Title: **Lab Report 0**
Notice: Dr. Bryan Runck
Author: Michael Felzan
Date: February 2, 2021

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

Abstract

This lab was intended for students to tour the "technology ecosystem," through executing a basic GIS function (buffering a network dataset) in three different environments: ArcGIS Pro, Jupyter Notebooks in ArcGIS Pro, and Arc Online Notebooks. Secondary goals were to become acquainted with lab report guidelines and formats for the course, and to learn how to use Git/Bash/GitHub to version control this/future lab reports.

Problem Statement

The objective of this lab was to perform a 'buffer' on any network dataset of our choosing, using 1. ArcGIS Pro, 2. Jupyter Notebooks in ArcGIS Pro, and 3. Esri Online Jupyter Notebooks. Once we had successfully performed a buffer in all three environments, we were to compare and contrast the processes in each of the environments.

The network dataset I chose to use is the "Bikeways" shapefile from MNDOT, which contains geographic information of all the bikeways in the Twin Cities metro area. Using this file, the illustrative figure below describes the Lab 0 task at hand:

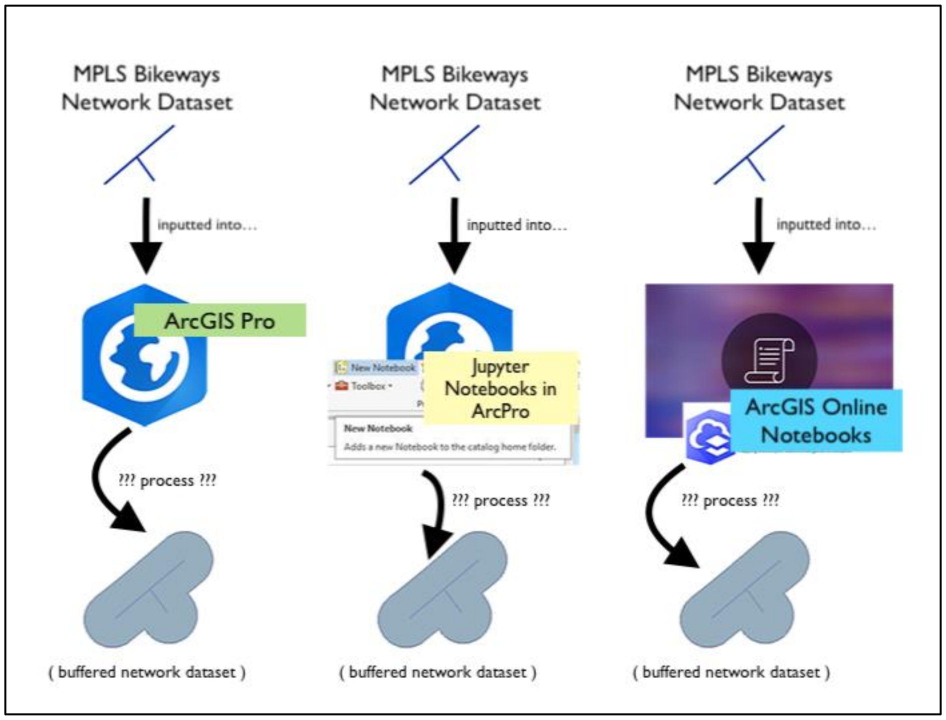

MPLS Bikeways Network Dataset
MPLS Bikeways Network Dataset
MPLS Bikeways Network Dataset
inputted into...
inputted into...
inputted into...
ArcGIS Pro
Jupyter Notebooks in ArcPro
ArcGIS Online Notebooks
New Notebook
Toolbox
New Notebook
Adds a new Notebook to the catalog home folder.
??? process ???
??? process ???
??? process ???
(buffered network dataset)
(buffered network dataset)
(buffered network dataset)

Figure 1. ***Infographic that depicts the lab objective: buffering a network dataset using three different environments (ArcPro, Notebooks in Pro, Notebooks in ArcOnline), and comparing the differences in processes of each***

Table 1. ***Required datasets to complete the Lab 0 objective.***

#	Requirement	Defined As	Spatial Data	Attribute Data	Dataset	Preparation
	Network Dataset	Comprehensive shapefile of MPLS bikeways from MNDOT	Line geometries		MN GeoSpatial Commons	none

Input Data

As mentioned above, the data used in this study is the Minneapolis "Bikeways" shapefile from MNDOT, made available through the MN GeoSpatial Commons. The dataset describes bicycle routes in the Twin Cities metropolitan area, and includes attributes such as name, type, width, length, stops, quality, etc. The dataset was initially digitized from 2003 to 2007, though it is still being updated (last update was this year).

