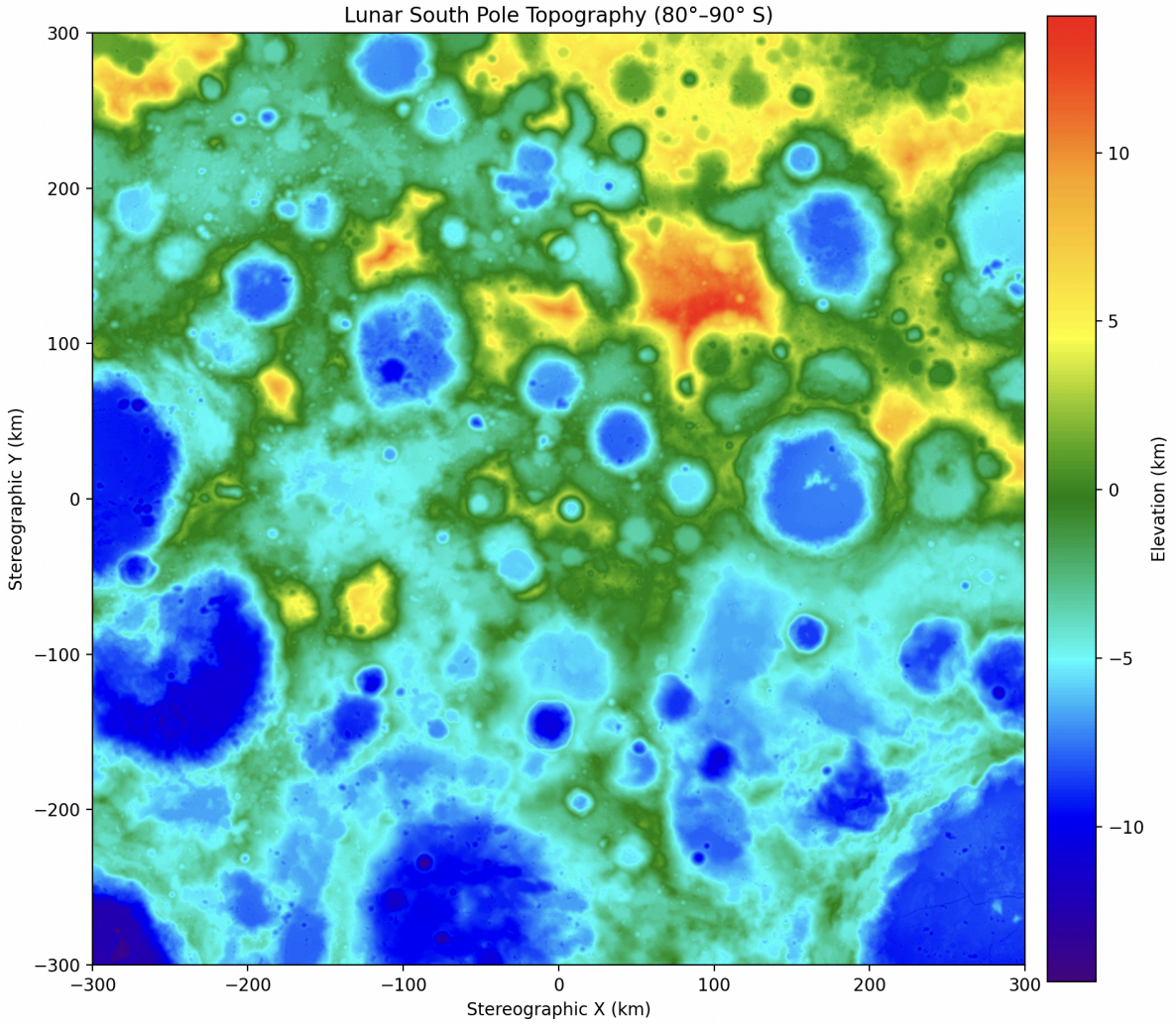
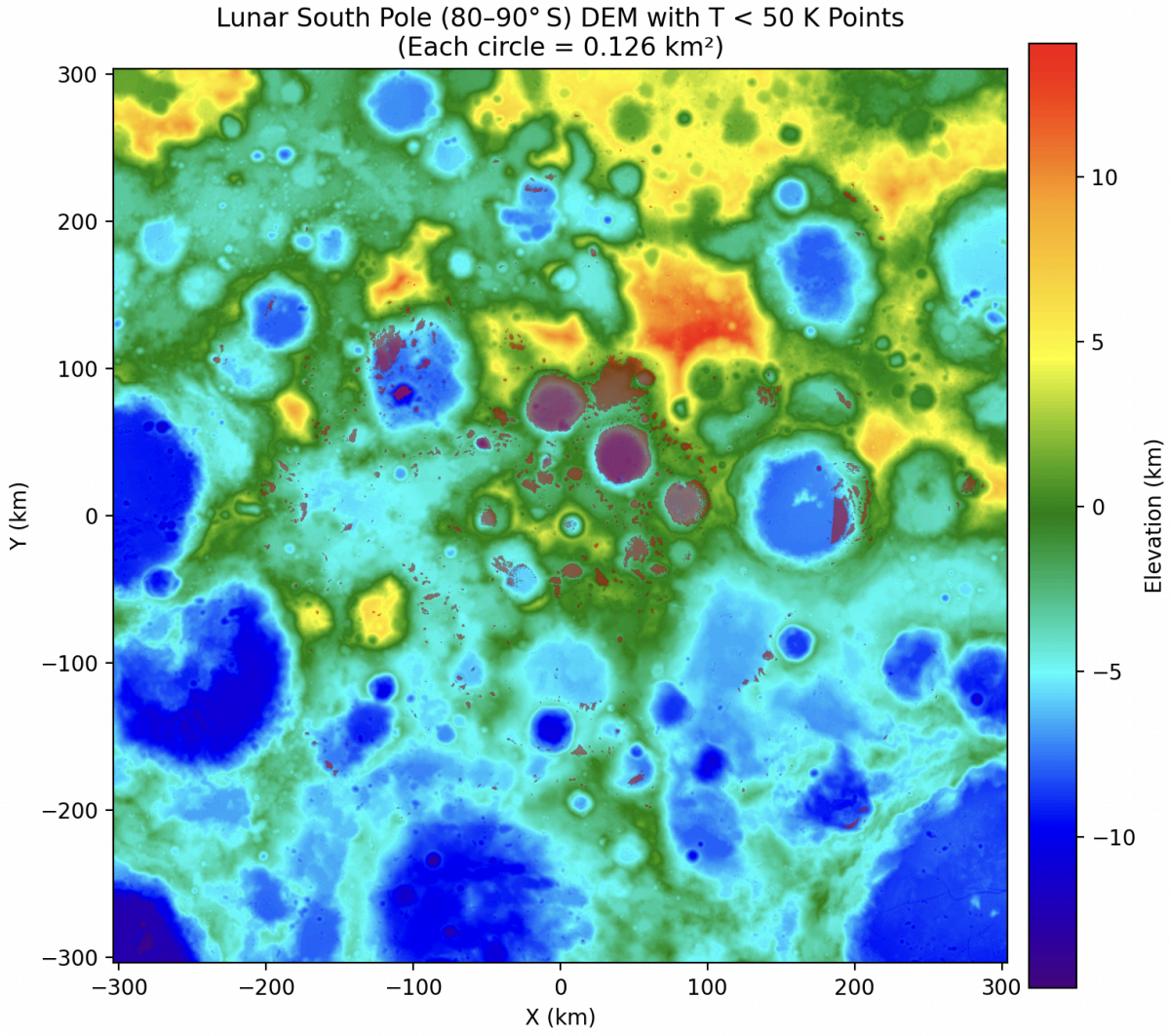
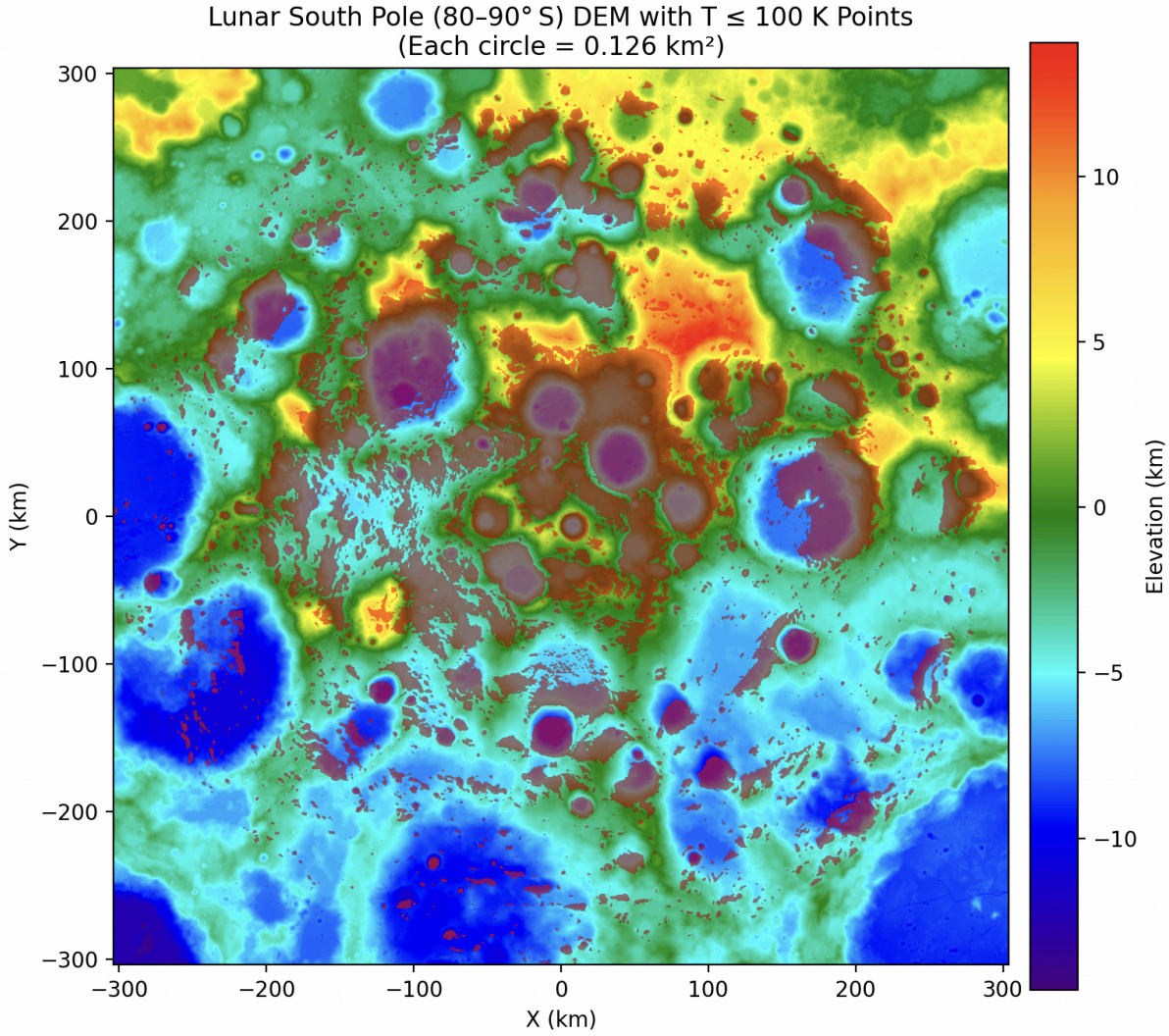
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap

# 1) Load just the topography .npz
topo = np.load("topography_80S.npz")
dem_data = topo["dem"]
x_coords = topo["x"]
y_coords = topo["y"]

# 2) Build a ROYGBIV colormap for elevation
roygbiv = ["indigo", "blue", "cyan", "green", "yellow", "orange", "red"]
roygbiv_cmap = LinearSegmentedColormap.from_list("roygbiv", roygbiv, N=256)

# 3) Plot only the DEM
fig, ax = plt.subplots(figsize=(10, 10))
topo_plot = ax.pcolormesh(
    x_coords, y_coords, dem_data,
    cmap=roygbiv_cmap,
    shading="auto"
)

# 4) Colorbar & labels
cbar = plt.colorbar(topo_plot, ax=ax, shrink=0.8, pad=0.02)
cbar.set_label("Elevation (km)")

ax.set_xlim(-300, 300)
ax.set_ylim(-300, 300)
ax.set_aspect("equal")
ax.set_title("Lunar South Pole Topography (80°–90° S)")
ax.set_xlabel("Stereographic X (km)")
ax.set_ylabel("Stereographic Y (km)")

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


Lunar South Pole Topography (80°–90° S)
Stereographic Y (km)
Stereographic X (km)
Elevation (km)
300
200
100
0
−100
−200
−300
10
5
0
−5
−10

