Evening Hade

Lab 3, Part 2

For GIS 5571

In this part of the lab, I will make an ETL for real-time data, clean & project that data, and interpolate it in different ways.

I always start with imports, setting up my project, map, and spatial reference, and defining my output path where everything will get saved.

```
import requests
import arcpy
import numpy as np
import pandas as pd
import os
from datetime import datetime, timedelta

#Set up arcpro project
project = arcpy.mp.ArcGISProject("CURRENT")
map = project.listMaps()[0]
ref = arcpy.SpatialReference(4326)  #WGS 1983

#Path where everything will be saved
outputPath = "C:/ArcGIS/Projects/Lab3-2"

if not os.path.exists(outputPath): #Test that outputPath exists
    os.makedirs(outputPath)
    print("Created output directory: ", outputPath)
else:
    print("Output directory is already assigned to: ", outputPath)
```

```
Output directory is already assigned to:  C:/ArcGIS/Projects/Lab3-2
```

After initial setup, I need to get my data. I made a list of all the stations currently in the NDAWN system. For the URL, I removed the temporal information "year", "start date", and "end date". This means that this URL will default to whatever day it is when you run the code.

```
#URL and get request
urlBase = "https://ndawn.ndsu.nodak.edu/table.csv?station="

stations = ( #List of stations (all of them)

"78,111,98,162,174,142,164,138,161,9,160,224,159,10,229,118,56,165,11,
12,58,"

"13,84,218,55,179,7,186,87,14,15,96,191,16,210,201,137,124,143,17,85,2
26,140,"
```

```
"134,18,136,219,65,104,99,192,19,227,129,20,101,166,178,81,21,97,22,75
,184,2,"

"211,172,139,158,23,157,220,62,86,24,89,126,223,167,93,183,90,25,205,8
3,107,"

"156,77,26,155,70,127,144,27,173,132,28,195,185,29,30,154,31,187,102,3
2,119,"

"4,217,80,33,59,153,105,82,225,34,198,72,135,35,76,120,209,141,109,36,
207,79,"

"193,71,212,37,38,189,39,130,73,188,40,41,54,228,69,194,145,214,113,12
8,42,43,"

"103,171,116,196,88,114,3,163,200,216,64,115,168,67,175,146,170,197,44
,206,133,"

"106,100,121,45,46,61,66,181,74,213,60,199,125,176,177,8,180,204,47,22
1,122,"

"108,5,152,48,151,147,68,169,49,50,91,182,117,63,150,51,6,222,52,92,11
2,131,"
    "123,95,53,203,190,208,57,149,148,202,215,110"
)

#Replace commas with '&station=' for use with requests
andStation = stations.replace(",", "&station=")

#Construct full URL
url = f"{urlBase}{andStation}"

#Additional parameters for 30-day temp normals
params = {
    "variable": "ddavt",
    "ttype": "daily",
    "quick_pick": "30_d"
}

#Extract data
response = requests.get(url, params=params)

if response.status_code == 200:
    data = response.text
    rawPath = os.path.join(outputPath, "NDAWN_data.csv")
    with open(rawPath, "w", encoding="utf-8") as f:
        f.write(data)
    print(f"Data saved to: {rawPath}")
```

```
else:
    print(f"Error: {response.status_code}, {response.text}")
```

```
Data saved to: C:/ArcGIS/Projects/Lab3-2\NDAWN_data.csv
```

I quickly learned that this .csv would need to be manipulated before it could be used. First, I need to remove the random rows with some supplementary information about the table. Then, I need to remove the row with the units. I didn't remove all the string rows because that would leave me without field names.

```
#Clean the data using pandas

csvPath = os.path.join(outputPath, "clean_data.csv")

stringsRemoved = [
    "Data from North Dakota Agricultural Weather Network
https://ndawn.ndsu.nodak.edu",
    "Daily Observation Table for",
    "Flag Definition Line: M - Missing; E - Estimated; N/A - Not
Available"
]

#Clean the CSV file
with open(rawPath, 'r', encoding='utf-8') as infile, open(csvPath,
'w', encoding='utf-8') as outfile:
    for line in infile:
        # Remove any leading/trailing whitespace characters
        line = line.strip()
        # Check if the line starts with any of the specified strings
        if not any(line.startswith(s) for s in stringsRemoved) and
line:
            outfile.write(line + '\n')

#Attempt to read the cleaned CSV into a DataFrame using pandas
try:
    df = pd.read_csv(csvPath, on_bad_lines='skip')
    print("CSV successfully read into DataFrame.")
except pd.errors.ParserError as e:
    print("Error reading CSV into DataFrame:", e)
```

```
CSV successfully read into DataFrame.
```

```
#More data cleaning
#Someone at NDAWN needs a performance review is2g

# Step 2: Remove rows where the "Longitude" field has the value "deg"
df = df[df['Longitude'] != 'deg']
df['Longitude'] = pd.to_numeric(df['Longitude'], errors='coerce')
df['Latitude'] = pd.to_numeric(df['Latitude'], errors='coerce')
df['Elevation'] = pd.to_numeric(df['Elevation'], errors='coerce')
```

```
print("Deleted units row. Converted Longitude, Latitude, and Elevation fields to numeric values.")

# Step 4: Save the cleaned DataFrame back to the CSV
df.to_csv(csvPath, index=False, encoding='utf-8')
print(f"Updated CSV saved at: {csvPath}")
```

```
Deleted units row. Converted Longitude, Latitude, and Elevation fields to numeric values.
Updated CSV saved at: C:/ArcGIS/Projects/Lab3-2\clean_data.csv
```

