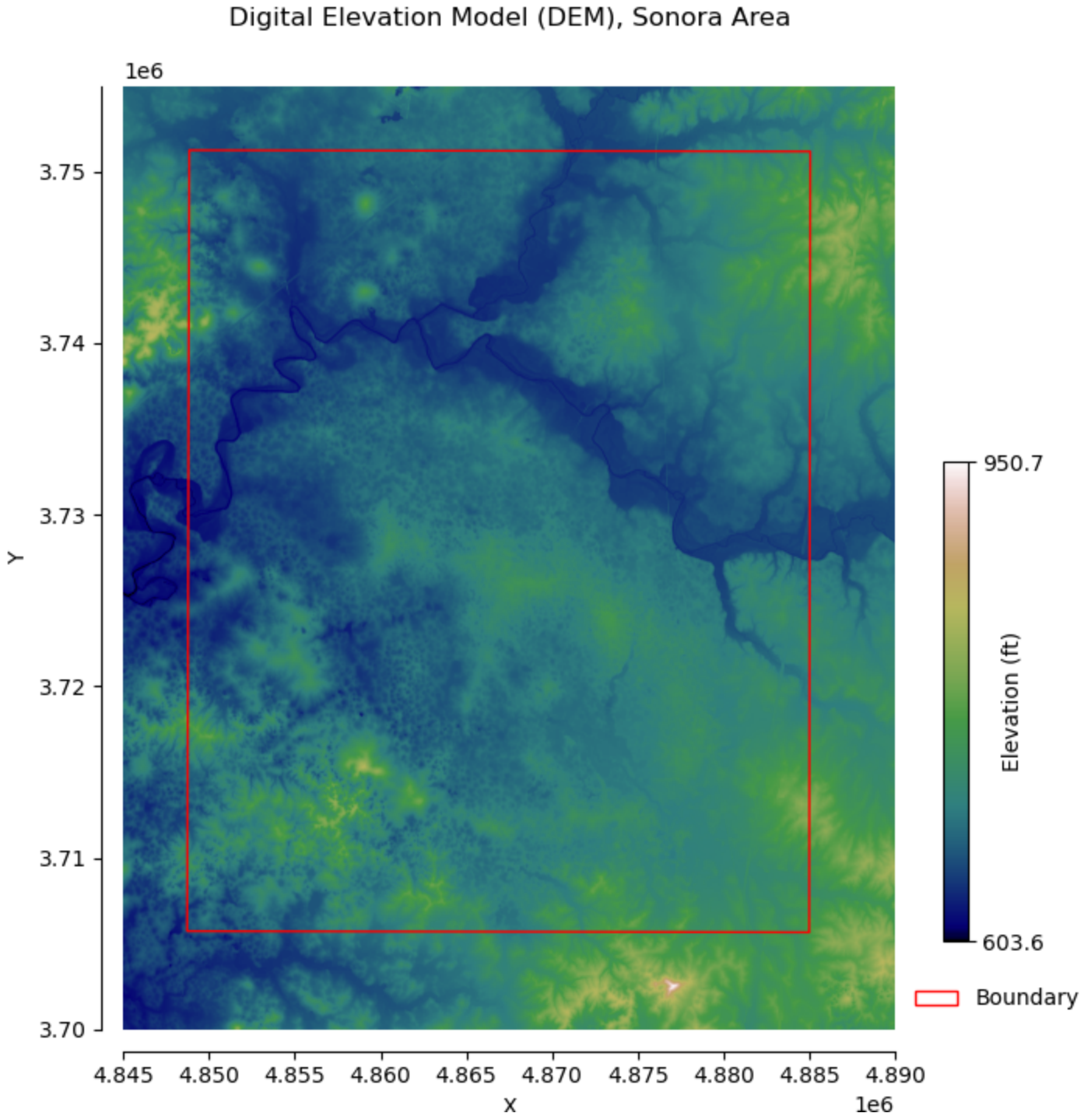
In [15]:
```
##############################################
# Download DEM GeoTIFF tiles for Sonora
##############################################

# path to DEM tile index GeoJSON for Sonora
index_path = r'../data/sonora/dem_index.geojson'

# field containing unique tile id for file naming
id_field = 'TileName'

# field containing GeoTIFF download url
url_field = 'Phase1_AWS_url'

# output directory to save downloaded tiles
output_dir = r'../data/sonora/dem_tiles'
if not os.path.isdir(output_dir):
    os.makedirs(output_dir)


###### call custom function to download tiles
download_data_tiles(index_path, id_field, url_field, output_dir)
```

In [16]:
```
##########################################
# Mosaic DEM Tiles into Single DEM GeoTIFF
##########################################

# list of DEM tile GeoTIFFs
dem_tile_paths = glob.glob(r'../data/sonora/dem_tiles/*.tif')

# path for output DEM mosaic
output_path = r'../data/sonora/dem.tif'


##### mosaic DEM tiles to single DEM
# NOTE: this produces a single DEM of the tiles that completely cover the target area + the expanded (buffe
mosaic_image_tiles(glob.glob(r'../data/sonora/dem_tiles/*.tif'), output_path, band_number=1)
```

In [19]:
```
#######################
# Visualize DEM Mosaic
#######################
```

```python
# path to buffered dem mosaic
dem_path = r'../data/sonora/dem.tif'

# path to target boundary (not buffered)
boundary_path = r'../data/sonora/boundary.geojson'


##### plot dem
fig, ax = plt.subplots(figsize=(8,8))

# dem
with rasterio.open(dem_path) as dem:

    # get dem data
    data = dem.read(1, masked=True)
    min_val = np.min(data)
    max_val = np.max(data)
    bounds = dem.bounds

    # plot dem with imshow to get colorbar for figure (not shown)
    dem_hidden = ax.imshow(data, cmap='gist_earth', interpolation='bicubic')

    # plot dem with rasterio.plot.show
    show(dem, ax=ax, cmap='gist_earth')

# plot DEM colorbar
cbar = fig.colorbar(dem_hidden, ticks=[min_val, max_val], shrink=0.5, anchor=(0, 0.2))
cbar.set_label('Elevation (ft)', labelpad=-20)

# plot dataset boundary & patch for legend
boundary = gpd.read_file(boundary_path)
boundary.plot(ax=ax, facecolor='none', edgecolor='red')
boundary_patch = Patch(facecolor='none', edgecolor='red', label='Boundary')
ax.legend(handles=[boundary_patch], loc='lower left', bbox_to_anchor=(1, 0), frameon=False)

# customize plot elements
ax.set_xlim(bounds.left, bounds.right)
ax.set_ylim(bounds.bottom, bounds.top)
ax.set_title('Digital Elevation Model (DEM), Sonora Area', y=1.05)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.spines['top'].set_visible(False)
```

```
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_position(('outward', 10))
ax.spines['left'].set_position(('outward', 10))

plt.show()
```


Digital Elevation Model (DEM), Sonora Area
1e6
3.75
3.74
3.73
3.72
3.71
3.70
Y
4.845
4.850
4.855
4.860
4.865
4.870
4.875
4.880
4.885
4.890
X
1e6
950.7
Elevation (ft)
603.6
Boundary

In [20]:
```
###################
# Delete DEM Tiles
###################

# directory path to dem tiles
dem_dir = r'../data/sonora/dem_tiles'


##### delete dem_tiles folder and all contents
if os.path.isdir(dem_dir):
    shutil.rmtree(dem_dir)
```

Multiscale Terrain Analysis

OVERVIEW

- Terrain features are derived from the DEM to characterize the topography and capture features that are critical for identifying and delineating surficial geologic map units. These features provide supplementary data about topography, weathering processes, water drainage patterns, and vegetation that enhance the interpretation of geologic processes and landforms, contributing to more accurate predictions.

- Terrain features may be calculated using a variety of GIS tools. The features below use the Python implementation of Whitebox Tools. Details from the user manual are presented below each tool, and the reader is referred to the home page for more information - https://www.whiteboxgeo.com/manual/wbt_book/preface.html

PURPOSE

- Five terrain features are calculated from the DEM, each providing unique insights into the landscape's characteristics:

    1. **Slope.** Measures the steepness or incline of the terrain. It is critical for identifying areas prone to erosion, landslides, or specific deposition patterns associated with certain geologic map units.

    2. **Profile Curvature.** Indicates the curvature of the terrain in the direction of the slope. It affects the acceleration or deceleration of water flow, which can influence erosion and sediment deposition patterns.

3. **Planform Curvature.** Measures the curvature perpendicular to the slope direction. It helps in identifying convergent and divergent flow patterns, which are important for understanding water drainage and sediment deposition.

4. **Standard Deviation of Slope.** Provides a measure of the variability in slope within a given area. It is useful for identifying areas with complex topography, which may correlate with different geologic processes or materials.

5. **Elevation Percentile.** Elevation percentile (EP) is a measure of local topographic position. It is useful for identifying areas of high or low relative topographic position such as ridges, valleys, or sinkholes.

- Each terrain feature described above will be calculated at multiple scales to capture local, intermediate, and global scales. This will be accomplished by first resampling the DEM to multiple coarser resolutions, and then calculating each terrain features at each of these resolutions. It is well established that landforms and geologic processes operate at multiple scales. For example, a high resolution slope image will identify areas of abrupt local change in elevation like the edges of a small stream or building foundation, whereas slope calculated at a coarser resolution could highlight areas within a larger flat plain or hill side.

In [21]:
```
##########################
# Initialize Whitebox
##########################
# USER MANUAL: https://whiteboxgeo.com/manual/wbt_book/preface.html

# import whitebox tools class as wbt object
wbt = whitebox.WhiteboxTools()

# toggle on/off geoprocessing tool outputs (optional)
wbt.verbose = False

# set working directory for input/output files to current working directory
wbt.set_working_dir(os.getcwd())

# print whitebox version to verify correctly loading
wbt.version()
```

Out[21]:
```
"WhiteboxTools v2.4.0 (c) Dr. John Lindsay 2017–2023\n\nWhiteboxTools is an advanced geospatial data analy
sis platform developed at\nthe University of Guelph's Geomorphometry and Hydrogeomatics Research \nGroup
(GHRG). See www.whiteboxgeo.com for more details.\n"
```

Multidirectional Hillshade

The purpose of the multi-directional hillshade image created below his is strictly to enhance visualizations.

This tool performs a hillshade operation (also called shaded relief) on an input digital elevation model (DEM) with multiple sources of illumination. The user must specify the name of the input DEM (--dem) and the output hillshade image name (--output). Other parameters that must be specified include the altitude of the illumination sources (--altitude; i.e. the elevation of the sun above the horizon, measured as an angle from 0 to 90 degrees) and the Z conversion factor (--zfactor). The Z conversion factor is only important when the vertical and horizontal units are not the same in the DEM, and the DEM is in a projected coordinate system.

The hillshade value (HS) of a DEM grid cell is calculate as, HS = tan(s) / [1 - tan(s)2]0.5 x [sin(Alt) / tan(s) - cos(Alt) x sin(Az - a)], where s and a are the local slope gradient and aspect (orientation) respectively and Alt and Az are the illumination source altitude and azimuth respectively. Slope and aspect are calculated using Horn's (1981) 3rd-order finate difference method.

Lastly, the user must specify whether or not to use full 360-degrees of illumination sources (--full_mode). When this flag is not specified, the tool will perform a weighted summation of the hillshade images from four illumination azimuth positions at 225, 270, 315, and 360 (0) degrees, given weights of 0.1, 0.4, 0.4, and 0.1 respectively. When run in the full 360-degree mode, eight illumination source azimuths are used to calculate the output at 0, 45, 90, 135, 180, 225, 270, and 315 degrees, with weights of 0.15, 0.125, 0.1, 0.05, 0.1, 0.125, 0.15, and 0.2 respectively.

- https://www.whiteboxgeo.com/manual/wbt_book/available_tools/geomorphometric_analysis.html#MultidirectionalHillshade

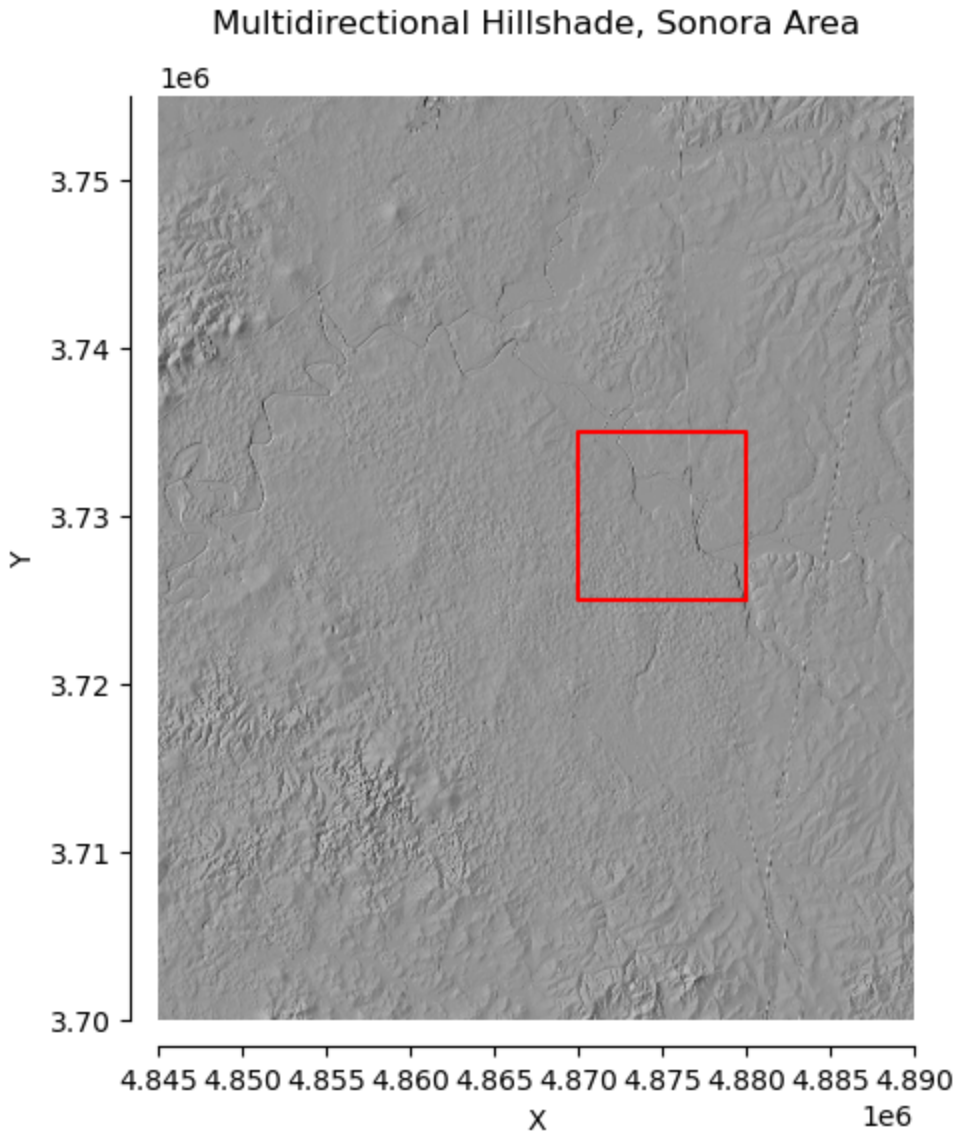
In [62]:
```
########################################
# Calculate Multi-directional Hillshade
########################################

# path to DEM
dem_path = r'../data/sonora/dem.tif'
dem_path = os.path.abspath(dem_path)

# path for output GeoTIFF
output_path = r'../data/sonora/mdhs.tif'
output_path = os.path.abspath(output_path)
```

```
##### calculate mdhs
wbt.multidirectional_hillshade(dem_path, output_path, altitude=30, zfactor=None, full_mode=False)
```

Out[62]: 0

In [24]:

```
#################
# Visualize MDHS
#################

# path to mdhs image
mdhs_path = r'../data/sonora/mdhs.tif'

# bounding box coordinates for plot (left, bottom, right, top)
# NOTE: bounding coordinates can be any area, but these represent an area with many types of landforms
bounds = (4870000, 3725000, 4880000, 3735000)


##### plot mdhs image
fig, ax = plt.subplots(figsize=(8,6))

# plot multidirectional hillshade
with rasterio.open(mdhs_path) as src:
    crs = src.crs
    show(src, ax=ax, cmap='binary_r')

# plot inset figure box & text description
x, y = box(*bounds).exterior.xy
inset = ax.plot(x, y, 'r-', label='Area of terrain feature plots (below)')
ax.text(x=-0.1, y=-0.2, s='*Red box shows area of example terrain feature plots below.', style='italic', t

# customize plot
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_position(('outward', 10))
ax.spines['left'].set_position(('outward', 10))
ax.set_title('Multidirectional Hillshade, Sonora Area', y=1.05)

plt.show()
```



Red box shows area of example terrain feature plots below.

Digital Elevation Model

Multiple scales of terrain features can be critical to accurately detect and characterize surficial geologic map units. The

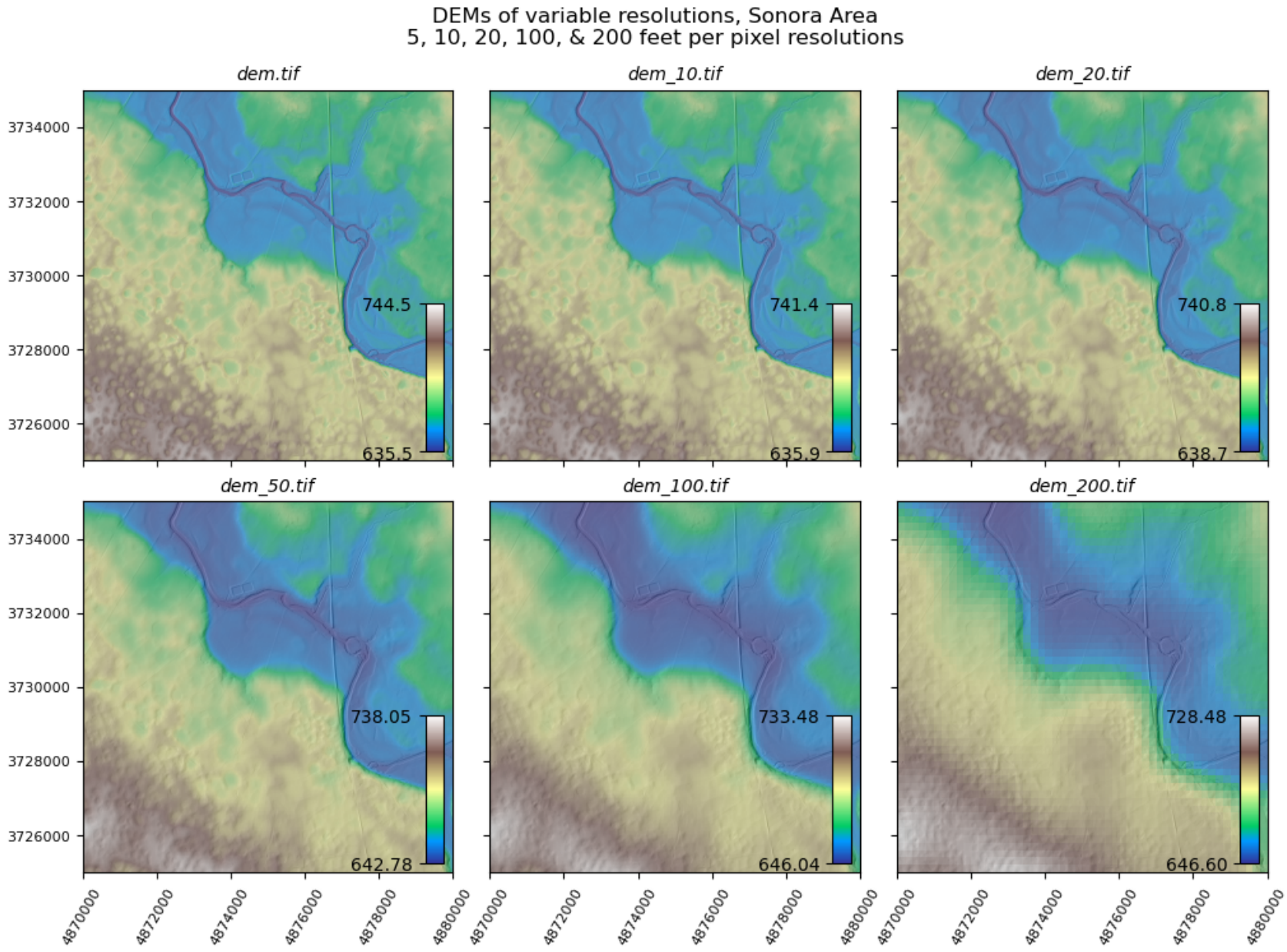
original DEM will then be downsampled to several lower resolutions to capture local, intermediate, and global scale topographic features that may be relevant to the ultimate goal of identifying surficial geologic map units.

