Title: **LAB IV.**

Notice: Dr. Bryan Runck
Author: Michael Felzan
Date: April 22, 2021
Project Repository**:** ***https://github.com/fezfelzan/GIS5572.git***

Abstract

This report outlines an ETL process for extracting tabular temperature data from the NDAWN API using Python "get" requests in a Jupyter Notebook environment. The methods demonstrated here explain techniques for organizing, cleaning, aggregating, and spatially mapping this data. This ETL additionally programmatically generates interpolated temperature surface maps based on a set of points with associated 'z-attributes.' The results of these methods, and a discussion of interpolation algorithms which considers the pre-existing literature on spatial interpolation are provided at the end of this report.

Problem Statement

The objective of Lab 4 is to create a fully functional real-time ETL which downloads 30-day temperature data from the NDAWN API, and creates interpolated temperature surface maps for every NDAWN stations average monthly temperature, maximum monthly temperature, and minimum monthly temperature.

Table 1. Defining the Problem

#	Requirement	Defined As	Spatial Data	Attribute Data	Dataset	Preparation
1	Weather Stations	Points which reflect the temperature at a given location for a given time.	Point geometry	Weather data, Lat/long, Station name		Created via XY Table to Point
2	Temperature	Daily or Monthly Fahrenheit temperature values recorded by weather stations		Min Temp/ Max Temp/ Ave Temp	(NDAWN)	CSV Manipulation; pandas "groupby" function

(No Input Data – all data generated with code through web scraping & get requests)

Methods

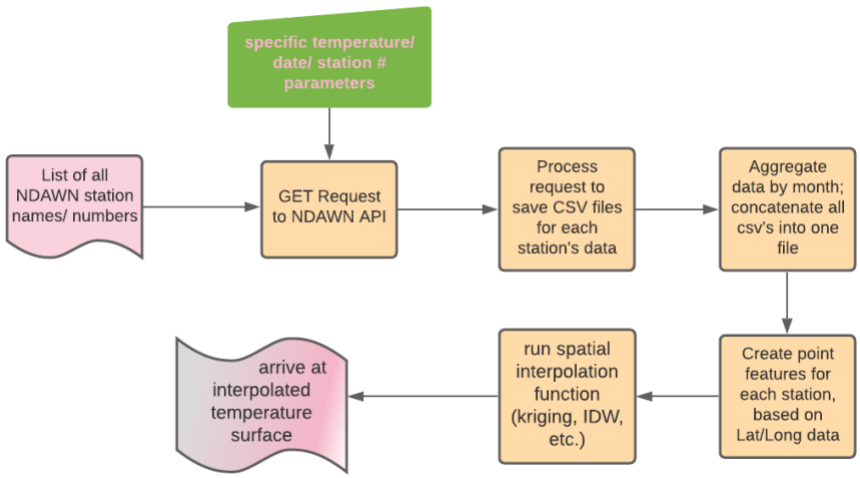


Figure 1. *Flow Diagram depicting the ETL process of retrieving real-time temperature data from NDAWN API via "get" requests. The tabular data retrieved from these requests is cleaned, combined, and aggregated using the Python 'csv' module. This ETL creates interpolated temperature surface maps from the NDAWN stations temperature data.*

Methods Overview:

The ETL process included in the included Jupyter Notebook (see github repo) retrieves temperature data from NDAWN's (North Dakota Agricultural Weather Network) API in 'semi-real time', by requesting data representing the past 30 days *from whichever day the function was run on*. This is made possible by importing the current date from Python's 'datetime' module, saving that date to a variable, and passing it as a parameter in the API requests.

I decided to build a function that creates a new directory/ folder tree for the current day the user decides to run the function on (titled as <date>_NDAWN). At the top of the Notebook, the user must input the directory path they wish for the function's folder structure to be created in. Once specified, the function should be able to run on its own, through the use of self-referential variables/code.

This function iteratively requests tabular data, describing the average, maximum, and minimum temperature values over the past 30 days for every single weather station tracked by NDAWN. This data is organized in folders with names corresponding to the data they house. This function makes good use of the 'csv' and 'Pandas' modules, as much of the function is built around iterating over CSVs and extracting specific rows of data, and then writing this data to either new CSVs or to memory using Pandas dataframes. The Pandas 'groupby' function is utilized in this ETL to aggregate monthly maximum, minimum, and average temperature values for each station.

