



EXPLORING MODERN AND HISTORICAL EFFECTIVE RAINFALL PATTERNS FOR CAVE STUDIES: A GEOSPATIAL ANALYSIS APPROACH

Integrating Climate Data and Geospatial Programming Techniques to
Understand Water Infiltration Dynamics



MAY 4, 2024

UNIVERSITY OF IOWA, GEOSPATIAL PROGRAMMINGCOURSE: 5055
Instructor Caglar Koylu

Introduction:

The aim of this project is to harness geospatial programming techniques to integrate two key datasets: the Global Aridity Index and Potential Evapotranspiration Database (Zomer, et al., 2022) and the WorldClim 2 climate surfaces (Fick & Hijmans, 2017). These datasets will be utilized to generate comprehensive maps encompassing temperature, precipitation, and effective rainfall. Finally, modern effective rainfall will be calculated for 2023. The overarching objective is to employ these mapped variables to support cave studies, particularly in calculating effective rainfall for selected cave systems. Effective rainfall, defined as the disparity between precipitation and potential evapotranspiration (PET), holds significance in understanding water infiltration patterns within cave environments. To calculate modern effective rainfall you will need temperature and precipitation data from weather stations. For example in this project, two weather stations with data for Perryville, Missouri were used.

Data Requirements:

Global Aridity Index and Potential Evapotranspiration Database (Version 3): This dataset provides essential information regarding potential evapotranspiration and aridity index on a global scale. The data will serve as a fundamental component in assessing water availability and climatic conditions.

Zomer, R.J., Xu, J., & Trabucco, A. (2022). Version 3 of the Global Aridity Index and Potential Evapotranspiration Database. *Scientific Data*, 9(1), 409. [DOI: 10.1038/s41597-022-01493-1](https://doi.org/10.1038/s41597-022-01493-1)

WorldClim 2 Climate Surfaces: The WorldClim 2 dataset offers high-resolution climate data, including temperature and precipitation variables. These data layers are crucial for understanding the climatic context of the study area and for calculating effective rainfall.

Fick, S.E., & Hijmans, R.J. (2017). WorldClim 2: new 1-km spatial resolution climate surfaces for global land areas. *International Journal of Climatology*, 37(12), 4302-4315. [DOI:10.1002/joc.5086](https://doi.org/10.1002/joc.5086)

Data from Proximal Stations: Temperature and precipitation data will be derived from proximal stations to calculate modern PET and effective rainfall. Specifically, stations:

KMOPERRY49 (37.69°N, -89.87°W)

PERRYVILLE WATER PLT, MO (37.73°N, -89.92°W)

Methods:

PET Calculation: PyETo is a Python library for calculating reference crop evapotranspiration (ET_o), sometimes referred to as potential evapotranspiration (PET). The library provides numerous functions for estimating missing meteorological data. For estimating ET_o/PET the Thornthwaite (Thornthwaite, 1948) method will be implemented.

Effective Rainfall Calculation: This will be done in Excel by subtracting potential evapotranspiration from precipitation values. This calculation will provide insights into the actual water input available for infiltration into the selected cave systems.

Historic Effective Rainfall Calculation: The Historic Effective Rainfall Calculation is a little more complicated, because it requires the subtraction of two raster layers that need to be clipped and converted to a local UTM to avoid errors in the calculation that might result in wrong conclusions about differences in the historic and modern effective rainfall.

Workflow and Results:

First of all, we will calculate annual PET. An example of the code is provided below.

Here's a step-by-step guide based on the Thornthwaite (1948) equation example for estimating monthly Potential Evapotranspiration (PET) in 2023 for Perryville, Missouri from the first weather station KMOPERRY49 (latitude 37.69 degrees N):

1. **Convert Latitude to Radians:**

- Convert the latitude from degrees to radians using the deg2rad function.