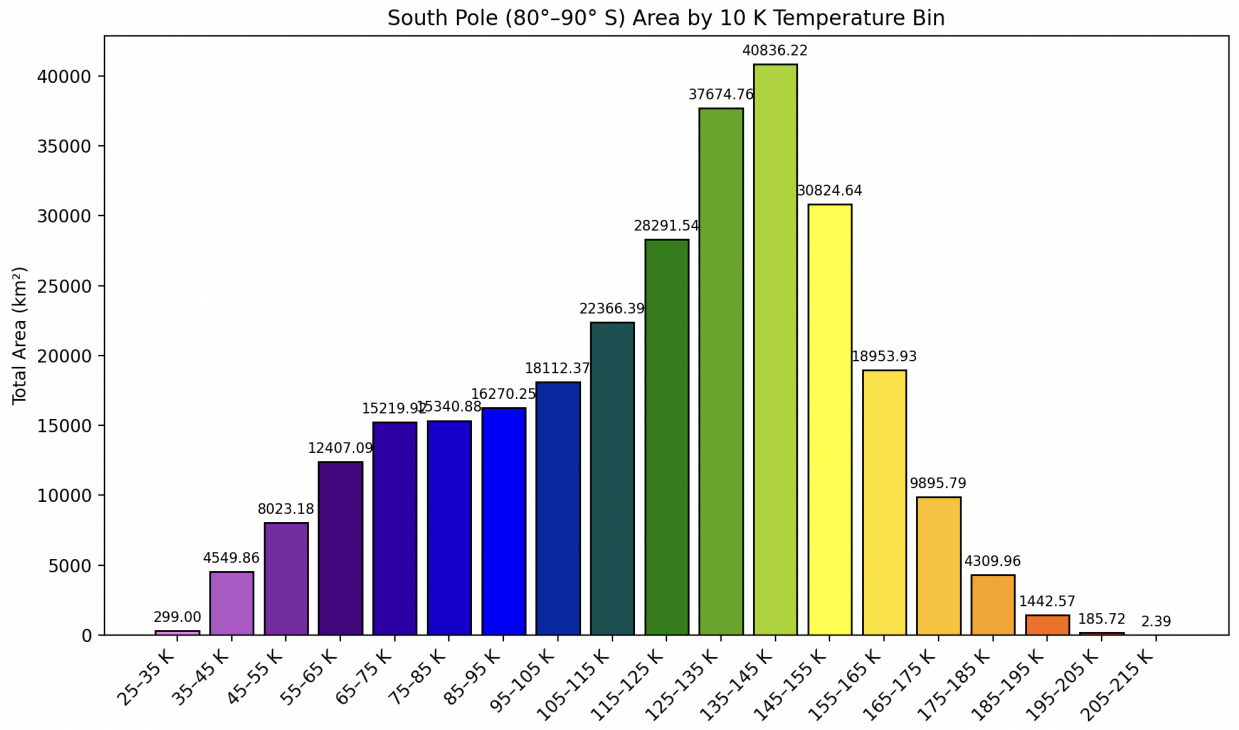
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.patches import Circle

# 1) Load the South-Pole DEM (80–90° S)
topo = np.load("topography_80S.npz")
dem = topo["dem"]   # 2D elevation array
x   = topo["x"]     # 1D X coords (km)
y   = topo["y"]     # 1D Y coords (km)

# 2) Load the DLRE temperature centroids
dlre   = np.load("dlre_temp_80S.npz")
x_temp = dlre["x"]     # km
y_temp = dlre["y"]     # km
t_avg  = dlre["temp"]  # K

# 3) Filter to only points below 50 K
mask50 = t_avg < 50.0
x50, y50 = x_temp[mask50], y_temp[mask50]

# 4) Compute true-area circle radius for 0.126 km²
area_pt = 0.126                  # km²
r_km    = np.sqrt(area_pt/np.pi) # km

# 5) Build a ROYGBIV colormap for the DEM
roygbiv = ["indigo","blue","cyan","green","yellow","orange","red"]
cmap_dem = LinearSegmentedColormap.from_list("roygbiv", roygbiv, N=256)

# 6) Plot the DEM
fig, ax = plt.subplots(figsize=(8,8))
pcm = ax.pcolormesh(
   x, y, dem,
   cmap=cmap_dem,
   shading="auto"
)
cbar = fig.colorbar(pcm, ax=ax, shrink=0.8, pad=0.02)
cbar.set_label("Elevation (km)")

# 7) Overlay T<50 K points as red true-area circles
for xt, yt in zip(x50, y50):
   circ = Circle(
```

```
        (xt, yt), r_km,
        facecolor="red", edgecolor=None, alpha=0.7,
        transform=ax.transData, zorder=2
    )
    ax.add_patch(circ)

# 8) Formatting
ax.set_aspect("equal", "box")
ax.set_xlim(x.min(), x.max())
ax.set_ylim(y.min(), y.max())
ax.set_xlabel("X (km)")
ax.set_ylabel("Y (km)")
ax.set_title("Lunar South Pole (80–90° S) DEM with T < 50 K Points\n(Each circle =
0.126 km²)")

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


Lunar South Pole (80–90° S) DEM with T < 50 K Points
(Each circle = 0.126 km²)
Y (km)
X (km)
Elevation (km)
300
200
100
0
−100
−200
−300
10
5
0
−5
−10


Lunar South Pole (80–90° S) DEM with T ≤ 100 K Points
(Each circle = 0.126 km²)
Y (km)
X (km)
Elevation (km)
300
200
100
0
−100
−200
−300
10
5
0
−5
−10

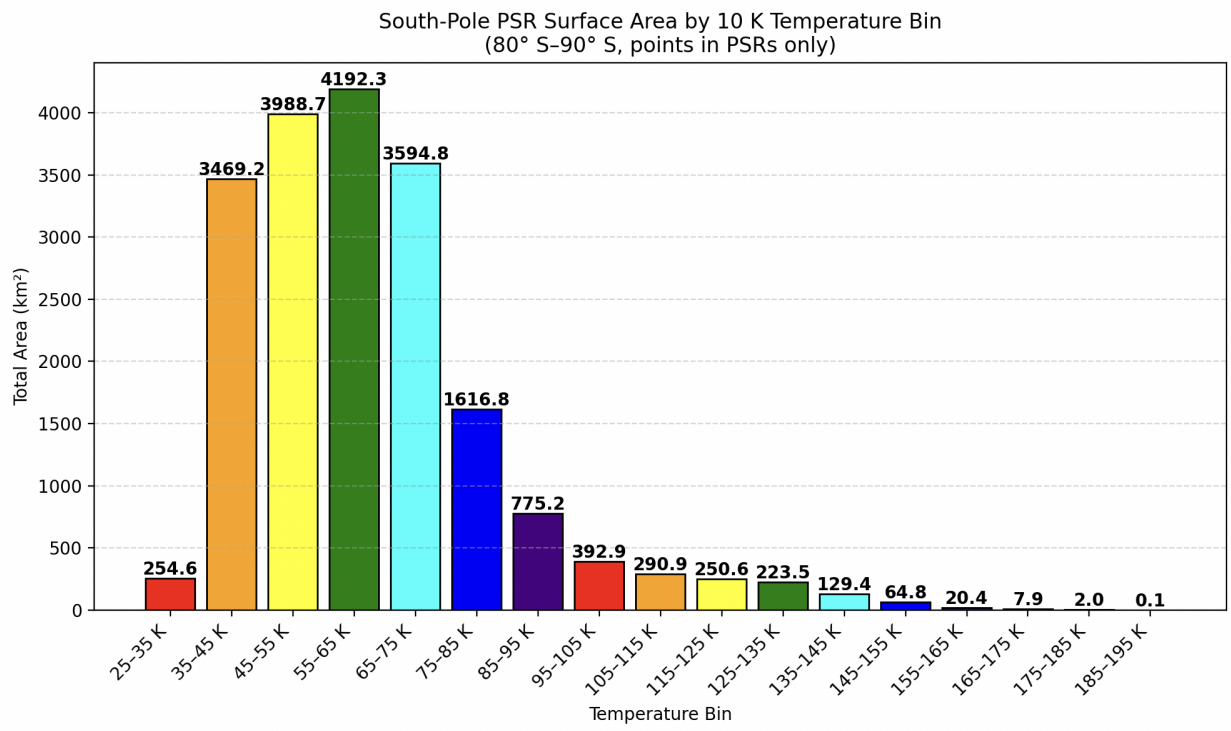
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap

# 1) Load DLRE south-pole temperature data (80°–90° S)
data = np.load("dlre_temp_80S.npz")
t_avg = data["temp"]  # in Kelvin

# 2) Build 10 K bins from 25 K up to the next multiple above max
step = 10
min_edge = 25
max_edge = int(np.ceil(t_avg.max() / step) * step)
bin_edges = np.arange(min_edge, max_edge + step, step)  # e.g.
[25,35,45,…,max_edge+10]

# 3) Count points per bin and convert to area
counts, _ = np.histogram(t_avg, bins=bin_edges)
area_per_point = 0.126  # km² footprint per Diviner sample
areas = counts * area_per_point

# 4) Prepare ROYGBIV colors stretched to number of bins
roygbiv = ["violet","indigo","blue","green","yellow","orange","red"]
cmap = LinearSegmentedColormap.from_list("roygbiv", roygbiv, N=len(areas))
colors = [cmap(i) for i in range(len(areas))]

# 5) Plot
labels = [f"{bin_edges[i]}–{bin_edges[i+1]} K" for i in range(len(bin_edges)-1)]
x = np.arange(len(areas))

fig, ax = plt.subplots(figsize=(10,6))
bars = ax.bar(x, areas, color=colors, edgecolor="black")
for xi, a in zip(x, areas):
   ax.text(xi, a + max(areas)*0.01, f"{a:.2f}", ha="center", va="bottom", fontsize=8)

ax.set_xticks(x)
ax.set_xticklabels(labels, rotation=45, ha="right")
ax.set_ylabel("Total Area (km²)")
ax.set_title("South Pole (80°–90° S) Area by 10 K Temperature Bin")
plt.tight_layout()
plt.savefig("south_pole_temp_area_breakdown.png", dpi=300)
plt.show()
```




```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import rasterio
from scipy import ndimage

# ---------------------------------------------------------------------------
# 1) Load the PSR raster and label connected PSR regions
# ---------------------------------------------------------------------------
PSR_TIF = "LPSR_75S_120M_201608.tiff"
with rasterio.open(PSR_TIF) as src:
    psr_mask      = src.read(1)
    psr_transform = src.transform

# binary mask >0 means PSR
labeled_array, _ = ndimage.label(psr_mask > 0)

# ---------------------------------------------------------------------------
# 2) Load the temperature point cloud from DLRE
# ---------------------------------------------------------------------------
dlre   = np.load("dlre_temp_80S.npz")
x_temp = dlre["x"]    # x (km) in polar stereographic
y_temp = dlre["y"]    # y (km)
```

```
t_avg  = dlre["temp"] # average temp (K)

# ──────────────────────────────────────────────────────────────────────────────
# 3) Map each DLRE point into the PSR raster (fast pixel lookup)
# ──────────────────────────────────────────────────────────────────────────────
# Convert km → m
x_m = x_temp * 1000
y_m = y_temp * 1000

# pixel-fractional coordinates
col_f = (x_m - psr_transform.c) / psr_transform.a
row_f = (y_m - psr_transform.f) / psr_transform.e

# integer pixel indices
col = np.floor(col_f).astype(int)
row = np.floor(row_f).astype(int)

# mask out-of-bounds
valid = (
   (row >= 0) & (row < labeled_array.shape[0]) &
   (col >= 0) & (col < labeled_array.shape[1])
)

pt_psr = np.zeros_like(t_avg, dtype=int)
pt_psr[valid] = labeled_array[row[valid], col[valid]]

# ──────────────────────────────────────────────────────────────────────────────
# 4) Select only points inside any PSR (i.e. pt_psr > 0)
# ──────────────────────────────────────────────────────────────────────────────
in_psr_mask = pt_psr > 0
print(f"Total temperature points inside PSRs: {in_psr_mask.sum()}")

# ──────────────────────────────────────────────────────────────────────────────
# 5) Define temperature bins [25-35), [35-45), … up to highest K
# ──────────────────────────────────────────────────────────────────────────────
t_min = 25
t_max = int(np.ceil(t_avg[in_psr_mask].max() / 10)) * 10
bins  = np.arange(t_min, t_max + 10, 10)
labels = [f"{bins[i]}-{bins[i+1]} K" for i in range(len(bins)-1)]

# ROYGBIV palette for the bins
roygbiv = ["red","orange","yellow","green","cyan","blue","indigo"]
```

```
colors = roygbiv[: len(labels)]

# ──────────────────────────────────────────────────────────────────────────────
# 6) Compute area per bin (each point = 0.126 km²)
# ──────────────────────────────────────────────────────────────────────────────
AREA_PER_POINT = 0.126  # km²

areas = []
for lo, hi in zip(bins[:-1], bins[1:]):
   count = np.sum((t_avg >= lo) & (t_avg < hi) & in_psr_mask)
   areas.append(count * AREA_PER_POINT)

# ──────────────────────────────────────────────────────────────────────────────
# 7) Plot the bar chart
# ──────────────────────────────────────────────────────────────────────────────
fig, ax = plt.subplots(figsize=(10, 6))

bars = ax.bar(labels, areas, color=colors, edgecolor="black")

# annotate area atop each bar
for bar, area in zip(bars, areas):
   ax.text(
       bar.get_x() + bar.get_width() / 2,
       area + 0.2,  # slight vertical offset
       f"{area:.1f}",
       ha="center", va="bottom", fontsize=10, fontweight="bold"
   )

ax.set_title("South-Pole PSR Surface Area by 10 K Temperature Bin\n(80° S–90° S,
points in PSRs only)")
ax.set_ylabel("Total Area (km²)")
ax.set_xlabel("Temperature Bin")
ax.set_xticklabels(labels, rotation=45, ha="right")
ax.grid(axis="y", linestyle="--", alpha=0.5)

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



Total PSR area within DLRE footprint: 20991.00 km²

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.path import Path
from matplotlib.patches import Circle
from matplotlib.lines import Line2D
from skimage.measure import find_contours
import rasterio
from scipy import ndimage