- Original DEM has a spatial resolution of 5x5 feet per pixel.
- DEM will be downsampled to resolutions of 10x10, 20x20, 50x50, 100x100, and 200x200 feet per pixel.
- Terrain features will be calculated for each DEM resolution.

In [25]:
```
################################
# Create Multiple Scales of DEM
################################

# path to original dem
dem_path = r'../data/sonora/dem.tif'

# list of resolution(s) to downsample DEM (in native spatial units, which is feet here)
# NOTE: with default 5x5 window for terrain feature calculations, these resolutions will effectively resul
downsample_resolutions = [10, 20, 50, 100, 200]

# # gaussian filter standard deviation
sigma = 2

##### downsample DEM and save new GeoTIFF images
# iterate through resolutions, downsample DEM, and save as GeoTIFF
for res in downsample_resolutions:

    # create output path
    output_path = f"../data/sonora/dem_{res}.tif"

    if not os.path.isfile(output_path):

        # resample dem with given resolution
        resample_image(dem_path, res, output_path)

        # apply gaussian filter to downsampled DEM
        filter_image(input_path=output_path, sigma=sigma)
```

In [27]:
```
############################
# Visualize Downsampled DEMs
```

```
############################

# list of paths to DEMs
dem_paths = glob.glob(r'../data/sonora/dem*.tif')
dem_paths.sort(key=lambda x: list(map(int, re.findall(r'\d+', x))))   # sort DEMs in order of resolution

# path to multi-directional hillshade (used as base layer to aid visualization)
mdhs_path = r'../data/sonora/mdhs.tif'

# bounding box coordinates for plot (left, bottom, right, top)
bounds = (4870000, 3725000, 4880000, 3735000)

# colormap for terrain feature
cmap = 'terrain'

# title for plot
title = 'DEMs of variable resolutions, Sonora Area\n5, 10, 20, 100, & 200 feet per pixel resolutions'


##### plot DEMs
plot_multi_terrain_features(mdhs_path, dem_paths, bounds, cmap, title)
```


DEMs of variable resolutions, Sonora Area
5, 10, 20, 100, & 200 feet per pixel resolutions
dem.tif
dem_10.tif
dem_20.tif
dem_50.tif
dem_100.tif
dem_200.tif
744.5
635.5
741.4
635.9
740.8
638.7
738.05
642.78
733.48
646.04
728.48
646.60
3734000
3732000
3730000
3728000
3726000
4870000
4872000
4874000
4876000
4878000
4880000

Slope

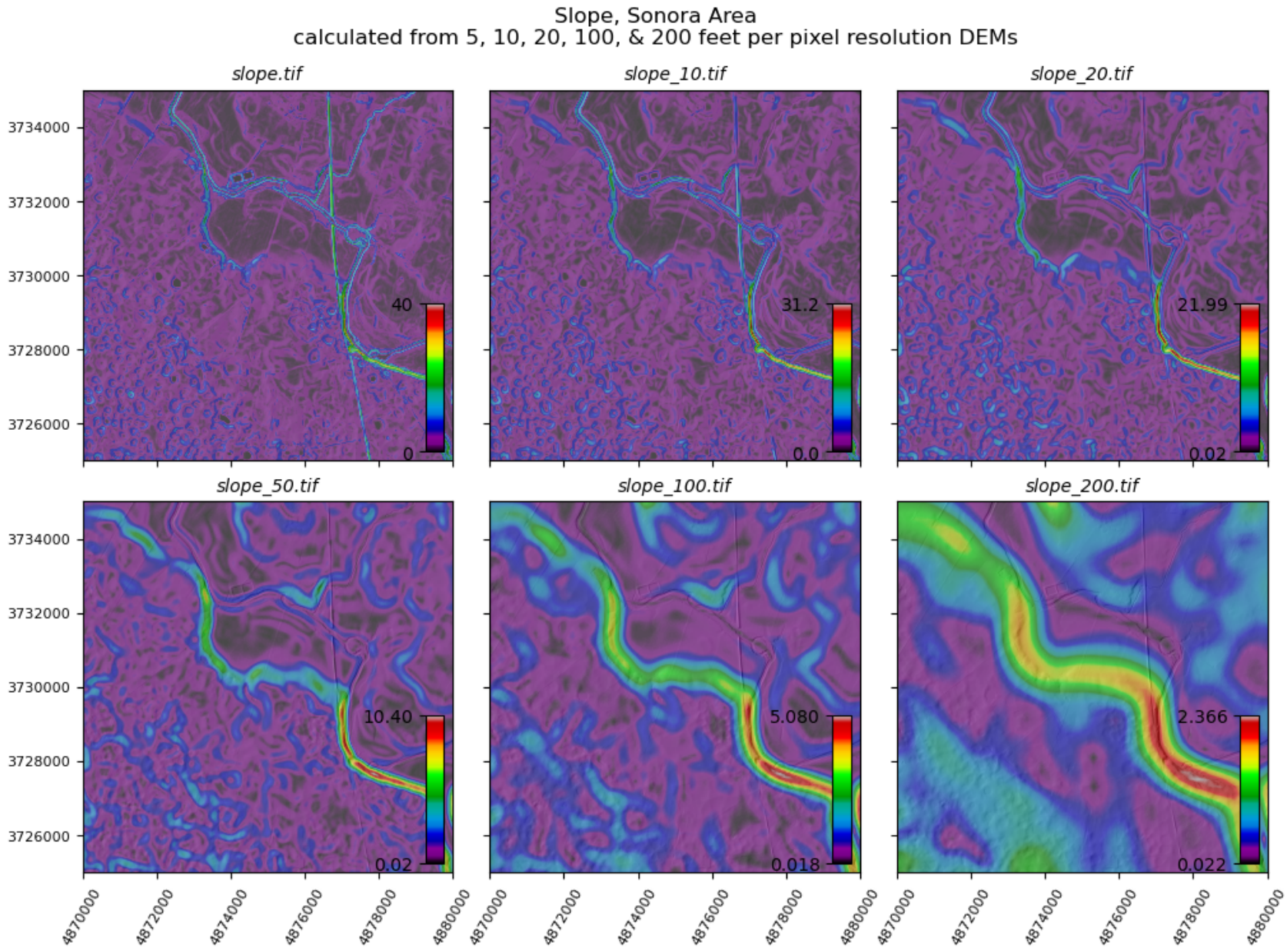
This tool calculates slope gradient (i.e. slope steepness in degrees, radians, or percent) for each grid cell in an input digital elevation model (DEM). For DEMs in projected coordinate systems, the tool uses the 3rd-order bivariate Taylor polynomial method described by Florinsky (2016). Based on a polynomial fit of the elevations within the 5x5 neighbourhood surrounding each cell, this method is considered more robust against outlier elevations (noise) than other methods.

- https://www.whiteboxgeo.com/manual/wbt_book/available_tools/geomorphometric_analysis.html#Slope

In [28]:
```
###################
# Calculate Slopes
###################

# paths to DEMs
dem_paths = glob.glob(r'../data/sonora/dem*.tif')

# path to target image (for reference)
reference_path = r'../data/sonora/geology.tif'

# gaussian filter standard deviation
sigma = 2


##### calculate slope
for dem in dem_paths:

    # create output path
    output_path = dem.replace('dem', 'slope')

    # create absolute paths - Whitebox does not like relative paths
    dem = os.path.abspath(dem)
    output_path = os.path.abspath(output_path)

    # calculate slope (default 5x5 pixel window)
    wbt.slope(dem=dem, output=output_path)

    # upsample & align to target image
    image_to_reference_image(input_path=output_path, reference_path=reference_path)

    # apply gaussian filter to upsampled image
    filter_image(input_path=output_path, sigma=sigma)
```

In [29]:

```
####################################
# Visualize Downsampled Slope Images
####################################

# paths to slope images
slope_paths = glob.glob(r'../data/sonora/slope*.tif')
slope_paths.sort(key=lambda x: list(map(int, re.findall(r'\d+', x))))

# path to multi-directional hillshade
mdhs_path = r'../data/sonora/mdhs.tif'

# bounding box coordinates for plot (left, bottom, right, top)
bounds = (4870000, 3725000, 4880000, 3735000)

# colormap for terrain feature
cmap = 'nipy_spectral'

# title for plot
title = 'Slope, Sonora Area\ncalculated from 5, 10, 20, 100, & 200 feet per pixel resolution DEMs'


##### plot slope images
plot_multi_terrain_features(mdhs_path, slope_paths, bounds, cmap, title)
```



Profile Curvature

This tool calculates the profile (or vertical) curvature, or the rate of change in slope along a flow line, from a digital elevation model (DEM). It is the curvature of a normal section having a common tangent line with a slope line at a given point on the surface (Florinsky, 2017). This variable has an unbounded range that can take either positive or negative values. Positive values of the index are indicative of flow acceleration while negative profile curvature values indicate flow deceleration. Profile curvature is measured in units of m-1. For DEMs in projected coordinate systems, the tool uses the 3rd-order bivariate Taylor polynomial method described by Florinsky (2016). Based on a polynomial fit of the elevations within the 5x5 neighbourhood surrounding each cell, this method is considered more robust against outlier elevations (noise) than other methods.

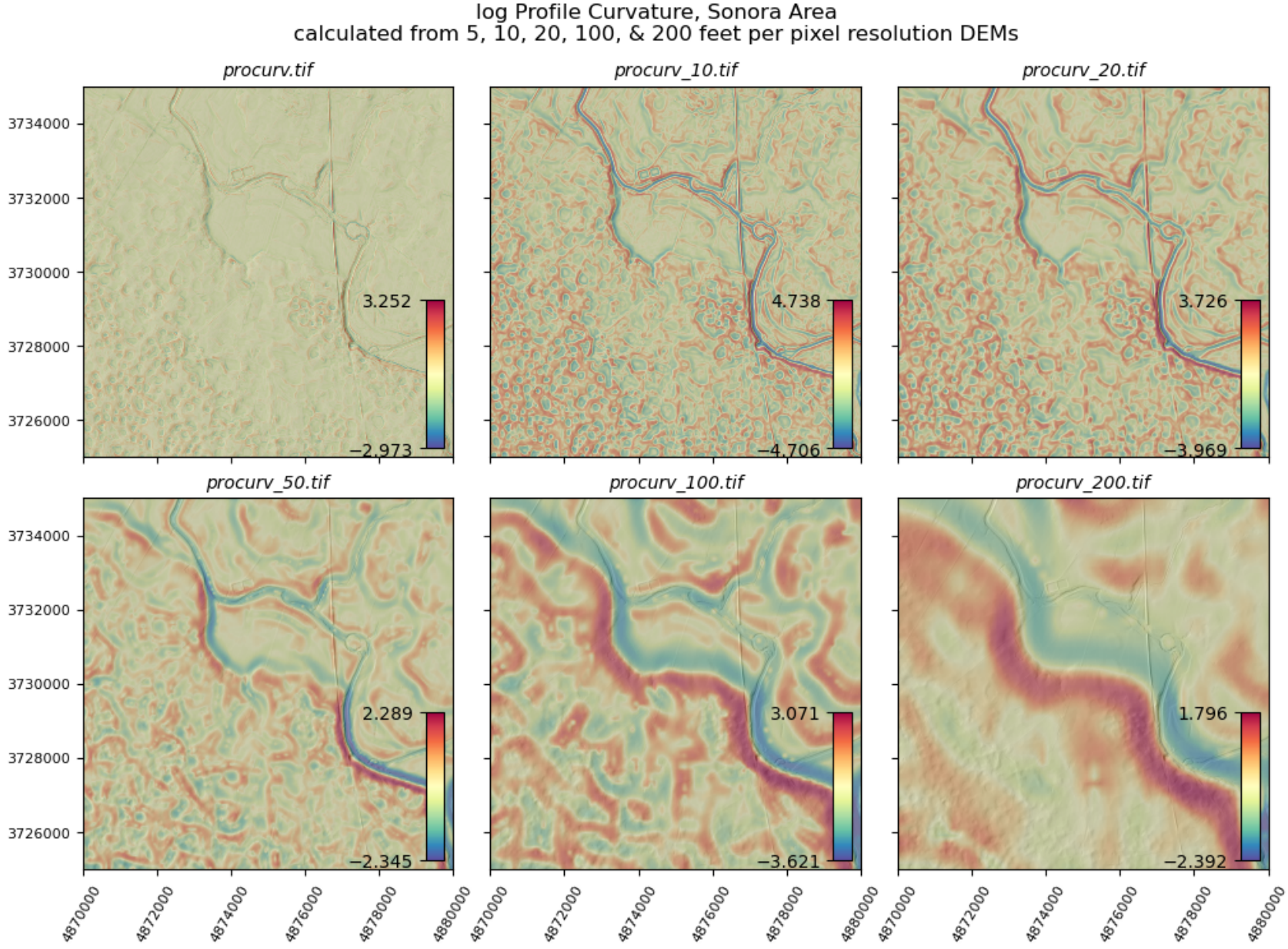
- https://www.whiteboxgeo.com/manual/wbt_book/available_tools/geomorphometric_analysis.html#ProfileCurvature

In [30]:
```
################################
# Calculate Profile Curvatures
################################

# paths to DEMs
dem_paths = glob.glob(r'../data/sonora/dem*.tif')

# path to target image (for reference)
reference_path = r'../data/sonora/geology.tif'

# gaussian filter standard deviation
sigma = 2


##### calculate log profile curvature
for dem in dem_paths:

    # create output path
    output_path = dem.replace('dem', 'procurv')

    # create absolute paths - Whitebox does not like relative paths
    dem = os.path.abspath(dem)
    output_path = os.path.abspath(output_path)

    # calculate profile curvature (default 5x5 pixel window)
    wbt.profile_curvature(dem=dem, output=output_path, log=True)

    # upsample & align to target image
    image_to_reference_image(input_path=output_path, reference_path=reference_path)
```

```
    # apply gaussian filter to upsampled image
    filter_image(input_path=output_path, sigma=sigma)
```

In [31]:

```
################################################
# Visualize Downsampled Profile Curvature Images
################################################

# paths to profile curvature images
procurv_paths = glob.glob(r'../data/sonora/procurv*.tif')
procurv_paths.sort(key=lambda x: list(map(int, re.findall(r'\d+', x))))

# path to multi-directional hillshade
mdhs_path = r'../data/sonora/mdhs.tif'

# bounding box coordinates for plot (left, bottom, right, top)
bounds = (4870000, 3725000, 4880000, 3735000)

# colormap for terrain feature
cmap = 'Spectral_r'

# title for plot
title = 'log Profile Curvature, Sonora Area\ncalculated from 5, 10, 20, 100, & 200 feet per pixel resolutio


##### plot profile curvature images
plot_multi_terrain_features(mdhs_path, procurv_paths, bounds, cmap, title)
```



Planform Curvature

This tool calculates the plan (or contour) curvature from a digital elevation model (DEM). Plan curvature is the curvature of a contour line at a given point on the topographic surface (Florinsky, 2017). This variable has an unbounded range that can take either positive or negative values. Positive values of the index are indicative of flow divergence while negative plan curvature values indicate flow convergence. Thus plan curvature is similar to tangential curvature, although the literature suggests that the latter is more numerically stable (Wilson, 2018). Plan curvature is measured in units of m-1. method described by Florinsky (2016). Based on a polynomial fit of the elevations within the 5x5 neighbourhood surrounding each cell, this method is considered more robust against outlier elevations (noise) than other methods.

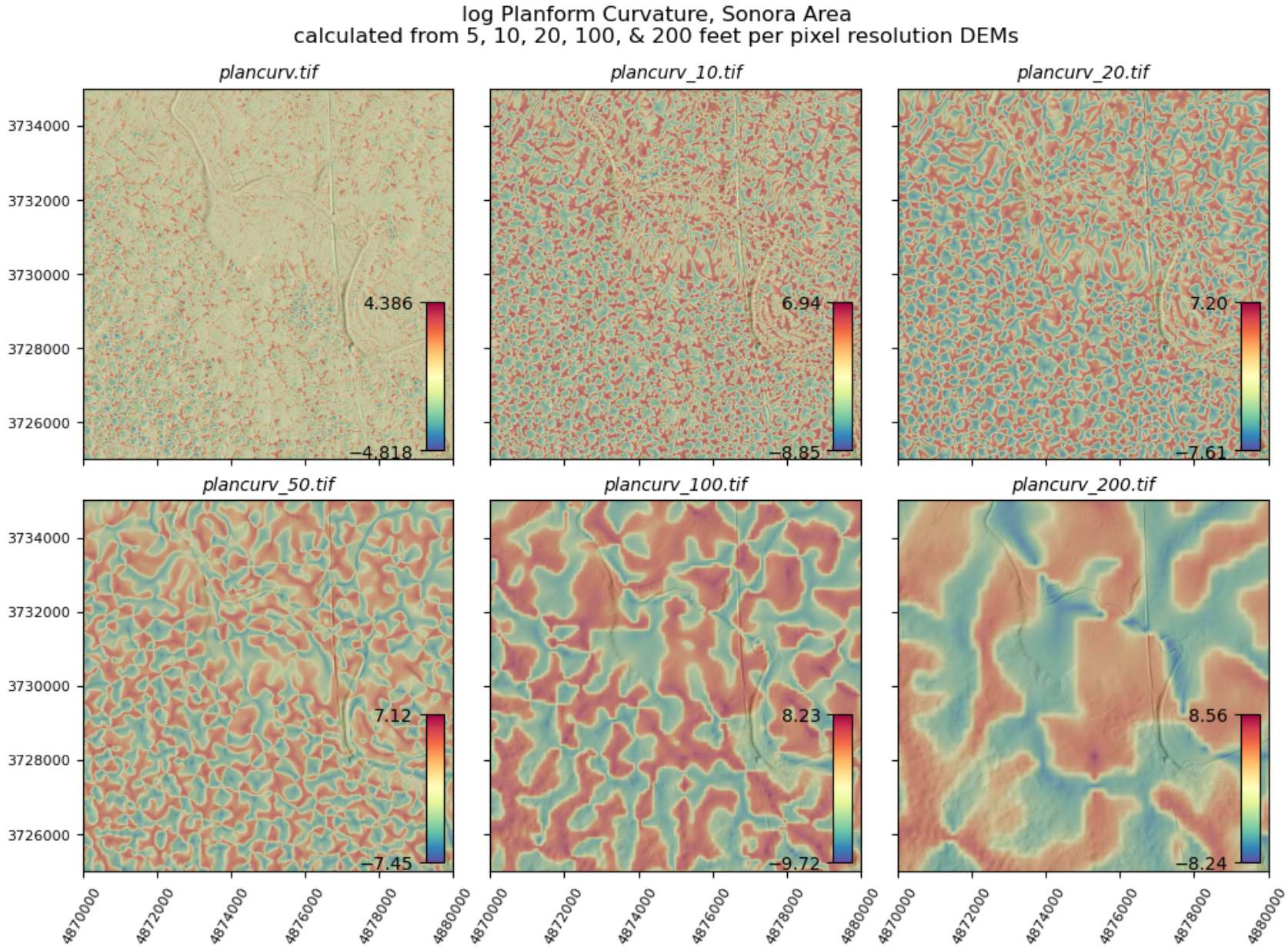
- https://www.whiteboxgeo.com/manual/wbt_book/available_tools/geomorphometric_analysis.html#PlanCurvature

In [32]:
```
################################
# Calculate Planform Curvatures
################################

# paths to DEMs
plancurv_paths = glob.glob(r'../data/sonora/dem*.tif')
plancurv_paths.sort(key=lambda x: list(map(int, re.findall(r'\d+', x))))

# path to target image (for reference)
reference_path = r'../data/sonora/geology.tif'

# gaussian filter standard deviation
sigma = 2


##### calculate log plan curvature
for dem in dem_paths:

    # create output path
    output_path = dem.replace('dem', 'plancurv')

    # create absolute paths - Whitebox does not like relative paths
    dem = os.path.abspath(dem)
    output_path = os.path.abspath(output_path)

    # calculate log plan curvature (default 5x5 pixel window)
    wbt.plan_curvature(dem=dem, output=output_path, log=True)

    # upsample & align to target image
```

```
    image_to_reference_image(input_path=output_path, reference_path=reference_path)

    # apply gaussian filter to upsampled image
    filter_image(input_path=output_path, sigma=sigma)
```

In [33]:

```
################################################
# Visualize Downsampled Planform Curvature Images
################################################

# paths to planform curvature images
plancurv_paths = glob.glob(r'../data/sonora/plancurv*.tif')
plancurv_paths.sort(key=lambda x: list(map(int, re.findall(r'\d+', x))))

# path to multi-directional hillshade
mdhs_path = r'../data/sonora/mdhs.tif'

# bounding box coordinates for plot (left, bottom, right, top)
bounds = (4870000, 3725000, 4880000, 3735000)

# colormap for terrain feature
cmap = 'Spectral_r'

# title for plot
title = 'log Planform Curvature, Sonora Area\ncalculated from 5, 10, 20, 100, & 200 feet per pixel resoluti


##### plot profile curvature images
plot_multi_terrain_features(mdhs_path, plancurv_paths, bounds, cmap, title)
```



Standard Deviation of Slope

Calculates the standard deviation of slope from an input DEM, a metric of roughness described by Grohmann et al., (2011).