Because NDAWN also includes the coordinates of each weather station on their webpage, this data is additionally extracted and tacked onto all outputted CSVs and Pandas dataframes. Because all of this code operates in the "arcgispro-py3" Python package environment, this ETL is able to use Ersi's 'arcpy' module to perform spatial analyses. The final steps of this code include creating point feature class layers (shapefiles) based on the latitude/longitude values associated with each weather station, which have embedded in their attribute tables the name of the weather station, and the aggregated monthly temperature values for each station (three point feature classes are made—one represents mean monthly average temperature, and the other two describe the absolute maximum and minimum temperature values for that station over the past 30 days).

The final portion of this code creates an interpolated temperature surface for each of the three feature classes (ave monthly temp, monthly max temp, monthly min temp), using Esri's Kriging algorithm. The decision to use this interpolation method, as opposed to one like IDW, was informed by the various literature discussed in this report's 'Discussion' section.

Instead of pulling screenshots of code from the program into this report, the included Jupyter Notebook has been organized into labeled sections which will be referenced in this report, and the following methods section will follow the order of steps taken in the Notebook.

Jupyter Notebook "Table of Contents:"

1.) GENERAL INITIALIZATION:
   - A. Importing Packages
   - B. Manually inputting path to the base of script users working directory
   - C. Grabbing current date from the datetime module; creating a new folder for today's date

2.) DIRECTORY/ STATION DICTIONARY INITIALIZATION
   - D. Setting up dictionaries / lists to house all relevant info about every station
   - E. Iteratively sending requests to NDAWN server for each of the station numbers (via inputting the station numbers into the URL parameters) to receive the city/state that corresponds to each station number.
   - F. Writing a CSV file at base of directory which contains all station numbers and their city/state names
   - G. Creating a 'STATIONS' folder at the base of working directory
   - H. Iteratively creating folders inside STATIONS folder for each of the stations (named by just their station #):

3.) REQUESTING AND SEQUESTERING AVE. TEMPERATURE DATA FOR THE PAST 30 DAYS FOR EACH WEATHER STATION

I. Iterating over list of stations and sending requests to NDAWN to retrieve average temperature data for each station for the past 30 days
J. Creating "clean" CSV files up to contain *only* a descriptive header and relevant data.
K. Concatenating all stations' Ave Temp Groupby CSVS into one master CSV. This CSV contains each station name, its Latitude, Longitude, and mean 30 Day Ave Temp:

4.) REQUESTING AND SEQUESTERING ABSOLUTE MIN AND MAX TEMPERATURE VALUES OVER THE PAST 30 DAYS FOR EACH WEATHER STATION

L. Creating new directories for the Max Temp and Min Temp station data
M. Sending requests to NDAWN for CSVs of daily MAXIMUM temp over 30 days for each of the weather stations
N. Sending requests to NDAWN for CSVs of daily MINIMUM temp over 30 days for each of the weather stations
O. Using Pandas 'GroupBy' function to grab the maximum value among all 30 days of Max temp data for each; writing new CSVs to 'MaxTempGroupbyCSVs' folder
P. Using Pandas 'GroupBy' function to grab the minimum value among all 30 days of Min temp data for each; writing new CSVs to 'MinTempGroupbyCSVs' folder
Q. Concatenating all station MAX Temp Groupby CSVs into one CSV. This CSV contains each station name, its Latitude, Longitude, and maximum temp over 30 days
R. Concatenating all station MIN Temp Groupby CSVs into one CSV. This CSV contains each station name, its Latitude, Longitude, and minimum temp over 30 days

5.) USING ARCPY TO CREATE POINT FEATURE CLASSES OUT OF CONCATENATED CSV FILES

S. Setting up directory to store Arcpy-related files
T. Using the Arcpy Management "XYTableToPoint" function to convert the 'Ave Temp Groupby Concatenation' CSV into a point feature class, using the Longitude and Latitude fields for the X and Y
U. Using the Arcpy Management "XYTableToPoint" function to convert the 'Max Temp Groupby Concatenation' CSV into its own point feature class
V. Using the Arcpy Management "XYTableToPoint" function to convert the 'Min Temp Groupby Concatenation' CSV into its own point feature class