Table 2. ***Further describing the input data ("bikeways" network dataset) and its purpose in the analysis.***

#	Title	Purpose in Analysis	Link to Source
1	"Bikeways" (MPLS)	To be used as source data to demonstrate the process of buffering a network dataset in three different environments	MN GeoSpatial Commons

Methods

(SEE FIG. 2 ON NEXT PAGE)

I. ArcGIS Pro

Using ArcGIS Pro to accomplish the task of buffering a network dataset differs from the two other environments, mainly because you are fully using a "GUI" (graphical user interface) to input and process the data. When you open ArcGIS Pro, your first step is to import or route to your data. To import the "bikeways" network dataset, we must "Add a Folder Connection" to where the data resides on our computer. This is essentially, saving within our project file the *filepath* to this folder, except we are navigating a file explorer GUI instead of hard-coding the filepath. Once the data is located and incorporated into our project file, we can visualize the data using ArcPro's fully interactive interface, and also examine the attribute information embedded in the data.

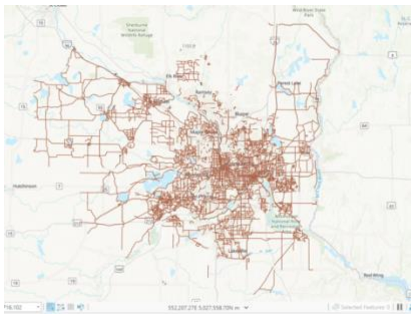
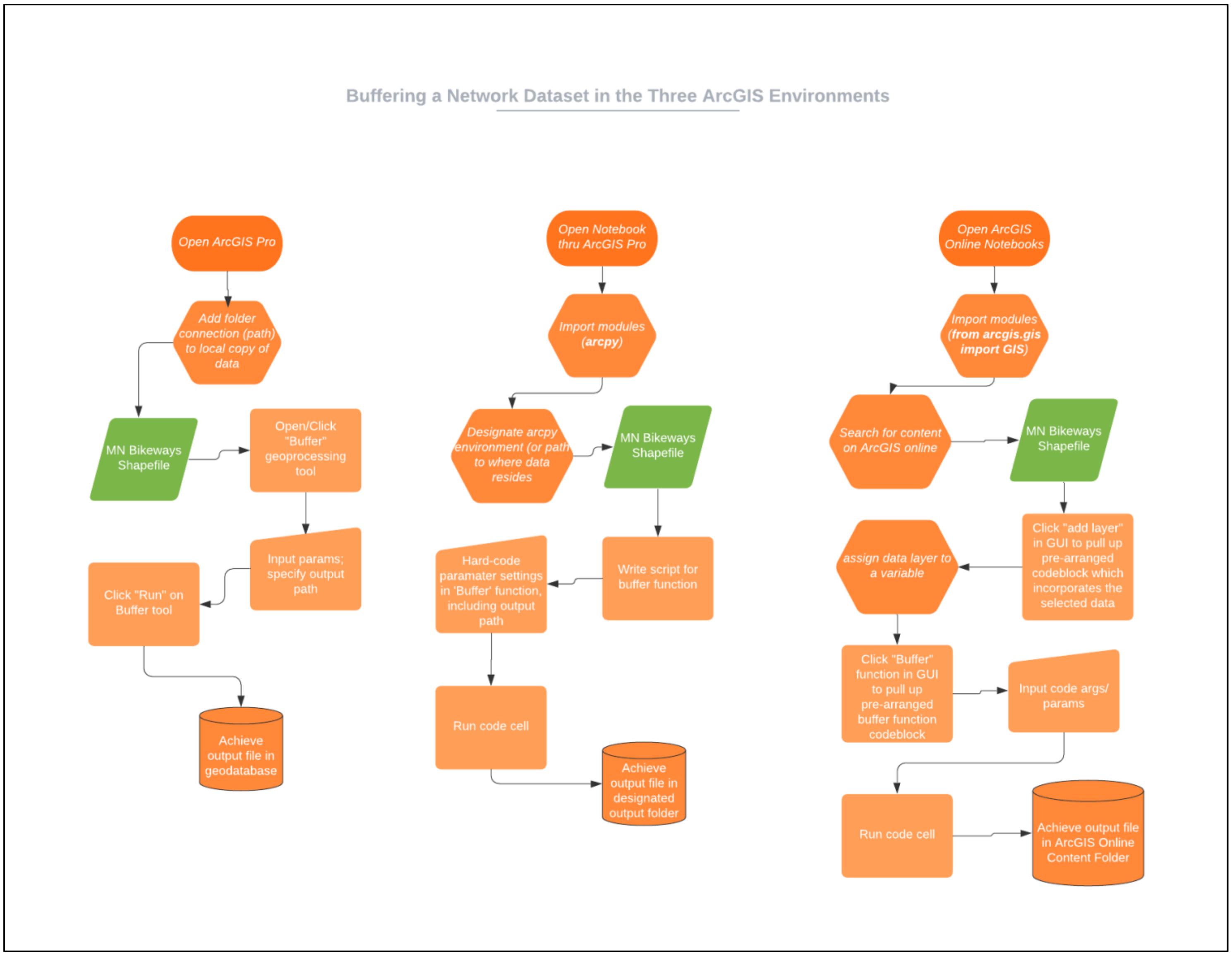


