Title: **LAB 02, PART I.**

Notice: Dr. Bryan Runck
Author: **Michael Felzan**
Date: **March 6, 2021**

Project Repository: https://github.com/fezfelzan/GIS5572.git

Abstract

This report demonstrates an ETL operating in an ArcGIS Pro Jupyter Notebooks which retrieves .LAS LiDAR data from the MN DNR FTP server, converts this data into TIN and DEM raster data, and then saves these files to local disk. The process of programmatically creating layouts of layer files in ArcGIS (with proper camera extent), and exporting these layouts as PDFs, is additionally outlined. The second section of this lab explores ArcGIS's functionality for viewing .LAS files in 2D and 3D. The third section of this report explores the process of downloading 30-Year Normals .bil rasters from the PRISM server, and the several steps involved in converting the .bil files into a space time cube. The results (and their validity) are discussed in the final portion of this report.

Problem Statement

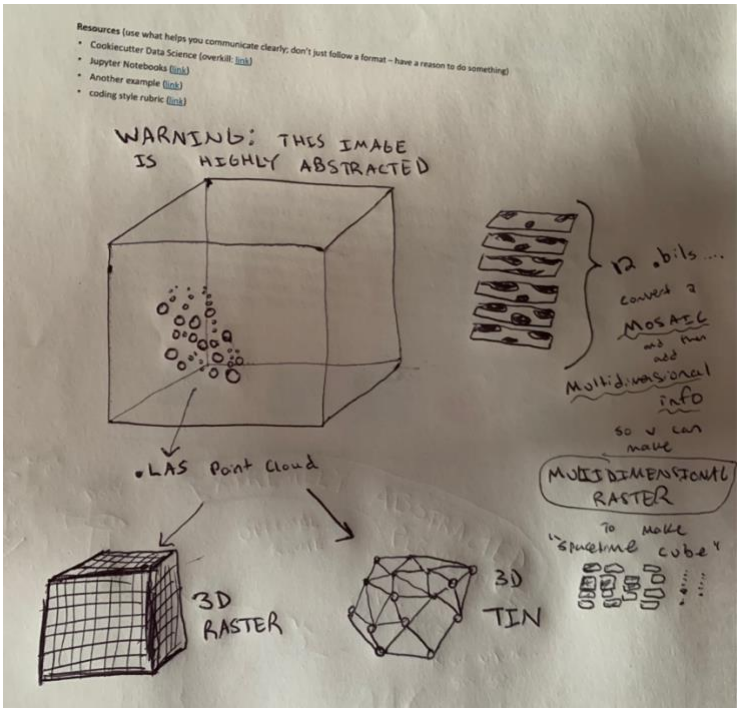


Figure 1: *Illustrative figure of the lab objective*

The objective of the first part of Lab 2 can be broken into three sections. The first of these sections deals with building an ETL in ArcPro Jupyter Notebooks (arcpy) which:

1. Downloads a .LAS file from the MN DNR FTP server
2. Converts the .LAS file into both a DEM and TIN (exporting files to disk)
3. Exports PDFs of the DEM and TIN with "correct" visualizations

The second portion of Lab 2 pt 1 involves:

4. Performing a side-by-side exploratory data analysis of the downloaded .LAS file in 2D and 3D
5. Exploring ArcGIS's included functionality for visualizing .LAS files

And the final portion involves building an ETL which:

6. Downloads annual 30-Year Normals .bil files from the PRISM website
7. Converts the .bil files into a spacetime cube & exports the file to local disc
8. Exports an animation of the weather data timeseries

Table 1. *Defining the Problem*

#	Requirement	Defined As	Spatial Data	Attribute Data	Dataset	Preparation
1	Elevation	Raw input data (.LAS) from MN DNR FTP server	Elevation point cloud		MN DNR	
2	Digital Elevation Model	Raster data with pixel values relating to elevation	Location, elevation			Convert .LAS to .lasd, raster
3	Precipitation	Amount of precipitation across the US over the span of 12 months, raster .bils	(location)	Time, Value (amt. precipitation)		Select only 12 .bils which reflect monthly averages over 30 yrs.

Input Data

The first part of this lab involves demonstrating a Python-based ETL which converts a .LAS file into a TIN and a DEM. The actual .LAS could represent any data, so a sample file (with a small file size) from the MN DNR's FTP server was chosen. For the third part of this lab, the objective is to use a Python ETL to download 30 Year Normals .bils from the PRISM website, to convert these rasters into a spacetime cube. The 12 .bils from the PRISM website are the only input data needed for this portion of the objective.

Table 2. *Input Data*

#	Title	Purpose in Analysis	Link to Source
1	4342-13-03.las	Sample .las point cloud data to be converted to DEM and TIN.	https://resources.gisdata.mn.gov/pub/data/elevation/lidar/examples/lidar_sample/las/
2	PRISM US 30 Year Normals Precipitation data	12 .bil raster files which represent average monthly precipitation in the US. Files will be used to create a spacetime cube	https://prism.oregonstate.edu/normals/

Methods

SECTION I.

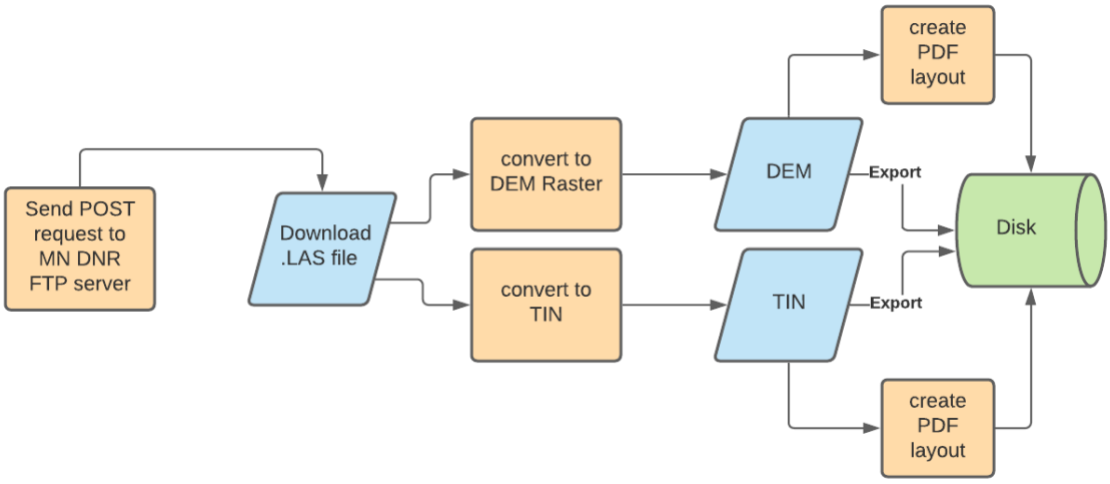


Figure 2. *Flow Diagram depicting the ETL process of downloading a .LAS file from the MN DNR FTP server, converting the file to both a DEM raster and a TIN, and exporting these files (along with PDF layouts of their visualization) to local disk*