Now that the data in the .csv can easily be used by Esri with no random string values, I can add the table to the project and turn it into a feature layer:

```
if not arcpy.Exists(csvPath):
    print(f"Error: The file {csvPath} was not found.")
else:
    arcpy.TableToTable_conversion(in_rows=csvPath, out_path=project.defaultGeodatabase, out_name="clean_data")
    print(f"Table added to the project.")
```

```
Table added to the project.
```

```
#Turn the table into a feature layer

arcpy.management.XYTableToPoint(
    in_table=csvPath,
    out_feature_class=r"C:\ArcGIS\Projects\Lab3-2\NDAWN_Points",
    x_field="Longitude",
    y_field="Latitude"
)

print("Converted table to point feature layer.")
```

```
Converted table to point feature layer.
```

With this point feature layer, I can run spatial interpolations.

I did Kernel with Barriers first because I like how its extent "follows" the points without using a polygon/polyline barrier layer. I chose it because it predicts values and errors while also having one value per location.

```
#Kernel interpolation with barriers

arcpy.ga.KernelInterpolationWithBarriers(
    in_features="NDAWN_Points",
    z_field="Avg_Temp",
    out_ga_layer=None,
    out_raster=r"C:\ArcGIS\Projects\Lab3-2\kernelNDAWN",
```

```
    cell_size=0.021010908,
    in_barrier_features=None,
    kernel_function="POLYNOMIAL5",
    bandwidth=None,
    power=1,
    ridge=50,
    output_type="PREDICTION"
)

print(f"Created kernel interpolation with barriers and saved to
{outputPath}.")
```

```
Created kernel interpolation with barriers and saved to
C:/ArcGIS/Projects/Lab3-2.
```

Next is IDW, which I have done a lot of and see as the simplest recipe to make a spatial interpolation:

```
#Interpolation with IDW

arcpy.ddd.Idw(
    in_point_features="NDAWN_Points",
    z_field="Avg_Temp",
    out_raster=r"C:\ArcGIS\Projects\Lab3-2\idwNDAWN",
    power=2,
    search_radius="VARIABLE 12",
    in_barrier_polyline_features=None
)

print(f"Created inverse distance weighted interpolation and saved to
{outputPath}.")
```

```
Created inverse distance weighted interpolation and saved to
C:/ArcGIS/Projects/Lab3-2.
```

Now for kriging, which is a stochastic interpolation:

```
#Interpolation with ordinary Kriging
#I wrote that is "kringing" at first and I feel like I cannot be the
only one to ever do that

with arcpy.EnvManager(scratchWorkspace=outputPath):
    ordinaryNDAWN = arcpy.sa.Kriging(
        in_point_features="NDAWN_Points",
        z_field="Avg_Temp",
        kriging_model="Spherical # # # #", #This populated
automatically
        search_radius="VARIABLE 12",
```

```
        out_variance_prediction_raster=None
    )
    ordinaryNDAWN.save(r"C:\ArcGIS\Projects\Lab3-2\ordinaryNDAWN")

print(f"Created ordinary spherical kriging interpolation and saved to
{outputPath}.")
```

```
---------------------------------------------------------------------------
-----
RuntimeError                              Traceback (most recent call
last)
In  [18]:
Line 12:    ordinaryNDAWN.save(r"C:\ArcGIS\Projects\Lab3-2\
ordinaryNDAWN")

RuntimeError: Invalid pointer
---------------------------------------------------------------------------
-----
```

The above produces an error message every time I re-run it, but it DOES work the first time around.

Next, I wanted to see if there was a difference between the two main kinds of kriging offered by ArcGIS Pro, ordinary (above) and universal:

```
#Interpolation with universal Kriging

with arcpy.EnvManager(scratchWorkspace=outputPath):
    univNDAWN = arcpy.sa.Kriging(
        in_point_features="NDAWN_Points",
        z_field="Avg_Temp",
        kriging_model="LinearDrift 0.021011 # # #", #This populated
automatically
        search_radius="VARIABLE 12",
        out_variance_prediction_raster=None
    )
    univNDAWN.save(r"C:\ArcGIS\Projects\Lab3-2\univNDAWN")

print(f"Created universal linear drift kriging interpolation and saved
to {outputPath}.")
```

```
Created universal linear drift kriging interpolation and saved to
C:/ArcGIS/Projects/Lab3-2.
```

That's all, folks!