Fig. 2 depicts what our "bikeways" network dataset looks like after being pulled into ArcPro. The next step is to perform a "buffer" function on all of the line geometries in the dataset. To do this, we need to navigate to the ArcPro ribbon and select "Tools," which activates a pop-up window where we can scroll through various built-in ArcPro geoprocessing functions (again, completely GUI).

Fig. 3 "*bikeways" visualized within ArcPro*

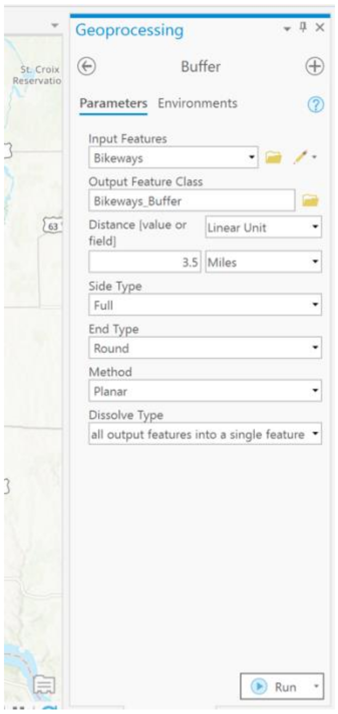


Buffering a Network Dataset in the Three ArcGIS Environments
Open ArcGIS Pro
Add folder connection (path) to local copy of data
MN Bikeways Shapefile
Open/Click "Buffer" geoprocessing tool
Input params; specify output path
Click "Run" on Buffer tool
Achieve output file in geodatabase
Open Notebook thru ArcGIS Pro
Import modules (arcpy)
Designate arcpy environment (or path to where data resides
MN Bikeways Shapefile
Write script for buffer function
Hard-code paramater settings in 'Buffer' function, including output path
Run code cell
Achieve output file in designated output folder
Open ArcGIS Online Notebooks
Import modules (from arcgis.gis import GIS)
Search for content on ArcGIS online
MN Bikeways Shapefile
Click "add layer" in GUI to pull up pre-arranged codeblock which incorporates the selected data
assign data layer to a variable
Click "Buffer" function in GUI to pull up pre-arranged buffer function codeblock
Input code args/ params
Run code cell
Achieve output file in ArcGIS Online Content Folder

Fig. 2 ***Flow map depicting the similarities/differences in processes involved with buffering a dataset using three different ArcGIS environments.***