Part 1: Downloading .LAS file from the MN DNR FTP server

As covered in Lab 01, the MN DNR FTP server resembles a hierarchical folder tree, where navigating the contents of the server involves concatenating components onto the URL (much like navigating directory paths on your own computer). The base of this URL/ folder structure is:

https://resources.gisdata.mn.gov/pub/data/

One location we can find sample .LAS LiDAR data on this server is in the "lidar_sample" repository. We may navigate to that page by navigating from the pub/data:

elevation/ → lidar/ → examples/ → lidar_sample/ → las/

```
In [2]: gis_mngov_home = "https://resources.gisdata.mn.gov/pub/data/"

        path_to_lidar = "elevation/lidar/examples/lidar_sample/las/"

        full_path = gis_mngov_home + path_to_lidar
        full_path

Out[2]: 'https://resources.gisdata.mn.gov/pub/data/elevation/lidar/examples/lidar_sample/las/'
```

For this repository, we may send a "request" to receive the HTML text of this page, and then parse this text to print only the "data" items in the page:

```
In [4]: PrintNavItems(full_path)

        /pub/data/elevation/lidar/examples/lidar_sample/
        4342-12-05.las
        4342-12-06.las
        4342-12-07.las
        4342-13-03.las
        4342-13-04.las
        4342-13-05.las
        4342-13-06.las
        4342-13-07.las
        4342-14-03.las
        4342-14-04.las
        4342-14-05.las
        4342-15-03.las
        4342-15-04.las
        4342-16-03.las
        4342-16-04.las
```

It is unclear what these .LAS files will be before actually downloading them, but, trusting that the DNR has included small files for their samples, we may just pick any of these filenames and add it to the end of our previously constructed filepath, in order to send a request which receives back the .LAS data.

Because the DNR server uses the CKAN API, we must use a POST request instead of a GET. Once we have mapped the request object to a variable, we may write it to our local disc:

Creating path to download a specific .las file via post request (as CKAN requires POSTs instead of GET)

In [5]:
```
las_selection = "4342-13-03.las"

final_path = full_path + las_selection
final_path
```

Out[5]:
```
'https://resources.gisdata.mn.gov/pub/data/elevation/lidar/examples/lidar_sample/las/4342-13-03.las'
```

In [6]:
```
#saving data received from post to var
post_req_obj = requests.post(final_path)

#writing data to local disk
with open(r"C:\Users\michaelfelzan\Documents\arc ii labby ii\4342-13-03.las", 'wb') as f:
    f.write(post_req_obj.content)
```

Confirming file has been downloaded properly:

In [7]:
```
env.workspace = r"C:\Users\michaelfelzan\Documents\arc ii labby ii"

arcpy.ListFiles()
```

Out[7]:
```
['4342-13-03.las', 'arciilabii_clean_runthru']
```

Part 2: Convert .LAS file into both DEM raster and TIN

Now we will convert our downloaded .LAS file into both a DEM raster and a TIN.

Because ArcGIS Pro seems to handle "LAS Datasets" better than raw .LAS files (some functions cannot be executed on .LAS files), we will first go ahead and convert our .LAS file into a .lasd :

Creating Las Dataset:

```
arcpy.management.CreateLasDataset(r"C:\Users\michaelfelzan\Documents\arc ii labby ii\4342-13-03.las",
                                  'my_las_dataset.lasd',
                                  #{spatial_reference},  if not assigned,
                                  #  assumes original .las crs
                                 )
```

8]:

Output

C:\Users\michaelfelzan\Documents\arc ii labby ii\my_las_dataset.lasd

DEM Raster

To convert to a DEM raster from a .LAS file, we may use the LAS Dataset to Raster function. Our input will be the previously created .lasd, we will output a file as a .TIF, and create the raster using the **elevation** data embedded in our .LAS file. We may also interpolate the cell values using "binning," where we may take the average elevation value across a user-determined cell size, and assign the pixel value to the average. Choosing a larger cell-size is essentially resampling or generalizing our data to save disk space. In our case, we will use a cell size of "10":

Converting .LAS to DEM:

```
arcpy.conversion.LasDatasetToRaster('my_las_dataset.lasd',
                                    'my_output_raster.tif',
                                    'ELEVATION',
                                    'BINNING AVERAGE LINEAR',
                                    'FLOAT',
                                    'CELLSIZE',
                                    10,
                                    1)
```

[10]:

Output

C:\Users\michaelfelzan\Documents\arc ii labby ii\my_output_raster.tif

TIN

To convert to a TIN from a .LAS file, we can use the LAS Dataset to TIN (3D Analyst) function. Our input data will be the .lasd. This function also requires some choices in terms of parameters – we must decide on how we will "thin" our data (if at all), meaning, how we will aggregate/subset points from the .LAS point cloud to become TIN nodes. Because the ArcPro documentation recommends:

> Note:
> Consider using a **Window Size** thinning type (`thinning_type="WINDOW_SIZE"` in Python) when you need more predictable control of how LAS points are thinned in the generation of the output TIN.

(1.)

We will use Window Size as our thinning method. When iterating over 'windows' of points in the point cloud, we may define minimum elevation value, so that points below that elevation will be discarded in the data processing. Additionally, we may set a cap on the amount of nodes created for the TIN. ArcPro's default 'max_nodes' is 5 million, which is likely too high for our purposes – we may bump it down a factor of 10 to 500,000.

Converting .LAS to TIN:

In [11]:

```
arcpy.LasDatasetToTin_3d('my_las_dataset.lasd',
                         'my_output_tin',
                         'WINDOW_SIZE',
                         'MIN',
                         13.996,
                         500000,
                         1)
```

Out[11]:

Output

C:\Users\michaelfelzan\Documents\arc ii labby ii\my_output_tin

Part 3: Exports PDFs of the DEM and TIN with "correct" visualizations

As depicted in **Figure 2.,** along with exporting the DEM and TIN files to disk, we will also use arcpy.mp to create PDF layouts of each, taking care to scale our layouts to the correct map extent.