# ------------------------------------------------------------------------------
# 1) Load the filtered PSR metadata, pick top 10 + 13th + 20th by area
# ------------------------------------------------------------------------------
meta = pd.read_csv("valid_psrs_cold_stable_updated.csv")
meta = meta.sort_values("area_km2", ascending=False).reset_index(drop=True)
pick_idxs = list(range(10)) + [12, 19]                    # 0-9, plus 12th & 19th
indices
pick_idxs = [i for i in pick_idxs if i < len(meta)]      # guard in case fewer rows
top_ids = meta.loc[pick_idxs, "psr_id"].astype(int).tolist()
```

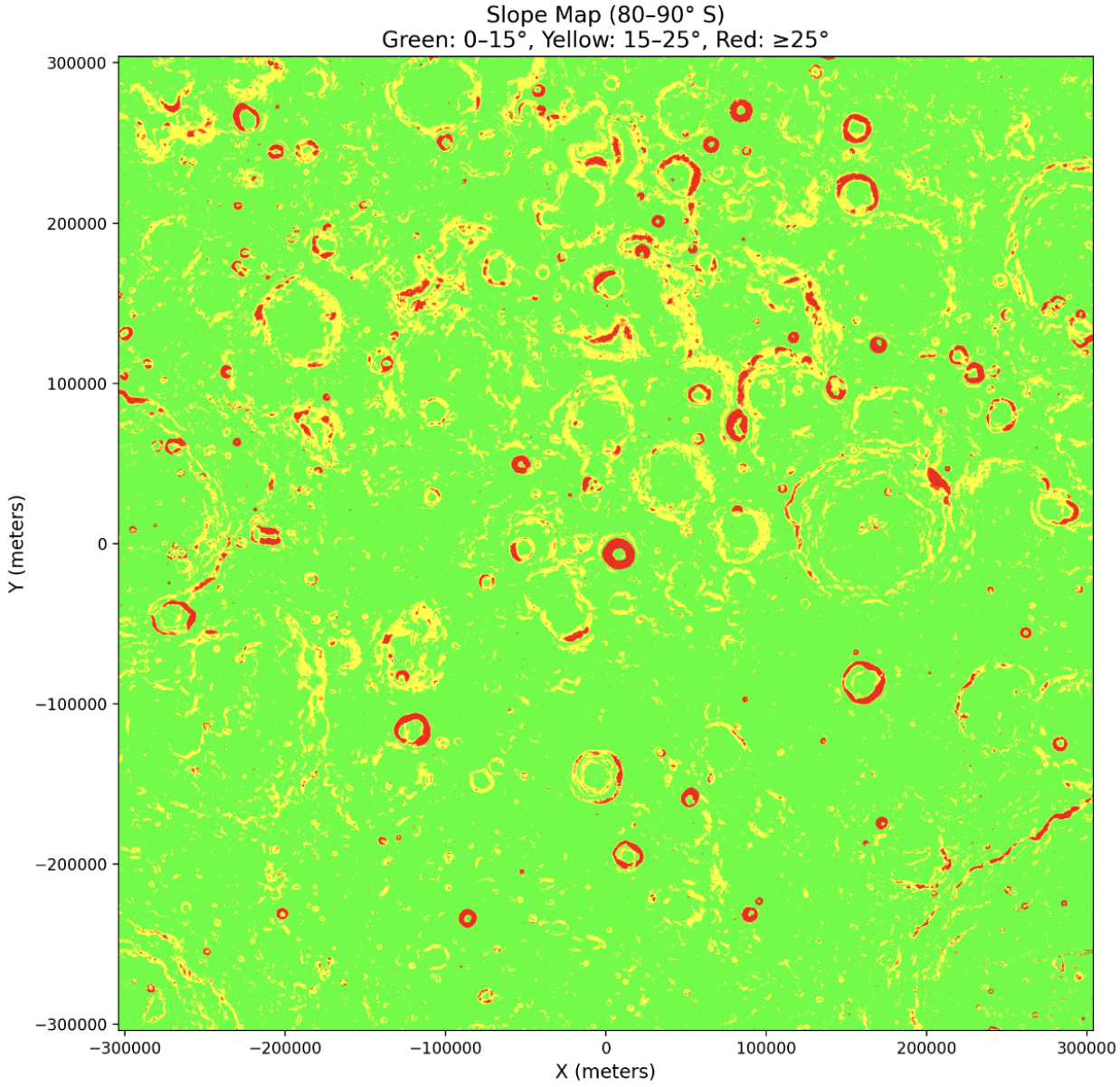
```
print("Zooming on PSR IDs:", top_ids)

# ------------------------------------------------------------------------
# 2) Load DEM + make ROYGBIV colormap
# ------------------------------------------------------------------------
topo = np.load("topography_80S.npz")
dem, x, y = topo["dem"], topo["x"], topo["y"]
roygbiv = ["indigo","blue","cyan","green","yellow","orange","red"]
cmap_dem = LinearSegmentedColormap.from_list("roygbiv", roygbiv, N=256)

# ------------------------------------------------------------------------
# 3) Load & label PSR raster
# ------------------------------------------------------------------------
with rasterio.open("LPSR_75S_120M_201608.tiff") as src:
   psr_mask, transform = src.read(1), src.transform
labeled_array, _ = ndimage.label(psr_mask > 0)

# ------------------------------------------------------------------------
# 4) Load DLRE temperatures
# ------------------------------------------------------------------------
dlre    = np.load("dlre_temp_80S.npz")
x_temp  = dlre["x"]     # km
y_temp  = dlre["y"]     # km
t_avg   = dlre["temp"]  # K

# ------------------------------------------------------------------------
# 5) Define our two bins and colors
# ------------------------------------------------------------------------
bins       = [25, 35, 45]                # [25–35), [35–45)
bin_labels = [f"{bins[i]}–{bins[i+1]} K" for i in range(len(bins)-1)]
colors     = ["red", "orange"]

# ------------------------------------------------------------------------
# 6) True-area radius in km for 0.126 km²
# ------------------------------------------------------------------------
area_pt = 0.126     # each circle covers 0.126 km²
r_km    = np.sqrt(area_pt / np.pi)

# ------------------------------------------------------------------------
# 7) Compute and save bounding boxes for each PSR in top_ids
# ------------------------------------------------------------------------
bounds = []
```

```
for psr_id in top_ids:
    mask = (labeled_array == psr_id)
    if not mask.any():
        continue
    contours = find_contours(mask.astype(float), 0.5)
    if not contours:
        continue
    c = max(contours, key=len)
    rows, cols = c[:,0], c[:,1]
    xs_m = transform.c + cols * transform.a
    ys_m = transform.f + rows * transform.e
    xs, ys = xs_m/1000, ys_m/1000
    bounds.append({
        "psr_id": psr_id,
        "xmin": xs.min(), "xmax": xs.max(),
        "ymin": ys.min(), "ymax": ys.max()
    })

df_bounds = pd.DataFrame(bounds)
df_bounds.to_csv("top12_psr_bounds.csv", index=False)
print("Wrote top12_psr_bounds.csv")

# ------------------------------------------------------------------------
# 8) Zoom & plot each PSR individually
# ------------------------------------------------------------------------
for _, row in df_bounds.iterrows():
    psr_id = int(row.psr_id)
    xmin, xmax = row.xmin - 5, row.xmax + 5
    ymin, ymax = row.ymin - 5, row.ymax + 5

    fig, ax = plt.subplots(figsize=(6,6))
    ax.pcolormesh(x, y, dem, cmap=cmap_dem, shading="auto")
    ax.set_aspect("equal", "box")
    ax.set_xlim(xmin, xmax)
    ax.set_ylim(ymin, ymax)
    ax.set_title(f"PSR {psr_id}: 25-45 K only", fontsize=12)
    ax.set_xlabel("X (km)")
    ax.set_ylabel("Y (km)")

    # draw PSR boundary
    contour = max(find_contours((labeled_array==psr_id).astype(float), 0.5), key=len)
    rows, cols = contour[:,0], contour[:,1]
```

```
    xs = (transform.c + cols*transform.a)/1000
    ys = (transform.f + rows*transform.e)/1000
    ax.plot(xs, ys, color="white", linewidth=1.5)

    # path to select only in-PSR points
    path = Path(np.column_stack((xs, ys)))
    pts = np.column_stack((x_temp, y_temp))

    # plot each temp-bin as true-area circles
    for lo, hi, c in zip(bins[:-1], bins[1:], colors):
        sel = path.contains_points(pts) & (t_avg >= lo) & (t_avg < hi)
        for xt, yt in pts[sel]:
            circ = Circle((xt, yt), r_km,
                          facecolor=c, edgecolor=None, alpha=0.8,
                          transform=ax.transData, zorder=2)
            ax.add_patch(circ)

    # custom legend with bigger markers
    legend_handles = [
        Line2D([0],[0], marker='o', color='w', label=lab,
               markerfacecolor=c, markersize=10)
        for c,lab in zip(colors, bin_labels)
    ]
    ax.legend(handles=legend_handles, title="Temp bins", loc="upper right")

    plt.tight_layout()
    outname = f"PSR_{psr_id}_25-45K_zoom.png"
    fig.savefig(outname, dpi=300)
    plt.close(fig)
    print("Saved", outname)
```

****** top 10 plus 2 blow ups*******

```python
import numpy as np
import matplotlib.pyplot as plt
import rasterio

# --- Step 1: Open the slope map COG ---
slope_file = "LDSM_80S_80MPP_ADJ.tiff"  # Path to your slope GeoTIFF
with rasterio.open(slope_file) as src:
   slope_data = src.read(1).astype(float)  # degrees
   bounds     = src.bounds

# --- Step 2: Define Color Mapping Based on Slope Values ---
output_rgb = np.zeros((slope_data.shape[0], slope_data.shape[1], 3), dtype=float)

# New masks:
mask_green  = (slope_data >= 0)  & (slope_data < 15)  # 0–15°
mask_yellow = (slope_data >= 15) & (slope_data < 25)  # 15–25°
mask_red    = (slope_data >= 25)                      # ≥25°

# Assign colors:
output_rgb[mask_green]  = [0, 1, 0]  # Green
output_rgb[mask_yellow] = [1, 1, 0]  # Yellow
output_rgb[mask_red]    = [1, 0, 0]  # Red