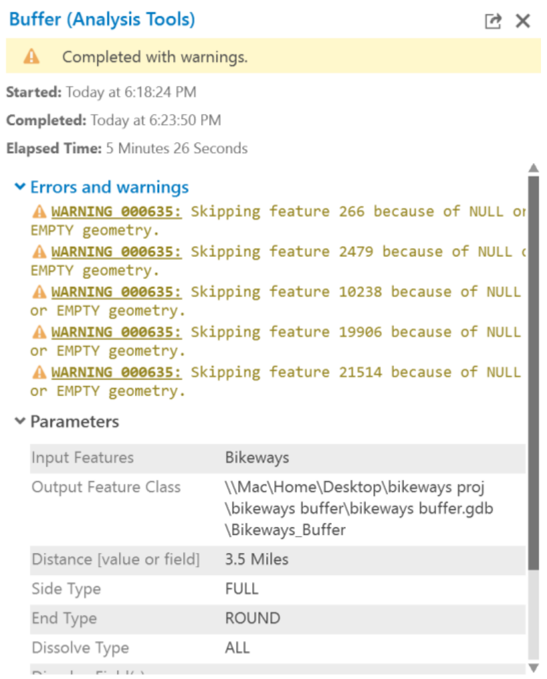
Using the ArcPro GUI to navigate through the various geoprocessing tools, we find the one we're looking for ("buffer"), and double-click it. This brings up another pop-up window, one for the buffer function. Here, we must fill out all of the blank boxes with the needed information to run the function. The information inputted to these boxes are **parameters**/ **arguments** for the underlying code that makes up the function. Conveniently, this window is also interactive, and the user may click on drop-down arrows within the blank boxes, to be prompted with the "possible" or "recommended" inputs for the function. An important argument the user must fill out here is the "Output Feature Class," which is essentially the name of the output layer, and the path to where that outputted file will go. Clicking the folder icon by this box, we may navigate to the output folder location using a file explorer GUI, and then enter the output name like you would in any other file explorer context. However, clicking within the Output Feature Class box will reveal that the full file path is saved within that parameter.

Once all of the function parameters are filled out, we may click "Run" at the bottom of the geoprocessing window. This might take a moment to run if your dataset has many line geometries that need buffering (as the "bikeways" dataset has). Once the function has completed running, or if the function fails to run properly, another pop-up window will appear, detailing what happened during the execution of the function.



Input Features: Bikeways
Output Feature Class: Bikeways_Buffer
Distance [value or field]: Linear Unit
3.5 Miles
Side Type: Full
End Type: Round
Method: Planar
Dissolve Type: all output features into a single feature

Fig. 4 ***Buffer geoprocessing window***



Completed with warnings.

Started: Today at 6:18:24 PM

Completed: Today at 6:23:50 PM

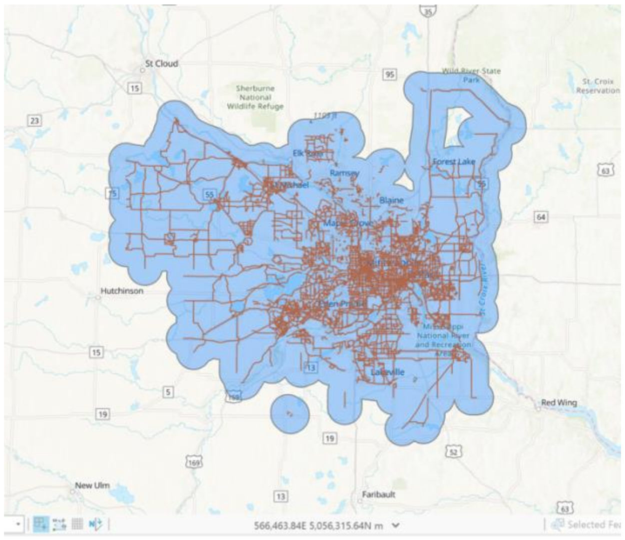
Elapsed Time: 5 Minutes 26 Seconds

```
WARNING 000635: Skipping feature 266 because of NULL o
EMPTY geometry.
WARNING 000635: Skipping feature 2479 because of NULL 
EMPTY geometry.
WARNING 000635: Skipping feature 10238 because of NULL
or EMPTY geometry.
WARNING 000635: Skipping feature 19906 because of NULL
or EMPTY geometry.
WARNING 000635: Skipping feature 21514 because of NULL
or EMPTY geometry.
```

Input Features	Bikeways
Output Feature Class	\\Mac\Home\Desktop\bikeways proj \bikeways buffer\bikeways buffer.gdb \Bikeways_Buffer
Distance [value or field]	3.5 Miles
Side Type	FULL
End Type	ROUND
Dissolve Type	ALL

Fig. 5. ***Pop-up window that appears after running "buffer" analysis. In this case, the function was properly executed, though "completed with warnings" (some features were not buffered because they contained null geometries)***

Fig. 5 shows an example of what one may receive as a pop-up window after running a function. In this case, there were no errors in the code/ parameters we entered before, though the analysis "completed with warnings." ArcPro has skipped executing the buffer function on 5 specific line geometries within the dataset, because they had NULL geometries. The function was still able to operate on all of the other line geometries because Esri built into the buffer function something like, "IF geometry = NULL: FLAG with WARNING, CONTINUE."