This process involves a series of steps, relating to assigning the map project (aprx), project 'map', layer files, map layout, and layout elements to individual variables in the Jupyter Notebook, and then performing functions on these map objects (such as map.addLayer( )). The specific syntax of these functions may be viewed in the Jupyter Notebook file included in the GitHub Repo.

The first of these steps is to convert our TIN and DEM raster into ArcGIS "layer" files (.lyr/.lyrx), as this is the format that all of the functions described above require :

raster:

```
arcpy.SaveToLayerFile_management("my_output_raster.tif",
                                 r"C:\Users\michaelfelzan\Documents\arc ii labby ii\arciilabii_clea
                                 "ABSOLUTE")
                                 #Lyr will store absolute path to source data
```

tin:

```
arcpy.SaveToLayerFile_management("my_output_tin",
                                 r"C:\Users\michaelfelzan\Documents\arc ii labby ii\arciilabii_clea
                                 "ABSOLUTE")
```

Once we have our files in .lyrx format, we may add them to our map/ layout map frame.

In order to set the extent of the layout mapframe to the Zoom Extent of our layers, we can use the functions getLayerExtent() and camera.SetExtent(). Once each our TIF and DEM have been added to the map frame, with the extent set to the layer's zoom extent, we may export the layout as a PDF by using the exportToPDF() function:

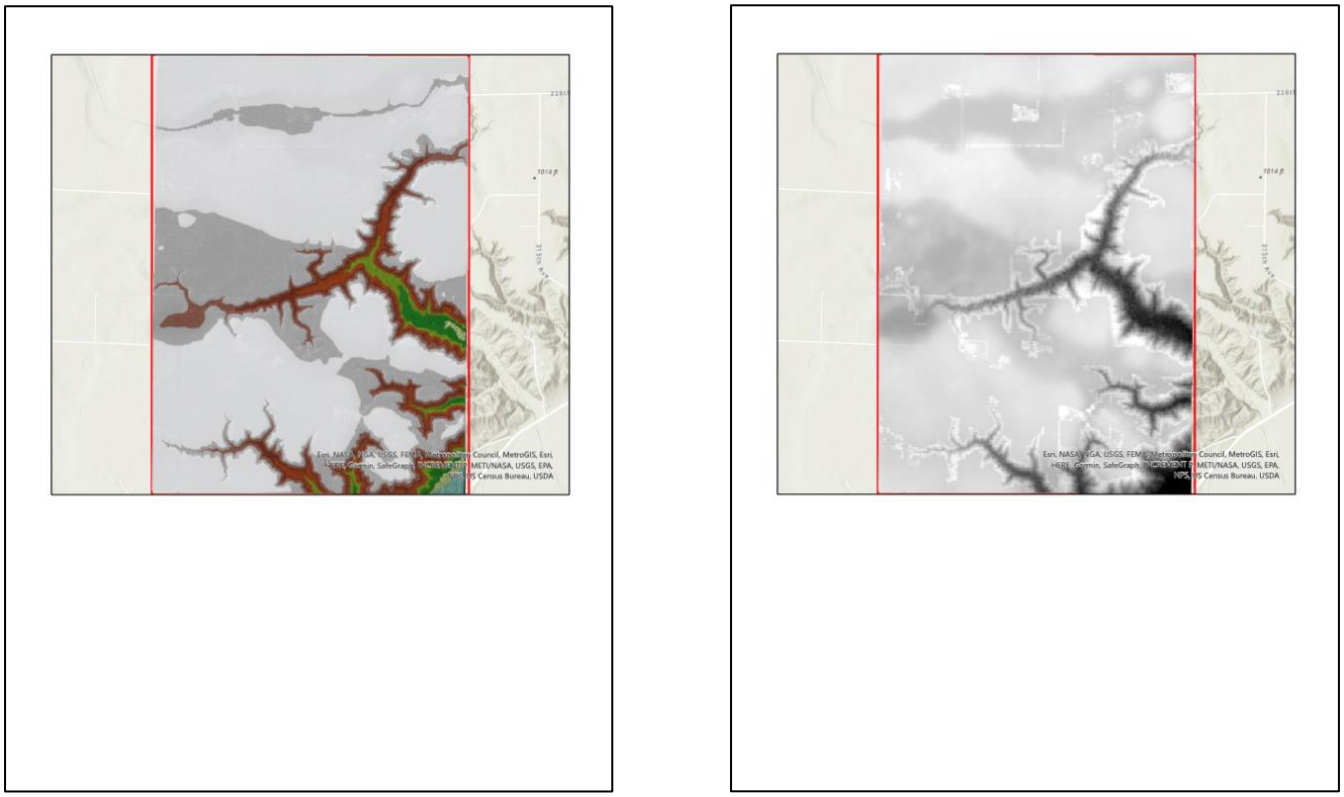
```
rast = m.listLayers("my_output_raster.tif")[0]

rast_layer_extent = mapframe.getLayerExtent(rast, True)

mapframe.camera.setExtent(rast_layer_extent)

lyt.exportToPDF(r"C:\Users\michaelfelzan\Documents\arc ii labby ii\RASTER_DEM.pdf")
```

```
: 'C:\\Users\\michaelfelzan\\Documents\\arc ii labby ii\\RASTER_DEM.pdf'
```



A. ***B.***

Figure 3: ***A****: result of the TIN layout exported as a PDF.* ***B****: result of the DEM raster layout exported as a PDF.*

SECTION II.