# --- Step 3: Plot the Composite Slope Map ---
extent = [bounds.left, bounds.right, bounds.bottom, bounds.top]

plt.figure(figsize=(10, 10))
plt.imshow(output_rgb, extent=extent, origin="upper", aspect="equal")
plt.title("Slope Map (80–90° S)\nGreen: 0–15°, Yellow: 15–25°, Red: ≥25°",
fontsize=14)
plt.xlabel("X (meters)", fontsize=12)
plt.ylabel("Y (meters)", fontsize=12)
plt.tight_layout()
plt.show()
```


Slope Map (80–90° S)
Green: 0–15°, Yellow: 15–25°, Red: ≥25°
300000
200000
100000
0
−100000
−200000
−300000
Y (meters)
−300000
−200000
−100000
0
100000
200000
300000
X (meters)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import Circle, Patch
from matplotlib.path import Path
import rasterio
from rasterio.windows import from_bounds
from scipy import ndimage
from skimage.measure import find_contours
from matplotlib.colors import LinearSegmentedColormap

# ------------------------------------------------------------------------
# 1) Load & label PSR raster
# ------------------------------------------------------------------------
PSR_TIF = "LPSR_75S_120M_201608.tiff"
with rasterio.open(PSR_TIF) as src:
   psr_mask      = src.read(1)
   psr_transform = src.transform

labeled_array, _ = ndimage.label(psr_mask > 0)

# compute PSR areas
pixel_area_km2 = (120/1000)**2
ids, counts    = np.unique(labeled_array[labeled_array>0], return_counts=True)
areas_km2      = counts * pixel_area_km2
psr_df         = pd.DataFrame({"psr_id": ids, "area_km2": areas_km2})
psr_df         = psr_df[psr_df.area_km2>1.0].sort_values("area_km2", ascending=False)

# ------------------------------------------------------------------------
# 2) Load all temperature points and map each into a PSR id
# ------------------------------------------------------------------------
dlre    = np.load("dlre_temp_80S.npz")
x_temp  = dlre["x"]      # km
y_temp  = dlre["y"]      # km
t_avg   = dlre["temp"]   # K

# convert to meters and pixel coordinates
x_m   = x_temp * 1000
y_m   = y_temp * 1000
col_f = (x_m - psr_transform.c) / psr_transform.a
row_f = (y_m - psr_transform.f) / psr_transform.e
col   = np.floor(col_f).astype(int)
```

```
row   = np.floor(row_f).astype(int)

# mask out-of-bounds
valid_pts = (
   (row>=0)&(row<labeled_array.shape[0]) &
   (col>=0)&(col<labeled_array.shape[1])
)
pt_psr = np.zeros_like(t_avg, dtype=int)
pt_psr[valid_pts] = labeled_array[row[valid_pts], col[valid_pts]]

# only keep 25–35 K
mask25_35 = (t_avg>=25)&(t_avg<35)
print("Total 25–35 K points:", mask25_35.sum())

# ------------------------------------------------------------------------
# 3) Filter to PSRs that contain ≥1 of those points, then pick top 10 + 13th +20th
# ------------------------------------------------------------------------
has_pt    = pd.Series(pt_psr[mask25_35]).value_counts()
valid_ids = has_pt.index.values

# all PSRs (with area) that have at least one 25–35 K point
valid_psrs = psr_df[psr_df.psr_id.isin(valid_ids)].reset_index(drop=True)

# sorted by area descending
sorted_ids = valid_psrs.psr_id.values.astype(int)

# take top 10
final_ids = list(sorted_ids[:10])
# add 13th if exists (index 12)
if len(sorted_ids) > 12:
   final_ids.append(int(sorted_ids[12]))
# add 20th if exists (index 19)
if len(sorted_ids) > 19:
   final_ids.append(int(sorted_ids[19]))

print("Final PSRs to plot:", final_ids)

# ------------------------------------------------------------------------
# 4) Compute bounding boxes for those selected
# ------------------------------------------------------------------------
bds = []
for pid in final_ids:
```

```
    mask     = (labeled_array == pid)
    contours = find_contours(mask.astype(float), 0.5)
    if not contours:
        continue
    c        = max(contours, key=len)
    rows,cols= c[:,0], c[:,1]
    xs       = (psr_transform.c + cols*psr_transform.a)/1000
    ys       = (psr_transform.f + rows*psr_transform.e)/1000
    bds.append({
        "psr_id": pid,
        "xmin": xs.min(), "xmax": xs.max(),
        "ymin": ys.min(), "ymax": ys.max()
    })

df_bounds = pd.DataFrame(bds)
df_bounds.to_csv("selected_psr_bounds.csv", index=False)
print("Wrote selected_psr_bounds.csv")

# ------------------------------------------------------------------------------
# 5) Drivable-slope color mapping
# ------------------------------------------------------------------------------
def slope_rgb(arr):
    rgb = np.zeros(arr.shape + (3,), float)
    rgb[arr<15]              = [0,1,0]    # green
    rgb[(arr>=15)&(arr<25)] = [1,1,0]    # yellow
    rgb[arr>=25]             = [1,0,0]    # red
    return rgb

# ------------------------------------------------------------------------------
# 6) True-area circle radius
# ------------------------------------------------------------------------------
area_pt = 0.126        # km² per temperature point
r_km    = np.sqrt(area_pt/np.pi)

# ------------------------------------------------------------------------------
# 7) Plot each PSR zoom
# ------------------------------------------------------------------------------
SLOPE_TIF = "LDSM_80S_80MPP_ADJ.tiff"
with rasterio.open(SLOPE_TIF) as slope_src:
    for _, row in df_bounds.iterrows():
        pid        = int(row.psr_id)
        xmin,xmax  = row.xmin-5, row.xmax+5
```

```
    ymin,ymax  = row.ymin-5, row.ymax+5

    # read only needed window from slope TIF
    left, bottom = xmin*1000, ymin*1000
    right, top   = xmax*1000, ymax*1000
    win          = from_bounds(left,bottom,right,top, slope_src.transform)
    slope_patch  = slope_src.read(1, window=win)

    # build km-grid for plotting
    h,w = slope_patch.shape
    xs = np.linspace(xmin, xmax, w)
    ys = np.linspace(ymax, ymin, h)  # flipped
    fig, ax = plt.subplots(figsize=(6,6))

    ax.imshow(
        slope_rgb(slope_patch),
        extent=(xmin,xmax,ymin,ymax),
        origin="upper", aspect="equal"
    )

    # PSR outline
    mask     = (labeled_array==pid)
    c        = max(find_contours(mask.astype(float),0.5), key=len)
    rows,cols= c[:,0], c[:,1]
    px       = (psr_transform.c + cols*psr_transform.a)/1000
    py       = (psr_transform.f + rows*psr_transform.e)/1000
    ax.plot(px, py, color="white", linewidth=1.5)

    # prepare path & list of pts in this PSR
    path  = Path(np.column_stack((px, py)))
    pts25 = [(x_temp[i], y_temp[i])
             for i in np.where(mask25_35 & (pt_psr==pid))[0]]

    # plot 25-35 K circles as true-area
    for xpt, ypt in pts25:
        if xmin<=xpt<=xmax and ymin<=ypt<=ymax and path.contains_point((xpt,ypt)):
            circ = Circle((xpt, ypt), r_km,
                          facecolor="red", edgecolor=None, alpha=0.8,
                          transform=ax.transData, zorder=2)
            ax.add_patch(circ)

    # label center
```

```
        cx, cy = px.mean(), py.mean()
        ax.text(cx, cy, str(pid),
                color="white", weight="bold",
                ha="center", va="center", fontsize=12)

        # legend
        legend_handles = [
          Patch(facecolor=[0,1,0], edgecolor="k", label="0–15° slope"),
          Patch(facecolor=[1,1,0], edgecolor="k", label="15–25° slope"),
          Patch(facecolor=[1,0,0], edgecolor="k", label="≥25° slope"),
          Patch(facecolor="red",  edgecolor="none", label="25–35 K pt")
        ]
        ax.legend(handles=legend_handles,
                  loc="upper right",
                  title=f"PSR {pid}")

        ax.set_xlim(xmin, xmax)
        ax.set_ylim(ymin, ymax)
        ax.set_xlabel("X (km)")
        ax.set_ylabel("Y (km)")
        ax.set_title(f"PSR {pid} – drivable + 25–35 K")

        out = f"PSR_{pid}_drivable_25-35K.png"
        fig.savefig(out, dpi=300, bbox_inches="tight")
        plt.close(fig)
        print("Saved", out)
```

*** SLOPE MAP WITH COLD POINTS BLOWN UP***

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
import rasterio
from rasterio.windows import from_bounds
from scipy import ndimage
from skimage.measure import find_contours
from matplotlib.colors import ListedColormap, BoundaryNorm

# ------------------------------------------------------------------------
# 1) Load & label PSR raster → get PSR areas
# ------------------------------------------------------------------------
PSR_TIF = "LPSR_75S_120M_201608.tiff"
with rasterio.open(PSR_TIF) as src:
   psr_mask      = src.read(1)
   psr_transform = src.transform

labeled_array, _ = ndimage.label(psr_mask > 0)

pixel_area_km2 = (120/1000)**2
ids, counts    = np.unique(labeled_array[labeled_array>0], return_counts=True)
areas_km2      = counts * pixel_area_km2

psr_df = pd.DataFrame({
   "psr_id":   ids.astype(int),
   "area_km2": areas_km2
}).query("area_km2>1.0")\
.sort_values("area_km2", ascending=False)\
.reset_index(drop=True)

# ------------------------------------------------------------------------
# 2) Load DLRE temps & map into PSR IDs
# ------------------------------------------------------------------------
dlre   = np.load("dlre_temp_80S.npz")
x_temp = dlre["x"]    # km
y_temp = dlre["y"]    # km
t_avg  = dlre["temp"] # K

# convert to meters → pixel coords in PSR raster
x_m   = x_temp * 1000
y_m   = y_temp * 1000
```

```
col_f = (x_m - psr_transform.c) / psr_transform.a
row_f = (y_m - psr_transform.f) / psr_transform.e
col   = np.floor(col_f).astype(int)
row   = np.floor(row_f).astype(int)

valid_pts = (
   (row>=0)&(row<labeled_array.shape[0]) &
   (col>=0)&(col<labeled_array.shape[1])
)

pt_psr = np.zeros_like(t_avg, dtype=int)
pt_psr[valid_pts] = labeled_array[row[valid_pts], col[valid_pts]]

mask25_35 = (t_avg >= 25) & (t_avg < 35)
print("Total 25–35 K points:", mask25_35.sum())

# ------------------------------------------------------------------------
# 3) Pick top 10 + 13th + 20th largest PSRs that have ≥1 cold point
# ------------------------------------------------------------------------
has_pt    = pd.Series(pt_psr[mask25_35]).value_counts()
valid_ids = has_pt.index.values
cold_psrs = psr_df[psr_df.psr_id.isin(valid_ids)].reset_index(drop=True)

# first 10, then index 12 and 19 if they exist
indices = list(range(min(10, len(cold_psrs))))
for extra in (12, 19):
   if extra < len(cold_psrs):
       indices.append(extra)
indices = sorted(set(indices))

psr_ids = cold_psrs.loc[indices, "psr_id"].astype(int).values
print("PSRs to plot:", psr_ids)

# ------------------------------------------------------------------------
# 4) Compute bounding boxes for each selected PSR (in km)
# ------------------------------------------------------------------------
bounds = []
for pid in psr_ids:
   mask = (labeled_array == pid)
   cnts = find_contours(mask.astype(float), 0.5)
   if not cnts:
       continue
```

```
    c = max(cnts, key=len)
    rows, cols = c[:,0], c[:,1]
    xs = (psr_transform.c + cols*psr_transform.a)/1000
    ys = (psr_transform.f + rows*psr_transform.e)/1000
    bounds.append({
        "psr_id": pid,
        "xmin":   xs.min(),
        "xmax":   xs.max(),
        "ymin":   ys.min(),
        "ymax":   ys.max()
    })
df_bounds = pd.DataFrame(bounds)

# ------------------------------------------------------------------------------
# 5) Build discrete 5-bin colormaps + norms
# ------------------------------------------------------------------------------
sky_colors = ["purple","blue","green","yellow","orange"]
sky_bins   = [0,0.2,0.4,0.6,0.8,1.0]
sky_cmap   = ListedColormap(sky_colors)
sky_norm   = BoundaryNorm(sky_bins, sky_cmap.N)

ill_colors = ["purple","blue","green","yellow","orange"]
ill_bins   = [0,20,40,60,80,100]
ill_cmap   = ListedColormap(ill_colors)
ill_norm   = BoundaryNorm(ill_bins, ill_cmap.N)

# ------------------------------------------------------------------------------
# 6) Plot each PSR side-by-side: sky & illumination
# ------------------------------------------------------------------------------
area_pt = 0.126
r_km    = np.sqrt(area_pt/np.pi)

SKY_TIF   = "SKYV_65S_240M.tiff"
SOLAR_TIF = "AVGVISIB_75S_120M_201608.tiff"

for _, row in df_bounds.iterrows():
    pid = int(row.psr_id)
    xmin, xmax = row.xmin - 5, row.xmax + 5
    ymin, ymax = row.ymin - 5, row.ymax + 5

    # — 6a) Sky visibility window
    with rasterio.open(SKY_TIF) as sky_src:
```

```
        win = from_bounds(xmin*1000, ymin*1000, xmax*1000, ymax*1000,
sky_src.transform)
        raw = sky_src.read(1, window=win).astype(float)
        sr  = raw * sky_src.scales[0] + sky_src.offsets[0]     # 0–2π
        vis = np.clip(sr/(2*np.pi), 0, 1)

    # — 6b) Avg illumination window
    with rasterio.open(SOLAR_TIF) as sol_src:
        win2 = from_bounds(xmin*1000, ymin*1000, xmax*1000, ymax*1000,
sol_src.transform)
        raw2 = sol_src.read(1, window=win2).astype(float)
        pct  = np.clip(raw2 * sol_src.scales[0] * 100, 0, 100)

    # — 6c) Make the figure
    fig, (ax1, ax2) = plt.subplots(1,2,figsize=(12,6), tight_layout=True)

    # —— sky panel
    im1 = ax1.imshow(vis,
                     extent=(xmin,xmax,ymin,ymax),
                     origin="upper",
                     cmap=sky_cmap,
                     norm=sky_norm,
                     aspect="equal")
    c = max(find_contours((labeled_array==pid).astype(float), 0.5), key=len)
    px = (psr_transform.c + c[:,1]*psr_transform.a)/1000
    py = (psr_transform.f + c[:,0]*psr_transform.e)/1000
    ax1.plot(px, py, 'k-', lw=1.5)
    ax1.set_title(f"PSR {pid}: Sky visibility")
    fig.colorbar(im1, ax=ax1, ticks=sky_bins, label="Fraction of sky visible")

    sel25 = (pt_psr==pid) & mask25_35
    for xt, yt in zip(x_temp[sel25], y_temp[sel25]):
        circ = Circle((xt, yt), r_km,
                      facecolor="red", edgecolor="none",
                      alpha=0.8, transform=ax1.transData, zorder=2)
        ax1.add_patch(circ)

    # —— illumination panel
    im2 = ax2.imshow(pct,
                     extent=(xmin,xmax,ymin,ymax),
                     origin="upper",
                     cmap=ill_cmap,
```

```
                  norm=ill_norm,
                  aspect="equal")
ax2.plot(px, py, 'k-', lw=1.5)
ax2.set_title(f"PSR {pid}: Avg illumination")
fig.colorbar(im2, ax=ax2, ticks=ill_bins, label="% of time sunlit")

for xt, yt in zip(x_temp[sel25], y_temp[sel25]):
    circ = Circle((xt, yt), r_km,
                  facecolor="red", edgecolor="none",
                  alpha=0.8, transform=ax2.transData, zorder=2)
    ax2.add_patch(circ)