W. Creating file geodatabase in 'ARCPRO' folder ('grid' files, which the Kriging function will output in the next step, have(?) to be stored in esri filegeodatabases

6.) USING ARCPY TO SPATIALLY INTERPOLATE MONTHLY AVE, MAX, AND MIN TEMPERATURE VALUES

X. Running arcpy.sa.Kriging() function on Ave Temp weather stations FC, outputting interpolated surface map of average monthly temperature

Y. Running arcpy.sa.Kriging() function on Max Temp weather stations FC, outputting interpolated surface map of monthly max temperature

Z. Running arcpy.sa.Kriging() function on Min Temp weather stations FC, outputting interpolated surface map of monthly minimum temperature

I. GENERAL INITIALIZATION

The first part of this Jupyter Notebook ETL involves importing all needed packages, and manually inputting the users directory path where the program should create a repository. As noted in the Methods Overview, because this program is designed to pull temperature data for all of NDAWN's tracked stations over the *past 30 days*, we need to give the program information about what the current date is (the date which the program is run). To do this, we may utilize Python's 'datetime' module and map the current date to a variable, using the datetime.datetime.now() function (*Jupyter 1.C*). We will utilize the &begin_date parameter in the NDAWN site URL to specify the date for which we want to capture 30 prior days of data. Because this &begin_date= parameter takes a date value formatted as *YYYY-MM-DD,* we will format our datetime object as such, using .strftime("%Y-%m-%d") *(Jupyter 1.C)*

Figure 2: *Screenshot of the NDAWN webpage and URL bar, showing that the &begin_date parameter takes a date value, formatted as YYYY-MM-DD.*

II. DIRECTORY/ STATION DICTIONARY INITIALIZATION

This ETL is designed to creates a new directory/ folder tree for the current day the user decides to run the function on (titled as <date>_NDAWN). This is done through using the 'os' module's . mkdir() function (*Jupyter 2.G.)*. The first set of NDAWN requests/ data extraction/data clean up/data aggregation is designed to pull data for each stations **average temperature** values for the past 30 days. Because we are gathering data for 140 weather stations, I decided to configure this program to create folders for each station, within a folder named 'STATIONS.' (*Jupyter 2.H)*. To set up a way of easily iterating over all of the stations (to perform tasks like creating file & directory names out of each station number), this function saves to memory a dictionary structure, where the key is the station number, and the value is the station city/state *(Jupyter 2.E)*. A list of all possible NDAWN stations was obtained by copy/pasting a URL to a page displaying 'All Station Data,' iteratively parsing this string to extract the station numbers, and then sending GET requests to https://ndawn.ndsu.nodak.edu/station-info.html to receive the station name (*Jupyter 2.E.)*. Because 140 total requests are sent to receive the names of each station, this block of code ends up being computationally intensive.

III. REQUESTING AND SEQUESTERING AVE. TEMPERATURE DATA FOR THE PAST 30 DAYS FOR EACH WEATHER STATION

This function iteratively requests tabular data for average, maximum, and minimum temperature values over the past 30 days for every single weather station tracked by NDAWN. This data is organized in folders with names corresponding to the data they house. To request the tabular data (CSV file) for a given station, for the past 30 days for a given day, for a given temperature "variable" (average daily temp, maximum daily temp, etc.), parameters must be set up in a python dictionary structure, and sent along with a 'get' request to the URL base:

https://ndawn.ndsu.nodak.edu/table.csv

```
ndawn_csvtable_urlbase = "https://ndawn.ndsu.nodak.edu/table.csv"

for stationnum in just_stationnumbers:
    foldpath = os.path.join(stationsfolderpath, stationnum)
    master_params = {
        "station" : stationnum,
        "variable" : "ddavt",
        "dfn" : "",
        "year" : "2021",      # NDAWN has a built-in option to
        "ttype" : "daily",    # "quickpick" the past 30 days' data
        "quick_pick" : "30_d",        # Past 30 days,
        "begin_date" : formatteddate}  # Based on current date.

    reqreq = requests.get(ndawn_csvtable_urlbase,
                          params = master_params)

    csvname = os.path.join(foldpath, f'{stationnum}_initial.csv')
    with open(csvname, "w",newline="") as open_csv:
        open_csv.write(reqreq.content.decode('utf-8'))
```