2. **Calculate Monthly Mean Daylight Hours:**

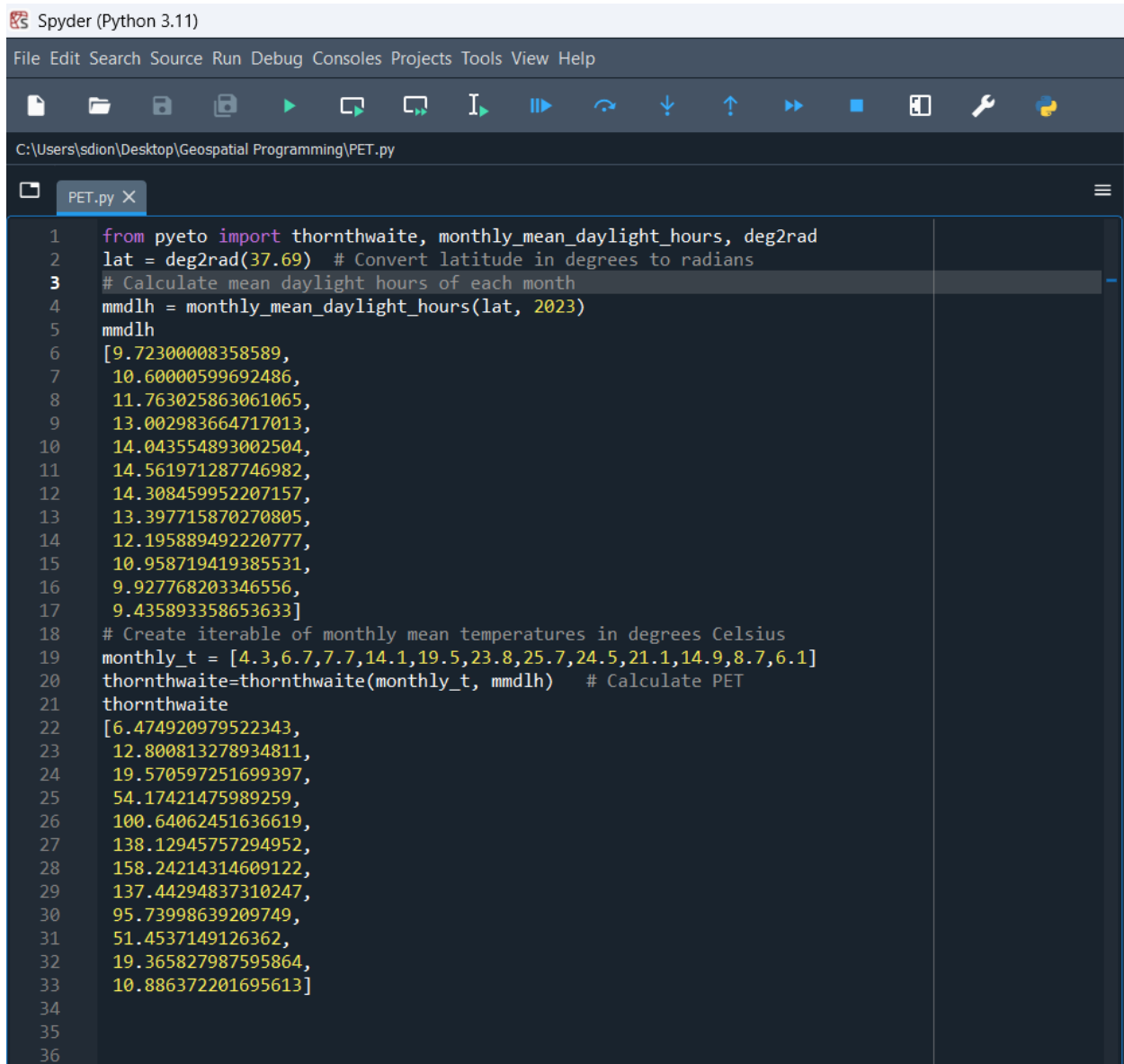
- Use the monthly_mean_daylight_hours function to calculate the mean daylight hours for each month of the year based on the converted latitude and the year (2023 in this case).

3. Collect Monthly Mean Temperature Data:

- Prepare a list of monthly mean temperatures in degrees Celsius for Perryville in 2023.

4. Apply Thornthwaite Equation:

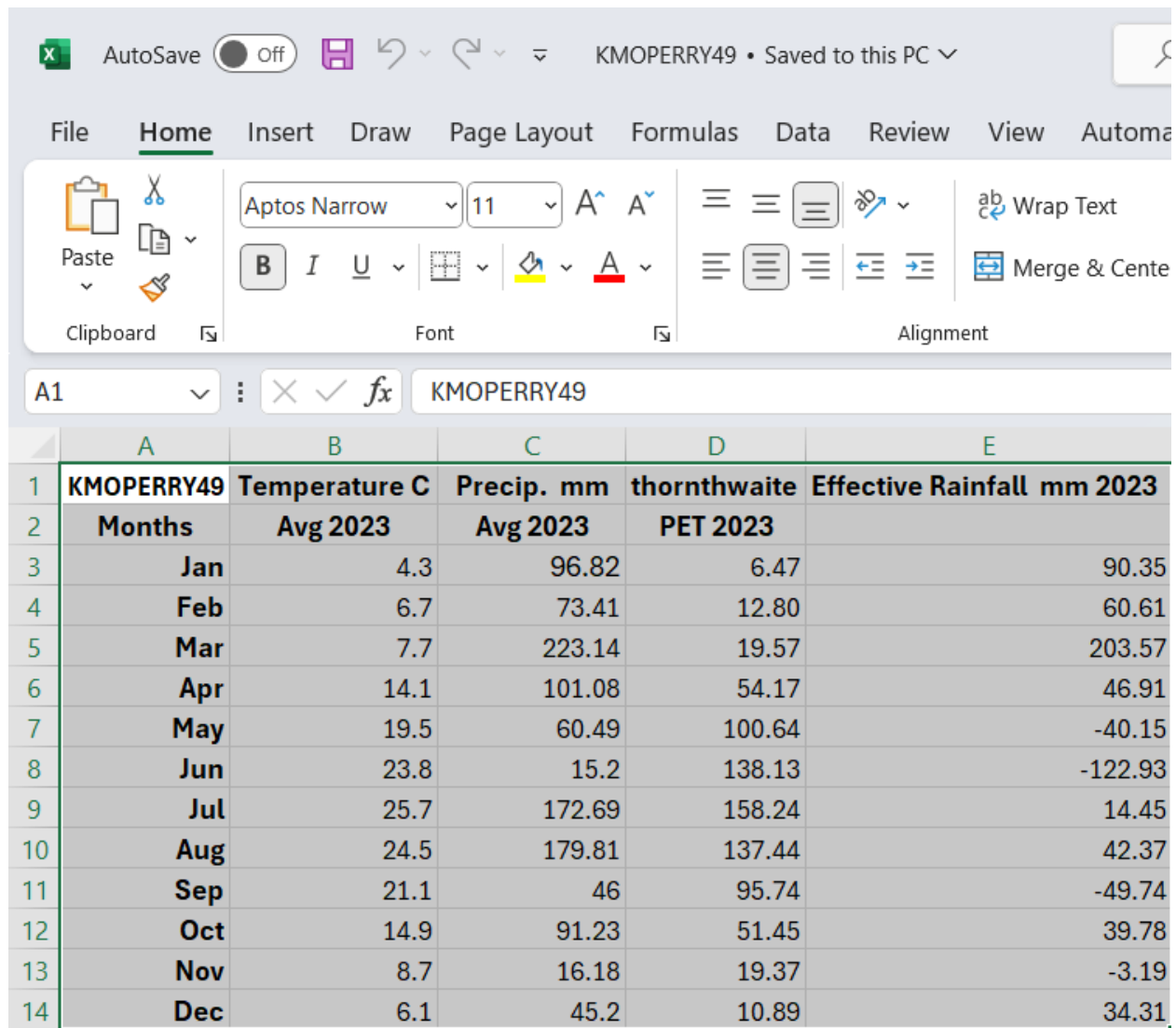
- Utilize the thornthwaite function, providing the monthly mean temperatures and the calculated mean daylight hours for each month as inputs. This function will compute the PET for each month.



```
1 from pyeto import thornthwaite, monthly_mean_daylight_hours, deg2rad
2 lat = deg2rad(37.69) # Convert latitude in degrees to radians
3 # Calculate mean daylight hours of each month
4 mmdlh = monthly_mean_daylight_hours(lat, 2023)
5 mmdlh
6 [9.72300008358589,
7  10.60000599692486,
8  11.763025863061065,
9  13.002983664717013,
10 14.043554893002504,
11 14.561971287746982,
12 14.308459952207157,
13 13.397715870270805,
14 12.195889492220777,
15 10.958719419385531,
16 9.927768203346556,
17 9.435893358653633]
18 # Create iterable of monthly mean temperatures in degrees Celsius
19 monthly_t = [4.3,6.7,7.7,14.1,19.5,23.8,25.7,24.5,21.1,14.9,8.7,6.1]
20 thornthwaite=thornthwaite(monthly_t, mmdlh) # Calculate PET
21 thornthwaite
22 [6.474920979522343,
23 12.800813278934811,
24 19.570597251699397,
25 54.17421475989259,
26 100.64062451636619,
27 138.12945757294952,
28 158.24214314609122,
29 137.44294837310247,
30 95.73998639209749,
31 51.4537149126362,
32 19.365827987595864,
33 10.886372201695613]
34
35
36
```