fig.suptitle(f"PSR {pid} — Cold points + Sky & Illumination", fontsize=14)
out = f"PSR_{pid}_sky_illum_bins.png"
fig.savefig(out, dpi=300, bbox_inches="tight")
plt.close(fig)
print("Saved", out)
```

**** SKY VIS AND SOLAR ILLUMNATION BLOWN UP SIDEBY SIDE*********

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.path import Path
from matplotlib.patches import Circle
from matplotlib.lines import Line2D
from skimage.measure import find_contours
import rasterio
from scipy import ndimage

# ------------------------------------------------------------------------
# 1) Load the filtered PSR metadata, pick top 10 + 13th + 20th by area
# ------------------------------------------------------------------------
meta = pd.read_csv("valid_psrs_cold_stable_updated.csv")
meta = meta.sort_values("area_km2", ascending=False).reset_index(drop=True)
pick_idxs = list(range(10)) + [12, 19]                    # 0-9, plus 12th & 19th
indices
pick_idxs = [i for i in pick_idxs if i < len(meta)]
top_ids = meta.loc[pick_idxs, "psr_id"].astype(int).tolist()
print("Zooming on PSR IDs:", top_ids)

# ------------------------------------------------------------------------
# 2) Load DEM + make ROYGBIV colormap
# ------------------------------------------------------------------------
topo = np.load("topography_80S.npz")
dem, x, y = topo["dem"], topo["x"], topo["y"]
roygbiv = ["indigo","blue","cyan","green","yellow","orange","red"]
cmap_dem = LinearSegmentedColormap.from_list("roygbiv", roygbiv, N=256)

# ------------------------------------------------------------------------
# 3) Load & label PSR raster
# ------------------------------------------------------------------------
with rasterio.open("LPSR_75S_120M_201608.tiff") as src:
   psr_mask, transform = src.read(1), src.transform
labeled_array, _ = ndimage.label(psr_mask > 0)

# ------------------------------------------------------------------------
# 4) Load DLRE temperatures
# ------------------------------------------------------------------------
dlre    = np.load("dlre_temp_80S.npz")
x_temp  = dlre["x"]     # km
```

```
y_temp  = dlre["y"]     # km
t_avg   = dlre["temp"]  # K

# ------------------------------------------------------------------------------
# 5) Define our two bins and colors
# ------------------------------------------------------------------------------
bins       = [25, 35, 45]
bin_labels = [f"{bins[i]}–{bins[i+1]} K" for i in range(len(bins)-1)]
colors     = ["red", "orange"]

# ------------------------------------------------------------------------------
# 6) True-area radius in km for 0.126 km²
# ------------------------------------------------------------------------------
area_pt = 0.126
r_km    = np.sqrt(area_pt / np.pi)

# ------------------------------------------------------------------------------
# 7) Compute and save bounding boxes for each PSR in top_ids
# ------------------------------------------------------------------------------
bounds = []
for psr_id in top_ids:
   mask = (labeled_array == psr_id)
   if not mask.any():
       continue
   contours = find_contours(mask.astype(float), 0.5)
   if not contours:
       continue
   c = max(contours, key=len)
   rows, cols = c[:,0], c[:,1]
   xs_m = transform.c + cols * transform.a
   ys_m = transform.f + rows * transform.e
   xs, ys = xs_m/1000, ys_m/1000
   bounds.append({
       "psr_id": psr_id,
       "xmin": xs.min(), "xmax": xs.max(),
       "ymin": ys.min(), "ymax": ys.max()
   })

df_bounds = pd.DataFrame(bounds)
df_bounds.to_csv("top12_psr_bounds.csv", index=False)
print("Wrote top12_psr_bounds.csv")
```

```
# ------------------------------------------------------------------------------
# 8) Zoom & plot each PSR individually, now with a topo colorbar
# ------------------------------------------------------------------------------
for _, row in df_bounds.iterrows():
   psr_id = int(row.psr_id)
   xmin, xmax = row.xmin - 5, row.xmax + 5
   ymin, ymax = row.ymin - 5, row.ymax + 5

   fig, ax = plt.subplots(figsize=(6,6))
   mesh = ax.pcolormesh(x, y, dem, cmap=cmap_dem, shading="auto")
   # add topo colorbar
   cbar = fig.colorbar(mesh, ax=ax, shrink=0.8, pad=0.02)
   cbar.set_label("Elevation (km)")

   ax.set_aspect("equal", "box")
   ax.set_xlim(xmin, xmax)
   ax.set_ylim(ymin, ymax)
   ax.set_title(f"PSR {psr_id}: 25-45 K only", fontsize=12)
   ax.set_xlabel("X (km)")
   ax.set_ylabel("Y (km)")

   # draw PSR boundary
   contour = max(find_contours((labeled_array==psr_id).astype(float), 0.5), key=len)
   rows, cols = contour[:,0], contour[:,1]
   xs = (transform.c + cols*transform.a)/1000
   ys = (transform.f + rows*transform.e)/1000
   ax.plot(xs, ys, color="white", linewidth=1.5)

   # select only in-PSR points
   path = Path(np.column_stack((xs, ys)))
   pts = np.column_stack((x_temp, y_temp))

   # plot each temp-bin as true-area circles
   for lo, hi, c in zip(bins[:-1], bins[1:], colors):
       sel = path.contains_points(pts) & (t_avg >= lo) & (t_avg < hi)
       for xt, yt in pts[sel]:
           circ = Circle((xt, yt), r_km,
                         facecolor=c, edgecolor=None, alpha=0.8,
                         transform=ax.transData, zorder=2)
           ax.add_patch(circ)

   # custom legend with bigger markers
```

```
    legend_handles = [
        Line2D([0],[0], marker='o', color='w', label=lab,
               markerfacecolor=c, markersize=10)
        for c,lab in zip(colors, bin_labels)
    ]
    ax.legend(handles=legend_handles, title="Temp bins", loc="upper right")

    plt.tight_layout()
    outname = f"PSR_{psr_id}_25-45K_zoom.png"
    fig.savefig(outname, dpi=300)
    plt.close(fig)
    print("Saved", outname)
```

TOP TEN PLUS 2 BLOWUPS WITH COLOR BAR ***

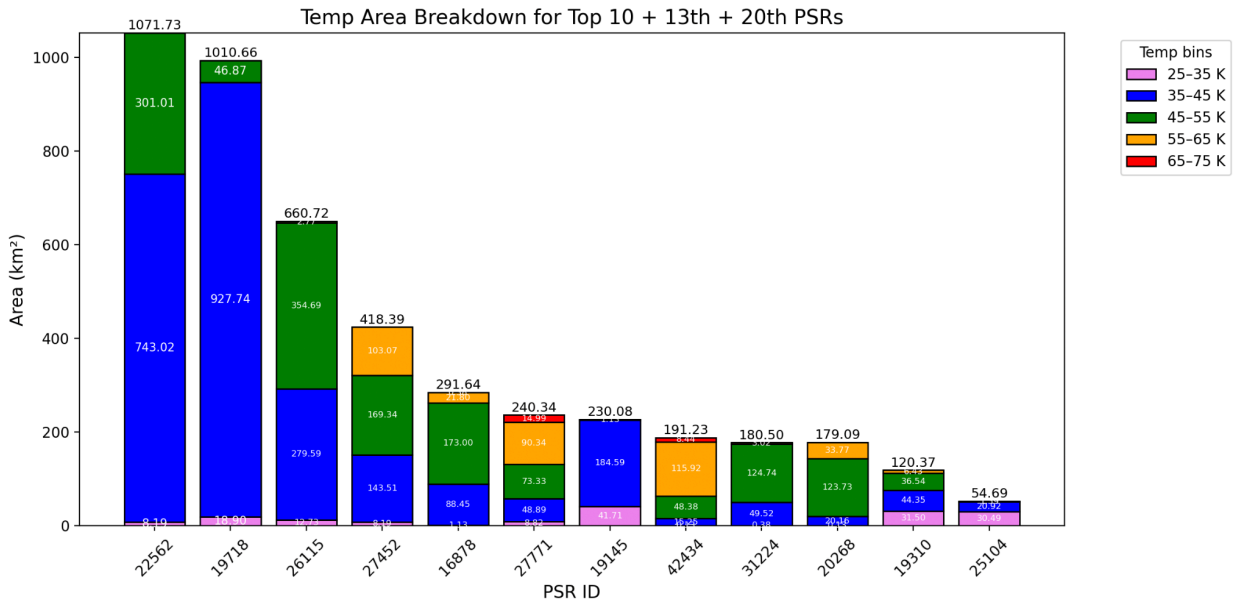
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import rasterio
from scipy import ndimage
import math

# Moon radius in meters (IAU 2000)
R_MOON = 1737400

# === Load PSR TIFF and its transform ===
with rasterio.open("LPSR_75S_120M_201608.tiff") as src:
   psr_mask   = src.read(1)
   transform  = src.transform

# === Mask out anything north of 80°S ===
height, width = psr_mask.shape

# build array of row indices
rows = np.arange(height)

# compute y (meters) at center of each row: y = f + row*e
y_m = transform.f + rows * transform.e  # note: transform.e is negative in polar
stereographic

# inverse stereographic latitude: φ = 2*atan(y/(2R)) - 90°
lat_rad = 2*np.arctan(y_m / (2*R_MOON)) - (math.pi/2)
lat_deg = np.degrees(lat_rad)

# for any row whose latitude > -80° (i.e. north of 80°S), zero the mask
north_rows = np.where(lat_deg > -80)[0]
psr_mask[north_rows[:,None], :] = 0

# === Label PSRs (only 80–90°S remains) ===
labeled_array, num_features = ndimage.label(psr_mask > 0)

# === Area per pixel (in km²) ===
pixel_area_km2 = (120 / 1000)**2  # 120 m → km

# === Calculate each PSR’s area ===
psr_ids, counts = np.unique(labeled_array[labeled_array>0], return_counts=True)
areas_km2       = counts * pixel_area_km2
```

```
# === Build DataFrame & sort ===
psr_df = pd.DataFrame({
    "PSR_ID":   psr_ids,
    "Area_km2": areas_km2
}).sort_values("Area_km2", ascending=False).reset_index(drop=True)

# === Print summary stats ===
print(f"📦 Total PSRs (80–90°S): {len(psr_df)}")
print(f" 🔹 Min Area: {psr_df['Area_km2'].min():.5f} km²")
print(f" 🔹 Max Area: {psr_df['Area_km2'].max():.2f} km²")
print(f" 🔹 Median Area: {psr_df['Area_km2'].median():.5f} km²")

# thresholds
for t in [0.5, 1, 2, 5, 10]:
    cnt = (psr_df["Area_km2"] >= t).sum()
    print(f"✅ PSRs ≥ {t} km²: {cnt}")

# total area
total_psr_area = psr_df["Area_km2"].sum()
print(f"Total PSR area (80–90°S): {total_psr_area:.2f} km²")

# === Histogram ===
plt.figure(figsize=(10,6))
plt.hist(psr_df["Area_km2"], bins=100,
         color="slateblue", edgecolor="black", log=True)
plt.title("PSR Area Distribution (80–90°S, log scale)")
plt.xlabel("Area (km²)")
plt.ylabel("Count of PSRs")
plt.grid(True, which="both", linestyle="--", alpha=0.5)
plt.tight_layout()
plt.savefig("psr_area_histogram_80-90S.png", dpi=300)
plt.show()
```

📦 Total PSRs (80–90°S): 47136
🔹 Min Area: 0.01440 km²
🔹 Max Area: 1071.73 km²
🔹 Median Area: 0.01440 km²
✅ PSRs ≥ 0.5 km²: 3355
✅ PSRs ≥ 1 km²: 2096
✅ PSRs ≥ 2 km²: 1253
✅ PSRs ≥ 5 km²: 582

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap

# ----- Load CSV & sort by area descending -----
df = pd.read_csv("valid_psrs_cold_stable_updated.csv")
df = df.sort_values("area_km2", ascending=False).reset_index(drop=True)

# ----- Select indices: 0-9 (top 10), plus 12 (13th) and 19 (20th) -----
pick_idxs = list(range(10)) + [12, 19]
# filter out any out-of-range indices
pick_idxs = [i for i in pick_idxs if i < len(df)]
sel = df.iloc[pick_idxs].reset_index(drop=True)

print("Selected PSR IDs:", sel["psr_id"].tolist())

# ----- Define 5 bins from 25→75 K in 10 K steps -----
bins = np.arange(25, 75 + 1, 10)
labels = [f"{bins[i]}-{bins[i+1]} K" for i in range(len(bins)-1)]
area_per_pt = 0.126

# ----- Build histogram areas for each selected PSR -----
hist_areas = []
for _, row in sel.iterrows():
   pts_str = str(row["all_temp_points"])
   if pts_str.strip():
       pts = np.array([float(x) for x in pts_str.split(",") if x.strip()])
       counts, _ = np.histogram(pts, bins=bins)
   else:
       counts = np.zeros(len(labels), int)
   hist_areas.append(counts * area_per_pt)
hist_areas = np.array(hist_areas)  # shape (n_sel, 5)

# ----- Violet→Blue→Green→Yellow→Red palette -----
colors = ["violet", "blue", "green", "orange", "red"]
cmap5 = LinearSegmentedColormap.from_list("VBGYR", colors, N=5)

# ----- Plot stacked bars -----
n = hist_areas.shape[0]
inds = np.arange(n)
bottom = np.zeros(n)
```

```
plt.figure(figsize=(12, 6))
for i, lab in enumerate(labels):
   plt.bar(inds, hist_areas[:, i], bottom=bottom,
           color=cmap5(i / 4), edgecolor="black", label=lab)
   bottom += hist_areas[:, i]

# ----- Annotate each segment, small font for bars 3+ -----
bottom2 = np.zeros(n)
for i in range(len(labels)):
   for j in range(n):
       a = hist_areas[j, i]
       if a > 0:
           ymid = bottom2[j] + a / 2
           if i == 0:
               ymid += 0.5  # nudge for the coldest bin
           fs = 8 if j < 2 else 6
           plt.text(inds[j], ymid, f"{a:.2f}",
                    ha="center", va="center",
                    color="white", fontsize=fs)
   bottom2 += hist_areas[:, i]

# ----- Total area labels above each bar -----
for j in range(n):
   tot = sel.loc[j, "area_km2"]
   plt.text(inds[j], bottom2[j] + 2, f"{tot:.2f}",
            ha="center", va="bottom",
            color="black", fontsize=9)

# ----- Final formatting -----
plt.xticks(inds, sel["psr_id"].astype(int), rotation=45)
plt.xlabel("PSR ID", fontsize=12)
plt.ylabel("Area (km²)", fontsize=12)
plt.title("Temp Area Breakdown for Top 10 + 13th + 20th PSRs", fontsize=14)
plt.legend(title="Temp bins", bbox_to_anchor=(1.05, 1), loc="upper left")
plt.tight_layout()
plt.savefig("top12_psr_temp_25-75K_violet_to_red.png", dpi=300)
plt.show()
```