Figure 3: *Code block in included Jupyter Notebook which sets up the parameters for a request which retrieves 'daily average temp' ('ddavt'), for a given station ('stationnum' var, which updates on each iteration, is passed here), and the "quick_pick" variable, which NDAWN conveniently has set up, is set to "30_d", to retrieve 30 days of data moving backwards in time from the "begin_date" ('formatteddate" variable, created from datetime module, is passed here)*

The NDAWN site conveniently has a built-in option called "quick pick," which allows you to obtain data for a station over the past 30 days. Because of this, we only need to pass a "begin_date" and do not have to worry about specifying an "end_date" (*Jupyter 3.I.).*

Each station's CSV retrieved in the above code block is stored in the folder corresponding to its station number. As these CSVs stand initially, they are not "cleaned up" enough to allow for table concatenation/aggregation. (**Figure 4.)** This function iteratively 'lops off' the excess header at the start of each of these CSVs, gives each 'clean' CSV a new descriptive header, and organizes the clean CSVs accordingly *(Jupyter 3.J).*

Figure 4: *Example of what one station's CSV file looks like after being initially returned from an NDAWN API request. This program deletes the first 4 rows of these csvs, and updates the 5th row to be a fitting/descriptive header.*

6_initial

1	Data from North Dakota Agricultural Weather Network https://ndawn.ndsu.nodak.edu						
2	Daily Observation Table for March 22 2021 to April 20 2021						
3	Flag Definition Line: M - Missing; E - Estimated; N/A - Not Available						
4	Station Name	Latitude	Longitude	Elevation	Year	Month	Day	Avg Temp
5		deg	deg	ft				Degrees F
6	Warren	48.137	-96.839	854	2021	3	22	33.936
7	Warren	48.137	-96.839	854	2021	3	23	38.119
8	Warren	48.137	-96.839	854	2021	3	24	31.245
9	Warren	48.137	-96.839	854	2021	3	25	33.287
10	Warren	48.137	-96.839	854	2021	3	26	35.042
11	Warren	48.137	-96.839	854	2021	3	27	29.277
12	Warren	48.137	-96.839	854	2021	3	28	25.396
13	Warren	48.137	-96.839	854	2021	3	29	51.312
14	Warren	48.137	-96.839	854	2021	3	30	21.451
15	Warren	48.137	-96.839	854	2021	3	31	19.19
16	Warren	48.137	-96.839	854	2021	4	1	31.622
17	Warren	48.137	-96.839	854	2021	4	2	51
18	Warren	48.137	-96.839	854	2021	4	3	43.257

Once a set of "clean" CSV files for each station has been created, these CSVs are iteratively mapped to Pandas dataframes, and the pandas 'groupby' function is utilized to aggregate monthly average temperature values for each station (these are also saved as separate CSVs). In each "groupby CSV," data is stored for the station name, the Latitude, Longitude, and 30-day average temp *(Jupyter 3.J).*

	Station Name	Latitude	Longitude	30 Day Avg Temp
0	Zeeland	46.013378	-99.687587	36.8081

Figure 5: *Example of what a station's "groupbt CSV" file looks like.*

Now that each station has its own file saved to the users disc, containing the station's location and monthly average temperature, all of these CSV files may be glued into one master table, which we may use to import into ArcGIS, and create a points feature class from, which

contains aggregated temperature values which we will use to create an interpolated temperature surface from.

```
avetempGBconcat = os.path.join(avetempgbpath,
                               'avetempGBconcat.csv')

allFiles = glob.glob(avetempgbpath + "/*.csv")
allFiles.sort()
with open(avetempGBconcat, 'wb') as outfile:
    for i, fname in enumerate(allFiles):
        with open(fname, 'rb') as infile:
            if i != 0:
                infile.readline()  # Throw away header on all but first file
            # Block copy rest of file from input to output without parsing
            shutil.copyfileobj(infile, outfile)
            print(fname + " has been imported.")
```