By following these steps, you can estimate monthly PET using the Thornthwaite equation for a specific location based on mean monthly temperature and daylight hours.

Now that we have PET we can subtract it from precipitation to find the effective rainfall. An example is provided below:



	A	B	C	D	E
1	KMOPERRY49	Temperature C	Precip. mm	thornthwaite	Effective Rainfall mm 2023
2	Months	Avg 2023	Avg 2023	PET 2023	
3	Jan	4.3	96.82	6.47	90.35
4	Feb	6.7	73.41	12.80	60.61
5	Mar	7.7	223.14	19.57	203.57
6	Apr	14.1	101.08	54.17	46.91
7	May	19.5	60.49	100.64	-40.15
8	Jun	23.8	15.2	138.13	-122.93
9	Jul	25.7	172.69	158.24	14.45
10	Aug	24.5	179.81	137.44	42.37
11	Sep	21.1	46	95.74	-49.74
12	Oct	14.9	91.23	51.45	39.78
13	Nov	8.7	16.18	19.37	-3.19
14	Dec	6.1	45.2	10.89	34.31

After the effective rainfall is calculated for the second station a csv file is created with the Station_ID, Geographic Coordinates and the calculated Average Annual Effective Rainfall as shown below.

	A	B	C	D	E
1	Station_ID	Annual_Effective_Rainfall_mm_2023	Latitude	Longitude	Elevation_m
2	PERRYVILLE WATER PLT	26.83	37.73	-89.92	153.0096
3	KMPERRY49	26.36	37.69	-89.87	188

For the next step we need to download annual ET0 and Precipitation for the period 1970-2000 from the global datasets. After the files are downloaded the user should open the jupyter notebook in ArcGIS Pro and follow the procedure outlined in there. Below a short step by step guide is provided:

Set up Environment and Licenses:

Import necessary modules from arcpy. Check out Spatial Analyst and Image Analyst extensions licenses.

Set Analysis Environments:

Set the workspace to the appropriate geodatabase. Enable overwriting output.

```
In [1]: ▶ # Import system modules
import arcpy
from arcpy.sa import *
from arcpy.ia import *
```

```
In [2]: ▶ # Check out the ArcGIS Spatial Analyst extension license
arcpy.CheckOutExtension("Spatial")
```

Out[2]: 'CheckedOut'

```
In [3]: ▶ # Set the analysis environments
arcpy.env.workspace = "C:/Users/sdion/Desktop/Geospatial Programming/Geospatial_Progra
# Check out the ArcGIS Image Analyst extension license
arcpy.CheckOutExtension("ImageAnalyst")
```

Out[3]: 'CheckedOut'

```
In [4]: ▶ arcpy.env.overwriteOutput = True
print(arcpy.env.workspace)
```

C:/Users/sdion/Desktop/Geospatial Programming/Geospatial_Programming_Project.gdb

Import Data into Geodatabase:

Convert raster and shapefile data to the geodatabase. Import CSV file as point data using XYTableToPoint tool.


```
In [5]: ▶ # Import tif, shapefiles and csv files into the geodatabase
from arcpy import env
arcpy.conversion.RasterToGeodatabase("et0_v3_yr.tif;wc2.1_30s_bio_12.tif",
                                     "Geospatial_Programming_Project.gdb", "")
arcpy.management.XYTableToPoint("Combined_Stations.csv", "Geospatial_Programming_Proje
arcpy.conversion.FeatureClassToFeatureClass("USA_Counties.shp",
                                             "Geospatial_Programming_Project.gdb",
                                             "USA_Counties")
```

Out[5]:


Messages

Project Feature Classes:

Project point and polygon feature classes to a desired coordinate system (NAD 1983 UTM Zone 15N).