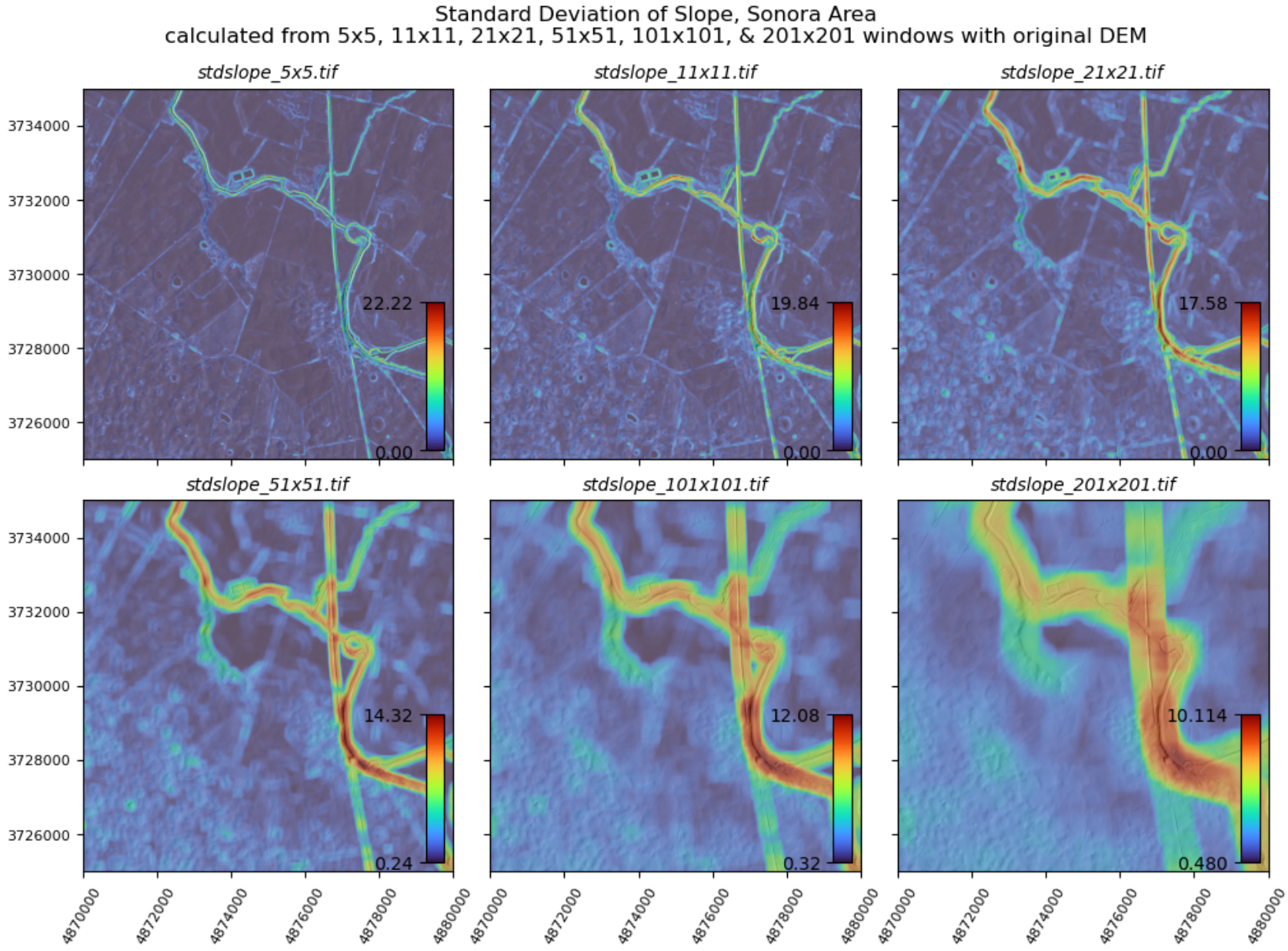
This feature is calculated with various sized window kernels and the original DEM (5 foot resolution); kernels represent approximately same area as the re-sampled terrain features calculated above.

- https://www.whiteboxgeo.com/manual/wbt_book/available_tools/geomorphometric_analysis.html#StandardDeviationOfSlope

In [34]:
```
################################################################
# Calculate Standard Deviation of Slope from Downsampled DEMs
################################################################

# paths to DEMs
dem_path = r'../data/sonora/dem.tif'

# path to target image (for reference)
reference_path = r'../data/sonora/geology.tif'

# window sizes
window_sizes = [5, 11, 21, 51, 101, 201]


##### calculate standard deviation of slope using various windows
for window in window_sizes:

    output_path = f'../data/sonora/stdslope_{window}x{window}.tif'

    dem = os.path.abspath(dem_path)
    output_path = os.path.abspath(output_path)

    wbt.standard_deviation_of_slope(dem, output_path, zfactor=None, filterx=window, filtery=window)

    image_to_reference_image(input_path=output_path, reference_path=reference_path)
```

In [35]:
```
############################################################
# Visualize Downsampled Standard Deviation of Slope Images
############################################################

# paths to standard deviation of slope images
stdslope_paths = glob.glob(r'../data/sonora/stdslope*x*.tif')
```

```
stdslope_paths.sort(key=lambda x: list(map(int, re.findall(r'\d+', x))))

# path to multi-directional hillshade
mdhs_path = r'../data/sonora/mdhs.tif'

# bounding box coordinates for plot (left, bottom, right, top)
bounds = (4870000, 3725000, 4880000, 3735000)

# colormap for terrain feature
cmap = 'turbo'

# title for plot
title = 'Standard Deviation of Slope, Sonora Area\ncalculated from 5x5, 11x11, 21x21, 51x51, 101x101, & 201


##### plot profile curvature images
plot_multi_terrain_features(mdhs_path, stdslope_paths, bounds, cmap, title)
```



Elevation Percentile

Elevation percentile (EP) is a measure of local topographic position (LTP). It expresses the vertical position for a DEM grid cell (z_0) as the percentile of the elevation distribution within the filter window, such that:

$$EP = count_{i \epsilon C}(z_i < z_0) \times (100/n_C)$$

where z_0 is the elevation of the window's center grid cell, z_i is the elevation of cell i contained within the window set C, and n_C is the number of grid cells contained within the window.

EP is unsigned and expressed as a percentage, bound between 0% and 100%. Quantile-based estimates (e.g., the median and interquartile range) are often used in nonparametric statistics to provide data variability estimates without assuming the distribution is normal. Thus, EP is largely unaffected by irregularly shaped elevation frequency distributions or by outliers in the DEM, resulting in a highly robust metric of LTP. In fact, elevation distributions within small to medium sized neighborhoods often exhibit skewed, multimodal, and non-Gaussian distributions, where the occurrence of elevation errors can often result in distribution outliers. Thus, based on these statistical characteristics, EP is considered one of the most robust representation of LTP.

This feature is calculated with various sized window kernels and the original DEM (5 foot resolution); kernels represent approximately same area as the re-sampled terrain features calculated above.

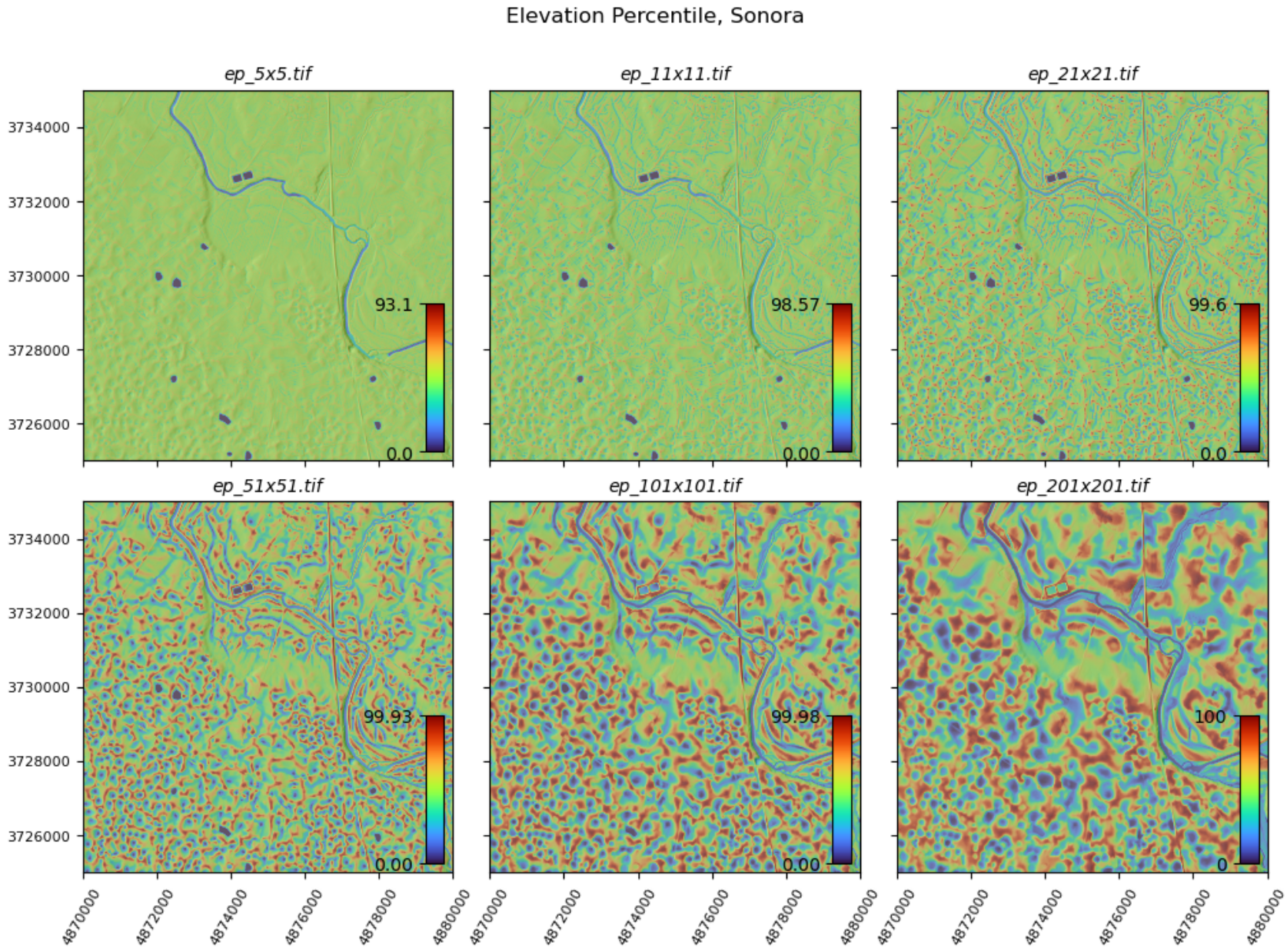
- https://www.whiteboxgeo.com/manual/wbt_book/available_tools/geomorphometric_analysis.html#ElevPercentile

In [36]:
```
######################################################################
# Calculate Elevation Percentile from DEM & Adjustable Windows
######################################################################

# paths to DEMs
dem_path = r'../data/sonora/dem.tif'

# path to target image (for reference)
reference_path = r'../data/sonora/geology.tif'

# window sizes
window_sizes = [5, 11, 21, 51, 101, 201]


##### calculate standard deviation of slope using various windows
for window in window_sizes:
```

```
    output_path = f'../data/sonora/ep_{window}x{window}.tif'

    dem = os.path.abspath(dem_path)
    output_path = os.path.abspath(output_path)

    wbt.elev_percentile(dem, output_path, filterx=window, filtery=window, sig_digits=2)

    image_to_reference_image(input_path=output_path, reference_path=reference_path)
```

In [37]:

```
######################################################
# Visualize Downsampled Elevation Percentile Images
######################################################

# paths to elevation percentile images
ep_paths = glob.glob(r'../data/sonora/ep*x*.tif')
ep_paths.sort(key=lambda x: list(map(int, re.findall(r'\d+', x))))

# path to multi-directional hillshade
mdhs_path = r'../data/sonora/mdhs.tif'

# bounding box coordinates for plot (left, bottom, right, top)
bounds = (4870000, 3725000, 4880000, 3735000)

# colormap for terrain feature
cmap = 'turbo'

# title for plot
title = 'Elevation Percentile, Sonora'


##### plot profile curvature images
plot_multi_terrain_features(mdhs_path, ep_paths, bounds, cmap, title)
```



Aerial Imagery

OVERVIEW

- Aerial imagery has been collected alongside LiDAR data as part of the KyFromAbove program, providing high-resolution coverage across Kentucky. The imagery is available with resolutions ranging from 1 foot to 3 inches, with a standard statewide resolution of 6 inches. Although the program captures true color imagery (RGB) across the state, some areas also include near-infrared (NIR) as a fourth band. These four-band images are accessible for download, offering detailed visual information about the landscape.

PURPOSE

- The four-band aerial imagery (RGB + NIR) plays a specific role in predicting certain surficial geologic map units. This imagery is particularly useful for identifying artificial fill, as man-made structures and disturbed land are easily distinguishable from natural landscapes. Additionally, the imagery may help in recognizing areas of alluvium or stream channels, especially where vegetation patterns (visible through the NIR band) can indicate the presence of water or recent sediment deposition. While the imagery may not be as effective for delineating all surficial map units, its value lies in enhancing the identification of human-altered landscapes and specific hydrological features.

- There may be a temporal discrepancy between the DEM and aerial imagery datasets, as they could have been captured at different times. This discrepancy can introduce challenges in analysis, particularly in areas where topographic changes have occurred between the collection dates, potentially affecting the consistency and accuracy of the derived features.

In [38]:
```
##################################################
# Download Aerial Imagery Tiles for Sonora
##################################################

# path to aerial imagery tile index GeoJSON
index_path = r'../data/sonora/aerial_index.geojson'

# field containing unique tile id
id_field = 'TileName'

# field containing TIFF download url
# NOTE: this field is not the same for all areas, and must be checked manually
url_field = 'Boxzip2020'

# output directory to save downloaded tiles
output_dir = r'../data/sonora/aerial_tiles'
```

```
if not os.path.isdir(output_dir):
    os.makedirs(output_dir)


##### call custom function to download tiles
download_data_tiles(index_path, id_field, url_field, output_dir)
```

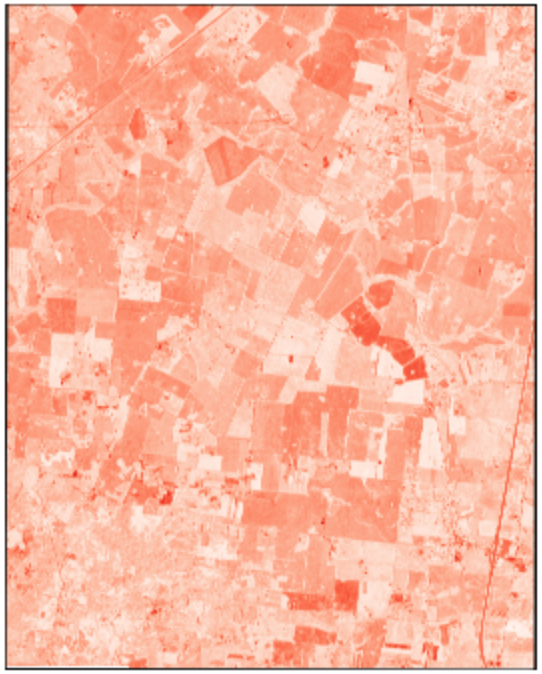
In [39]:
```
########################################
# Mosaic Aerial Tiles to Single GeoTIFF
########################################

# list of paths of aerial image tiles
aerial_tile_paths = glob.glob(r'../data/sonora/aerial_tiles/*.tif')

# path to image to use as alignment reference
reference_path = r'../data/sonora/geology.tif'


##### iterate through bands, mosaic tiles to single 1-band GeoTIFF, & align with reference image
band_names = ['r', 'g', 'b', 'nir']
for i, band in enumerate(band_names, start=1):
    aerial_path = f"../data/sonora/aerial{band}.tif"
    mosaic_image_tiles(aerial_tile_paths, aerial_path, band_number=i, resample=(5,5))
    image_to_reference_image(aerial_path, reference_path, output_path=None)
```

In [40]:
```
########################################
# Visualize Single Aerial Imagery GeoTIFF
########################################

# path to aerial image
aerial_paths = [r'../data/sonora/aerialr.tif',
                r'../data/sonora/aerialg.tif',
                r'../data/sonora/aerialb.tif',
                r'../data/sonora/aerialnir.tif']

# path to target boundary (not buffered)
boundary_path = r'../data/sonora/boundary.geojson'


##### plot aerial image
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(8,8))
```

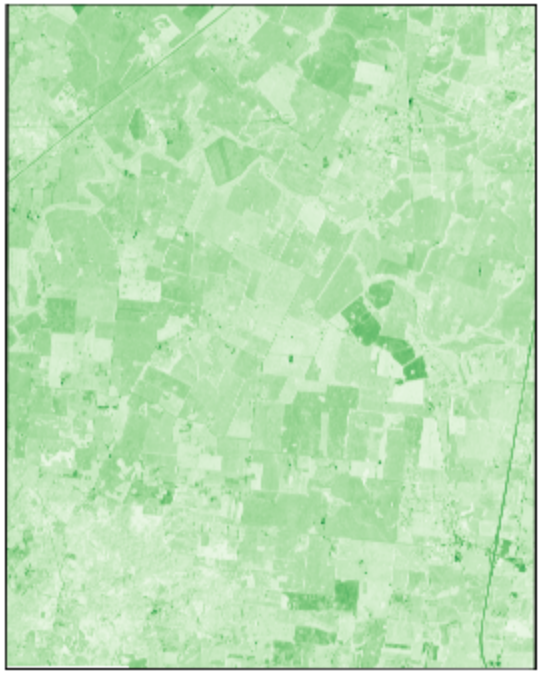
```
ax = ax.ravel()

# read dataset boundary
boundary = gpd.read_file(boundary_path)

# read & plot aerial
cmaps = ['Reds', 'Greens', 'Blues', 'coolwarm']
for idx, (aerial, cmap) in enumerate(zip(aerial_paths, cmaps)):
    with rasterio.open(aerial) as src:
        bounds = src.bounds
        show(src, ax=ax[idx], cmap=cmap)

    # plot boundary
    boundary.plot(ax=ax[idx], facecolor='none', edgecolor='k')

    # customize plot elements
    ax[idx].set_xlim(bounds.left, bounds.right)
    ax[idx].set_ylim(bounds.bottom, bounds.top)
    ax[idx].set_axis_off()
    label = os.path.basename(aerial)
    ax[idx].set_title(label, style='italic')

plt.suptitle('Aerial Imagery (RGB+NIR), Sonora Area\nresampled from 0.5 feet to 5 feet')
plt.tight_layout()
plt.show()
```

Aerial Imagery (RGB+NIR), Sonora Area
resampled from 0.5 feet to 5 feet

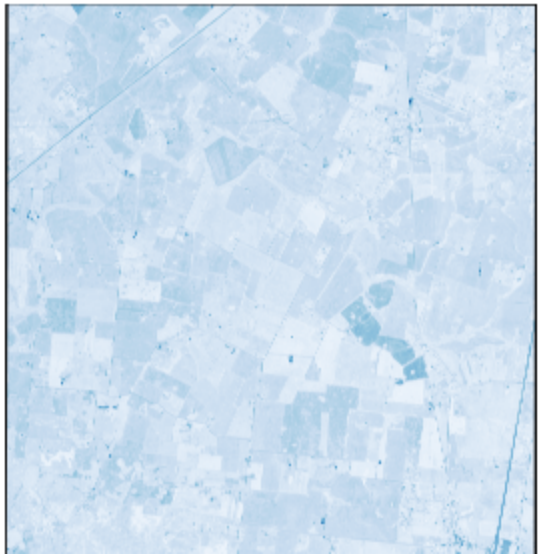
aerialr.tif



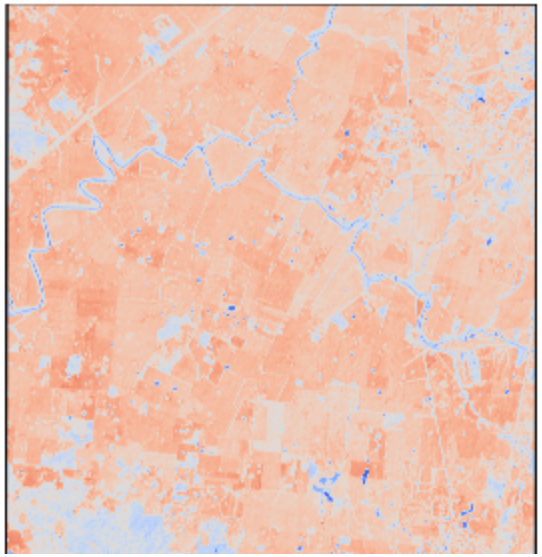
aerialg.tif

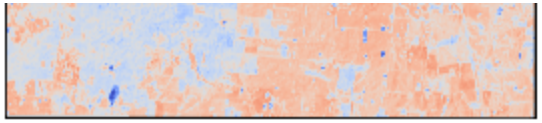


aerialb.tif



aerialnir.tif





In [41]:
```
###################################
# Delete Tiles for Space (optional)
###################################

# directory path to dem tiles
aerial_dir = r'../data/sonora/aerial_tiles'


##### delete aerial_tiles folder and all contents
if os.path.isdir(aerial_dir):
    shutil.rmtree(aerial_dir)
```

Hydrography

OVERVIEW

- The NHDPlus High Resolution (HR) dataset is a comprehensive geospatial resource that maps the flow of water across the United States, integrating stream networks, water bodies, and landscape features. Built using the National Hydrography Dataset (NHD) at a scale of 1:24,000 or finer, along with high-resolution elevation data from the 3D Elevation Program and the Watershed Boundary Dataset, NHDPlus HR provides detailed and accurate representations of hydrography. This dataset is crucial for hydrological analysis, environmental modeling, and understanding the interaction between water and the landscape. More information can be found in the USGS publication - https://pubs.usgs.gov/publication/ofr20191096.