Figure 6 : *Code block from Jupyter Notebook 3.K: concatenating all stations' Ave Temp Groupby CSVS into one master CSV. Though all the files are merged together, the header is only applied once.*

IV. REQUESTING AND SEQUESTERING ABSOLUTE MIN AND MAX TEMPERATURE VALUES OVER THE PAST 30 DAYS FOR EACH WEATHER STATION

The methodology for constructing data tables for each station's monthly maximum and minimum temperature is almost identical to step 3. (*see Jupyter section 4*). One of the only differences, besides the differing variable/directory names, is that we pass the maximum temperature ('ddmxt') and minimum temperature ('ddmnt') variables into the parameter assignment, shown in **Figure 3**. We also will have to use a different 'groupby' function than 'mean' to aggregate our maximum and minimum temperature values (we are looking for the *absolute maximum and minimum* values, out of all 30 days recorded).

```
updatedDF.groupby(['Station Name',
                   'Latitude',
                   'Longitude']).agg({'30 Day Max Temp': ['max']}).reset_index()
                      # grouping by MAX this time, to find the maximum
                      # value between the 30 days..~
                      # Grouping by max will again, not affect the lat and long
                      # values, because they are all the same for each day.
```

Figure 7: *Screenshot of the codeblock in Jupyter Notebook section 4.O.; here we are using the 'max' parameter with the groupby .agg() function instead of 'mean.'*

V. USING ARCPY TO CREATE POINT FEATURE CLASSES OUT OF CONCATENATED CSV FILES

Because all of our 'master' CSV files have fields which describe each station's latitude and longitude, we may easily convert these tables to point feature classes through using the arcpy.management.XYTableToPoint() function.

*** After searching for a while on NDAWNs website trying to find what datum they used for their coordinate data, I decided to just try out WGS84… it seems like the correct fit, but I am not confident that this is the original coordinate system of the data.***

```
avetempstations = os.path.join(arcpro_folder, "weatherstations_avetemp")

arcpy.management.XYTableToPoint(avetempGBconcat,
                                avetempstations,
                                "Longitude",
                                "Latitude",
                            None,
                                wgs84_GCS)
```

Figure 8: *Screenshot of codeblock in Notebook section 5.T.; running the XYTableToPoint() function.*

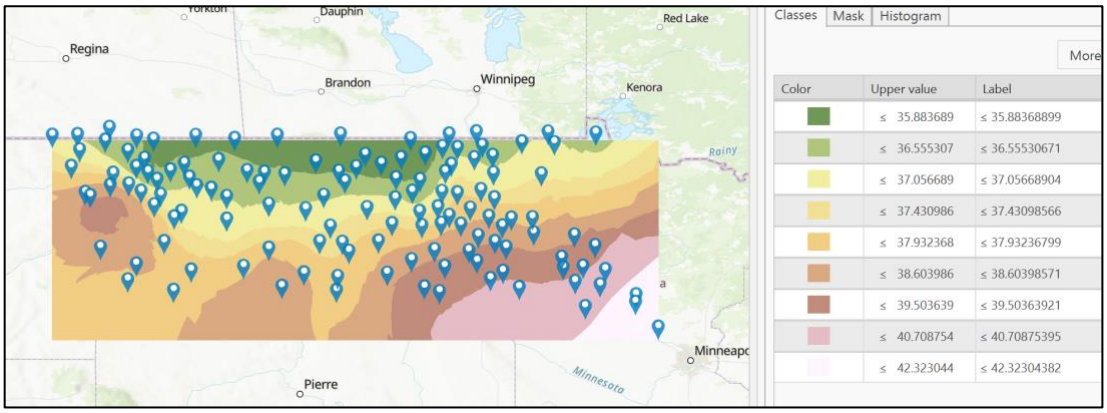
Three point feature classes are created from this program: one for stations//average monthly temp, another for monthly maximum temp, and a final layer representing monthly minimum temps.