In [6]:  *# Input DEM raster file (TIFF format) & Set local variables*

```
Annual_ET0 = "et0_v3_yr.tif"
Annual_Prec = "wc2.1_30s_bio_12.tif"
USA_Counties = "USA_Counties.shp"
```

In [7]:  *# Define input and output paths*

```
input_table = "Geospatial_Programming_Project.gdb/Weather_Stations"
output_table = "Geospatial_Programming_Project.gdb/stations_utm"

# Define the output coordinate system (NAD 1983 UTM Zone 15N)
output_coordinate_system = arcpy.SpatialReference("NAD 1983 UTM Zone 15N")

# Project the feature class to the desired coordinate system
stations_utm = arcpy.management.Project(input_table, output_table, output_coordinate_s

# You can also print the output variable if you want to confirm its name and location
print(stations_utm)
```

Geospatial_Programming_Project.gdb\stations_utm

hasM	False
hasZ	False
▼ fields	
OBJECTID	OID
Shape	Geometry
Station_ID	String
Annual_Effective_Rainfall_mm_2023	Double
Latitude	Double
Longitude	Double
Elevation_m	Double
Field6	String
Field7	String
Field8	String
Field9	String
▼ spatialReference	
name (Projected Coordinate System)	NAD_1983_UTM_Zone_15N
factoryCode (WKID)	26915
linearUnitName (Linear Unit)	Meter
spatialReference.GCS	
name (Geographic Coordinate System)	GCS_North_American_1983
factoryCode (WKID)	4269
angularUnitName (Angular Unit)	Degree
datumName (Datum)	D_North_American_1983

For additional help, see [arcpy.Describe](#)

Project Raster Data:

Project raster data (Annual_ET0 and Annual_Prec) to the desired coordinate system (NAD 1983 UTM Zone 15N). This conversion ensures that the spatial data is accurately represented in the context of the new reference frame. It typically involves mathematical transformations to adjust for differences in datum parameters, such as the ellipsoid used for modeling the Earth's shape and the orientation of the coordinate axes. The extent and coordinates of features within the TIFF files will be adjusted accordingly to match the new reference system.

```
In [9]: ► # output data
         counties_utm = "USA_Counties_utm.shp"

         # create a spatial reference object for the output coordinate system
         out_coordinate_system = arcpy.SpatialReference('NAD 1983 UTM Zone 15N')

         # run the tool
         arcpy.Project_management(USA_Counties, counties_utm, out_coordinate_system)
```

Out[9]:

Messages

▼ spatialReference

name (Projected Coordinate System)	NAD_1983_UTM_Zone_15N
factoryCode (WKID)	26915
linearUnitName (Linear Unit)	Meter

spatialReference.GCS

name (Geographic Coordinate System)	GCS_North_American_1983
factoryCode (WKID)	4269
angularUnitName (Angular Unit)	Degree
datumName (Datum)	D_North_American_1983

For additional help, see [arcpy.Describe](#)

```
In [12]: ► # Define the input raster and output raster paths
input_raster = Annual_ET0
output_raster = "Annual_ET0_utm"

# Remove invalid characters from the output raster name
output_raster = arcpy.ValidateTableName(output_raster)

# Define the spatial reference using a factory code
spatial_ref = arcpy.SpatialReference(26915) # Factory code for NAD 1983 UTM Zone 15N

# Project the raster
arcpy.ProjectRaster_management(input_raster, output_raster, spatial_ref)
```

Out[12]:

Messages

```
In [13]: ► # Define the input raster and output raster paths
input_raster1 = Annual_Prec
output_raster1 = "Annual_Prec_utm"

# Remove invalid characters from the output raster name
output_raster = arcpy.ValidateTableName(output_raster)

# Define the spatial reference using a factory code
spatial_ref = arcpy.SpatialReference(26915) # Factory code for NAD 1983 UTM Zone 15N

# Project the raster
arcpy.ProjectRaster_management(input_raster1, output_raster1, spatial_ref)
```

Out[13]:

Messages

Select and Clip Raster Data:

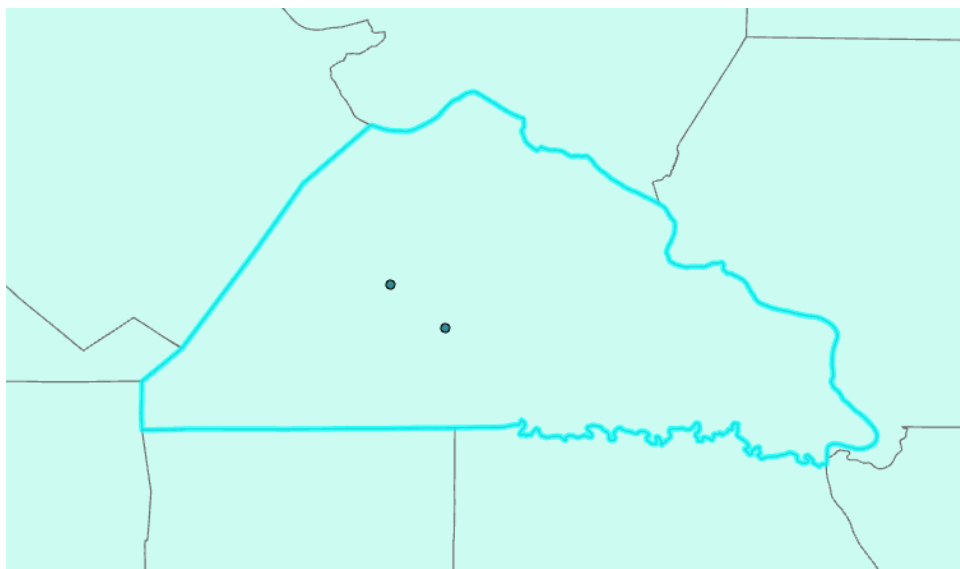
Select a specific county (Perry county in Missouri) using a SQL query. Clip the projected rasters to the extent of the selected county.

```
In [14]: ► # Select Perry county in Missouri by using the FIPS code and the name of the county
sql = "FIPS = '29157'"
perry_lyr = arcpy.management.SelectLayerByAttribute(counties_utm, "NEW_SELECTION", sql)

# Write the selected features to a new feature class
arcpy.management.CopyFeatures(perry_lyr, 'perry')
```