Parts 4 & 5: Performing a side-by-side exploratory data analysis of the downloaded .LAS file in 2D and 3D; Exploring ArcGIS's included functionality for visualizing .LAS files

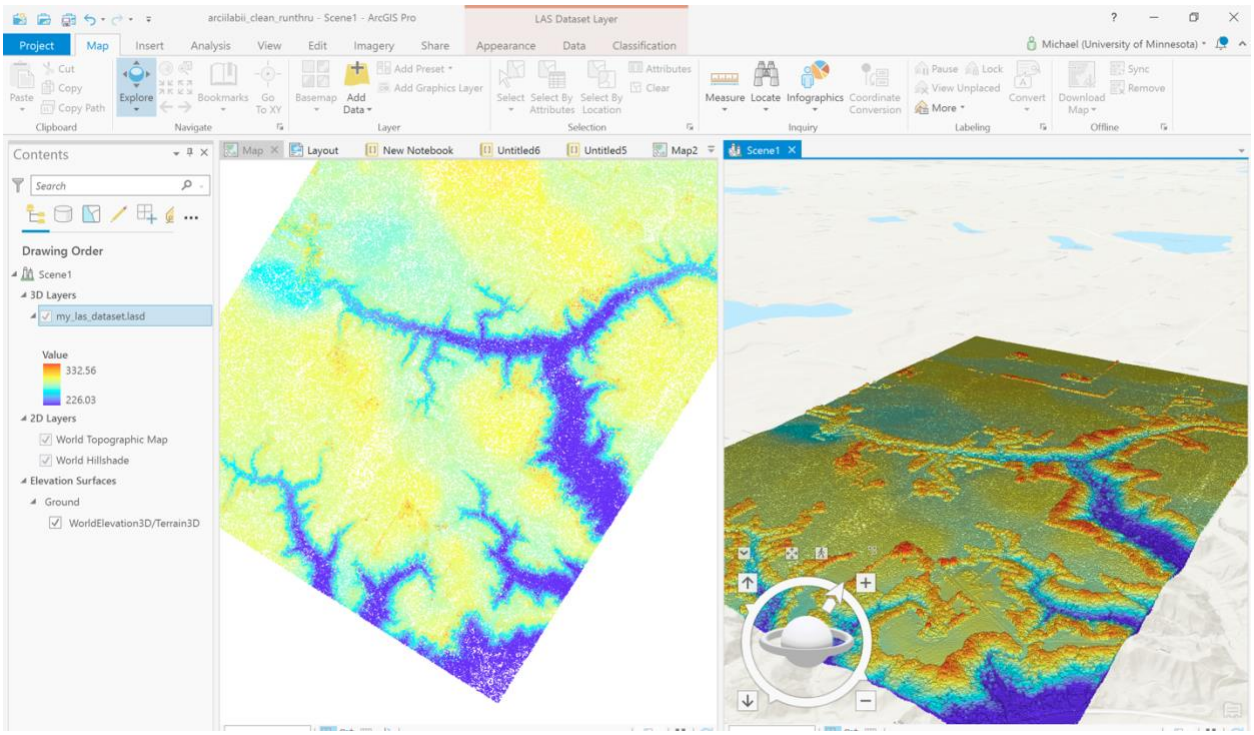


Figure 4: *Visualizing .LAS data in both 2D and 3D, using "linked views" (1)*

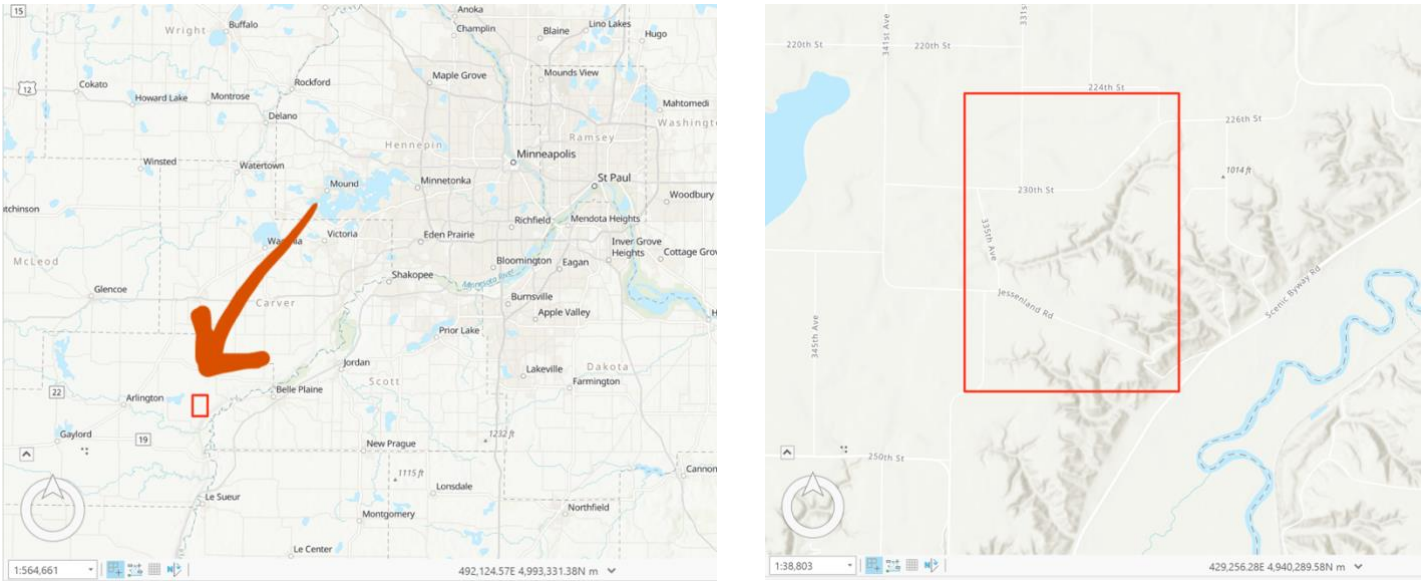


Figure 5: *Geographic extent of .LAS file, shown in context with basemap of MN/ Minneapolis. .LAS file appears to pertain to a river-carved ravine.*

When viewing this .LAS file in 2D and 3D simultaneously (using the ArcPro "link views" feature), we see branches of low elevation areas that wane in occupied space in a fractal-like

pattern – a visual pattern characteristic of river systems (**Figure 4).** Visualizing the .LAS area extent against the basemap, we can see that this file does appear to represent the elevation points of a ravine connected to the Minnesota River (**Figure 5).** The areas colored blue on **Figure 4** relate to areas of low elevation, with increasing elevation values moving up the rainbow towards red.