VI. USING ARCPY TO SPATIALLY INTERPOLATE MONTHLY AVE, MAX, AND MIN TEMPERATURE VALUES

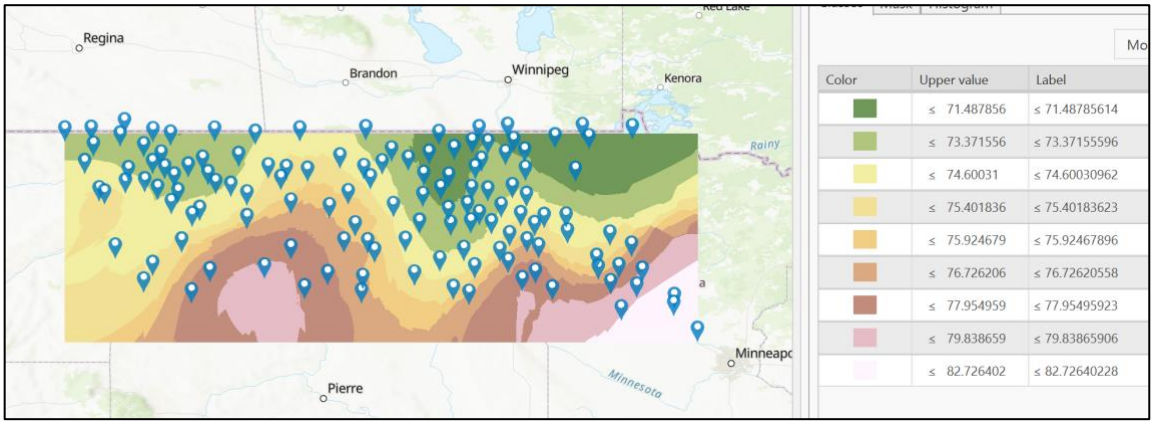
With the three point feature classes created in Jupyter Notebook section V., we are now able to calculate interpolated temperature surfaces. Based various pre-existing literature (discussed in the "Discussion" section), the spatial interpolation method chosen in this ETL was *Ordinary Kriging*. Sections 6.X,Y, and Z. in the Jupyter Notebook describes how the arcpy.sa.Kriging() function may be parameterized for each of the three temperature variable feature classes. Because Esri's kriging function outputs a .GRID file, the output location for this file must be an Esri filegeodatabase (the program creates a blank filegeodatabase for the user in Jupyter 6.W.).

*** Note: this Jupyter Notebook ETL only outputs .GRID files for the interpolated temperature surfaces of each weather station's monthly average temperature, monthly maximum temperature, and monthly minimum temperature – this program does ~not~ actually create fully-finished map documents. I decided that it is much more efficient, effective, and intuitive to change the symbology of the kriging rasters through the ArcGIS Pro GUI, and that process only takes a couple of steps….vs what I imagine would be multiple steps using arcpy.mp. ***

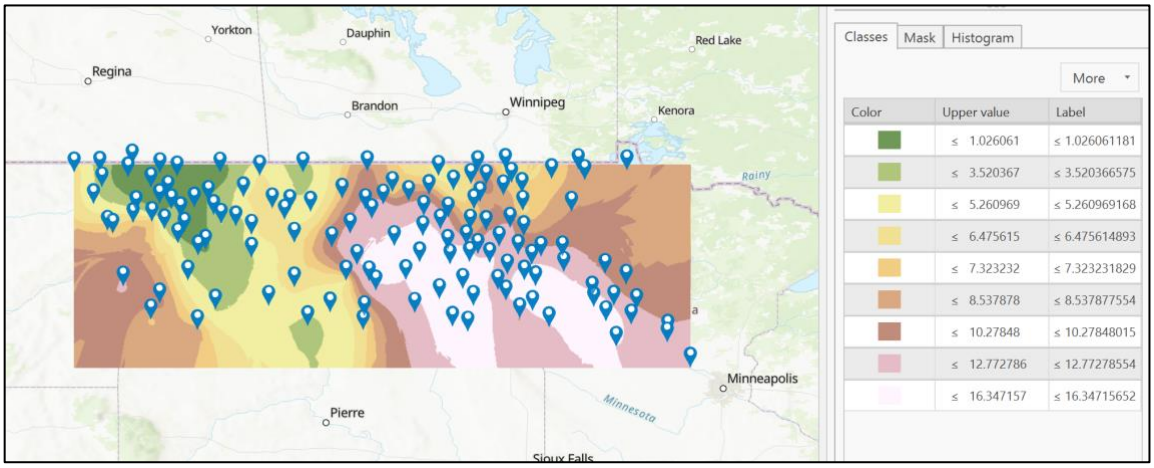
Results/ Results Verification



A.