Out[14]:

Messages

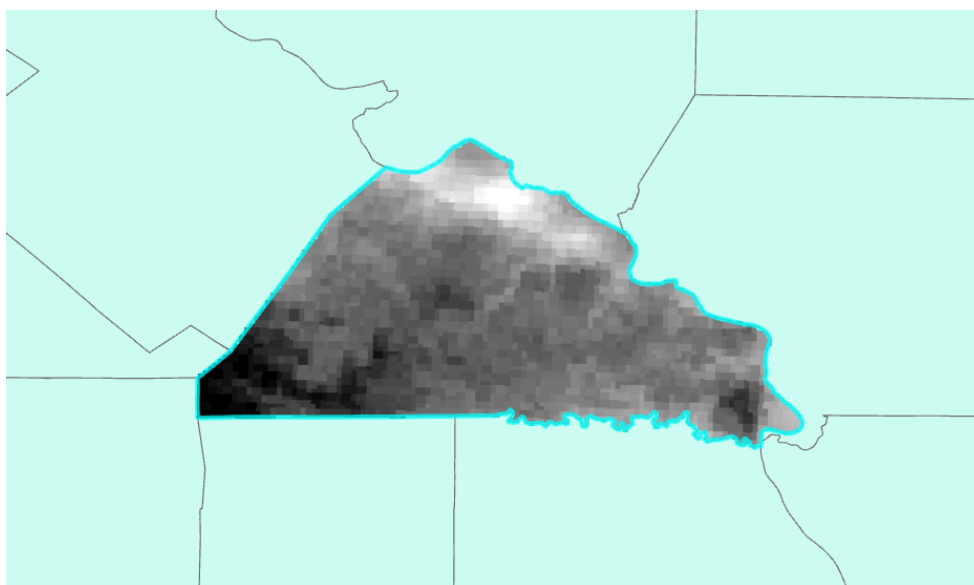


```
In [27]: # Set input and output raster paths
output_raster2 = "AnET0"

# Clip the raster
arcpy.management.Clip(output_raster, "#", output_raster2, "perry", "#", "ClippingGeome
```

Out[27]:

Messages



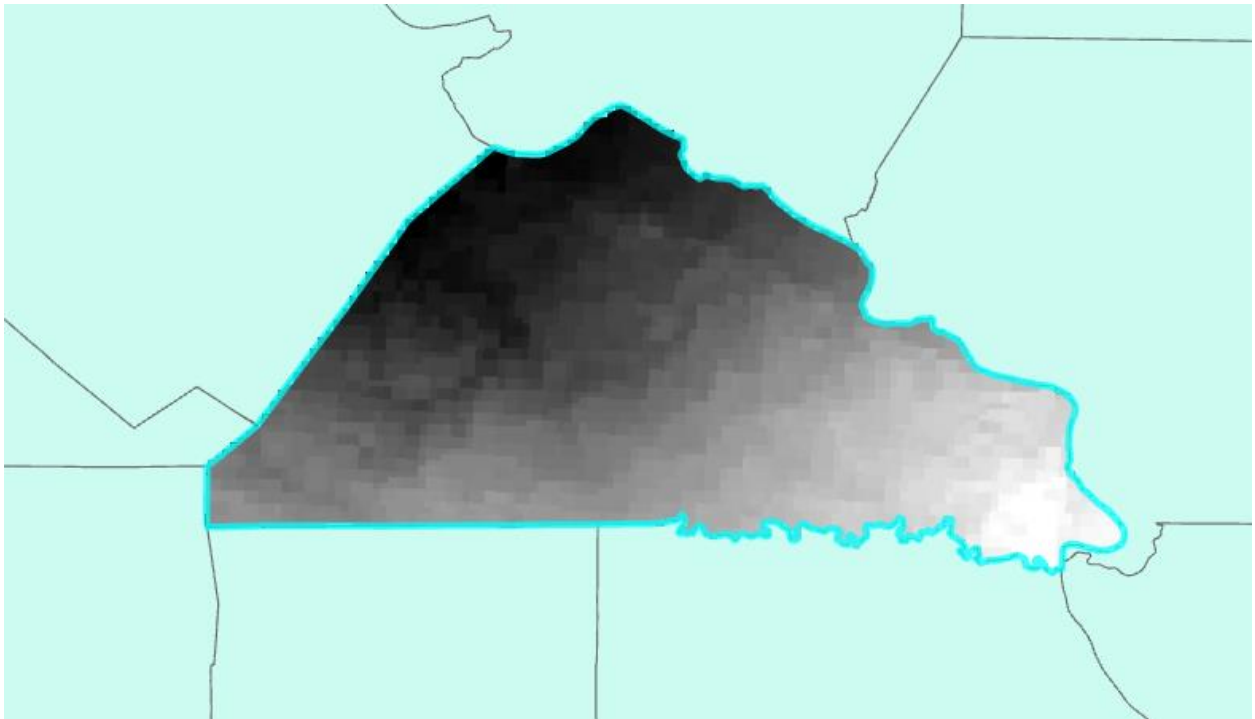
Succeeded at Saturday, May 4, 2024 1:09:55 PM (Elapsed Time: 4.05 seconds)

```
In [17]: # Set input and output raster paths
output_raster3 = "AnPrec"

# Clip the raster
arcpy.management.Clip(output_raster1, "#", output_raster3, "perry", "#", "ClippingGeom
```

Out[17]:

Messages



Perform Raster Operations:

Perform raster calculation (subtraction) to find the difference between two rasters (AnPrec_Perry and AnET0_Perry). Save the output raster.