PURPOSE

- The NHDPlus HR dataset includes two key features that will be integrated into this project:

    1. **NHDFlowline.** Stream centerlines represent the flow paths of rivers and streams, capturing the direction and connectivity of water flow across the landscape. These centerlines are essential for identifying and delineating areas of alluvium, as they highlight where water-driven sediment transport and deposition are most likely to occur.

2. **NHDWaterbody.** The waterbody polygons define the spatial extent of lakes, ponds, reservoirs, and other bodies of standing water. These features are useful for identifying low-lying areas where sediment may accumulate, as well as for recognizing artificial fill associated with water control structures like dams and levees.

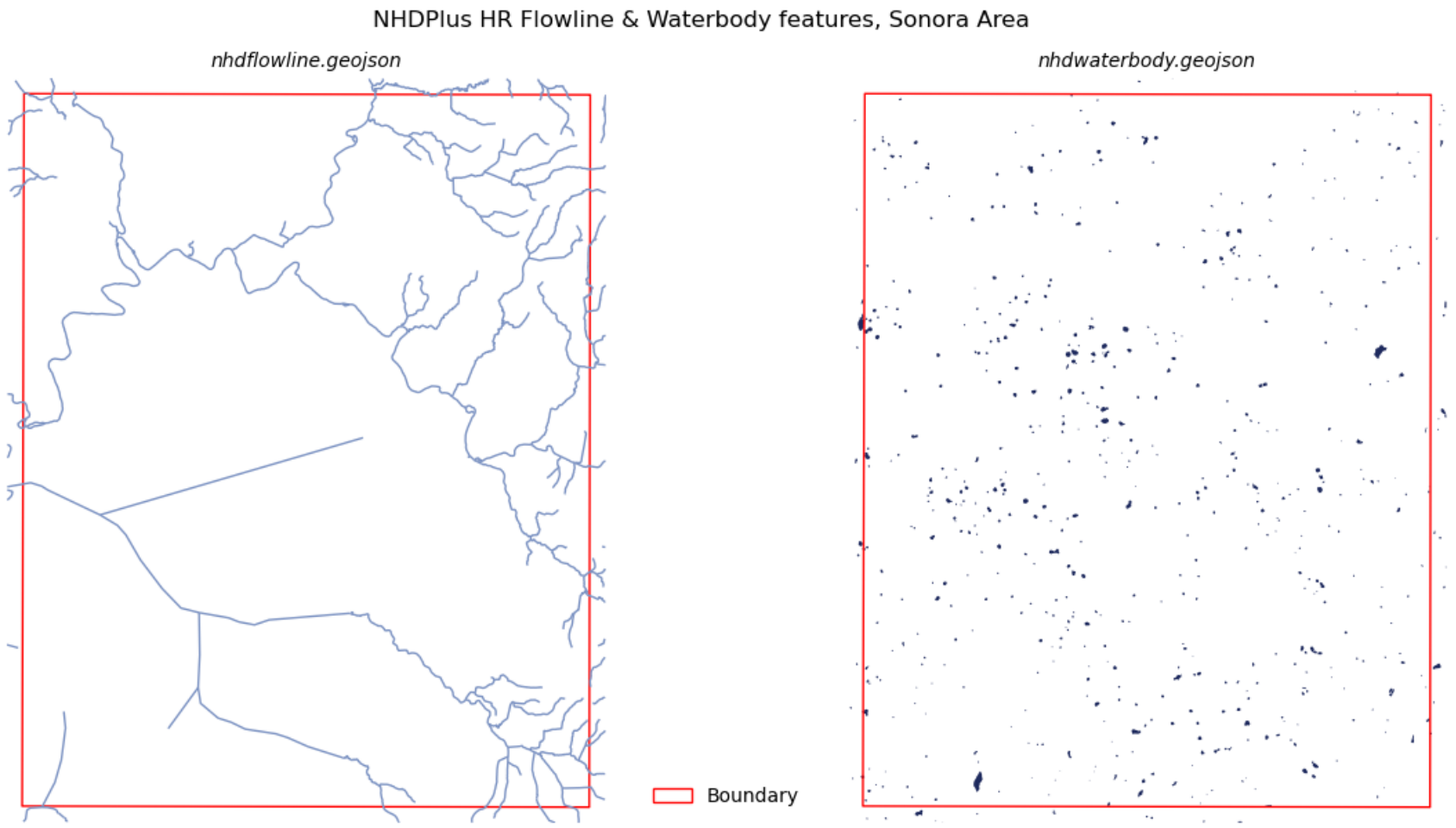
In [42]:
```
################################################
# Download NHDPlus HR Dataset for Sonora
################################################

# url for HUC 4 Basin 0511 (covers Sonora area)
nhd_url = r'https://prd-tnm.s3.amazonaws.com/StagedProducts/Hydrography/NHDPlusHR/VPU/Current/GDB/NHDPLUS_H

# directory to save hydrography
nhd_dir = r'../data/sonora/nhdplushr'
if not os.path.isdir(nhd_dir):
    os.makedirs(nhd_dir)


##### custom function to download zip, extract contents, and delete zip
download_zip(nhd_url, nhd_dir)
```

In [43]:
```
#####################################################################
# Clip to NHDFlowline & NHDWaterbody to Boundary & Save as GeoJSONs
#####################################################################

# path to NHDPlus HR geodatabase
gdb_path = glob.glob(r'../data/sonora/nhdplushr/*.gdb')[0]

# path to buffered target boundary GeoJSON
buffered_path = glob.glob(r'../data/sonora/*buffer*.geojson')[0]


##### clip NHDFlowline and NHDWaterbody layers to boundary
for layer in ['NHDFlowline', 'NHDWaterbody']:
    output_path = f"../data/sonora/{layer}.geojson"
    output_path = output_path.lower()
    if not os.path.isfile(output_path):
        clip_gis_to_boundary(gdb_path, buffered_path, output_path, layer)
```

In [44]:
```
########################################
# Visualize Flowline & Waterbody GeoJSONs
```

```
########################################

# path to flowline GeoJSON
flowline_path = r'../data/sonora/nhdflowline.geojson'

# alphabetized list of paths to hydrography GeoJSONs
waterbody_path = r'../data/sonora/nhdwaterbody.geojson'

# path to target dataset boundary
boundary_path = r'../data/sonora/boundary.geojson'


##### plot GeoJSONs
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15,7))
fig.subplots_adjust(wspace=0.1, hspace=0.1)

# flowlines
gdf_flowline = gpd.read_file(flowline_path)
gdf_flowline.plot(ax=ax[0], facecolor='none', edgecolor='#7C93C3', linewidth=1)
ax[0].set_title('nhdflowline.geojson', style='italic', fontsize=10)

bounds = gdf_flowline.total_bounds

# water bodies
gdf_waterbody = gpd.read_file(waterbody_path)
gdf_waterbody.plot(ax=ax[1], color='#1E2A5E')
ax[1].set_title('nhdwaterbody.geojson', style='italic', fontsize=10)

# dataset boundary
gdf_boundary = gpd.read_file(boundary_path)
for axes in ax:
    gdf_boundary.plot(ax=axes, facecolor='none', edgecolor='red', linewidth=1)
boundary_patch = Patch(facecolor='none', edgecolor='red', label='Boundary')
ax[0].legend(handles=[boundary_patch], loc='lower left', bbox_to_anchor=(1.05, 0), frameon=False)

# customize plot
for axes in ax:
    axes.set_axis_off()
    axes.set_xlim(bounds[0], bounds[2])
    axes.set_ylim(bounds[1], bounds[3])

plt.suptitle('NHDPlus HR Flowline & Waterbody features, Sonora Area', y=0.95)
```

```
plt.show()
```



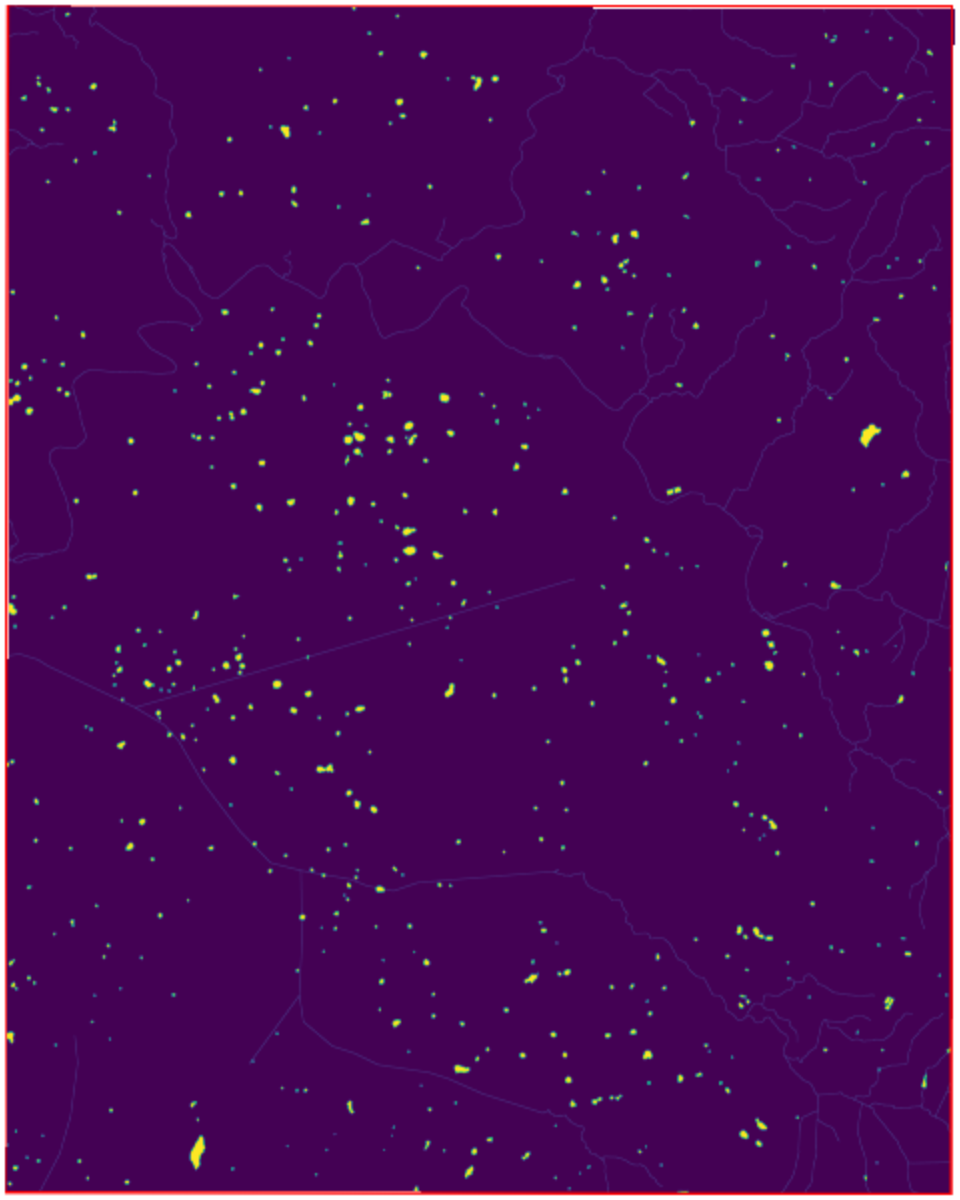
In [45]:
```
##########################################################
# Combine Flowlines & Waterbodies Into Single GeoTIFF
##########################################################

# list of paths to flowline and waterbody GeoJSONs
# NOTE: in this order, waterbody will be prioritized if features overlap
input_paths = [r'../data/sonora/nhdflowline.geojson',
               r'../data/sonora/nhdwaterbody.geojson']

# path to reference image (using labeled target dataset)
```

```
reference_path = r'../data/sonora/geology.tif'

# path for output GeoTIFF
output_path = r'../data/sonora/nhd.tif'


##### combine features and save as single GeoTIFF
# NOTE: if not binary output, JSON dictionary of labels and GeoTIFF values will also be created with same t
multiple_gis_to_reference_image(input_paths, reference_path, output_path, binary=True)
```

In [46]:

```
############################
# Visualize Single GeoTIFF
############################

# path to single GeoTIFF
nhd_path = r'../data/sonora/nhd.tif'

# path to dataset boundary GeoJSON
boundary_path = r'../data/sonora/boundary.geojson'


##### visualize combined GeoTIFF
fig, ax = plt.subplots(figsize=(15,7))

with rasterio.open(nhd_path) as src:
    show(src, ax=ax)

boundary = gpd.read_file(boundary_path)
boundary.plot(ax=ax, facecolor='none', edgecolor='red', linewidth=1)

handles = [Patch(facecolor='none', edgecolor='red', linewidth=1, label='Boundary')]
ax.legend(handles=handles, loc='center', bbox_to_anchor=(0.5,-0.05), frameon=False, fontsize=10)
ax.set_axis_off()
ax.set_title('NHDPlus HR Flowline & Waterbody features, Sonora Area')
plt.tight_layout()
plt.show()
```

NHDPlus HR Flowline & Waterbody features, Sonora Area



Boundary

In [47]:
```
#########################
# Delete NHD Data Folder
#########################

# path to nhd data folder (not GeoTIFF images)
nhd_dir = r'../data/sonora/nhdplushr'


##### remove nhdplushr folder and all contents (to save disk space)
if os.path.isdir(nhd_dir):
    shutil.rmtree(nhd_dir)
```

Roads & Railways

OVERVIEW

- OpenStreetMap (OSM) is a collaborative mapping project that provides freely accessible and up-to-date geospatial data. OSM data is continuously updated by a global community of contributors, making it a valuable resource for mapping current human developments and infrastructure. More information can be found at the OSM website - https://www.openstreetmap.org/about

PURPOSE

- Two layers from the OSM dataset will be utilized for this project, including road (gis_osm_roads_free_1) and railway (gis_osm_railways_free_1) centerlines. These features should be useful for identifying areas of artificial fill or areas of manmade alteration to the landscape. These features highlight regions where construction activities, such as road building or railway construction, may have introduced man-made changes to the landscape, such as grading, cutting, or filling.

- Since OSM data is continually updated, features may represent more recent developments that are not reflected in the KyFromAbove DEM or imagery datasets. Although This temporal mismatch could introduce some uncertainty in areas where significant changes have occurred since the DEM or aerial imagery were captured. However, the OSM data can still provide valuable insights into human modifications of the landscape, which are important for predicting surficial geologic map units.

In [48]:
```
##############################################
# Download OpenStreetMaps Dataset for Kentucky
##############################################

# url for OpenStreetMaps Kentucky dataset download
osm_url = r'http://download.geofabrik.de/north-america/us/kentucky-latest-free.shp.zip'

# directory to save statewide osm dataset
osm_dir = r'../data/sonora/osm_ky'
if not os.path.isdir(osm_dir):
    os.makedirs(osm_dir)


##### custom function to download zip, extract contents, and delete .zip
download_zip(osm_url, osm_dir)


##### only save shapefiles for roads & railroads
osm_files = glob.glob(os.path.join(osm_dir, '*.*'))
for path in osm_files:
    if ('gis_osm_roads_free_1' in path) or ('gis_osm_railways_free_1' in path):
        continue
    else:
        os.remove(path)
```

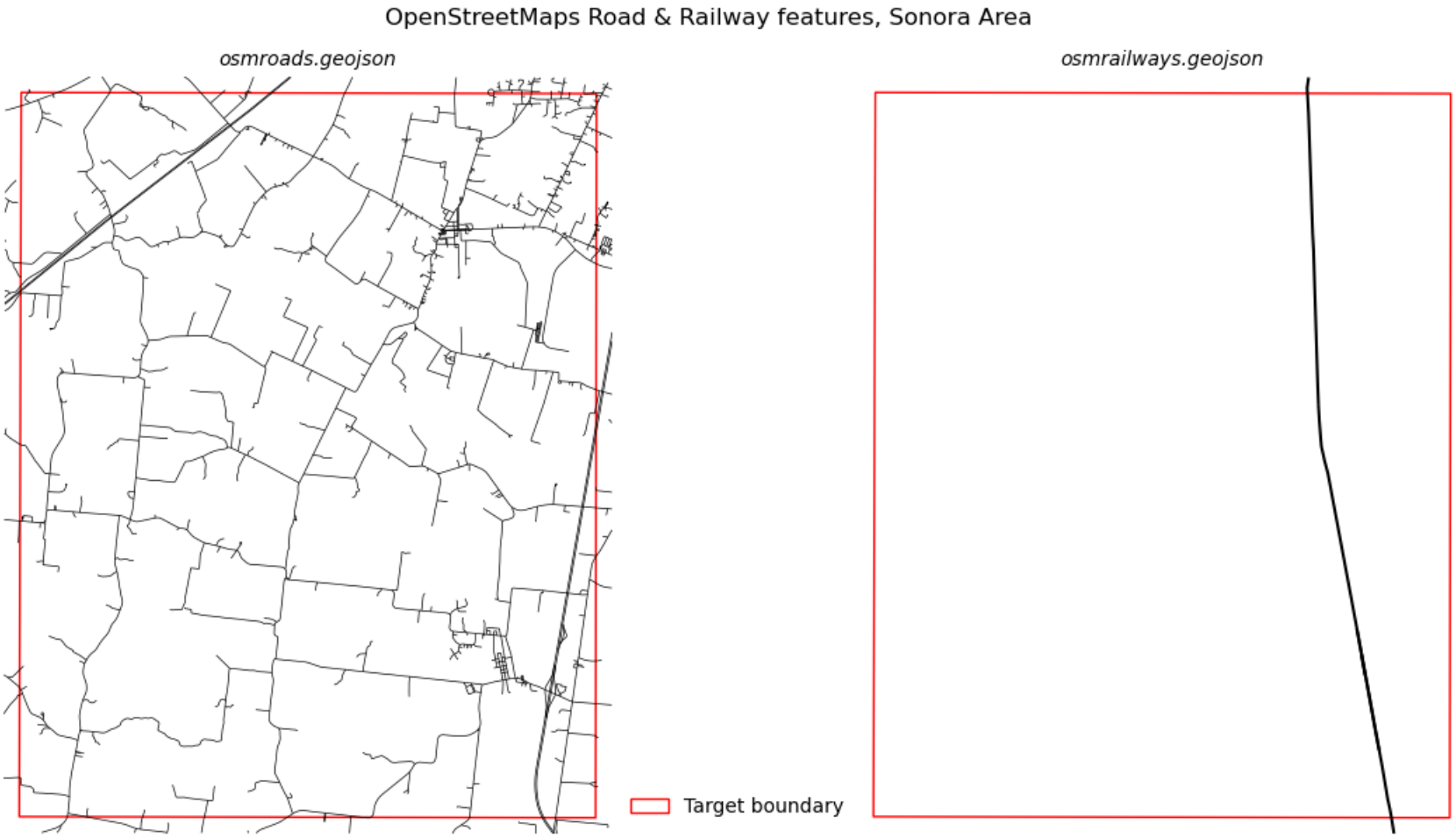
In [49]:
```
#########################################################################
# Clip to OSM Roads & Railways to Boundary to Target Dataset Boundary
#########################################################################

# path to NHDPlus HR geodatabase
osm_paths = glob.glob(r'../data/sonora/osm_ky/*.shp')

# path to target boundary GeoJSON
buffered_path = glob.glob(r'../data/sonora/*buffer*.geojson')[0]


##### clip roads & railways shapefiles to boundary
for path in osm_paths:
    if 'roads' in path:
        output_path = r'../data/sonora/osmroads.geojson'
    if 'railways' in path:
```

```
        output_path = r'../data/sonora/osmrailways.geojson'

    if not os.path.isfile(output_path):
        clip_gis_to_boundary(path, buffered_path, output_path)
```