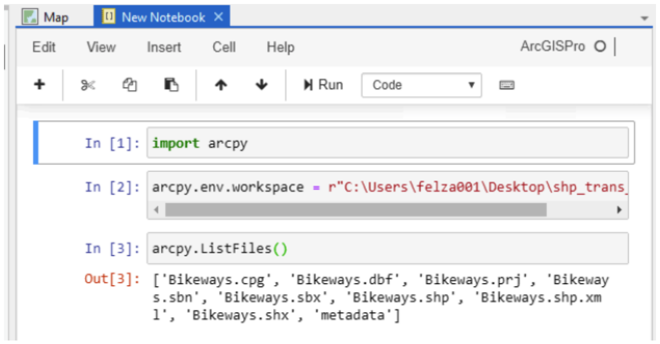
If we visualize the new layer outputted from the function, we can see that a 3.5 mile (dissolved) buffer was successfully created around our network dataset.

Fig. 6 ***Visualization of buffer layer created using ArcPro (visualized in ArcPro)***

II. <u>Jupyter Notebooks within ArcGIS Pro</u>

Now we will repeat the same process of buffering the "bikeways" network dataset using the Jupyter Notebooks within ArcGIS Pro. Our first step is to open a blank notebook.

If we glance at the *Figure 2* flowmap, we can see that the fundamentals of each process are similar, but there are slight differences in the execution of each method. Much like in part I., one of our first steps is to "route" to our data. However, there is an extra preparation step required when using an ArcGIS Pro Notebook:

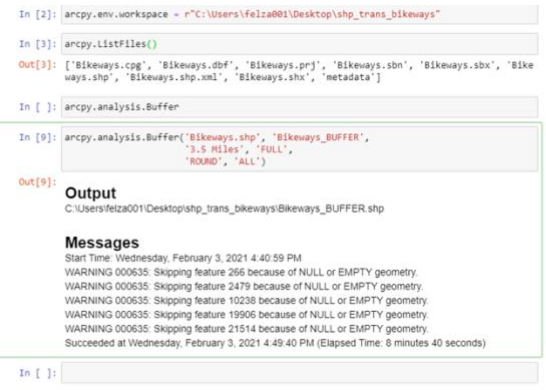


```
In [1]: import arcpy

In [2]: arcpy.env.workspace = r"C:\Users\felza001\Desktop\shp_trans_

In [3]: arcpy.ListFiles()
Out[3]: ['Bikeways.cpg', 'Bikeways.dbf', 'Bikeways.prj', 'Bikeway
        s.sbn', 'Bikeways.sbx', 'Bikeways.shp', 'Bikeways.shp.xm
        l', 'Bikeways.shx', 'metadata']
```

In order to gain access to all of the built-in ArcPro functionality, we must import it. We do this by writing "import arcpy" in a cell and running it. Something built into arcpy is the ability to assign your "workplace environment," which assigns your working directory to whatever path you enter. This is essentially the same as "adding a folder connection" like we did in the previous step. Once the working environment is assigned, we can then call an arcpy function to list all of the files in that directory (much like hitting the dropdown arrow in the folder connections) (**Fig 7**)

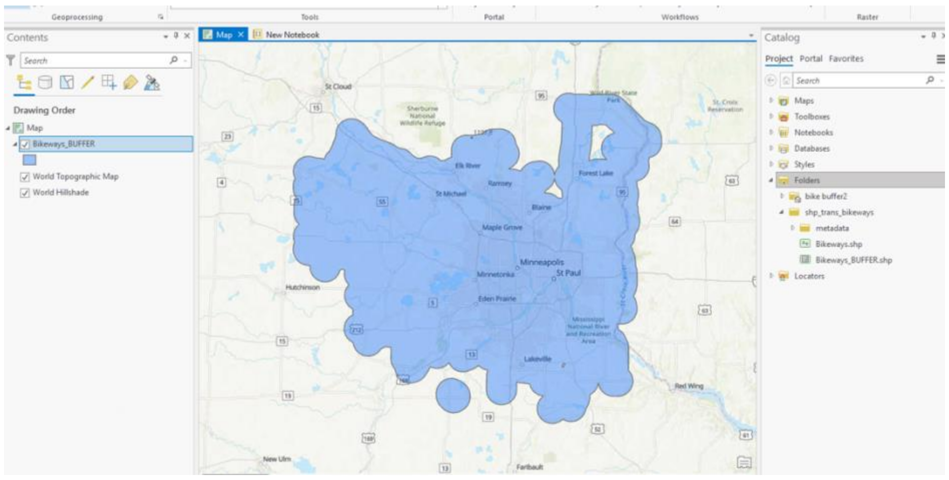
Fig. 7: U*sing 'arcpy' in an ArcPro Notebook.*