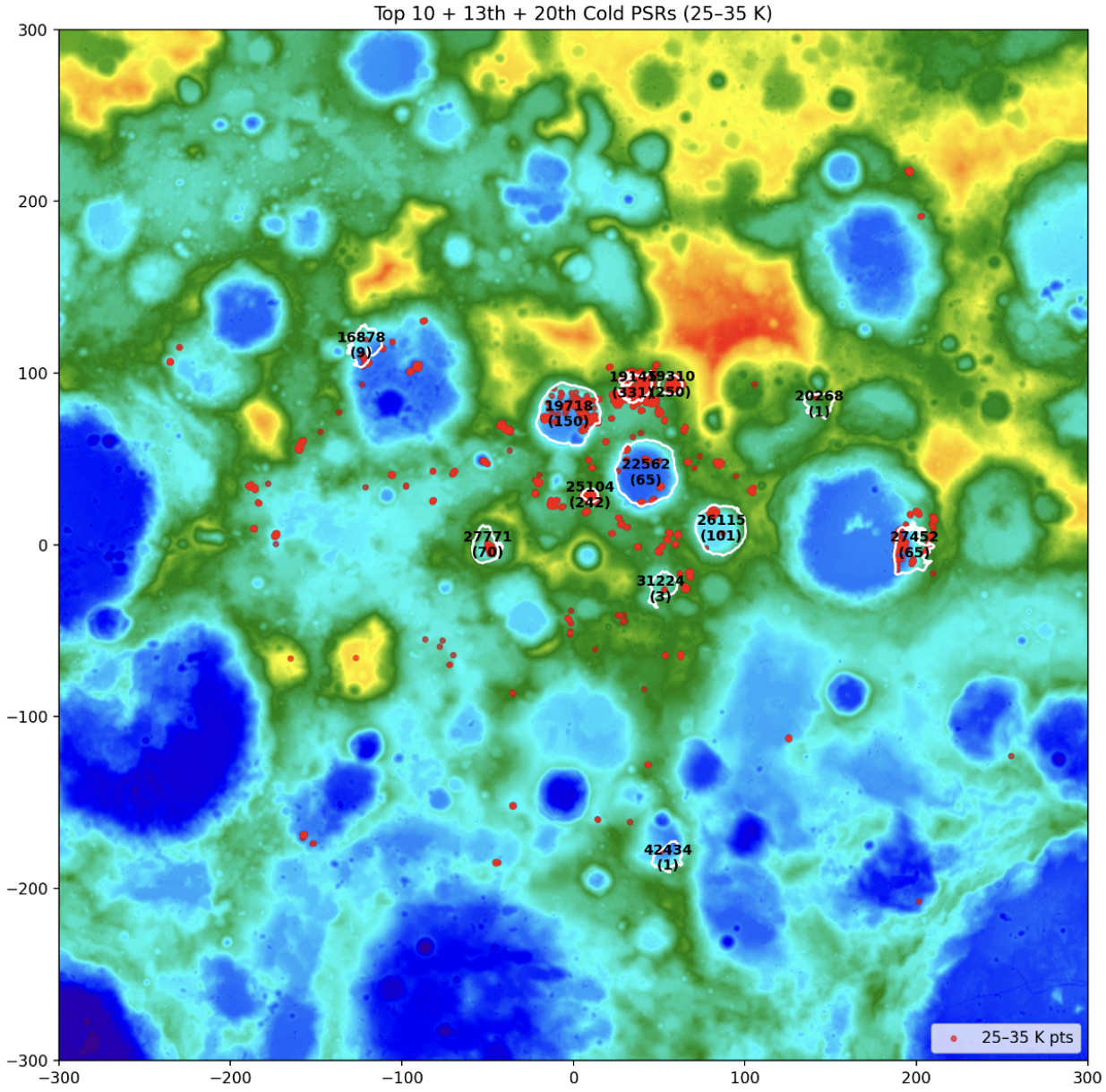

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.path import Path
from skimage.measure import find_contours
import pandas as pd
import rasterio
from scipy import ndimage

# 1) DEM + colormap
topo = np.load("topography_80S.npz")
dem, x, y = topo["dem"], topo["x"], topo["y"]
cmap_dem = LinearSegmentedColormap.from_list("roygbiv",
   ["indigo","blue","cyan","green","yellow","orange","red"], N=256)

# 2) PSR mask → labeled_array
with rasterio.open("LPSR_75S_120M_201608.tiff") as src:
   psr_mask, transform = src.read(1), src.transform
labeled_array, _ = ndimage.label(psr_mask>0)

# 3) DLRE temps & filter 25-35 K
dlre    = np.load("dlre_temp_80S.npz")
x_temp, y_temp, t_avg = dlre["x"], dlre["y"], dlre["temp"]
mask25_35 = (t_avg>=25)&(t_avg<35)
x_valid, y_valid = x_temp[mask25_35], y_temp[mask25_35]
print(f"Total 25-35 K pts: {len(x_valid)}")
```

```
# 4) compute PSR areas >1 km²
pix_area = (120/1000)**2
ids, cnts = np.unique(labeled_array[labeled_array>0], return_counts=True)
areas = cnts*pix_area
psr_df = pd.DataFrame({"psr_id":ids,"area_km2":areas})
psr_df = psr_df[psr_df.area_km2>1].sort_values("area_km2",ascending=False)

# 5) keep only PSRs that contain ≥1 valid pt
valid = []
for pid,ar in zip(psr_df.psr_id, psr_df.area_km2):
   m = (labeled_array==pid)
   conts = find_contours(m.astype(float),.5)
   if not conts: continue
   c = max(conts,key=len)
   xs = (transform.c + c[:,1]*transform.a)/1000
   ys = (transform.f + c[:,0]*transform.e)/1000
   path = Path(np.column_stack((xs,ys)))
   cnt = path.contains_points(np.column_stack((x_valid,y_valid))).sum()
   if cnt>0:
       valid.append((pid,ar,cnt,np.mean(xs),np.mean(ys),xs,ys))

# 6) sort & pick 1-10,13,20
valid = sorted(valid, key=lambda x:x[1], reverse=True)
sel_indices = list(range(10)) + [12,19]
picked = [valid[i] for i in sel_indices if i<len(valid)]

print("\nPicked PSRs:")
for pid,ar,cnt,_,_,_,_ in picked:
   print(f"  {pid}: {ar:.2f} km², {cnt} pts")
print("Total pts in these:", sum(p[2] for p in picked))

# 7) plot
fig, ax = plt.subplots(figsize=(10,10))
ax.pcolormesh(x,y,dem,cmap=cmap_dem,shading="auto")
ax.set_aspect("equal","box")
ax.set_xlim(-300,300); ax.set_ylim(-300,300)
ax.set_title("Top 10 + 13th + 20th Cold PSRs (25-35 K)")
ax.scatter(x_valid,y_valid, c="red", s=10, alpha=0.6, label="25-35 K pts")

for pid,ar,cnt,cx,cy,xs,ys in picked:
   ax.plot(xs,ys,"w-",lw=1.5)
   ax.text(cx,cy,f"{pid}\n({cnt})",ha="center",va="center",
```

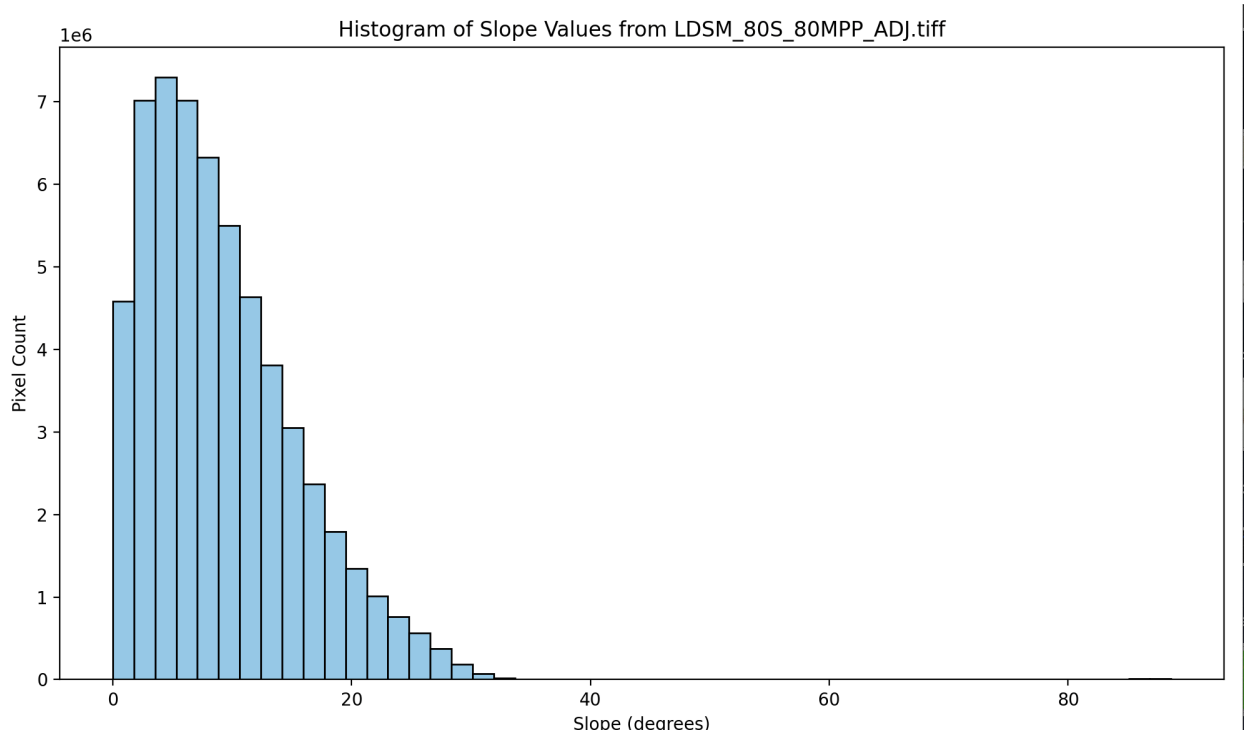
```
            color="black",weight="bold",fontsize=9)

ax.legend(loc="lower right")
plt.tight_layout()
plt.savefig("selected_cold_psrs.png",dpi=300)
plt.show()
```

(moon_env) (base) thomasleeds@dhcp-10-250-22-222 ~ % /Users/thomasleeds/opt/anaconda3/envs/moon_env/bin/python /Users/thomasleeds/moon_plot.py
Total 25–35 K pts: 2373

Picked PSRs:
  22562: 1071.73 km², 65 pts
  19718: 1010.66 km², 150 pts
  26115: 660.72 km², 101 pts
  27452: 418.39 km², 65 pts
  16878: 291.64 km², 9 pts
  27771: 240.34 km², 70 pts
  19145: 230.08 km², 331 pts
  42434: 191.23 km², 1 pts
  31224: 180.50 km², 3 pts
  20268: 179.09 km², 1 pts
  19310: 120.37 km², 250 pts
  25104: 54.69 km², 242 pts
Total pts in these: 1288
2025-04-16 23:53:52.783 python[265:3102339] +[IMKClient subclass]: chose IMKClient_Modern
2025-04-16 23:53:52.783 python[265:3102339] +[IMKInputSession subclass]: chose IMKInputSession_Modern


Top 10 + 13th + 20th Cold PSRs (25–35 K)
16878
(9)
19143
(331)
19310
(250)
19718
(150)
20268
(1)
22562
(65)
25104
(242)
26115
(101)
27452
(65)
27771
(70)
31224
(3)
42434
(1)
25–35 K pts
300
200
100
0
−100
−200
−300
−300
−200
−100
0
100
200
300

```python
import numpy as np
import matplotlib.pyplot as plt
import rasterio

# File path to your GeoTiff
slope_file = "LDSM_80S_80MPP_ADJ.tiff"

# --- Step 1: Open the GeoTiff and read the slope values ---
with rasterio.open(slope_file) as src:
   # Read band 1 as a float array
   slope_data = src.read(1).astype(float)
   # Replace nodata values (if any) with NaN
   nodata_val = src.nodata
   if nodata_val is not None:
       slope_data[slope_data == nodata_val] = np.nan
   print("Slope data shape:", slope_data.shape)

# --- Step 2: Plot Histogram of Slope Values ---
# Exclude NaN values from the histogram
slope_valid = slope_data[~np.isnan(slope_data)].ravel()

plt.figure(figsize=(10,6))
plt.hist(slope_valid, bins=50, color='skyblue', edgecolor='black')
plt.xlabel("Slope (degrees)")
plt.ylabel("Pixel Count")
plt.title("Histogram of Slope Values from LDSM_80S_80MPP_ADJ.tiff")
plt.tight_layout()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.path import Path
from skimage.measure import find_contours
import pandas as pd
import rasterio
from scipy import ndimage

# === Load data ===
topo = np.load("topography_80S.npz")
dem = topo["dem"]
x = topo["x"]
y = topo["y"]

dlre = np.load("dlre_temp_80S.npz")
x_temp = dlre["x"]
y_temp = dlre["y"]
temp_avg = dlre["temp"]

dlre_max = np.load("dlre_tempmax_80S.npz")
temp_max = dlre_max["temp_max"]
```

```
# Filter temp points: temp < 35K AND max < 100K
valid_mask = (temp_avg < 35) & (temp_max < 100)
x_valid = x_temp[valid_mask]
y_valid = y_temp[valid_mask]

# Load PSR raster and label regions
with rasterio.open("LPSR_75S_120M_201608.tiff") as src:
   psr_mask = src.read(1)
   transform = src.transform

labeled_array, num_features = ndimage.label(psr_mask > 0)
pixel_area_km2 = (120 / 1000) ** 2

psr_ids, counts = np.unique(labeled_array[labeled_array > 0], return_counts=True)
areas_km2 = counts * pixel_area_km2