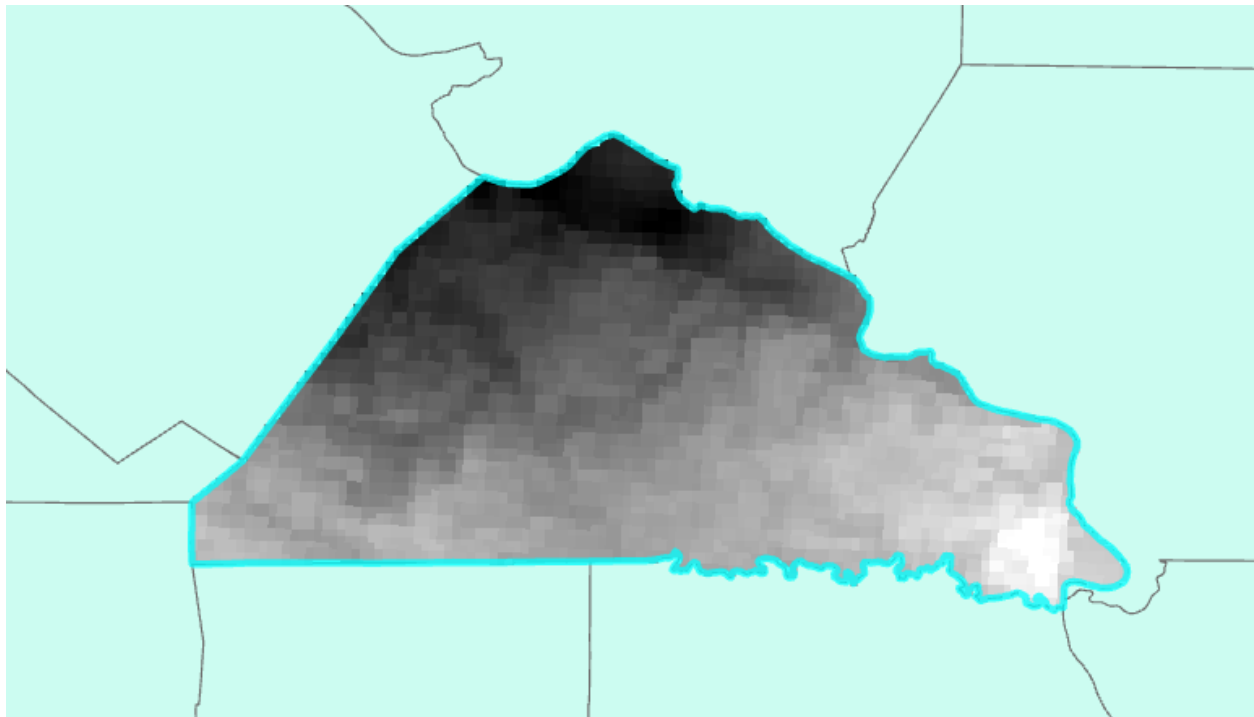
This code performs raster subtraction after clipping two TIFF files representing different spatial datasets. By clipping the rasters to a common extent defined by the "perry" geometry, the areas of interest in both datasets are aligned. Then, the Minus function calculates the difference between the clipped rasters, representing the change or variance between the two datasets within the clipped area. Finally, the result is saved as a new TIFF file named "Eff_Rain_Perry.tif". This process is

correct because it ensures that the analysis is performed only on the overlapping regions of the original datasets, avoiding potential inaccuracies caused by areas outside the common extent.

```
In [29]: ► # Set local variables
AnET0_Perry = output_raster2
AnPrec_Perry = output_raster3

In [30]: ► # Execute Minus
outMinus = Minus(AnPrec_Perry, AnET0_Perry)

# Save the output
outMinus.save("C:/Users/sdion/Desktop/Geospatial Programming/Geospatial_Programming_Pr
```



Create Points and Project Coordinates:

Define points using coordinates. Create point geometries and project them to the desired coordinate system (NAD 1983 UTM Zone 15N). Print the projected coordinates.

```
In [31]: ► import arcpy
from arcpy import env
from arcpy.sa import *
```

```
In [32]: ► # Create points using the provided coordinates
point1 = arcpy.Point(-89.791667, 37.666667)
point2 = arcpy.Point(-89.875, 37.75)

# Create point geometries from the points
point_geom1 = arcpy.PointGeometry(point1, arcpy.SpatialReference(4326)) # WGS 1984
point_geom2 = arcpy.PointGeometry(point2, arcpy.SpatialReference(4326)) # WGS 1984

# Define the output coordinate system (NAD 1983 UTM Zone 15N)
output_coordinate_system = arcpy.SpatialReference("NAD 1983 UTM Zone 15N")

# Project the point geometries to the desired coordinate system
projected_point1 = point_geom1.projectAs(output_coordinate_system)
projected_point2 = point_geom2.projectAs(output_coordinate_system)

# Print the projected coordinates
print("Projected coordinates of (-89.791667, 37.666667): ", projected_point1.centroid)
print("Projected coordinates of (-89.875, 37.75): ", projected_point2.centroid)
```

Projected coordinates of (-89.791667, 37.666667): 782991.062738216 4173676.54752523
NaN NaN
Projected coordinates of (-89.875, 37.75): 775329.935151029 4182676.98663146 NaN NaN