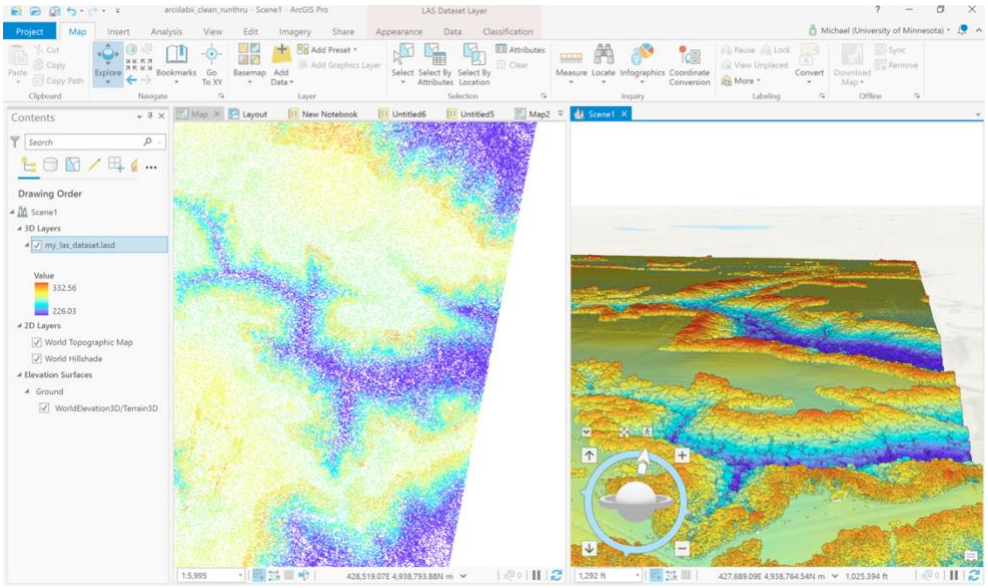


Figure 6: *Visualizing .LAS data in both 2D and 3D, using "linked views" (2)*

In the figures shown, we are mapping the .LAS file in 3D against ArcPro's default "ground elevation" surface. If we wanted to map the pointcloud against a different elevation surface (as in, a custom one built from the elevation values of the file), we would have to use the raster or the DEM

It appears that the elevation values "spike" around the perimeter of the ravines, right before they start to dip. I believe this could be due to vegetation around those areas, though it is hard make any claims with certainty with 1-band data (where our only band is elevation).

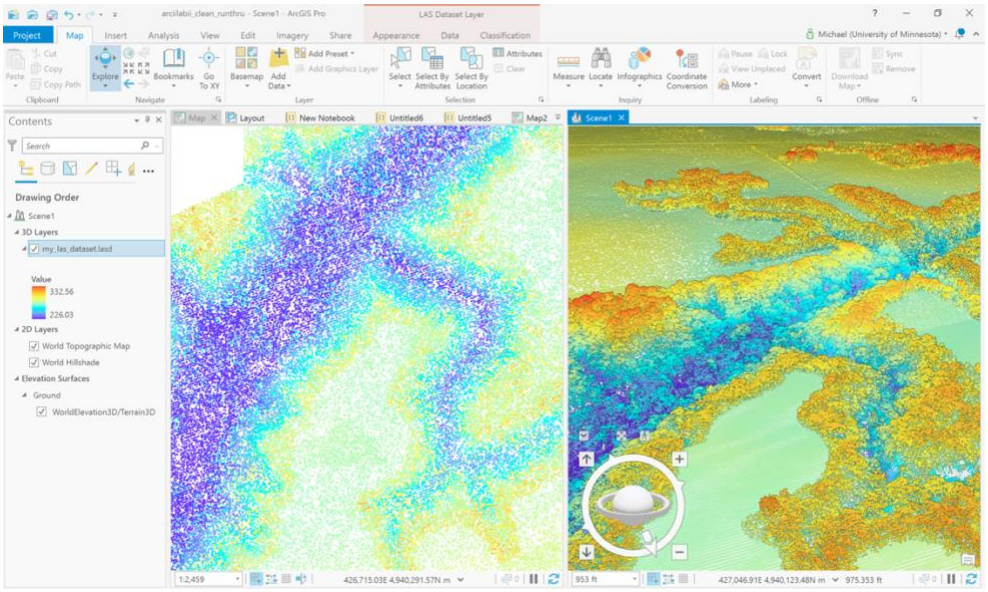


Figure 7: *Visualizing .LAS data in both 2D and 3D, using "linked views" (3)*

ArcPro offers a variety of features for visualizing .LAS files in 2D and 3D. One of the most powerful features (in my opinion) is the ability to open both a 2D and 3D view of your .LAS file in space, and *link the views*, so that when you are scrolling in one window, the other window mirrors the scroll & change in scale. ArcPro also allows you to adjust the appearance of the data, including changing the symbology of the dataset, filtering what data you wish to be visible, and the ability to perform point thinning. The user may also do fine-tuned edits to their data by navigating to the **data** tab, and perform functions like increasing the extent of the original data, dealing with bad/null data, or adding breaklines. You may also re-classify points in your .LAS dataset (that may have originally been mis-classified by ArcPro) by navigating to the **classification** tab.

SECTION III.

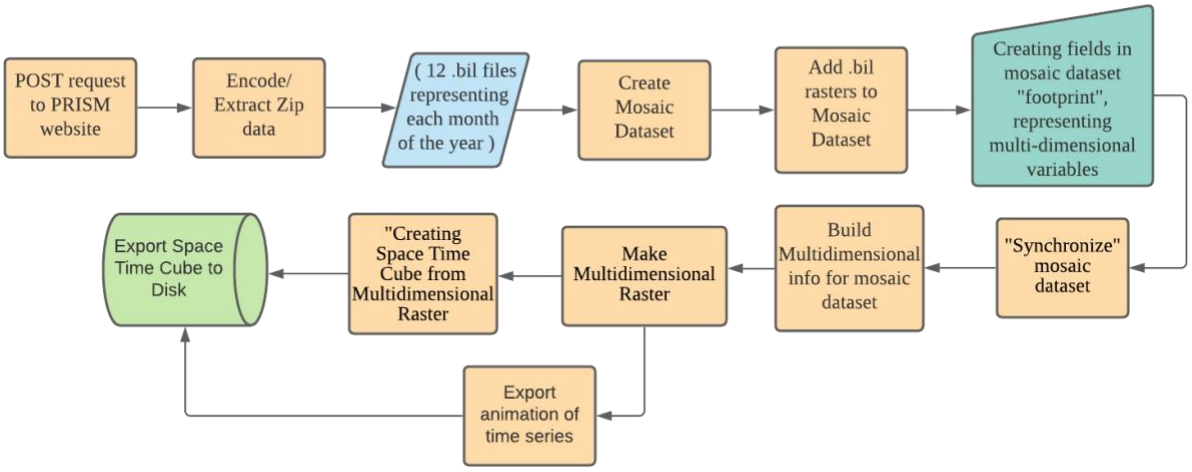


Figure 8. *Flow Diagram depicting the process of downloading 30-Year Normals .bil rasters from the PRISM server, and the multiple steps involved in converting the .bil files into a space time cube*

Part 6: Downloading annual 30-Year Normals .bil files from the PRISM website

The first step in the ETL process outlined in **Figure 8.** is to download the .bil raster files from the PRISM website. Navigating to the 30-Year Normals page on the PRISM webpage, we can see there is a button which says "Download All Normals Data (.bil)" Because we want 12 rasters that depict average monthly precipitation in the US over the span of 30 years, we select this button instead of "Download Data" (which would just give us a specific month). We will make our spacetime cube by 'stacking' 12 rasters into one dataset, embedding a time element for each image, in order to create a 'multidimensional raster' .crf file.