In [50]:

```
########################################
# Visualize OSM Road & Railway GeoJSONs
########################################

# path to roads GeoJSON
roads_path = r'../data/sonora/osmroads.geojson'

# path to railways GeoJSON
rail_path = r'../data/sonora/osmrailways.geojson'

# path to target dataset boundary
boundary_path = r'../data/sonora/boundary.geojson'


##### plot GeoJSONs
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15,7))
fig.subplots_adjust(wspace=0.1, hspace=0.1)

# roads
gdf_roads = gpd.read_file(roads_path)
gdf_roads.plot(ax=ax[0], facecolor='none', edgecolor='k', linewidth=0.5)
ax[0].set_title('osmroads.geojson', style='italic', fontsize=10)

bounds = gdf_roads.total_bounds

# railways
gdf_railways = gpd.read_file(rail_path)
gdf_railways.plot(ax=ax[1], color='k', linewidth=1.5)
ax[1].set_title('osmrailways.geojson', style='italic', fontsize=10)

# dataset boundary
gdf_boundary = gpd.read_file(boundary_path)

for axes in ax:
    gdf_boundary.plot(ax=axes, facecolor='none', edgecolor='red', linewidth=1)
    axes.set_axis_off()
    axes.set_xlim(bounds[0], bounds[2])
```

```
    axes.set_ylim(bounds[1], bounds[3])

boundary_patch = Patch(facecolor='none', edgecolor='red', label='Target boundary')
ax[0].legend(handles=[boundary_patch], loc='lower left', bbox_to_anchor=(1.0, 0), frameon=False)

plt.suptitle('OpenStreetMaps Road & Railway features, Sonora Area', y=0.95)
plt.show()
```



In [51]:

```
#######################################################
# Combine OSM Roads & Railroads Into Single GeoTIFF
#######################################################

# list of paths to flowline and waterbody GeoJSONs
```

```
# NOTE: in this order, railways will be prioritized if features overlap
input_paths = [r'../data/sonora/osmroads.geojson',
               r'../data/sonora/osmrailways.geojson']

# path to reference image (using labeled target dataset)
reference_path = r'../data/sonora/geology.tif'

# path for output GeoTIFF
output_path = r'../data/sonora/osm.tif'


##### combine features and save as single GeoTIFF
# NOTE: if not binary image, JSON dictionary of labels and GeoTIFF values will also be created with same fi
multiple_gis_to_reference_image(input_paths, reference_path, output_path, binary=True)
```

In [52]:

```
################################
# Visualize Single OSM GeoTIFF
################################

# path to single GeoTIFF
nhd_path = r'../data/sonora/osm.tif'

# path to dataset boundary GeoJSON
boundary_path = r'../data/sonora/boundary.geojson'


##### visualize combined GeoTIFF
fig, ax = plt.subplots(figsize=(15,7))

with rasterio.open(nhd_path) as src:
    show(src, ax=ax)

boundary = gpd.read_file(boundary_path)
boundary.plot(ax=ax, facecolor='none', edgecolor='red', linewidth=1)

handles = [Patch(facecolor='none', edgecolor='red', linewidth=1, label='Boundary')]
ax.legend(handles=handles, loc='center', bbox_to_anchor=(0.5, -0.05), frameon=False, fontsize=10)
ax.set_axis_off()
ax.set_title('OpenStreetMaps Road & Railway features, Sonora Area')
plt.tight_layout()
plt.show()
```

OpenStreetMaps Road & Railway features, Sonora Area

Boundary

In [53]:
```
#########################
# Delete OSM Data Folder
#########################

# path to nhd data folder (not GeoTIFF images)
osm_dir = r'../data/sonora/osm_ky'


##### remove nhdplushr folder and all contents (to save disk space)
if os.path.isdir(osm_dir):
    shutil.rmtree(osm_dir)
```

Image Alignment

Check Image Alignments

In [54]:
```
########################
# Check Image Alignment
########################
# NOTE: DEM was created from entire tiles for terrain feature calculations, and will not be aligned at this

# paths to images
image_paths = glob.glob(r'../data/sonora/*.tif')
image_paths.sort(key=lambda x: x.lower())


##### check image alignment
df = check_image_alignment(image_paths, target='geology')
df.head(len(df))
```

Out[54]:

	image	path	dtype	aligned	resolution_x	resolution_y	width	height	left	bo
0	aerialb	../data/sonora/aerialb.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
1	aerialg	../data/sonora/aerialg.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
2	aerialnir	../data/sonora/aerialnir.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
3	aerialr	../data/sonora/aerialr.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
4	dem	../data/sonora/dem.tif	float32	False	5.00	5.00	9000	11000	4845000.00	370000
5	dem_10	../data/sonora/dem_10.tif	float32	False	10.00	10.00	4500	5500	4845000.00	370000
6	dem_100	../data/sonora/dem_100.tif	float32	False	100.00	100.00	450	550	4845000.00	370000
7	dem_20	../data/sonora/dem_20.tif	float32	False	20.00	20.00	2250	2750	4845000.00	370000
8	dem_200	../data/sonora/dem_200.tif	float32	False	200.00	200.00	225	275	4845000.00	370000
9	dem_50	../data/sonora/dem_50.tif	float32	False	50.00	50.00	900	1100	4845000.00	370000
10	ep_101x101	../data/sonora/ep_101x101.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
11	ep_11x11	../data/sonora/ep_11x11.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
12	ep_201x201	../data/sonora/ep_201x201.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
13	ep_21x21	../data/sonora/ep_21x21.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
14	ep_51x51	../data/sonora/ep_51x51.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565

	image	path	dtype	aligned	resolution_x	resolution_y	width	height	left	bo
15	ep_5x5	../data/sonora/ep_5x5.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
16	geology	../data/sonora/geology.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
17	mdhs	../data/sonora/mdhs.tif	int16	False	5.00	5.00	9000	11000	4845000.00	370000
18	nhd	../data/sonora/nhd.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
19	osm	../data/sonora/osm.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
20	plancurv	../data/sonora/plancurv.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
21	plancurv_10	../data/sonora/plancurv_10.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
22	plancurv_100	../data/sonora/plancurv_100.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
23	plancurv_20	../data/sonora/plancurv_20.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
24	plancurv_200	../data/sonora/plancurv_200.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
25	plancurv_50	../data/sonora/plancurv_50.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
26	procurv	../data/sonora/procurv.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
27	procurv_10	../data/sonora/procurv_10.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
28	procurv_100	../data/sonora/procurv_100.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
29	procurv_20	../data/sonora/procurv_20.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565

	image	path	dtype	aligned	resolution_x	resolution_y	width	height	left	bo
30	procurv_200	../data/sonora/procurv_200.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
31	procurv_50	../data/sonora/procurv_50.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
32	slope	../data/sonora/slope.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
33	slope_10	../data/sonora/slope_10.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
34	slope_100	../data/sonora/slope_100.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
35	slope_20	../data/sonora/slope_20.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
36	slope_200	../data/sonora/slope_200.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
37	slope_50	../data/sonora/slope_50.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
38	stdslope_101x101	../data/sonora/stdslope_101x101.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
39	stdslope_11x11	../data/sonora/stdslope_11x11.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
40	stdslope_201x201	../data/sonora/stdslope_201x201.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
41	stdslope_21x21	../data/sonora/stdslope_21x21.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
42	stdslope_51x51	../data/sonora/stdslope_51x51.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
43	stdslope_5x5	../data/sonora/stdslope_5x5.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565

Align Images

In [55]:
```
################################
# Align Images with Target Image
################################

# paths to images that need clipped (should only be original DEM)
# NOTE: Downsampled DEMs and mdhs will not be used for modeling, so will not be aligned here.
input_paths = [r'../data/sonora/dem.tif']

# path to reference target image to align input
reference_path = r'../data/sonora/geology.tif'


##### register and align input images with labeled target dataset
for path in input_paths:
    image_to_reference_image(path, reference_path)
```

Verify Image Alignments

In [56]:
```
############################
# Re-Check Image Alignment
############################
# NOTE: all images should be aligned now, except those not used for modeling (downsampled DEMs and multidi

# paths to images
image_paths = glob.glob(r'../data/sonora/*.tif')
image_paths.sort(key=lambda x: x.lower())


##### check image alignment
df = check_image_alignment(image_paths, target='geology')
df.head(len(df))
```

Out[56]:

	image	path	dtype	aligned	resolution_x	resolution_y	width	height	left	bo
0	aerialb	../data/sonora/aerialb.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
1	aerialg	../data/sonora/aerialg.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
2	aerialnir	../data/sonora/aerialnir.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
3	aerialr	../data/sonora/aerialr.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
4	dem	../data/sonora/dem.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
5	dem_10	../data/sonora/dem_10.tif	float32	False	10.00	10.00	4500	5500	4845000.00	370000
6	dem_100	../data/sonora/dem_100.tif	float32	False	100.00	100.00	450	550	4845000.00	370000
7	dem_20	../data/sonora/dem_20.tif	float32	False	20.00	20.00	2250	2750	4845000.00	370000
8	dem_200	../data/sonora/dem_200.tif	float32	False	200.00	200.00	225	275	4845000.00	370000
9	dem_50	../data/sonora/dem_50.tif	float32	False	50.00	50.00	900	1100	4845000.00	370000
10	ep_101x101	../data/sonora/ep_101x101.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
11	ep_11x11	../data/sonora/ep_11x11.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
12	ep_201x201	../data/sonora/ep_201x201.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
13	ep_21x21	../data/sonora/ep_21x21.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
14	ep_51x51	../data/sonora/ep_51x51.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565

	image	path	dtype	aligned	resolution_x	resolution_y	width	height	left	bo
15	ep_5x5	../data/sonora/ep_5x5.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
16	geology	../data/sonora/geology.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
17	mdhs	../data/sonora/mdhs.tif	int16	False	5.00	5.00	9000	11000	4845000.00	370000
18	nhd	../data/sonora/nhd.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
19	osm	../data/sonora/osm.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
20	plancurv	../data/sonora/plancurv.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
21	plancurv_10	../data/sonora/plancurv_10.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
22	plancurv_100	../data/sonora/plancurv_100.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
23	plancurv_20	../data/sonora/plancurv_20.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
24	plancurv_200	../data/sonora/plancurv_200.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
25	plancurv_50	../data/sonora/plancurv_50.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
26	procurv	../data/sonora/procurv.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
27	procurv_10	../data/sonora/procurv_10.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
28	procurv_100	../data/sonora/procurv_100.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
29	procurv_20	../data/sonora/procurv_20.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565

	image	path	dtype	aligned	resolution_x	resolution_y	width	height	left	bo
30	procurv_200	../data/sonora/procurv_200.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
31	procurv_50	../data/sonora/procurv_50.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
32	slope	../data/sonora/slope.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
33	slope_10	../data/sonora/slope_10.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
34	slope_100	../data/sonora/slope_100.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
35	slope_20	../data/sonora/slope_20.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
36	slope_200	../data/sonora/slope_200.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
37	slope_50	../data/sonora/slope_50.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
38	stdslope_101x101	../data/sonora/stdslope_101x101.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
39	stdslope_11x11	../data/sonora/stdslope_11x11.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
40	stdslope_201x201	../data/sonora/stdslope_201x201.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
41	stdslope_21x21	../data/sonora/stdslope_21x21.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
42	stdslope_51x51	../data/sonora/stdslope_51x51.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565
43	stdslope_5x5	../data/sonora/stdslope_5x5.tif	float32	True	5.00	5.00	7265	9118	4848728.04	370565

In [58]:
```
############################
```

```
# Delete Unnecessary Images
############################

# paths to unnecessary images
unused_paths = df.loc[df['aligned']==False, 'path'].to_list()
unused_paths = [path for path in unused_paths if not 'mdhs' in path]

##### delete images
for path in unused_paths:
    if os.path.isfile(path):
        os.remove(path)
```