B.



C.

Figure 9: *A.) Interpolated temperature surface map for monthly average temperature, B.) Interpolated temperature surface map for monthly maximum temperatures, C.) Interpolated temperature surface map for monthly minimum temperatures.*

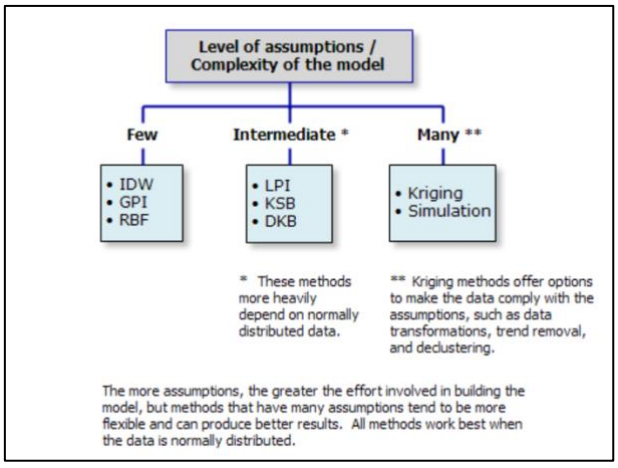
Figure 9 depicts the final products of the NDAWN weather stations temperature interpolation ETLs. The range of temperatures for each of the three maps is what we would expect for the northern Midwest from mid-March to mid-April.

As stated in the Methods, an assumption was made that the source spatial reference for the weather station coordinates was WGS84 – if this assumption is in fact wrong, the visual displays of these maps would likely be different.

Discussion

There are several interpolation algorithms to choose from when constructing an interpolated raster surface. A large body of literature exists which deliberates the 'optimal' interpolation method to use for mapping (predicted) average temperature across a geographic region. Although most of the literature I encountered concluded that some form of kriging was the optimal method in modelling/predicting temperature across space for their specific study, that is not to say kriging is the one-size-fits-all 'best' method for interpolating temperature data. The optimal interpolation method for a given study is always dependent on the data and geographic variables specific to that study.

That being said, there is an extensive body of literature which argues that the various kriging methods are likely to produce the most optimal interpolated temperature surfaces compared to Inverse Distance Weighting (IDW seems to be the only other likely competition for modelling temperature). Kim et al. (2010) come to the conclusion that, for the task of predicting climate factors in South Korea, "kriging showed gradual and smooth pattern," while "IDW showed rather irregularly distributed with discontinuous borders." Cao et al. (2009) found that, for interpolating temperature surfaces in China from a set of weather stations, between inverse distance weight, Spline, Kriging-exponential, Kriging-spherical, and Kriging-Gaussian in temperature interpolation, "Kriging-exponential and Kriging-spherical interpolation methods" were the "highest-accuracy methods", whereas IDW was "less accurate." Cronqvist (2018) found that, for creating weather forecasts at the Swedish Meteorological and Hydrological Institute, "regression kriging always had a smaller mean square error than IDW."



Source: ("Classification trees of the interpolation methods offered in Geostatistical Analyst," Esri)

As the diagram from the Esri documentation "Classification trees of the interpolation methods offered in Geostatistical" summarizes above, kriging is much more complex than IDW, and offers more flexibility in 'tweaking' the model to fit the needs of the specific study scenario. It is my understanding that this level of complexity often makes kriging the optimal interpolation method over IDW. However, as Esri states in the "Comparing interpolation methods" documentation, "more so than other interpolation methods, a thorough investigation of the spatial behavior of the phenomenon represented by the z-values should be done before you select the best estimation method for generating the output surface." Although kriging is more complex, to use the tool properly, the user must have a thorough understanding of what the algorithm is doing, and how to properly parameterize the algorithm based on the specific phenomenon being modelled, in order for the interpolation method to truly be "more powerful."