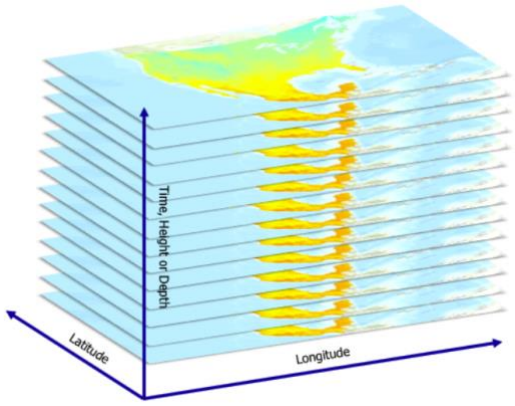
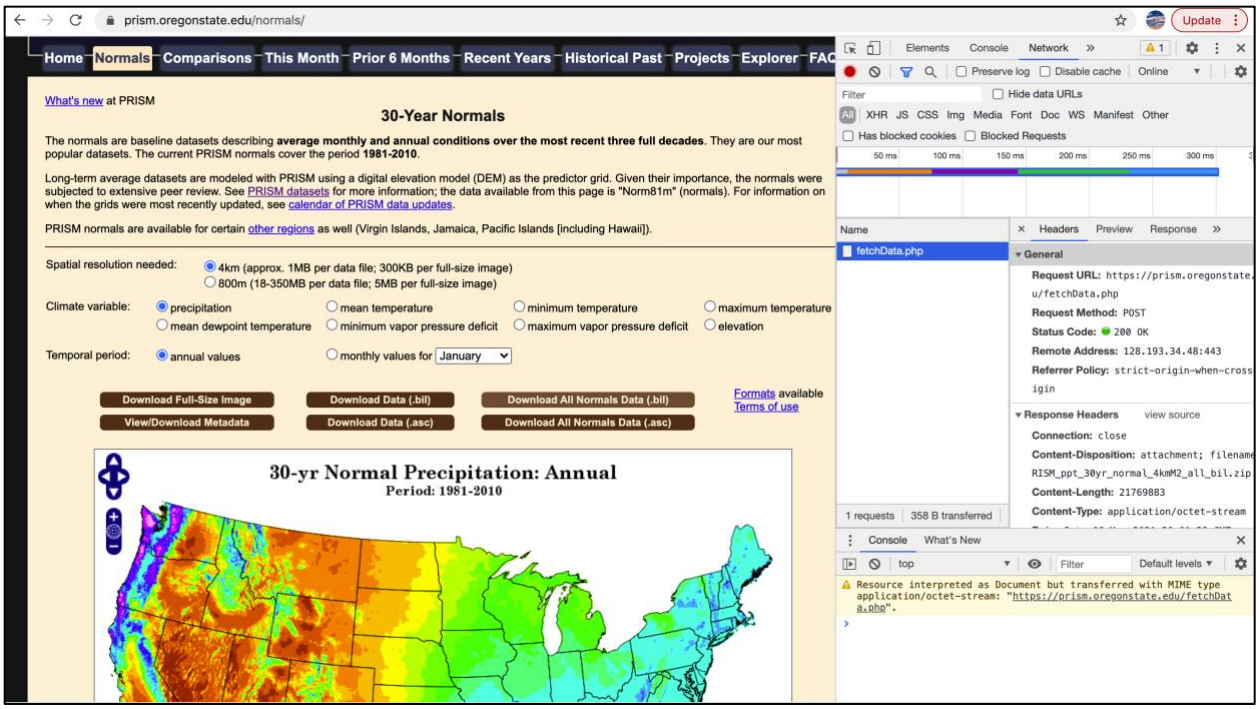


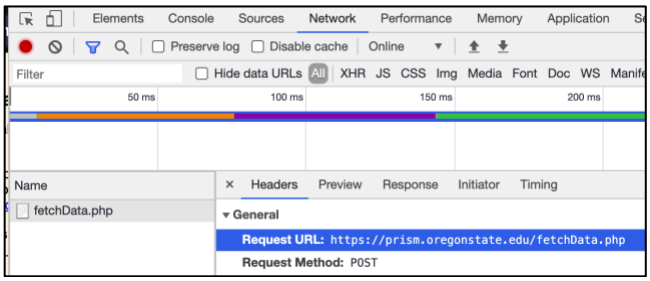
Figure 9. *Visualization of a Multidimensional Raster (source: Esri)*

However, if we wish to make every step of this process executable in a Jupyter Notebook, we need to develop a method that requests this data in a Python script and then saves the data to our

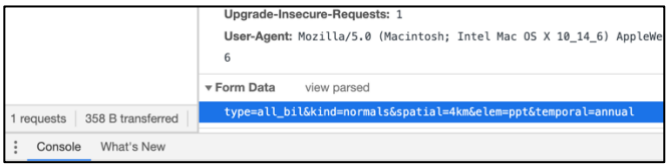
local disk. Right clicking the “Download All Normals Data” button does not give us any options to ‘copy the link address,’ so we will have to use Google Chrome’s developer tools to see how that button is linking to the .bil data.



A.



B.



C.