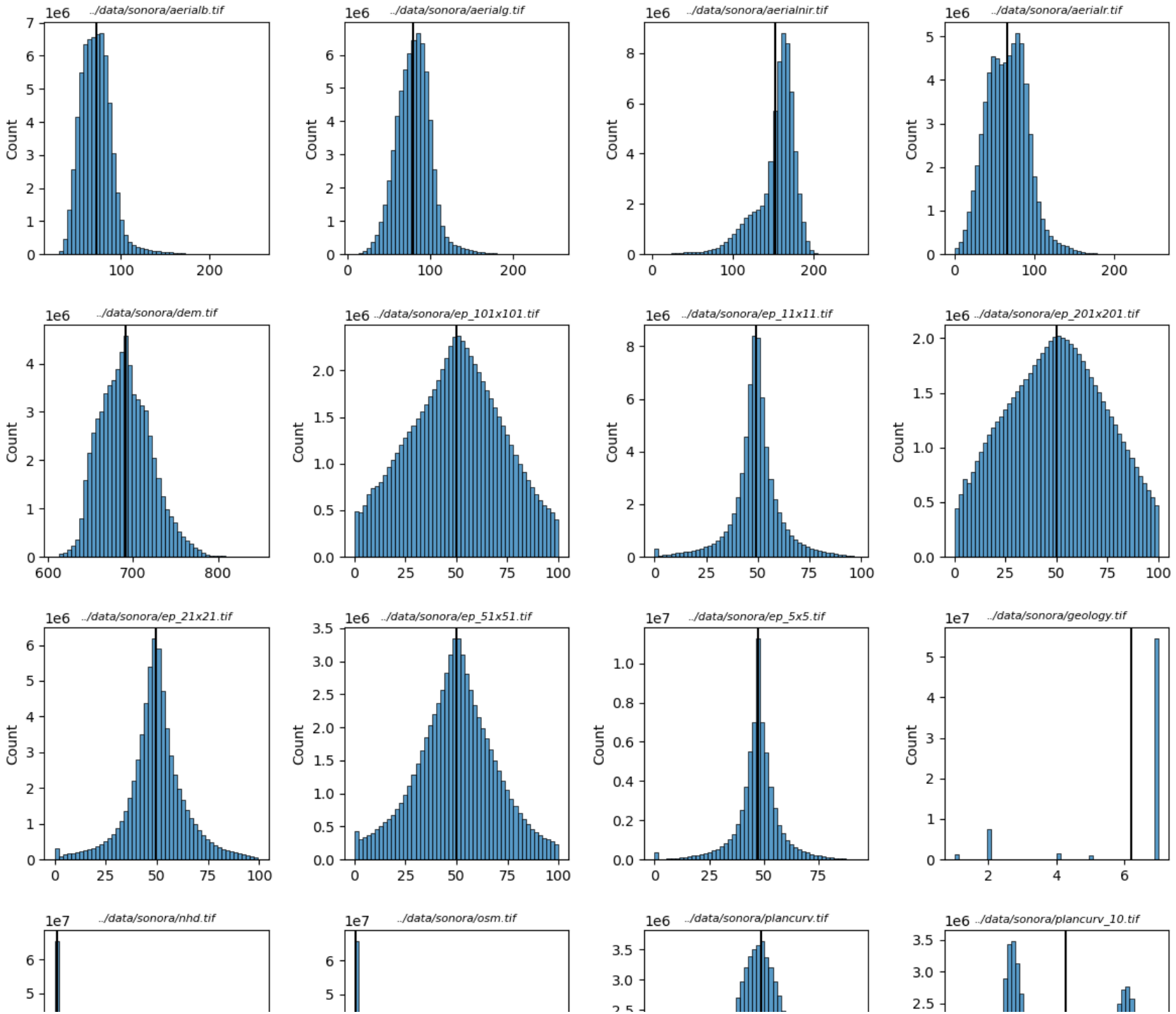
Image Statistics

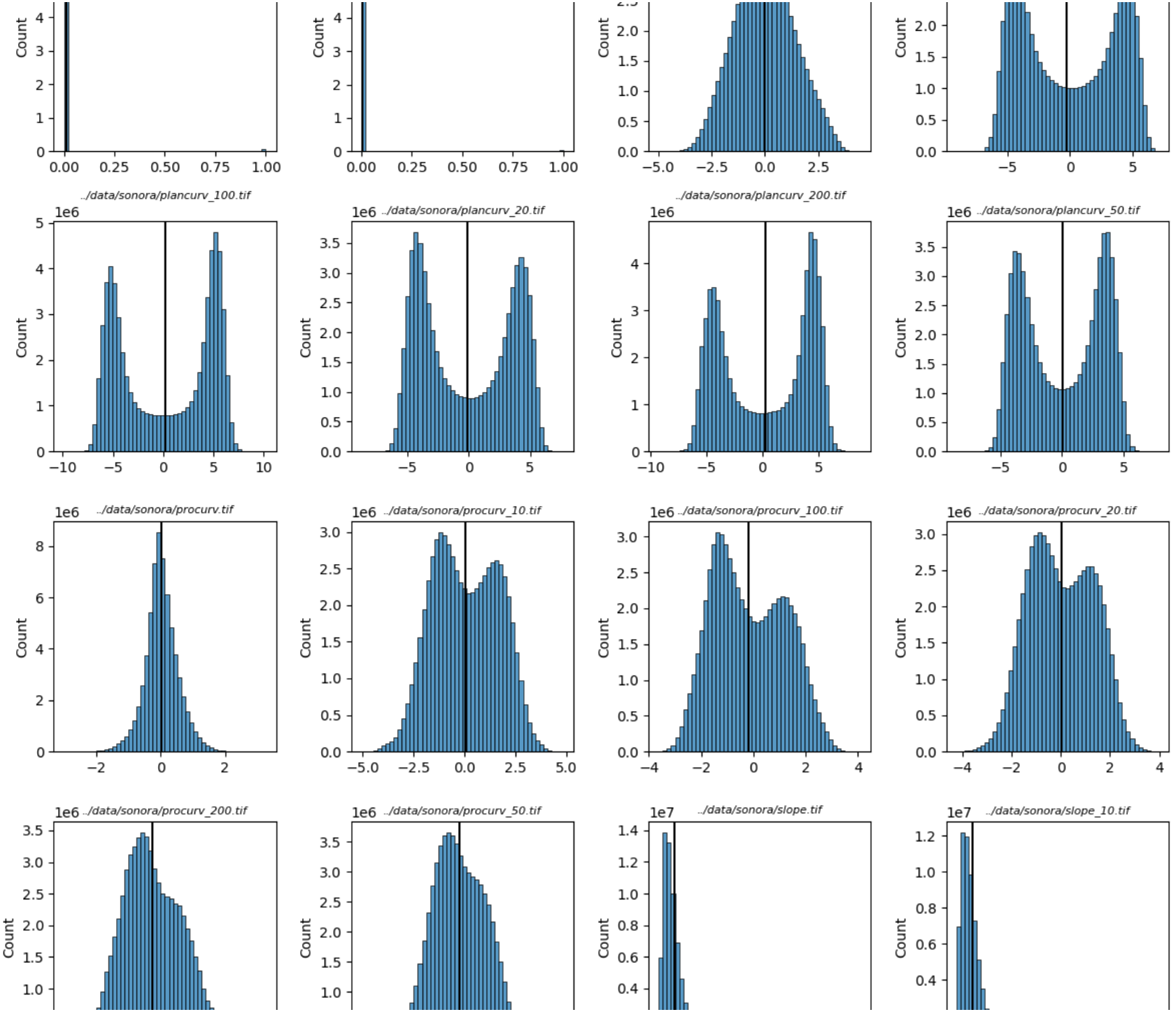
Distributions

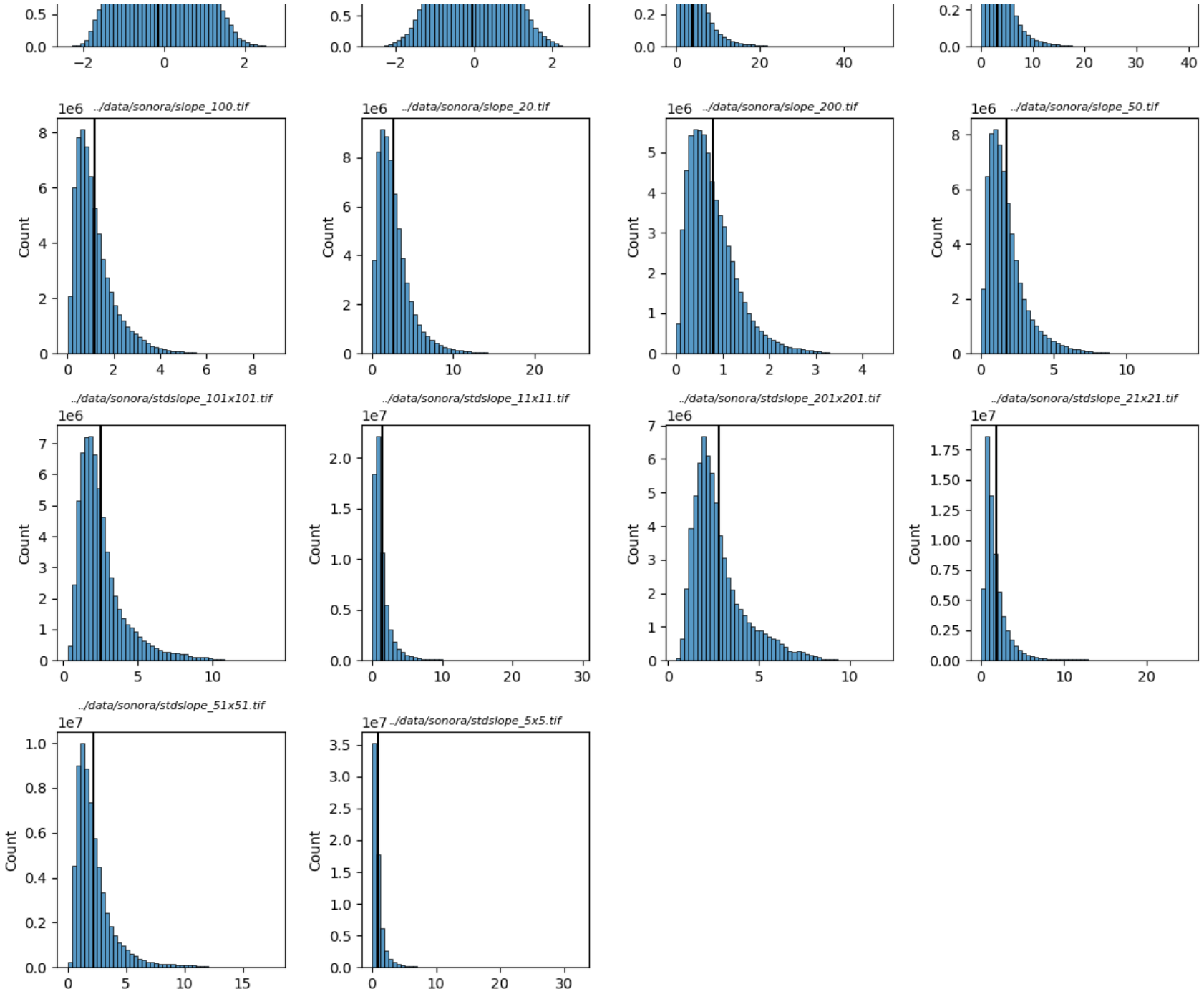
In [59]:

```
########################################
# Visualize Data Distributions of Images
########################################

# paths to images
image_paths = glob.glob(r'../data/sonora/*.tif')
image_paths = [path for path in image_paths if not 'mdhs' in path]
image_paths.sort(key=lambda x: x.lower())


##### plot histograms & means of images
fig, ax = plt.subplots(nrows=10, ncols=4, figsize=(12, 30))
ax = ax.ravel()

for idx, path in enumerate(image_paths):

    with rasterio.open(path) as src:
        data = src.read(1, masked=True)
        valid = data[~data.mask].data
        mean = np.mean(valid)

    sns.histplot(ax=ax[idx], data=valid, bins=50, edgecolor='k', linewidth=0.5)
```

```
    ax[idx].axvline(x=mean, color='k', linewidth=1.5)
    ax[idx].set_title(f"{path}", fontsize=8, style='italic')

for unused in ax[-2:]:
    fig.delaxes(unused)

plt.tight_layout()
plt.show()
```


../data/sonora/aerialb.tif
../data/sonora/aerialg.tif
../data/sonora/aerialnir.tif
../data/sonora/aerialr.tif
../data/sonora/dem.tif
../data/sonora/ep_101x101.tif
../data/sonora/ep_11x11.tif
../data/sonora/ep_201x201.tif
../data/sonora/ep_21x21.tif
../data/sonora/ep_51x51.tif
../data/sonora/ep_5x5.tif
../data/sonora/geology.tif
../data/sonora/nhd.tif
../data/sonora/osm.tif
../data/sonora/plancurv.tif
../data/sonora/plancurv_10.tif
Count


../data/sonora/plancurv_100.tif
../data/sonora/plancurv_20.tif
../data/sonora/plancurv_200.tif
../data/sonora/plancurv_50.tif
../data/sonora/procurv.tif
../data/sonora/procurv_10.tif
../data/sonora/procurv_100.tif
../data/sonora/procurv_20.tif
../data/sonora/procurv_200.tif
../data/sonora/procurv_50.tif
../data/sonora/slope.tif
../data/sonora/slope_10.tif
Count
1e6
1e7

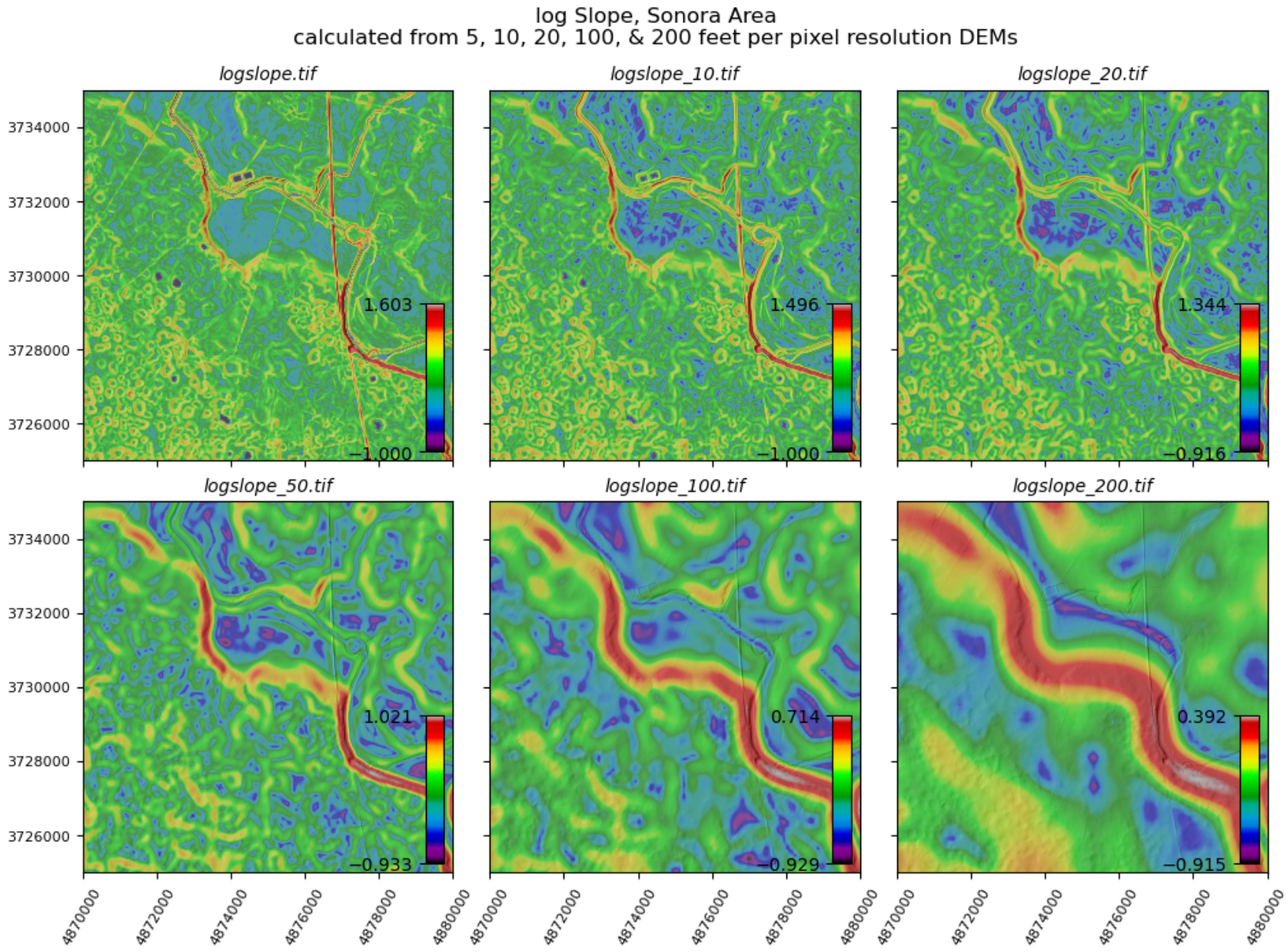

../data/sonora/slope_100.tif
../data/sonora/slope_20.tif
../data/sonora/slope_200.tif
../data/sonora/slope_50.tif
../data/sonora/stdslope_101x101.tif
../data/sonora/stdslope_11x11.tif
../data/sonora/stdslope_201x201.tif
../data/sonora/stdslope_21x21.tif
../data/sonora/stdslope_51x51.tif
../data/sonora/stdslope_5x5.tif
Count
1e6
1e7

Transformations

In [67]:
```
##########################
# log Transform Features
##########################

# paths to feature images to transform
slope_paths = glob.glob(r'../data/sonora/*slope*.tif')


##### apply log transformation to slope and standard deviation of slope (positive skew)
for path in slope_paths:

    with rasterio.open(path) as src:
        data = src.read(1, masked=True)
        dst_data = data + 0.1              # add scalar to prevent log(0)
        dst_data = np.log10(dst_data)      # transform by log base 10
        dst_meta = src.meta.copy()

    # modify output path
    output_basename = 'log' + os.path.basename(path)
    output_dir = os.path.dirname(path)
    output_path = f"{output_dir}/{output_basename}"

    # write new image GeoTIFF
    with rasterio.open(output_path, 'w', **dst_meta) as dst:
        dst.write(dst_data, 1)
```

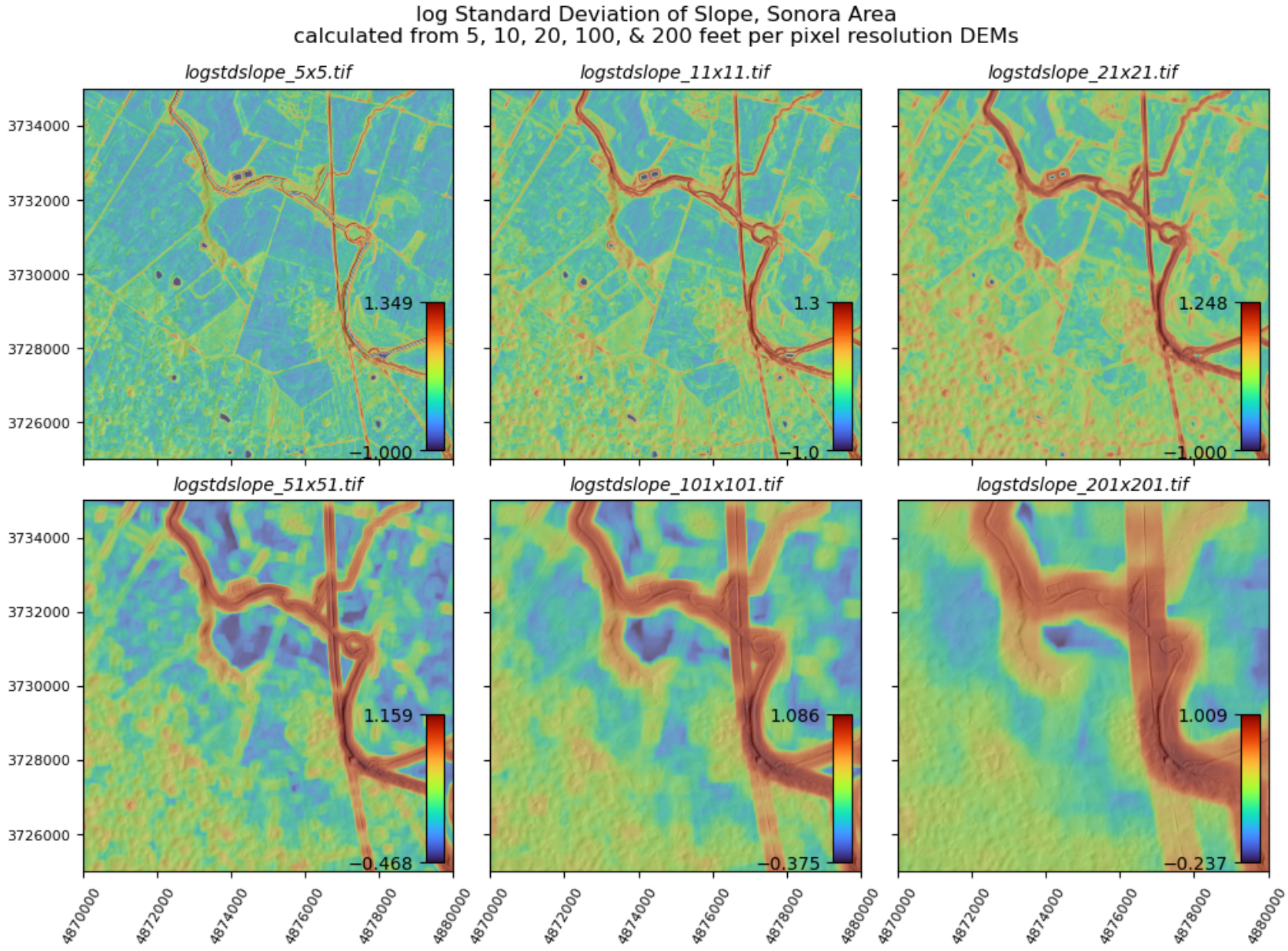
In [71]:
```
##############################
# Visualize Transformed Slope
##############################

# paths to transformed features
trans_paths = glob.glob(r'../data/sonora/*logslope*.tif')
trans_paths.sort(key=lambda x: list(map(int, re.findall(r'\d+', x))))

# path to multi-directional hillshade
mdhs_path = r'../data/sonora/mdhs.tif'

# bounding box coordinates for plot (left, bottom, right, top)
bounds = (4870000, 3725000, 4880000, 3735000)

# colormap for terrain feature
```

```
cmap = 'nipy_spectral'

# title for plot
title = 'log Slope, Sonora Area\ncalculated from 5, 10, 20, 100, & 200 feet per pixel resolution DEMs'


##### plot slope images
plot_multi_terrain_features(mdhs_path, trans_paths, bounds, cmap, title)
```



In [70]:
```
#######################################################
# Visualize Transformed Standard Deviation of Slope
#######################################################
```

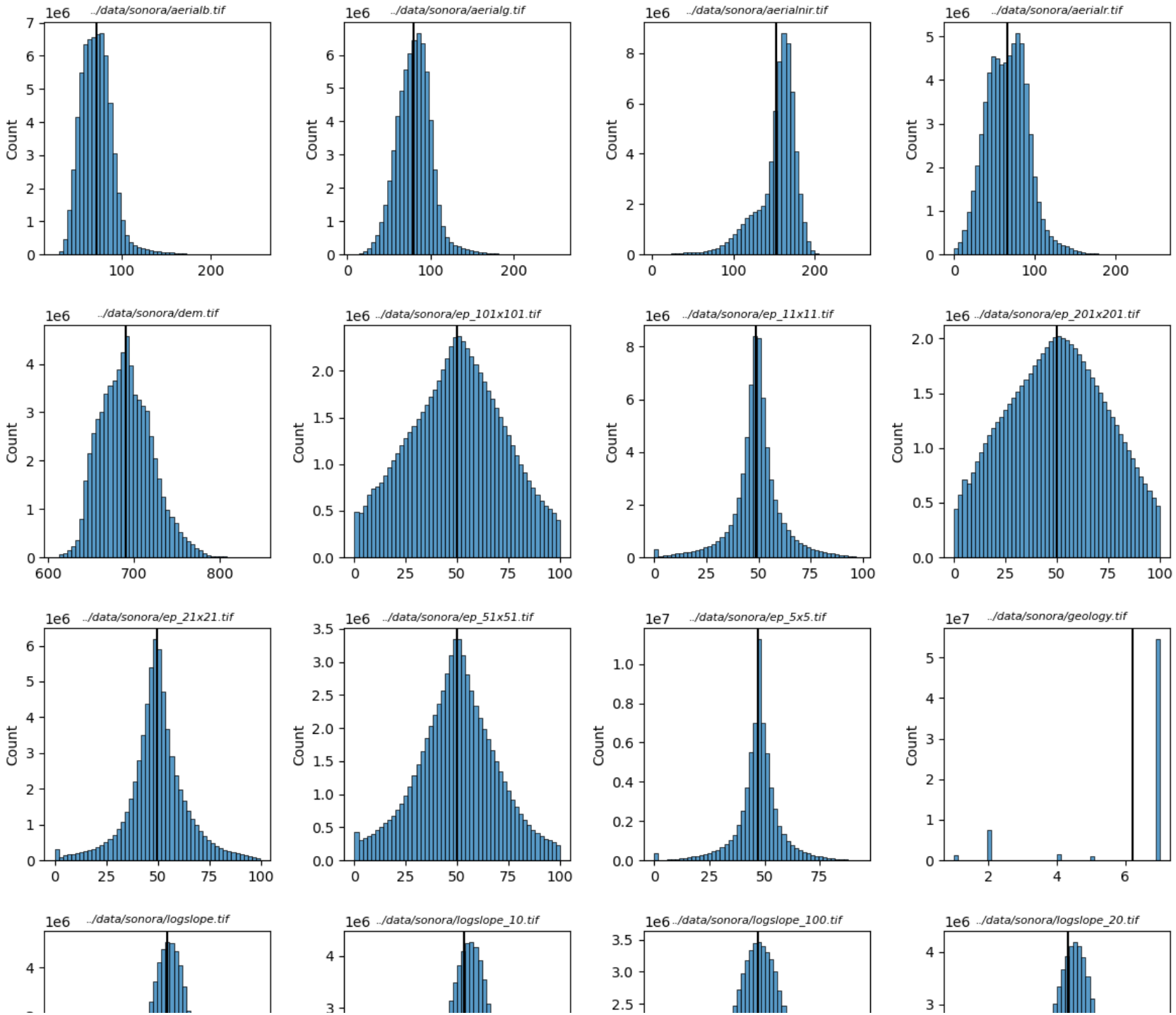
```
# paths to transformed features
trans_paths = glob.glob(r'../data/sonora/*logstdslope*.tif')
trans_paths.sort(key=lambda x: list(map(int, re.findall(r'\d+', x))))

# path to multi-directional hillshade
mdhs_path = r'../data/sonora/mdhs.tif'

# bounding box coordinates for plot (left, bottom, right, top)
bounds = (4870000, 3725000, 4880000, 3735000)

# colormap for terrain feature
cmap = 'turbo'

# title for plot
title = 'log Standard Deviation of Slope, Sonora Area\ncalculated from 5, 10, 20, 100, & 200 feet per pixe


##### plot slope images
plot_multi_terrain_features(mdhs_path, trans_paths, bounds, cmap, title)
```