Now that we have properly routed to our data, we need to execute the "buffer" function on the dataset. Though instead of using a GUI interface to enter in what dataset the function should be performed on, where the output will be routed, and all of the parameters/arguments for the function, we need to hard-code all of this information. The figure to the right depicts how one could call the "buffer" function from the arcpy library, and enter in the same parameters that we used in **Fig 4.**



```
In [2]: arcpy.env.workspace = r"C:\Users\felza001\Desktop\shp_trans_bikeways"

In [3]: arcpy.ListFiles()
Out[3]: ['Bikeways.cpg', 'Bikeways.dbf', 'Bikeways.prj', 'Bikeways.sbn', 'Bikeways.sbx', 'Bike
        ways.shp', 'Bikeways.shp.xml', 'Bikeways.shx', 'metadata']

In [ ]: arcpy.analysis.Buffer

In [9]: arcpy.analysis.Buffer('Bikeways.shp', 'Bikeways_BUFFER',
                              '3.5 Miles', 'FULL',
                              'ROUND', 'ALL')
Out[9]:
        Output
        C:\Users\felza001\Desktop\shp_trans_bikeways\Bikeways_BUFFER.shp

        Messages
        Start Time: Wednesday, February 3, 2021 4:40:59 PM
        WARNING 000635: Skipping feature 266 because of NULL or EMPTY geometry.
        WARNING 000635: Skipping feature 2479 because of NULL or EMPTY geometry.
        WARNING 000635: Skipping feature 10238 because of NULL or EMPTY geometry.
        WARNING 000635: Skipping feature 19906 because of NULL or EMPTY geometry.
        WARNING 000635: Skipping feature 21514 because of NULL or EMPTY geometry.
        Succeeded at Wednesday, February 3, 2021 4:49:40 PM (Elapsed Time: 8 minutes 40 seconds)

In [ ]:
```

Fig. 8: ***"hard-coding" buffer function on "bikeways" using same parameters as were entered in Fig. 4***

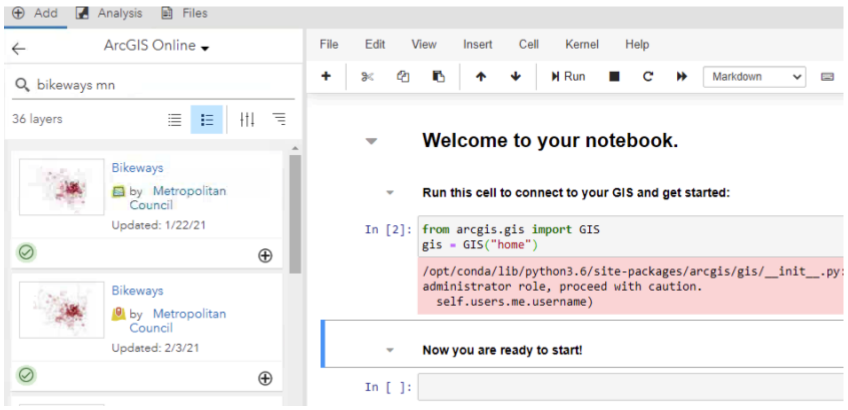


Because we are working in a Jupyter Notebook within ArcPro, we may flip back to our ArcPro "Map View" to visualize the layer that was just created/outputted. This output looks identical to what was created in **Fig. 6,** which reaffirms that this method attained the same result as the process used in section I.

Fig. 9: ***visualization of the buffer layer created using ArcPro Notebooks***

III. <u>Jupyter Notebooks in ArcGIS Online</u>

Finally, we will again buffer the "bikeways" network dataset, but this time perform the action using ArcOnline Notebooks. As with the above section II., our first step will be to import the packages/modules we need, which contain the code for the "buffer" function. However, the process immediately differs from section II., in that we are importing the "GIS" library instead of "arcpy." As arcpy utilizes the packages that are downloaded onto your computer when you download ArcGIS Pro, and the ArcOnline notebooks are designed to be entirely web-based, we need to call from a different library. This will slightly change our process moving forward.