Figure 10. ***(A.)*** *Viewing the “Network” tab on Google Chrome’s ‘Developer Tools’ while clicking the link to the “Download All Normals Data” button, to see what is happening on the back end of the website.* ***(B.)*** *Viewing the Request URL in the Network tab.* ***(C.)*** *Retrieving the URL parameters from the Network tab.*

Clicking on the button with the Google Chrome’s Developer Tools open (specifically, the ‘Network’ tab), we can view the **Request URL** that this button is linking **Figure 10(B)** (and also, the type of request needed to fetch the information). Additionally, we can view the parameters that are sent along with this request URL **Figure 10(C).**

With this information, we can send a POST request to the request URL, save the returned object to a variable, and then extract the contents of the zip file to our local disk:

Assigning URL path to PRISM data server to var:

```
In [54]: prism_base = r"https://prism.oregonstate.edu/fetchData.php"
```

Assigning URL path w/ relevant variables embedded (linking to 30-year normals data) to var:

```
In [55]: allnormals_params = r"type=all_bil&kind=normals&spatial=4km&elem=ppt&temporal=annual"

         # (variables discovered using Google Chromes Developer Tools ('Network'))
```

Constructing URL path which will yield 30-year normals .bil files when inputted into a 'request'

```
In [56]: path_2_bils = prism_base + "?" + allnormals_params
         path_2_bils
Out[56]: 'https://prism.oregonstate.edu/fetchData.php?type=all_bil&kind=normals&spatial=4km&elem=ppt&temporal=annual'
```

saving data returned from requesting above URL to var:

```
In [57]: bils_obj = requests.post(path_2_bils)
```

printing data to disk:

```
In [58]: output_path = r"C:\Users\michaelfelzan\Documents\arc ii labby ii\PRISM_BILS"

         thezip = zipfile.ZipFile(io.BytesIO(bils_obj.content))
         thezip.extractall(output_path)
```

```
In [ ]: env.workspace = r"C:\Users\michaelfelzan\Documents\arc ii labby ii\PRISM_BILS"

        bilfolder_path = r"C:\Users\felza001\Desktop\ArcII_Lab02_Working\PRISM_BILS\bil_to_tifs"
```

Part 7: Converting .bil files into a spacetime cube & exporting spacetime cube file to local disc

As depicted in the **Figure 8.** flow map, there are many steps associated with converting a set of raster images into a spacetime cube. The .bil files in their original form do not contain "multidimensional information" like files such as NetCDF would – we will have to hard-code the 'dimensions' into our data.

The first step in our process is to funnel only the relevant data retrieved from our POST request into a folder – we only want the .bil files for months 1 – 12. We will then create an empty mosaic dataset in our working file geodatabase to add all 12 .bil rasters to.

```
In [74]: arcpy.CreateMosaicDataset_management(
             "arciilabii_clean_runthru.gdb","mosaic_ds2",
             "GEOGCS['GCS_North_American_1983',DATUM['D_North_American_1983',SPHEROID['GRS_1980',6378137.0,298.257222101]],PRIMEM['Greenw
             None, '',
             "NONE", None)
```

Out[74]:

Output

C:\Users\michaelfelzan\Documents\arc ii labby ii\arciilabii_clean_runthru\arciilabii_clean_runthru.gdb\mosaic_ds2

Adding rasters to mosaic dataset:

In [76]:
```
arcpy.management.AddRastersToMosaicDataset("mosaic_ds2",
                                           "Raster Dataset",
                                           r"'C:\Users\michaelfelzan\Documents\arc ii labby ii\PRISM_BILS\PRISM_ppt_30yr_normal_
                                           "UPDATE_CELL_SIZES",
                                           "UPDATE_BOUNDARY",
                                           "NO_OVERVIEWS",
                                           None, 0, 1500, None,
                                           '', "SUBFOLDERS",
                                           "ALLOW_DUPLICATES",
                                           "NO_PYRAMIDS",
                                           "NO_STATISTICS",
                                           "NO_THUMBNAILS",
                                           '',
                                           "NO_FORCE_SPATIAL_REFERENCE",
                                           "NO_STATISTICS",
                                           None,
                                           "NO_PIXEL_CACHE",
                                           r"C:\Users\michaelfelzan\AppData\Local\ESRI\rasterproxies\mosaic_ds2")
```

Out[76]:

Output
a Layer object

Next, we must create fields for the “Variable” and “Dimensions” of our mosaic dataset (and also create a field for the one dimension we will have, which is time (StdTime)). We can both create new fields and calculate those fields by running the arcpy.management.CalculateField() tool once:

```
arcpy.management.CalculateField(r"mosaic_ds2\Footprint", "Variable", '"precipit"', "PYTHON3", '', "TEXT")
```

Output
a Layer object

```
arcpy.management.CalculateField(r"mosaic_ds2\Footprint", "Dimensions", '"StdTime"', "PYTHON3", '', "TEXT")
```

Output
a Layer object

```
arcpy.management.CalculateField(r"mosaic_ds2\Footprint",
                                "StdTime",
                                '!Name![28:30] + r"/01/2019"',
                                "PYTHON3",
                                '',
                                "TEXT")
```

Output
a Layer object

The “Variable” field for each raster can be filled with an arbitrary text string, though the Dimensions field needs to contain a text string that corresponds with the “time” field’s name. Because we will create a time field called StdTime, “StdTime” will be filled in to all of the raster’s ‘Dimensions’ field values. Lastly, the StdTime field can be arbitrary datetime values that are in a time step. Because the 30 Year Normals data represents monthly averages over 30 years, I decided to pick an arbitrary year (2019) and have each raster represent the first day of months January thru December.

Another step I took was to “synchronize” the mosaic dataset, so that the edits to the fields I made could be registered (using arcpy.management.SynchronizeMosaicDataset(), toggling on the “update fields” param).

Finally, to register the mosaic dataset as a *multi-dimensional* mosaic, I used the arcpy function BuildMultidimensionalInfo(), referencing the Variable and StdTime fields as the multidimensional info:

Build Multidimensional info for mosaic dataset:

```
arcpy.md.BuildMultidimensionalInfo("mosaic_ds2",
                                   "Variable",
                                   "StdTime 'time dimension' month",
                                   "precipit precipitation water")
```

3]:

Output
a Layer object

Once the multidimensional info has been built for the mosaic dataset, the mosaic can finally be converted into a Multidimensional Raster. There seem to be a couple ways to perform this transformation – I had success using the MakeMultidimensionalRasterLayer() function:

Make Multidimensional Raster:

```
arcpy.md.MakeMultidimensionalRasterLayer("mosaic_ds2",
                                         "multidimensional.crf",
                                         "precipit",
                                         "ALL",
                                         None, None,
                                         '', '', '',
                                         None, '',
                                         "-125.020833333333 24.0625 -66.479166666662 49.937500000002",
                                         "DIMENSIONS")
```

:[5]:

Output
a Layer object

Once a Multidimensional Raster layer has been created, it may be exported into an animation of the weather data timeseries, or converted into a spacetime cube to be saved to disk, using the arcpy.CreateSpaceTimeCubeMDRasterLayer_stpm() function:

Creating Space Time Cube from Multidimensional Raster:

```
arcpy.CreateSpaceTimeCubeMDRasterLayer_stpm("multidimensional.crf",
                                            r"C:\Users\michaelfelzan\Documents\arc ii labby ii\arciilabii_clean_runthru\precip_s
                                            "ZEROS")
```

[7]:

Output
C:\Users\michaelfelzan\Documents\arc ii labby ii\arciilabii_clean_runthru\precip_spacetimecube2.nc

Part 8: <u>Exporting an animation of the weather data timeseries</u>

For this step, I could not successfully develop a method in my Jupyter Notebook ETL which programmatically exports an animation of the weather data timeseries. I believe this would be a fairly straightforward step in a Jupyter Notebook environment that is not Esri's, as there are modules out there that allow you to construct an animated .GIF file from 12 .TIFs/.JPEGs/etc.