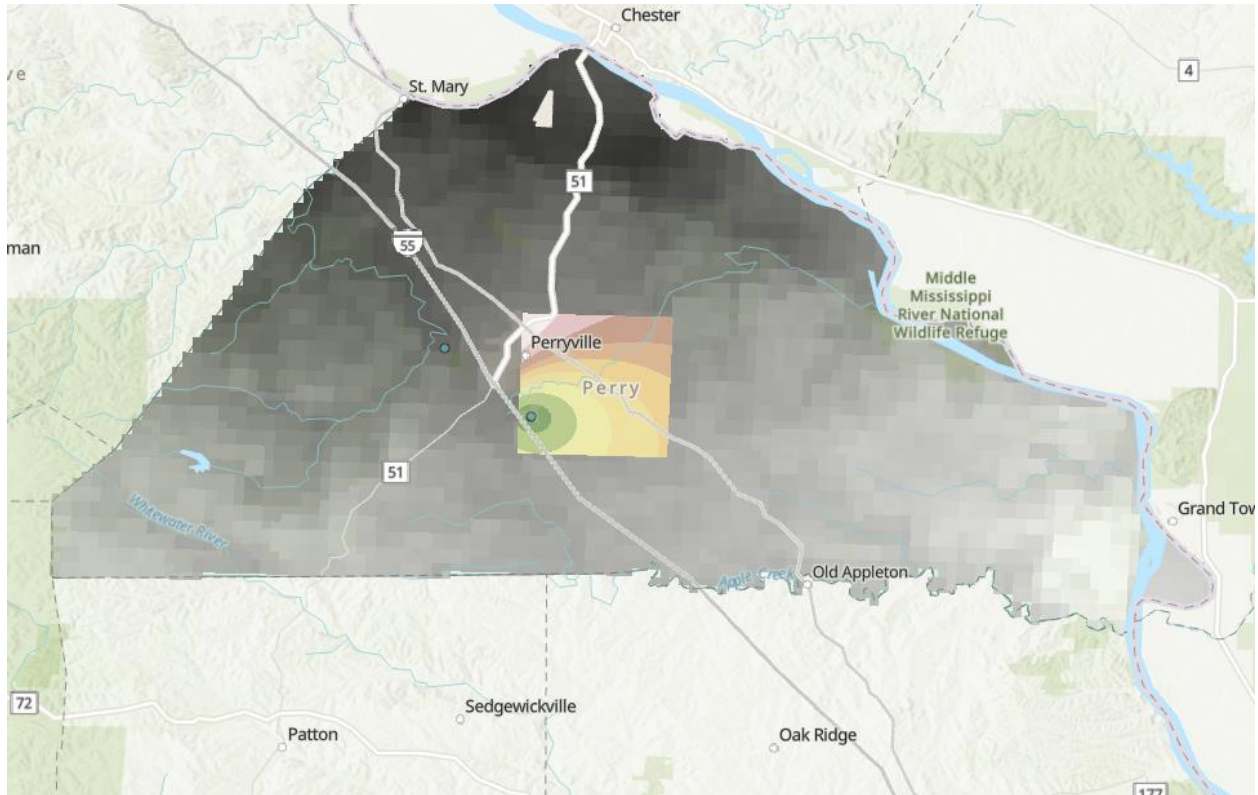
Set Extent and Perform Interpolation:

Configure the spatial environment, then apply IDW interpolation to the point dataset and store the output. Utilize a power of 1 for IDW interpolation, considering the small dataset with just two closely situated weather stations. IDW interpolation employs weights inversely proportional to distance, causing weights to diminish rapidly with increasing distance. The rate of decline in weights hinges on the power value; higher powers result in swifter weight reductions for distant points.

```
In [33]: ► # Set the extent environment
arcpy.env.extent = arcpy.Extent(782991.06, 4173676.54, 775329.93, 4182676.98)

# Perform IDW interpolation
outIDW = arcpy.sa.Idw("Geospatial_Programming_Project.gdb/stations_utm", "Annual_Effec

# Save the result
outIDW.save("Geospatial_Programming_Project.gdb/Idw_Project.shp")
```



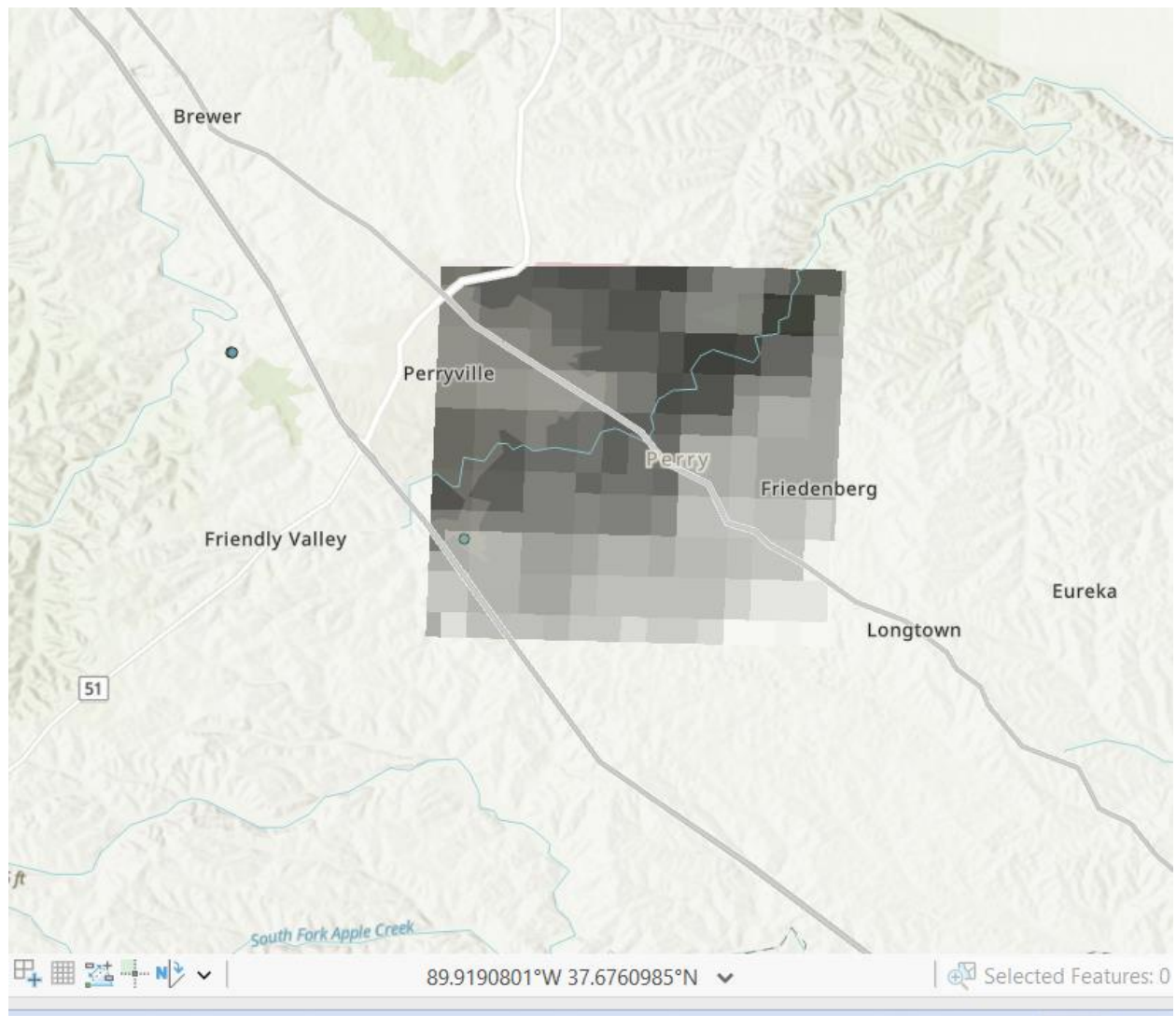
Compute Raster Difference:

Perform raster calculation (subtraction) to find the difference between two rasters (outMinus and outIDW). Save the output raster.

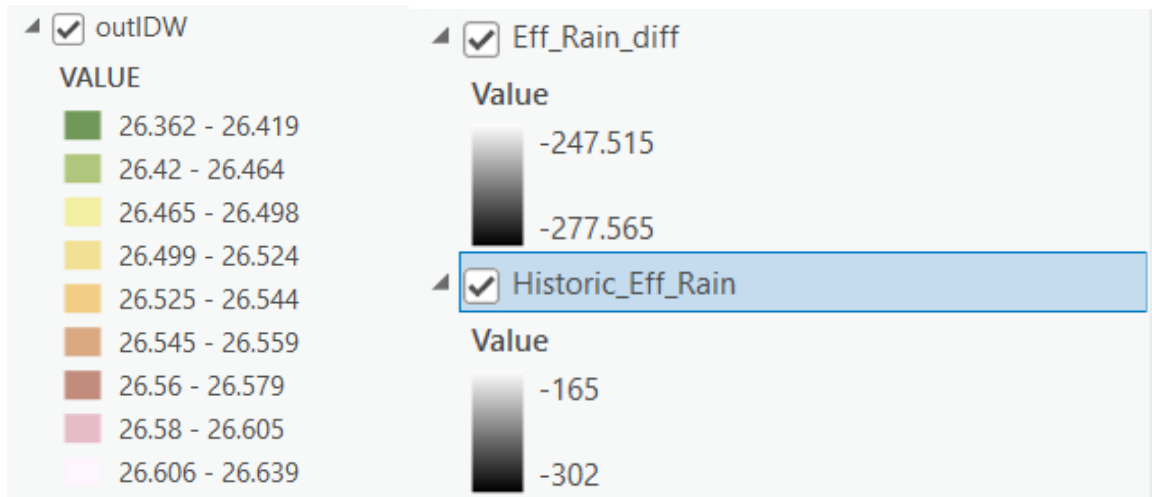
This above and below code snippet demonstrates spatial analysis procedures by first setting the extent environment using `arcpy.Extent` to confine subsequent operations to a defined geographic area of interest. It then conducts IDW interpolation, employing `arcpy.sa.Idw`, to estimate values within the study area based on surrounding point data. The interpolation method chosen, Inverse Distance Weighted (IDW), with a parameter of "1", is suitable for interpolating continuous surfaces like rainfall. The resulting IDW interpolation is saved as a shapefile named "Idw_Project.shp" within the specified geodatabase. These steps ensure accurate spatial analysis and proper management of resulting data for further processing or visualization.

```
In [36]: # Execute Minus to find the difference between the historic and modern effective rainfall
Eff_Rain_diff = Minus(outMinus,outIDW)

# Save the output
outMinus.save("C:/Users/sdion/Desktop/Geospatial Programming/Geospatial_Programming_Pr
```



Conclusions:



In this study, the integration of geospatial programming techniques facilitated the comprehensive analysis of key climatic variables, particularly effective rainfall, essential for understanding water infiltration patterns within cave environments. By harnessing datasets such as the Global Aridity Index and Potential Evapotranspiration Database and the WorldClim 2 climate surfaces, we were able to generate detailed maps encompassing temperature, precipitation, and effective rainfall for the selected region.

The calculated modern effective rainfall for 2023 provided valuable insights into current water availability, crucial for cave studies. The utilization of weather station data allowed for precise estimations, highlighting the significance of local climate conditions in influencing effective rainfall.

Furthermore, the comparison between modern effective rainfall and historical data spanning from 1970 to 2000 revealed intriguing trends. Despite the limitations of analyzing only a single year of modern data, the consistency of results within the historical context provided confidence in our

findings. Particularly noteworthy was the indication of more moist conditions in 2023 compared to the previous decades, suggesting potential shifts in regional climatic patterns.

These findings underscore the importance of ongoing monitoring and analysis of climatic variables, especially in ecologically sensitive environments like caves. The methodology employed in this study can serve as a robust framework for future research endeavors, contributing to a deeper understanding of hydrological processes and their implications for cave ecosystems.