Welcome to your notebook.

Run this cell to connect to your GIS and get started:

```
In [2]: from arcgis.gis import GIS
        gis = GIS("home")

        /opt/conda/lib/python3.6/site-packages/arcgis/gis/__init__.py:
        administrator role, proceed with caution.
          self.users.me.username)
```

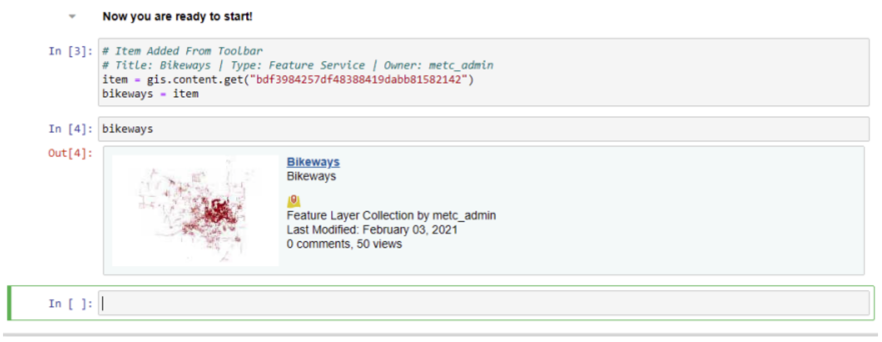
Now you are ready to start!

```
In [ ]:
```

Fig. 10: ***Importing "GIS" to get started with ArcOnline Notebooks***

The next step, as following the trends of the previous two sections, is to import/locate/route to our data. However, because ArcOnline is web-based, we cannot route to our "bikeways" network dataset if it *only* resides on our computer hard disk. We must either upload the data to our personal Esri Online content cloud, or, search for the original data via ArcGIS Online. I was able to do the latter and find the exact dataset I had been working with, posted by the Metropolitan Council (**Fig. 10)**.

Once the data is located, via using the GUI search-box attached to the notebook, the user may click the dataset and preview its content and metadata. If it is up to snuff, the user may click the (+) button at the bottom of the thumbnail, which triggers the ArcOnline notebook to copy the path to where that data resides (on the cloud), and inputs a new code cell a pre-made codeblock that allows the user to map that datapath to a variable **(Fig 11).**



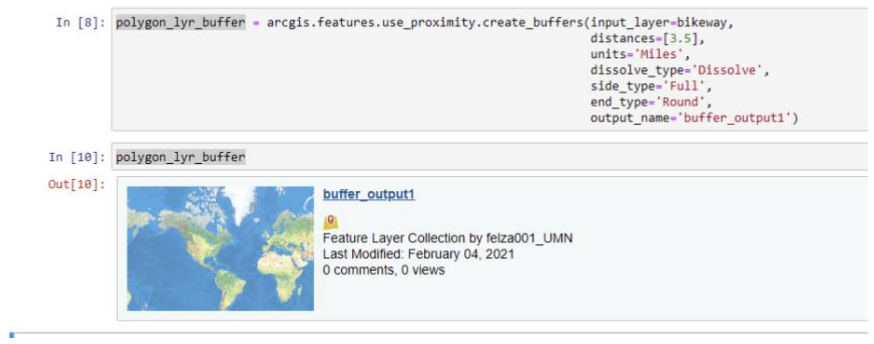
```
Now you are ready to start!

In [3]: # Item Added From Toolbar
        # Title: Bikeways | Type: Feature Service | Owner: metc_admin
        item = gis.content.get("bdf3984257df48388419dabb81582142")
        bikeways = item

In [4]: bikeways

Out[4]:
        Bikeways
        Bikeways

        Feature Layer Collection by metc_admin
        Last Modified: February 03, 2021
        0 comments, 50 views

In [ ]: |
```

Fig. 11: ***clicking the (+) button at the bottom of the data thumbnail in the ArcOnline search-bar triggers ArcOnline to copy the path to where that data resides to a new codeblock***

The user may then make slight edits to that codeblock, like renaming the gis “item” with a descriptive variable name. Running a cell containing *only* that new variables name will provide the user with an interactive, in-line pop-up of the data layer, including a thumbnail preview of what the layer looks like, and where it is posted on the Esri cloud (**Fig 11)**.