However, I was able to export an animation of the time series using ArcGIS Pro's GUI (yes I know….theoretically everything you can do in the ArcPro GUI you can supposedly do in a Notebook. However, I exhausted enough hours scouring the sparse documentation that exists for doing this, to the point where, I felt it was counter-productive to even try to use ArcPro to make the animation…isn't the point of programmatically performing functions usually to ***save*** time??). Though, because the process of making an animation is only a few clicks away in the ArcPro GUI, I decided to go with that:

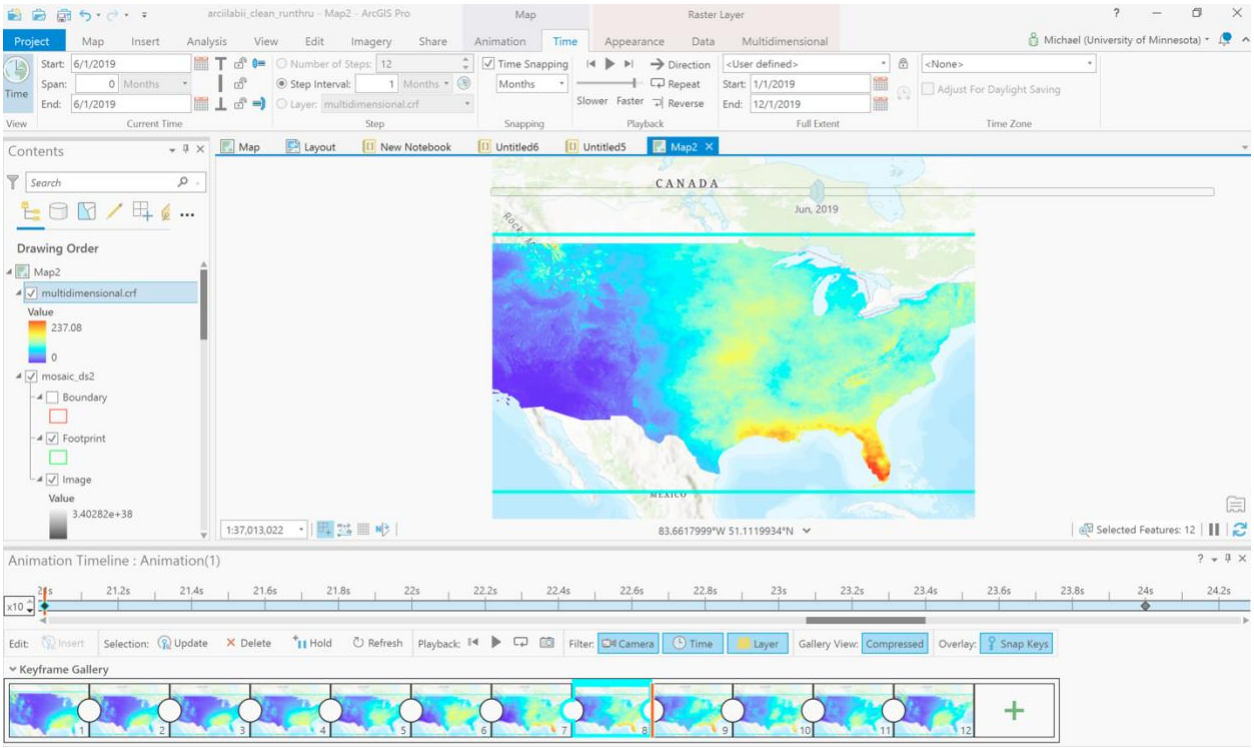


Figure 11. *Creating an animation of the 30 Year Normals time series (averages over 12 months) in the ArcPro GUI. To create keyframes for each time step, in the* ***Time*** *tab, the Start/End span may be specified, along with the number of steps, the step interval, and the unit for the step interval. After this is completed, in the* ***Animation*** *tab, you can simply click "import steps as keyframes."*

Results / Results Verification

(See GitHub for PDF layouts, spacetime cube file, .gif of time series animation)

The tangible results of the methods outlined in this report may be viewed in GitHub (the spacetime cube .nc file and the animation .gif), though one of the main objectives was to

establish an ETL in Jupyter Notebook which 1. downloads a .LAS file and converts it to both a TIN and DEM raster, and 2. download '30 Year Normals' weather data for the US and create a spacetime cube from 12 rasters (.bils). I believe this portion of the objective was clearly achieved (see Methods).

It may be hard to justify the "correctness" of my spacetime cube from a quantitative perspective, but the fact that I achieved the outputted file, and carefully followed the tutorials of others (online), I am led to believe I achieved the correct output.

Discussion and Conclusion

This report demonstrates how an ETL methodology which operates in ArcPro Jupyter Notebooks that downloads a .LAS file from the MN DNR FTP server, converts the file into DEM and TIN formats, exports PDF layouts of the layers, and explores the 2D vs. 3D visualization provided by ArcPro. This report also successfully demonstrates a Python-based ETL which downloads annual 30-Year Normals .bil files from the PRISM website, converts the .bil files into a spacetime cube & exports the file to local disc, and demonstrates how an animation of timeseries data may be created.

References

(1.) https://pro.arcgis.com/en/pro-app/latest/tool-reference/3d-analyst/las-dataset-to-tin.htm

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	
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	
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	
Verification	Results are correct in that they have been verified in comparison to some standard. The standard is clearly stated **(10 points)**, the method	20	

	of comparison is clearly stated **(5 points)**, and the result of verification is clearly stated **(5 points)**.		
		100	