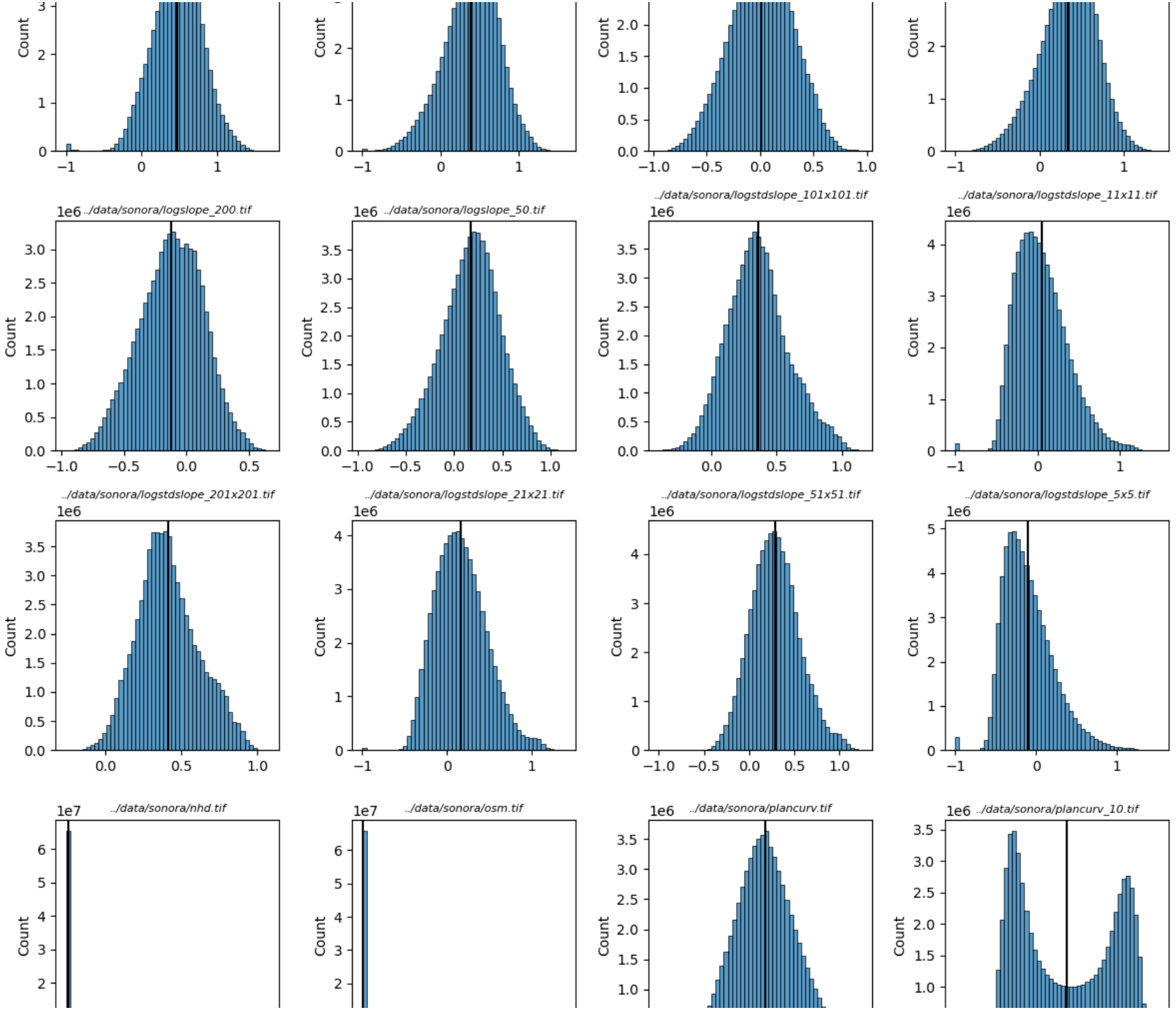


Verify Distributions

In [72]:

```
#################################################################
# Visualize Data Distributions of Images & Transformed Images
#################################################################

# paths to images
image_paths = glob.glob(r'../data/sonora/*.tif')
image_paths = [path for path in image_paths if not (os.path.basename(path).startswith('slope')) | (os.path.
image_paths.sort(key=lambda x: x.lower())


##### plot histograms & means of images
fig, ax = plt.subplots(nrows=10, ncols=4, figsize=(12, 30))
ax = ax.ravel()

for idx, path in enumerate(image_paths):

    with rasterio.open(path) as src:
        data = src.read(1, masked=True)
        valid = data[~data.mask].data
        mean = np.mean(valid)

    sns.histplot(ax=ax[idx], data=valid, bins=50, edgecolor='k', linewidth=0.5)
    ax[idx].axvline(x=mean, color='k', linewidth=1.5)
    ax[idx].set_title(f"{path}", fontsize=8, style='italic')

for unused in ax[-2:]:
    fig.delaxes(unused)

plt.tight_layout()
plt.show()
```


../data/sonora/aerialb.tif
../data/sonora/aerialg.tif
../data/sonora/aerialnir.tif
../data/sonora/aerialr.tif
../data/sonora/dem.tif
../data/sonora/ep_101x101.tif
../data/sonora/ep_11x11.tif
../data/sonora/ep_201x201.tif
../data/sonora/ep_21x21.tif
../data/sonora/ep_51x51.tif
../data/sonora/ep_5x5.tif
../data/sonora/geology.tif
../data/sonora/logslope.tif
../data/sonora/logslope_10.tif
../data/sonora/logslope_100.tif
../data/sonora/logslope_20.tif
Count


../data/sonora/logslope_200.tif
../data/sonora/logslope_50.tif
../data/sonora/logstdslope_101x101.tif
../data/sonora/logstdslope_11x11.tif
../data/sonora/logstdslope_201x201.tif
../data/sonora/logstdslope_21x21.tif
../data/sonora/logstdslope_51x51.tif
../data/sonora/logstdslope_5x5.tif
../data/sonora/nhd.tif
../data/sonora/osm.tif
../data/sonora/plancurv.tif
../data/sonora/plancurv_10.tif
Count
1e6
1e7

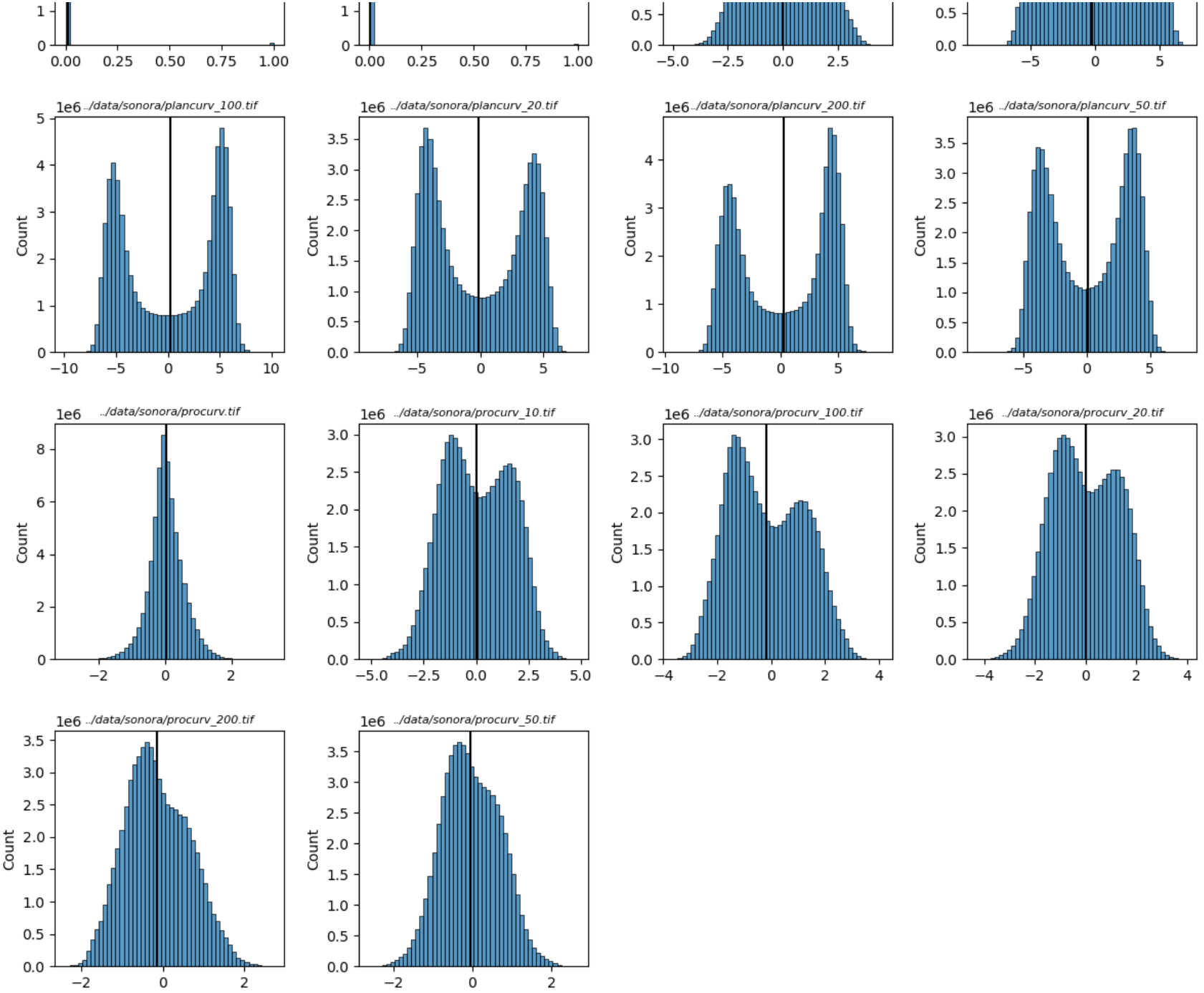

../data/sonora/plancurv_100.tif
../data/sonora/plancurv_20.tif
../data/sonora/plancurv_200.tif
../data/sonora/plancurv_50.tif
../data/sonora/procurv.tif
../data/sonora/procurv_10.tif
../data/sonora/procurv_100.tif
../data/sonora/procurv_20.tif
../data/sonora/procurv_200.tif
../data/sonora/procurv_50.tif
1e6
Count

Image Means & Standard Deviations

In [73]:
```
##################################################################
# Get Mean & Standard Deviations of Images for Normalization
##################################################################

# paths to images
image_paths = glob.glob(r'../data/sonora/*.tif')
image_paths = [path for path in image_paths if not 'mdhs' in path]
image_paths.sort(key=lambda x: x.lower())

# path to save statistics as .csv
stats_output_path = r'../data/sonora/image_stats.csv'


##### calculate mean and standard deviation for each image
# initialize lists to hold stats
means = []
sds = []

# iterate through each image, calculate stats, and append to lists
for path in image_paths:
    with rasterio.open(path) as src:
        data = src.read(1, masked=True)        # read image data array
        valid_data = data[~data.mask].data     # get valid data only (masked array)
        means.append(np.mean(valid_data))
        sds.append(np.std(valid_data))


##### save image stats as .csv with patches
df_stats = pd.DataFrame({'path': [os.path.basename(x) for x in image_paths], 'mean': means, 'std': sds})
df_stats.to_csv(stats_output_path, index=False)
```

Image Patches

Create Patches

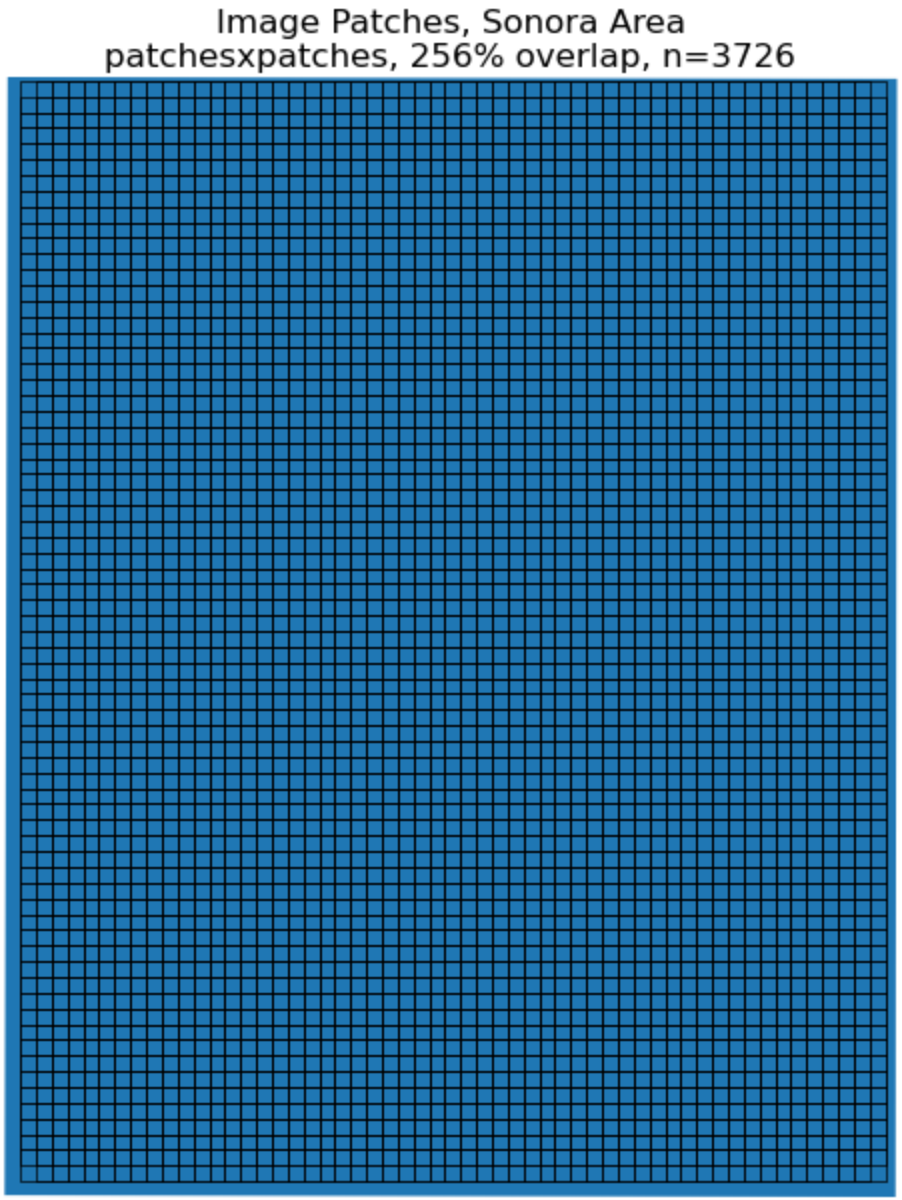
In [74]:
```
##############################
# Create GIS Patch Polygons
##############################
```

```
# path to reference image
reference_path = r'../data/sonora/geology.tif'

# define patch size (in pixels)
patch_size = 256

# proportion of patch overlap
overlap = 0.5

# path to Sonora dataset boundary (not buffered boundary)
boundary_path = r'../data/sonora/boundary.geojson'

# output path of patches GeoJSON
output_path = f"../data/sonora/sonora_patches_{patch_size}_{int(overlap*100)}.geojson"

# prefix for unique patch id
name_prefix = 'sonora'


# NOTE: Update patch id code to account for column & row - so can exclude certain columns/rows. For example


##### create patches geodataframe
create_image_patches(reference_path, patch_size, overlap, boundary_path, output_path, name_prefix)
```

In [75]:

```
####################
# Visualize patches
####################

# path to patches geojson
patches_path = glob.glob(r'../data/sonora/*patches*.geojson')[0]

# path to boundary (non-buffered)
boundary_path = r'../data/sonora/boundary.geojson'


##### plot patches overlying target area
# read patches and boundary GeoJSONs
patches = gpd.read_file(patches_path)
boundary = gpd.read_file(boundary_path)
```

```
# plot figure
fig, ax = plt.subplots(figsize=(12,8))
boundary.plot(ax=ax)
patches.plot(ax=ax, edgecolor='k', facecolor='none', linewidth=0.5)
ax.set_axis_off()

# get patch data for title
base = os.path.basename(patches_path)
base = os.path.splitext(base)[0]
base = base.split(sep='_')
size = base[1]
overlap = base[2]
ax.set_title(f"Image Patches, Sonora Area\n{size}x{size}, {overlap}% overlap, n={len(patches)}", y=0.95)
plt.show()
```



Encode Patch Features

In [76]:

```
################################################################
# Get Labels and Proportions of Map Unit Areas in Each Patch
################################################################

# path to patches GeoJSON
patches_path = glob.glob(r'../data/sonora/*patches*.geojson')[0]

# path to geology GeoJSON
geology_path = r'../data/sonora/geology.geojson'


##### read data as dataframes
patches = gpd.read_file(patches_path)     # patch polygons
geology = gpd.read_file(geology_path)     # geological map unit polygons


##### spatial overlay & get areas
# spatial overlay of geologic map units intersecting each patch
overlay = gpd.overlay(geology, patches, how='intersection')

# calculate area of each geologic map unit in each area
overlay['area_in_patch'] = overlay.geometry.area

# iterate through each patch
for patch in overlay['patch_id'].unique():

    # get array of geologic map unit areas
    geology_areas = overlay.loc[overlay['patch_id'] == patch, 'area_in_patch'].values

    # get total area covered by map units in each patch
    total_area = overlay.loc[overlay['patch_id'] == patch, 'area_in_patch'].sum(axis=0)

    # calculate proportions of each map unit in each patch
    overlay.loc[overlay['patch_id'] == patch, 'area_in_patch'] = geology_areas / total_area

    # convert to float32 for consistency with all other data
    overlay['area_in_patch'] = overlay['area_in_patch'].astype(np.float32)


##### create new dataframe for patches (rows) and map unit areas (columns)
initialize_dict = {'patch_id': patches['patch_id']}
df_areas = pd.DataFrame(initialize_dict)
```

```
df_areas[['af1', 'Qal', 'Qaf', 'Qat', 'Qc', 'Qca', 'Qr']] = 0
df_areas[['af1', 'Qal', 'Qaf', 'Qat', 'Qc', 'Qca', 'Qr']] = df_areas[['af1', 'Qal', 'Qaf', 'Qat', 'Qc', 'Q
# for symbol in geology['Symbol'].unique():
#     # df_areas[symbol] = 0
#     df_areas[symbol] = df_areas[symbol].astype(np.float32)


##### group overlay and insert areas into df
# group overlay gdf by patch and geologic map unit type
# NOTE: there can be multiple map units of same type within each patch, so need to also group by Symbol fo
grouped = overlay.groupby(['patch_id', 'Symbol']).agg({'area_in_patch':'sum'})

# iterate through grouped dataframe
for (patch, symbol), row in grouped.iterrows():

    # get area of unique map unit in specific patch
    area = row['area_in_patch'].item()

    # insert into dataframe
    df_areas.loc[df_areas['patch_id']==patch, symbol] = area


##### get binary label dataframe
labels = df_areas.iloc[:, 1:].apply(np.ceil)
df_labels = df_areas.copy()
df_labels.iloc[:, 1:] = labels


##### get patch basename and save as .csv files
base = os.path.basename(patches_path)
base = os.path.splitext(base)[0]
df_areas.to_csv(f"../data/sonora/{base}_areas.csv", index=False)
df_labels.to_csv(f"../data/sonora/{base}_labels.csv", index=False)
```