# Filter to PSRs > 1 km²
valid_psr_ids = psr_ids[areas_km2 > 1.0]

# === Create ROYGBIV colormap for topography ===
roygbiv_colors = ["indigo", "blue", "cyan", "green", "yellow", "orange", "red"]
roygbiv_cmap = LinearSegmentedColormap.from_list("roygbiv", roygbiv_colors, N=256)

# === Plot ===
fig, ax = plt.subplots(figsize=(10, 10))
mesh = ax.pcolormesh(x, y, dem, cmap=roygbiv_cmap, shading="auto")
plt.colorbar(mesh, ax=ax, label="Elevation (km)")

# Plot temp points
ax.scatter(x_valid, y_valid, color="red", s=10, alpha=0.8, label="Temp <35K & Max
<100K")

# Store metadata for output CSV
saved_psrs = []
plotted = 0

for psr_id in valid_psr_ids:
   mask = labeled_array == psr_id
   contours = find_contours(mask.astype(float), 0.5)

   for contour in contours:
```

```
        rows, cols = contour[:, 0], contour[:, 1]
        x_coords = transform.c + cols * transform.a
        y_coords = transform.f + rows * transform.e

        x_coords_km = x_coords / 1000
        y_coords_km = y_coords / 1000

        path = Path(np.column_stack((x_coords_km, y_coords_km)))
        contained_mask = path.contains_points(np.column_stack((x_valid, y_valid)))

        if np.any(contained_mask):
            ax.plot(x_coords_km, y_coords_km, color="white", linewidth=1.5)

            lon_center = x_coords.mean() / 1000
            lat_center = y_coords.mean() / 1000
            area = mask.sum() * pixel_area_km2

            saved_psrs.append({
                "psr_id": psr_id,
                "area_km2": area,
                "x_center_km": lon_center,
                "y_center_km": lat_center,
                "cold_points": contained_mask.sum()
            })

            plotted += 1
            break

print(f"✅ Plotted {plotted} valid PSRs > 1 km² containing cold & stable points")

# === Save data ===
df_psrs = pd.DataFrame(saved_psrs)
df_psrs.to_csv("valid_psrs_cold_stable.csv", index=False)
print("📄 Saved metadata for valid PSRs to valid_psrs_cold_stable.csv")

# Format
ax.set_title("Valid PSRs (>1 km²) with Cold Stable Points (<35K)")
ax.set_xlabel("Stereographic X (km)")
ax.set_ylabel("Stereographic Y (km)")
ax.set_xlim(-300, 300)
ax.set_ylim(-300, 300)
ax.set_aspect("equal")
```

```
ax.legend(loc="lower right", frameon=True)
plt.tight_layout()
plt.savefig("fixed_psrs_overlay_plot.png", dpi=300)
plt.show()
```

2373 vs 2308 for 25-35 avg and 100k max

```
import pandas as pd
import numpy as np

# Load the cleaned data
df = pd.read_csv("dlre_prp_south.tab", sep=r"\s+", header=None, skiprows=1,
low_memory=False)

# Assign column names
df.columns = [
   "tri1_x", "tri1_y", "tri1_z", "tri2_x", "tri2_y", "tri2_z", "tri3_x", "tri3_y",
"tri3_z",
   "tri_clon", "tri_clat", "tri_calt",
   "temp_avg", "temp_max", "ice_depth"
]

# Clean and convert
df.replace("NULL", np.nan, inplace=True)
df.dropna(subset=["tri1_x", "tri1_y", "tri2_x", "tri2_y", "tri3_x", "tri3_y",
"tri_clat"], inplace=True)

for col in ["tri1_x", "tri1_y", "tri2_x", "tri2_y", "tri3_x", "tri3_y", "tri_clat"]:
   df[col] = pd.to_numeric(df[col].astype(str).str.rstrip(','), errors="coerce")

# Filter for 80–90°S
df = df[(df["tri_clat"] >= -90) & (df["tri_clat"] <= -80)]

# Extract triangle vertices in km
A = df[["tri1_x", "tri1_y"]].values
B = df[["tri2_x", "tri2_y"]].values
C = df[["tri3_x", "tri3_y"]].values

# Shoelace formula in km² (no conversion needed)
area_km2 = 0.5 * np.abs(
```

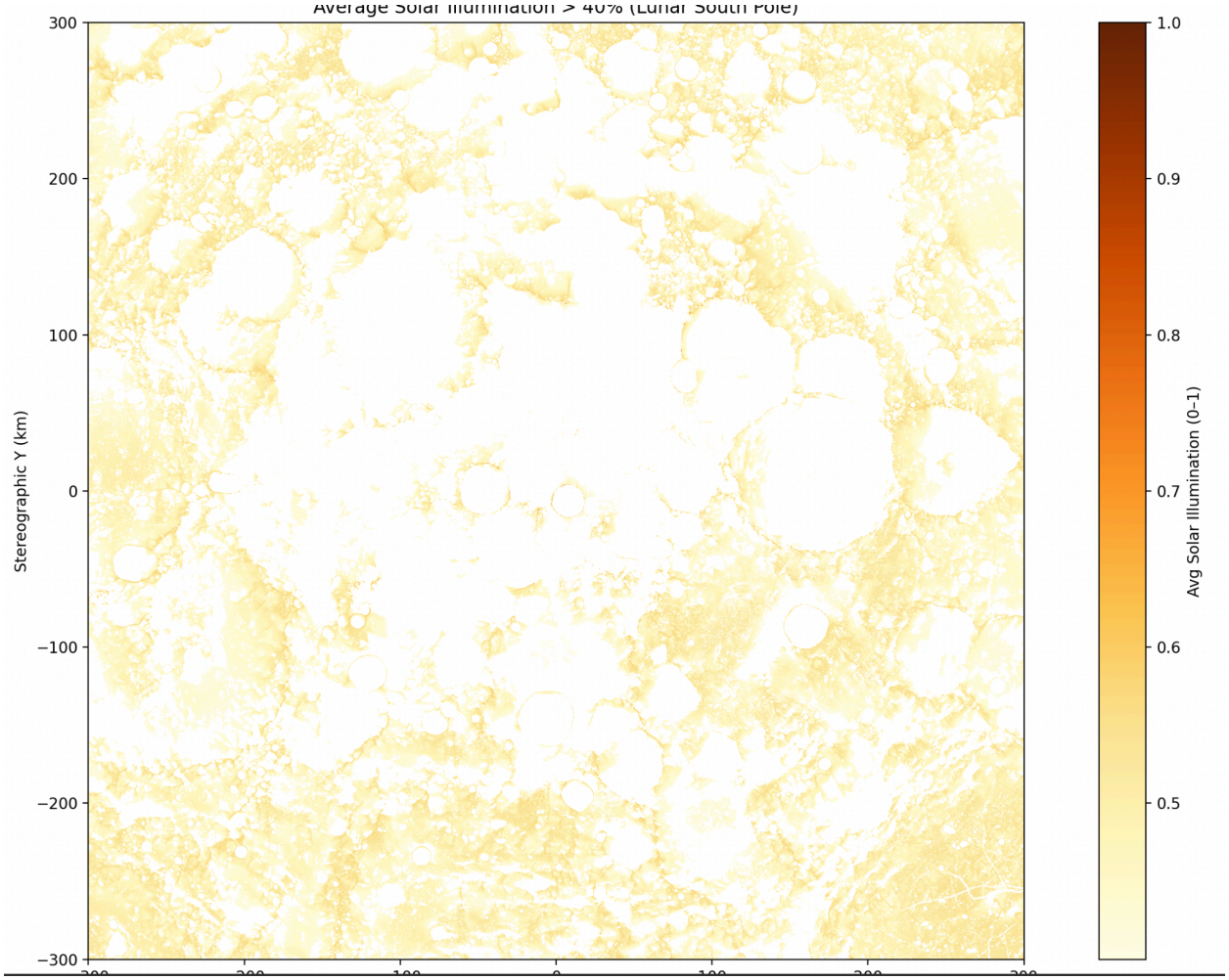
```
    A[:, 0] * (B[:, 1] - C[:, 1]) +
    B[:, 0] * (C[:, 1] - A[:, 1]) +
    C[:, 0] * (A[:, 1] - B[:, 1])
)

# Output
print(f"📦 Total triangles in 80–90°S: {len(area_km2)}")
print(f" 🔹 Min triangle area: {area_km2.min():.8f} km²")
print(f" 🔹 Max triangle area: {area_km2.max():.8f} km²")
print(f" 🔹 Average triangle area: {area_km2.mean():.8f} km²")
```

(moon_env) (base) thomasleeds@dhcp-10-250-74-101 ~ %
/Users/thomasleeds/opt/anaconda3/envs/moon_env/bin/python /Users/thomasleeds/moon_plot.py
📦 Total triangles in 80–90°S: 2261956
- 🔹 Min triangle area: 0.11050755 km²
- 🔹 Max triangle area: 0.13916959 km²
- 🔹 Average triangle area: 0.12618988 km²

(moon_env) (base) thomasleeds@dhcp-10-250-74-101 ~ %

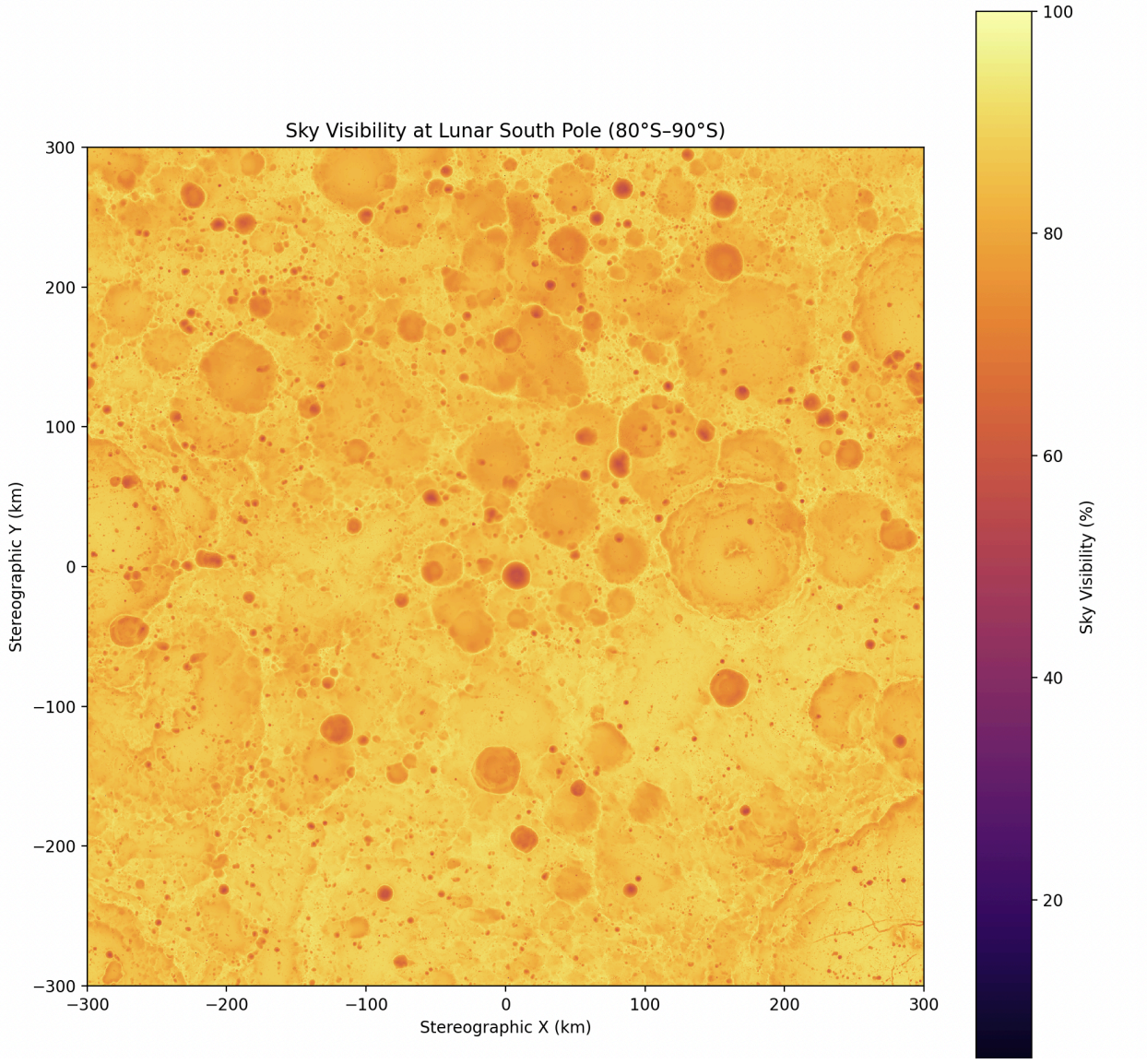
```
import rasterio
import numpy as np
import matplotlib.pyplot as plt

# Load the TIFF file
file_path = "AVGVISIB_75S_120M_201608.tiff"
with rasterio.open(file_path) as src:
    illum_data = src.read(1)
    transform = src.transform

# Replace no-data with NaN
illum_data = np.where(illum_data == src.nodata, np.nan, illum_data)

# Normalize if necessary (assuming 0-100 or 0-255 range instead of 0-1)
if np.nanmax(illum_data) > 1:
    illum_data = illum_data / np.nanmax(illum_data)

# Create X and Y coordinates in kilometers
rows, cols = illum_data.shape
x = np.arange(cols) * transform[0] + transform[2]
y = np.arange(rows) * transform[4] + transform[5]
```

```
x_km = x / 1000
y_km = y / 1000
x_coords, y_coords = np.meshgrid(x_km, y_km)