Given the literature explained above, I decided to have my program automatically output interpolated temperature surfaces created through ordinary kriging, because I wanted the result would *look* smoother than IDW. However, I do *not* have a good understanding on the innerworkings and complexities of kriging, and do not know if I could defend the choice to use this method over IDW. IDW works by generating surface (cell) values based on the distance of the cell being calculated to all other known/determined cell values. The closer the cell being calculated is to a pre-determined point/value pair, the more *weight* that point has in determining the average value mapped to that cell.

Spline is a third possible interpolation method one has access to in the Esri toolbox. A defining feature of "Spline" is that it reduces overall surface curvature. As Childs (2004) explains, like IDW, Spline is a "deterministic method that creates surfaces from samples based on the extent of similarity or degree of smoothing," though, while a spline surface "passes exactly through each sample point, an IDW will pass through none of the points."

Conclusion

This report demonstrates an ETL methodology that operates in a Jupyter Notebook, which extracts real-data temperature data from NDAWN, performs data clean-up on the extracted tabular data, and saves the data in an organized folder structure. This ETL also creates point-geometry shapefiles for each of the NDAWN weather stations, with relevant temperature data embedded into the feature attribute tables. This function successfully interpolates the temperature data associated with the created point feature classes through arcpy's ordinary kriging algorithm.

References

Cao, W., Hu, J., & Yu, X. (2009, August). A study on temperature interpolation methods based on GIS. In *2009 17th International Conference on Geoinformatics* (pp. 1-5). IEEE.

Childs, C. (2004). Interpolating surfaces in ArcGIS spatial analyst. *ArcUser, July-September*, *3235*(569), 32-35.

Classification trees of the interpolation methods offered in Geostatistical Analyst. Esri.Retrieved 2021 from https://pro.arcgis.com/en/pro-app/latest/help/analysis/geostatistical-analyst/classification-trees-of-the-interpolation-methods-offered-in-geostatistical-analyst.htm

Comparing interpolation methods. Esri. Retrieved 2021 from https://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/comparing-interpolation-methods.htm

Cronqvist, F. (2018). Interpolation of temperature data for improved weather forecasts.

Kim, S. N., Lee, W. K., Shin, K. I., Kafatos, M., Seo, D. J., & Kwak, H. B. (2010). Comparison of spatial interpolation techniques for predicting climate factors in Korea. *Forest Science and Technology*, *6*(2), 97-109.

Scheeres, A. (2016). Kriging: Spatial Interpolation in Desktop GIS. Azavea. https://www.azavea.com/blog/2016/09/12/kriging-spatial-interpolation-desktop-gis/

Witold Frączek, H. S. (2019). Interpolate Temperatures Using the Geostatistical Wizard. Esri. https://www.esri.com/content/dam/esrisites/en-us/media/pdf/learn-arcgis/interpolate-temperatures-using-the-geostatistical-wizard.pdf

Self-score

Category	**Description**	**Points Possible**	**Score**
Structural Elements	All elements of a lab report are included **(2 points each)**: Title, Notice: Dr. Bryan Runck, Author, Project Repository, Date, Abstract, Problem Statement, Input Data w/ tables, Methods w/ Data, Flow Diagrams, Results, Results Verification, Discussion and Conclusion, References in common format, Self-score	28	**22**
Clarity of Content	Each element above is executed at a professional level so that someone can understand the goal, data, methods, results, and their validity and implications in a 5 minute reading at a cursory-level, and in a 30 minute meeting at a deep level **(12 points)**. There is a clear connection from data to results to discussion and conclusion **(12 points)**.	24	**24**
Reproducibility	Results are completely reproducible by someone with basic GIS training. There is no ambiguity in data flow or rationale for data operations. Every step is documented and justified.	28	**28**
Verification	Results are correct in that they have been verified in comparison to some standard. The standard is clearly stated **(10 points)**, the method of comparison is clearly stated **(5 points)**, and the result of verification is clearly stated **(5 points)**.	20	**15**
		100	**89**