Extract Patches

In [77]:

```
############################
# Extract GeoTIFF Patches
############################
```

```
# list of paths to images
image_paths = glob.glob(r'../data/sonora/*.tif')
image_paths = [path for path in image_paths if not 'mdhs' in path]
image_paths.sort(key=lambda x: x.lower())

# path to patch polygons
patches_path = glob.glob(r'../data/sonora/*patches*.geojson')[0]

# path to patch labels
label_path = glob.glob(r'../data/sonora/*labels.csv')[0]

# directory path for patches
patches_dir = r'../data/patches_sonora'
if not os.path.isdir(patches_dir):
    os.makedirs(patches_dir)



##### get patch polygon geodataframe
patches = gpd.read_file(patches_path)


##### get labels as dataframe
labels = pd.read_csv(label_path)


##### extract selected patches
for path in image_paths:
    extract_patch(path, patches, patches_dir)


##### save csv of all patch_ids of extracted patches
labels['patch_id'].to_csv(f"{patches_dir}/patch_ids.csv", header=False, index=False)


##### save individual label for each patch
for _, row in labels.iterrows():
    patch_name = row['patch_id']
    data = row.iloc[1:]
    data.to_csv(f"{patches_dir}/{patch_name}_labels.csv", header=False, index=False)
```

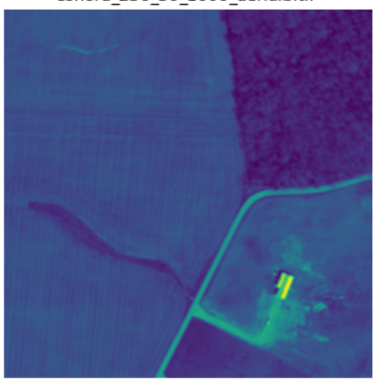
In [78]:
```
############################
# Validate Size of Patches
############################

# paths of patches
patches_paths = glob.glob(r'../data/patches_sonora/*.tif')

# desired size of patches
size = 256


##### iterate through each patch and check height and width
bad_patches = []                                    # initialize list for patches of incorrect size
for path in patches_paths:                          # iterate through patches
    with rasterio.open(path) as src:                # read patch GeoTIFF
        height = src.height                         # get height
        width = src.width                           # get width
        if (height != size) or (width != size):     # test size
            bad_patches.append(path)                # append incorreectly sized patches to list


##### plot incorrectly sized patches (if any)
if len(bad_patches) > 0:
    fig, ax = plt.subplots(figsize=(10,6))
    boundary = gpd.read_file(r'../data/sonora/boundary.geojson')
    for path in bad_patches:
        with rasterio.open(path) as src:
            data = src.read(1)
            binary_mask = np.where(data>0, 1, 0)
            show(binary_mask, transform=src.transform, ax=ax, cmap='gray')
    boundary.plot(ax=ax, facecolor='none', edgecolor='red', linewidth=0.25)
    ax.set_axis_off()
    ax.set_title(f"Incorrectly Sized Patches, Sonora Area\nn={len(bad_patches)}")
    plt.tight_layout()
    plt.show()
else:
    print('No incorrectly sized patches...')
```

```
No incorrectly sized patches...
```

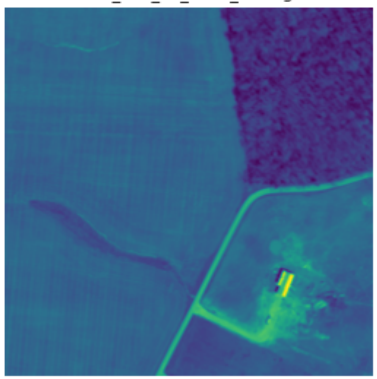
In [79]:
```
#########################
# Visualize Random Patch
```

```
#########################

# paths to cleaned & size-verified patches
patch_paths = glob.glob(r'../data/patches_sonora/*.tif')
patch_paths.sort(key=lambda x: x.lower())    # sort by name to get paired patches
patch_paths = patch_paths[:38]

##### plot first random patch
fig, ax = plt.subplots(nrows=10, ncols=4, figsize=(12,30))
fig.subplots_adjust(wspace=0.1)
ax = ax.ravel()

for idx, image in enumerate(patch_paths):
    with rasterio.open(image) as src:
        show(src, ax=ax[idx])
        ax[idx].set_axis_off()
        ax[idx].set_title(os.path.basename(image), style='italic', fontsize=8)

for unused in ax[-2:]:
    fig.delaxes(unused)

plt.tight_layout()
plt.show()
```

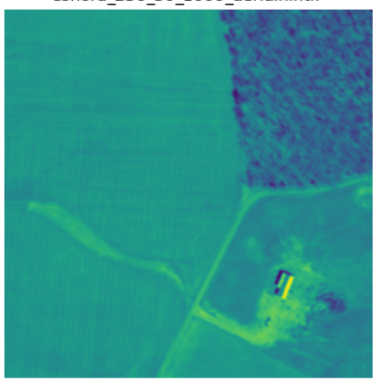
sonora_256_50_1000_aerialb.tif



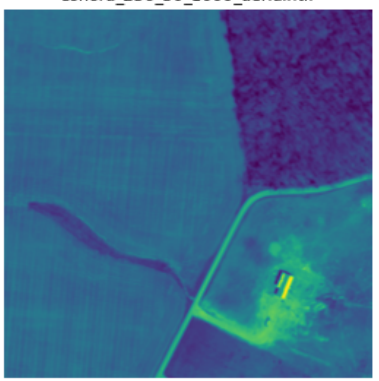
sonora_256_50_1000_aerialg.tif



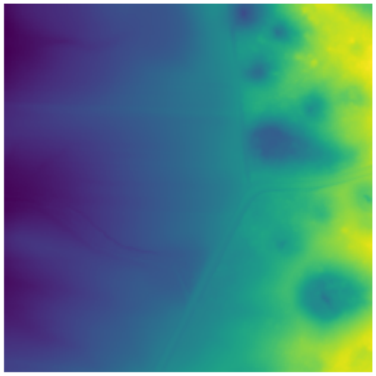
sonora_256_50_1000_aerialnir.tif



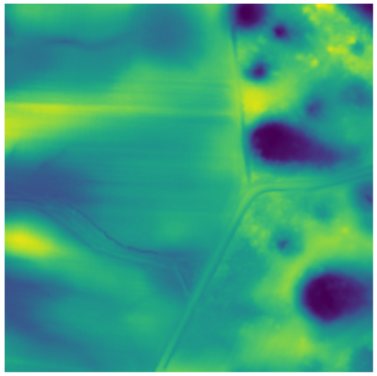
sonora_256_50_1000_aerialr.tif



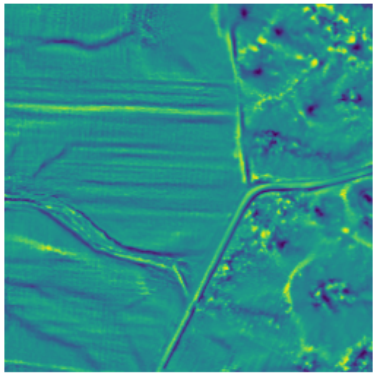
sonora_256_50_1000_dem.tif



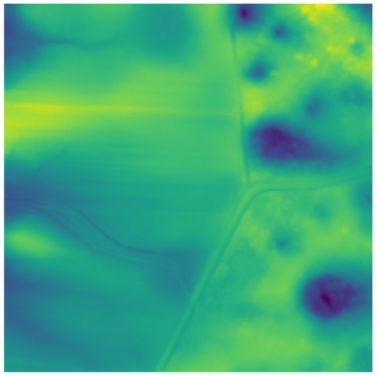
sonora_256_50_1000_ep_101x101.tif



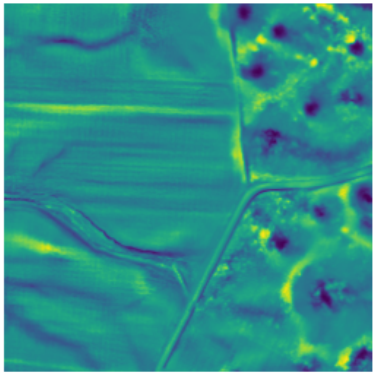
sonora_256_50_1000_ep_11x11.tif



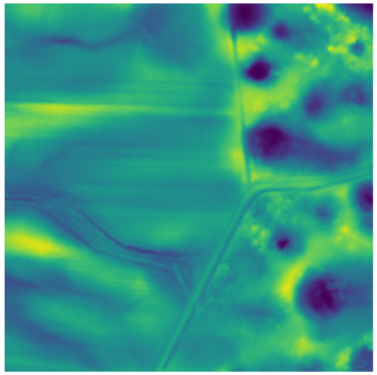
sonora_256_50_1000_ep_201x201.tif



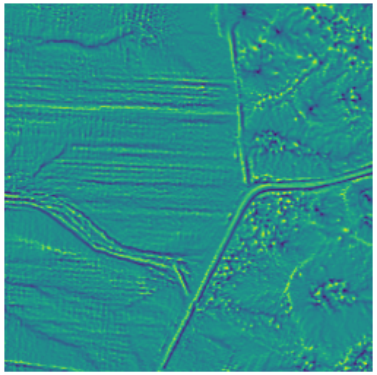
sonora_256_50_1000_ep_21x21.tif



sonora_256_50_1000_ep_51x51.tif

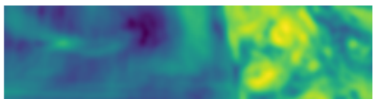


sonora_256_50_1000_ep_5x5.tif



sonora_256_50_1000_geology.tif

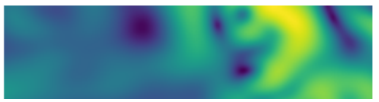
sonora_256_50_1000_logslope.tif

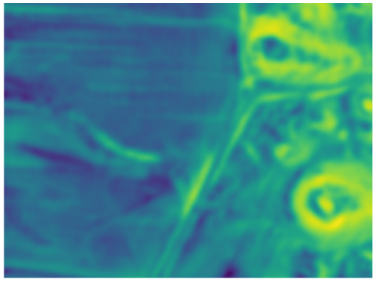


sonora_256_50_1000_logslope_10.tif

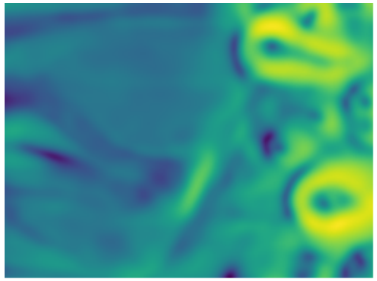
sonora_256_50_1000_logslope_100.tif

sonora_256_50_1000_logslope_20.tif

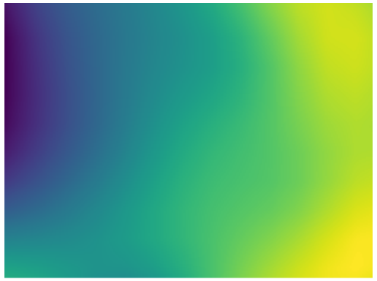




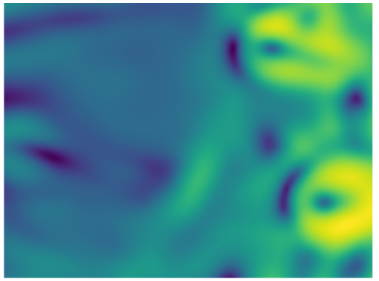
sonora_256_50_1000_logslope_200.tif



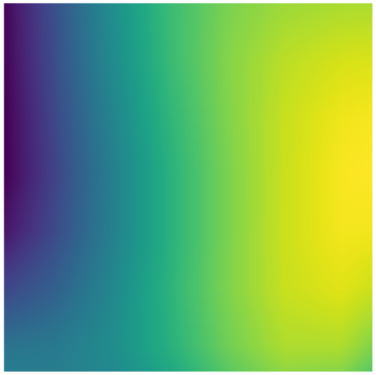
sonora_256_50_1000_logslope_50.tif



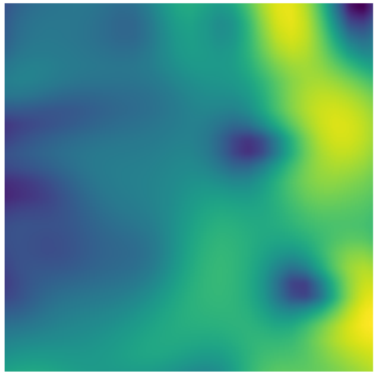
sonora_256_50_1000_logstdslope_101x101.tif



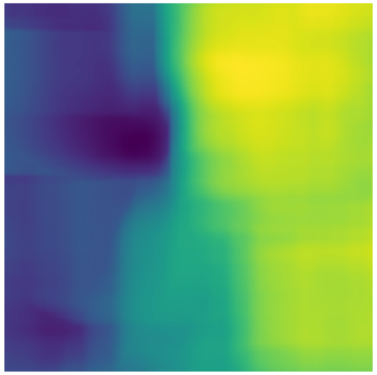
sonora_256_50_1000_logstdslope_11x11.tif



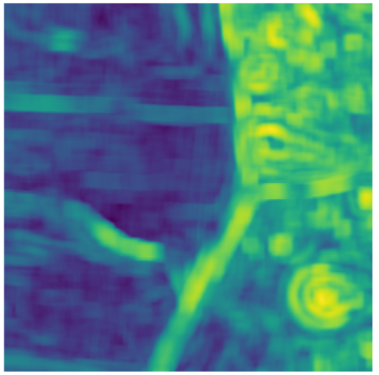
sonora_256_50_1000_logstdslope_201x201.tif



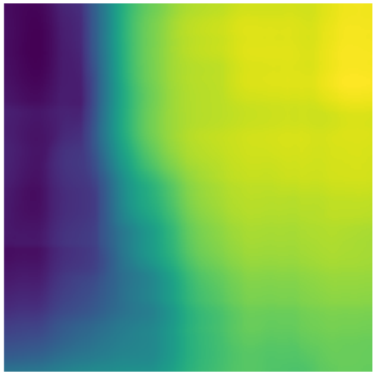
sonora_256_50_1000_logstdslope_21x21.tif



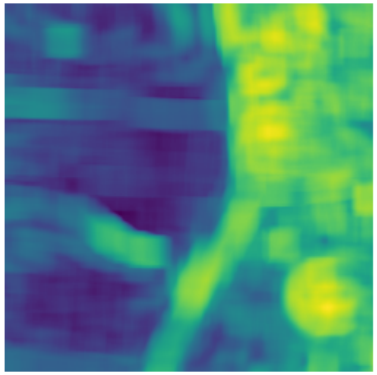
sonora_256_50_1000_logstdslope_51x51.tif



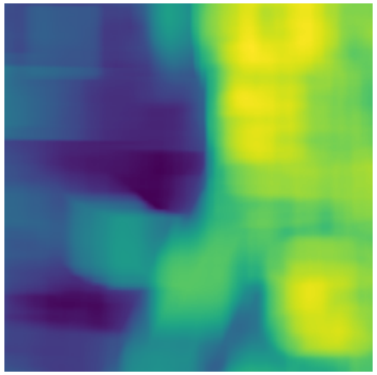
sonora_256_50_1000_logstdslope_5x5.tif



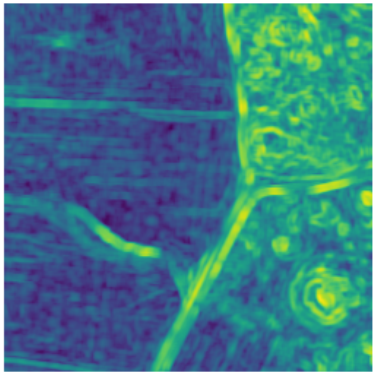
sonora_256_50_1000_nhd.tif



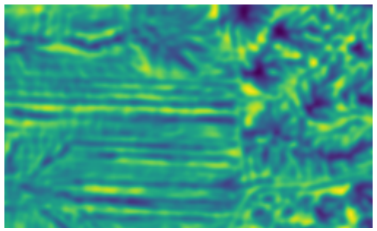
sonora_256_50_1000_osm.tif

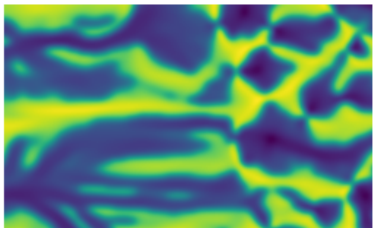


sonora_256_50_1000_plancurv.tif

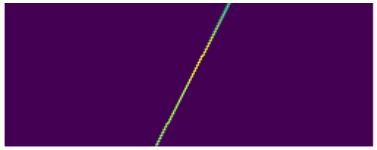


sonora_256_50_1000_plancurv_10.tif

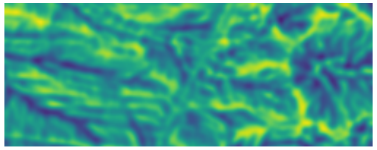




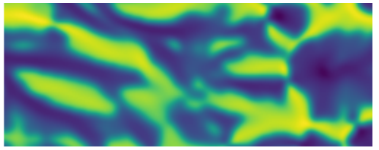
sonora_256_50_1000_plancurv_100.tif



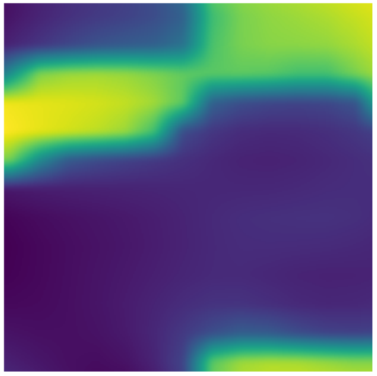
sonora_256_50_1000_plancurv_20.tif



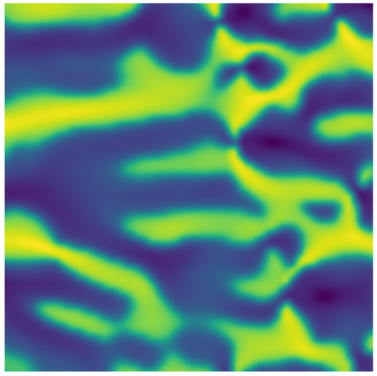
sonora_256_50_1000_plancurv_200.tif



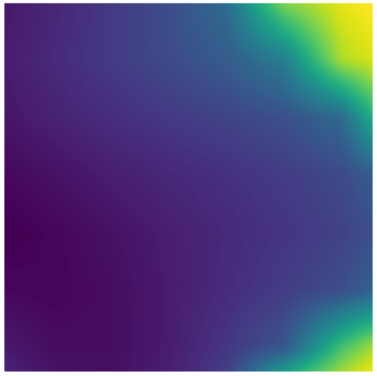
sonora_256_50_1000_plancurv_50.tif



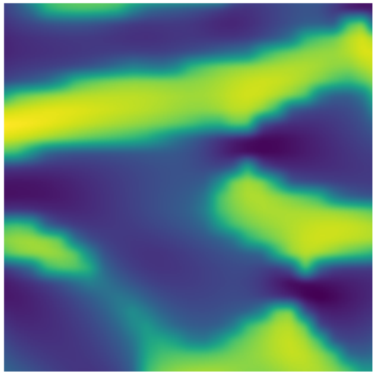
sonora_256_50_1000_procurv.tif



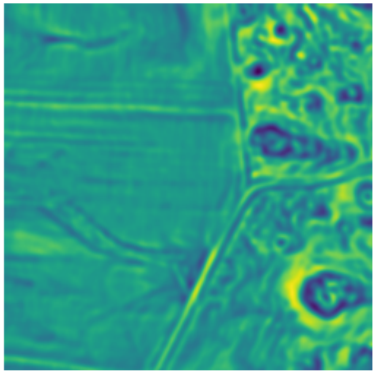
sonora_256_50_1000_procurv_10.tif



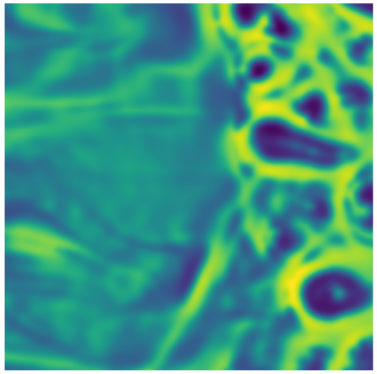
sonora_256_50_1000_procurv_100.tif



sonora_256_50_1000_procurv_20.tif



sonora_256_50_1000_procurv_200.tif



sonora_256_50_1000_procurv_50.tif