# Mask values below 40% (0.4)
mask = illum_data >= 0.4
masked_data = np.where(mask, illum_data, np.nan)

# Plot
fig, ax = plt.subplots(figsize=(10, 10))
illum_plot = ax.pcolormesh(x_coords, y_coords, masked_data, cmap="YlOrBr",
shading="auto")
cbar = plt.colorbar(illum_plot, ax=ax, label="Avg Solar Illumination (0-1)")

# Format
ax.set_title("Average Solar Illumination > 40% (Lunar South Pole)")
ax.set_xlim(-300, 300)
ax.set_ylim(-300, 300)
ax.set_aspect("equal")
ax.set_xlabel("Stereographic X (km)")
ax.set_ylabel("Stereographic Y (km)")

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



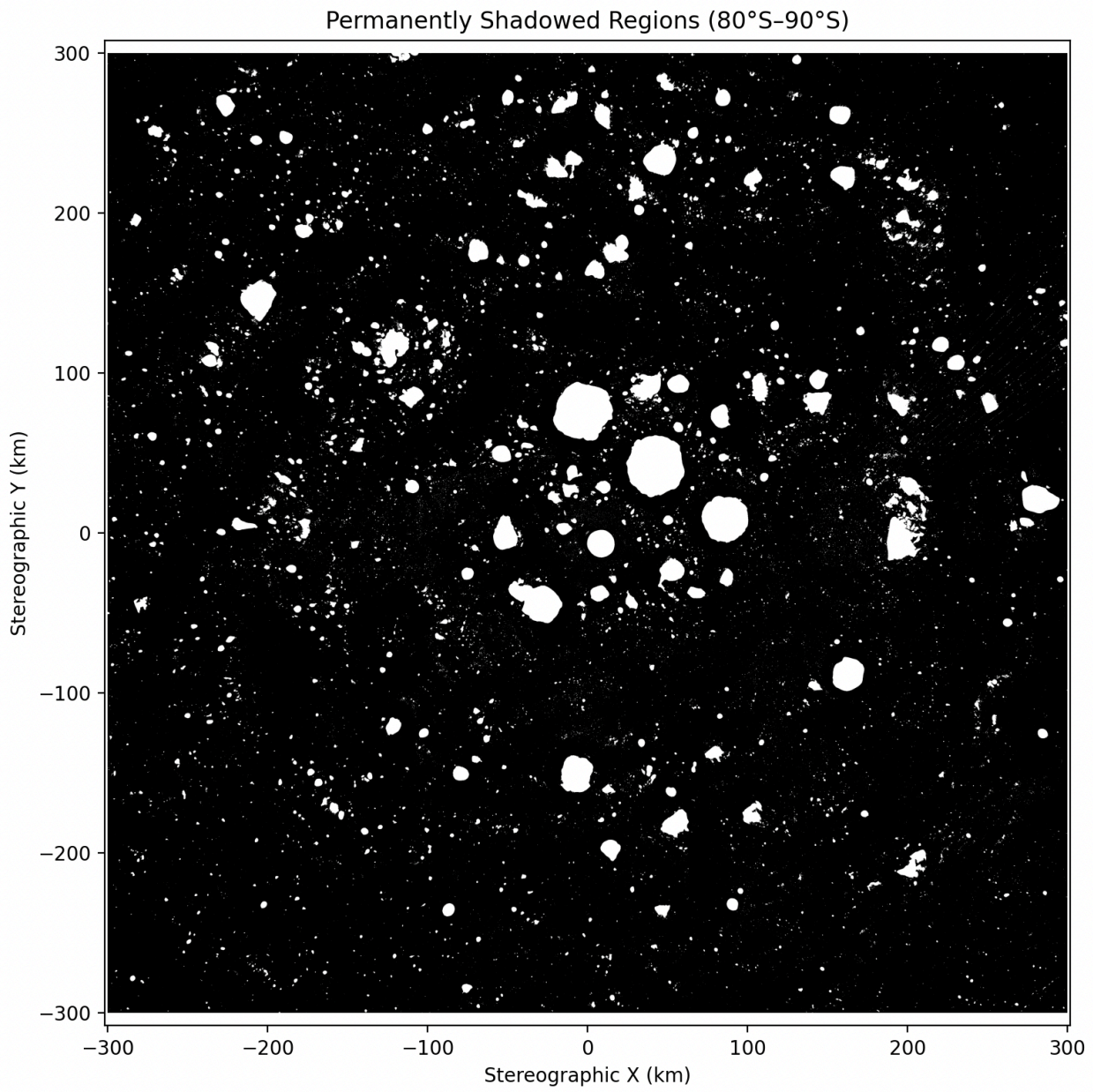

```python
import rasterio
import numpy as np
import matplotlib.pyplot as plt

# Load the sky visibility TIFF
file_path = "SKYV_65S_240M.tiff"
with rasterio.open(file_path) as src:
    sky_data = src.read(1)
    transform = src.transform
    nodata = src.nodata

# Handle no-data
sky_data = np.where(sky_data == nodata, np.nan, sky_data)

# --- Step 1: Normalize values ---
# First check min/max (you can print these)
```

```
min_val, max_val = np.nanmin(sky_data), np.nanmax(sky_data)

# Normalize to 0-1 range (if it's not already)
# If it's already in 0-1 or 0-100, we'll clamp it
if max_val > 100:
   sky_data = (sky_data - min_val) / (max_val - min_val)
else:
   sky_data = np.clip(sky_data, 0, 1)

# --- Step 2: Convert to percent ---
sky_data *= 100  # Now values range from 0-100

# Convert coordinates to km
rows, cols = sky_data.shape
x = np.arange(cols) * transform[0] + transform[2]
y = np.arange(rows) * transform[4] + transform[5]
x_km = x / 1000
y_km = y / 1000
x_coords, y_coords = np.meshgrid(x_km, y_km)

# --- Step 3: Plot ---
fig, ax = plt.subplots(figsize=(10, 10))

img = ax.pcolormesh(x_coords, y_coords, sky_data, cmap="inferno", shading="auto",
vmin=0, vmax=100)
cbar = plt.colorbar(img, ax=ax, label="Sky Visibility (%)")

ax.set_xlim(-300, 300)
ax.set_ylim(-300, 300)
ax.set_aspect("equal")
ax.set_title("Sky Visibility at Lunar South Pole (80°S–90°S)")
ax.set_xlabel("Stereographic X (km)")
ax.set_ylabel("Stereographic Y (km)")

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



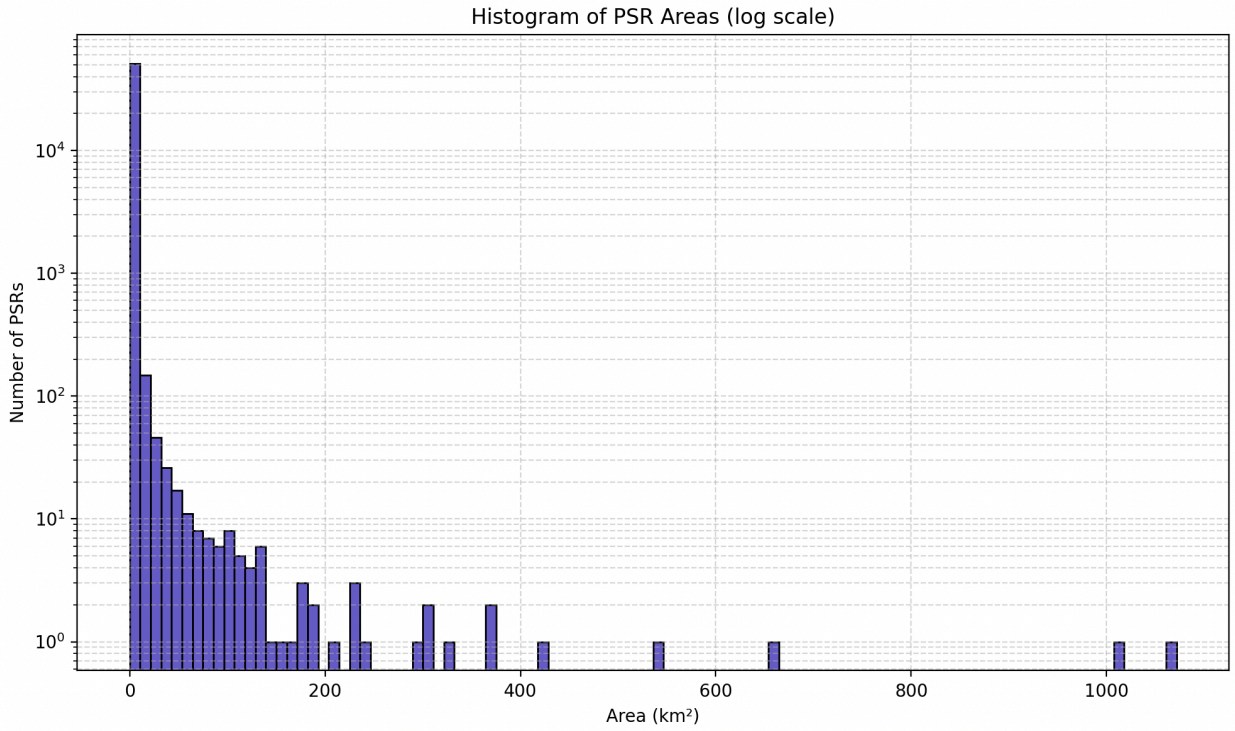

```
import rasterio
import matplotlib.pyplot as plt
import numpy as np

# Load the PSR image (TIFF)
filename = "LPSR_75S_120M_201608.tiff"
with rasterio.open(filename) as src:
   psr_data = src.read(1)
   transform = src.transform

# Convert to km
height, width = psr_data.shape
cols, rows = np.meshgrid(np.arange(width), np.arange(height))
```

```
xs, ys = rasterio.transform.xy(transform, rows, cols)
xs = np.array(xs) / 1000  # m → km
ys = np.array(ys) / 1000

# Mask for central 600x600 km region (-300 to +300 km)
mask = (xs >= -300) & (xs <= 300) & (ys >= -300) & (ys <= 300)

# Crop the arrays based on the mask bounding box
valid_rows, valid_cols = np.where(mask)
row_min, row_max = valid_rows.min(), valid_rows.max()
col_min, col_max = valid_cols.min(), valid_cols.max()

cropped_psr = psr_data[row_min:row_max+1, col_min:col_max+1]
cropped_xs = xs[row_min:row_max+1, col_min:col_max+1]
cropped_ys = ys[row_min:row_max+1, col_min:col_max+1]

# Plot tightly cropped map
plt.figure(figsize=(8, 8))
plt.imshow(cropped_psr, cmap="gray",
          extent=(cropped_xs.min(), cropped_xs.max(), cropped_ys.min(),
cropped_ys.max()))
plt.title("Permanently Shadowed Regions (80°S-90°S)")
plt.xlabel("Stereographic X (km)")
plt.ylabel("Stereographic Y (km)")
plt.axis("equal")
plt.grid(False)
plt.tight_layout()
plt.show()
```




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import rasterio
from scipy import ndimage

# === Load PSR TIFF ===
with rasterio.open("LPSR_75S_120M_201608.tiff") as src:
   psr_mask = src.read(1)
   transform = src.transform
```

```python
# === Label PSRs ===
labeled_array, num_features = ndimage.label(psr_mask > 0)

# === Area per pixel (in km²) ===
pixel_area_km2 = (120 / 1000) ** 2  # 120m pixels

# === Calculate area of each PSR ===
psr_ids, counts = np.unique(labeled_array[labeled_array > 0], return_counts=True)
areas_km2 = counts * pixel_area_km2

# === Create DataFrame ===
psr_df = pd.DataFrame({
    "PSR_ID": psr_ids,
    "Area_km2": areas_km2
}).sort_values(by="Area_km2", ascending=False).reset_index(drop=True)

# === Print Summary ===
print(f"📦 Total PSRs: {len(psr_df)}")
print(f" 🔹 Min Area: {psr_df['Area_km2'].min():.5f} km²")
print(f" 🔹 Max Area: {psr_df['Area_km2'].max():.2f} km²")
print(f" 🔹 Median Area: {psr_df['Area_km2'].median():.5f} km²")

# Optional: How many PSRs are above key thresholds?
thresholds = [0.5, 1, 2, 5, 10]
for t in thresholds:
    count = (psr_df["Area_km2"] >= t).sum()
    print(f"✅ PSRs ≥ {t} km²: {count}")

# === Plot Histogram of PSR Areas ===
plt.figure(figsize=(10, 6))
plt.hist(psr_df["Area_km2"], bins=100, color="slateblue", edgecolor="black", log=True)
plt.title("Histogram of PSR Areas (log scale)")
plt.xlabel("Area (km²)")
plt.ylabel("Number of PSRs")
plt.grid(True, which="both", linestyle="--", alpha=0.5)
plt.tight_layout()
plt.savefig("psr_area_histogram.png", dpi=300)
plt.show()
```



(moon_env) (base) thomasleeds@dhcp-10-250-74-101 ~ %
/Users/thomasleeds/opt/anaconda3/envs/moon_env/bin/python /Users/thomasleeds/moon_plot.py
📦 Total PSRs: 51562
🔹 Min Area: 0.01440 km²
🔹 Max Area: 1071.73 km²
🔹 Median Area: 0.01440 km²
✅ PSRs ≥ 0.5 km²: 3710
✅ PSRs ≥ 1 km²: 2306
✅ PSRs ≥ 2 km²: 1378
✅ PSRs ≥ 5 km²: 643
✅ PSRs ≥ 10 km²: 333
2025-04-06 16:55:35.118 python[38660:14021639] +[IMKClient subclass]: chose IMKClient_Modern
2025-04-06 16:55:35.119 python[38660:14021639] +[IMKInputSession subclass]: chose IMKInputSession_Modern
(moon_env) (base) thomasleeds@dhcp-10-250-74-101 ~ %