Now that we have captured our data within a variable, we may call the “buffer” function from the GIS library, and hard-code all of the buffer parameters we need for the function to run. However, because we are calling ‘buffer’ from GIS instead of arcpy, the syntax of the function will differ slightly than section II.

Here, we will the function using “arcgis.features.use_proximity.create_buffers()” instead of “arcpy.analysis.Buffer().” The syntax of the parameter inputs also differs slightly, as here we enter in our distance value and units as two separate arguments, whereas with arcpy, we entered in the string “3.5 Miles” as one argument input.

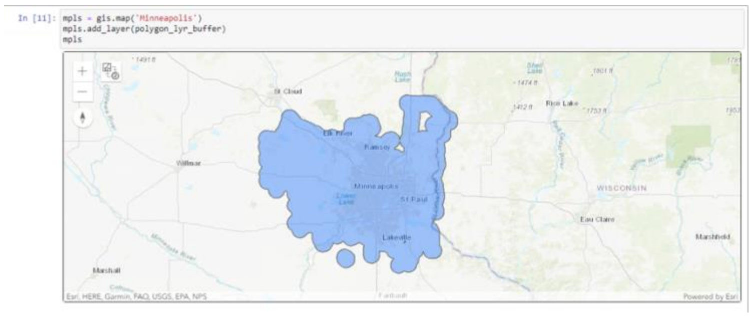


```
In [8]: polygon_lyr_buffer = arcgis.features.use_proximity.create_buffers(input_layer=bikeway,
                                                                            distances=[3.5],
                                                                            units='Miles',
                                                                            dissolve_type='Dissolve',
                                                                            side_type='Full',
                                                                            end_type='Round',
                                                                            output_name='buffer_output1')

In [10]: polygon_lyr_buffer

Out[10]:
         buffer_output1

         Feature Layer Collection by felza001_UMN
         Last Modified: February 04, 2021
         0 comments, 0 views
```

Fig. 12: ***hard-coding the “buffer” function and its related parameters, using ArcOnline Notebooks***

After the function is done running, we are given no indication that it worked/ any warning flags. However, if we call the variable we assigned the function codeblock to, we can see **(Fig. 12)** that the function outputted a layer with the name we gave it in the parameters, and that the data layer resides in our Esri Online content cloud.

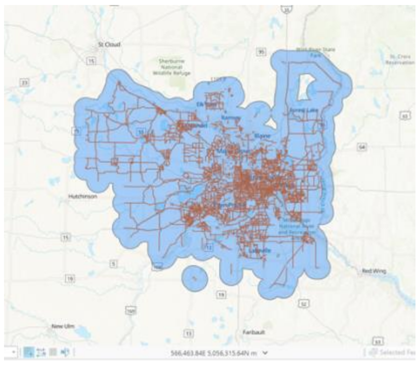
Mapping this layer to a gis basemap (codeblock demonstration shown in **Fig. 13**) we are assured that the function, although involving a slightly different process, produced the same buffer result as the ArcPro and ArcPro Notebooks did.



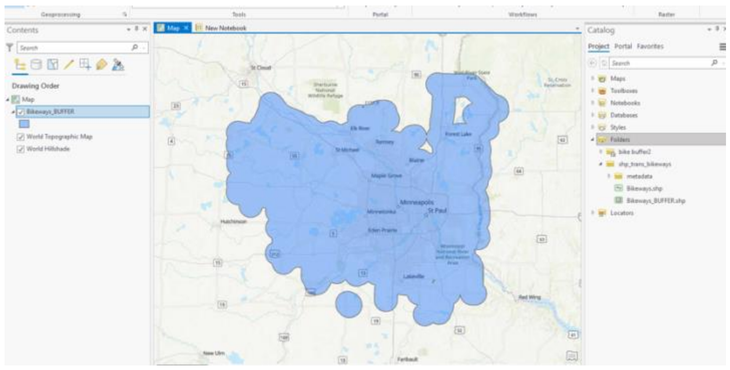
```
In [11]: mpls = gis.map('Minneapolis')
         mpls.add_layer(polygon_lyr_buffer)
         mpls
```

Fig. 13: ***visualization of the buffer output, created/visualized within ArcOnline Notebooks***

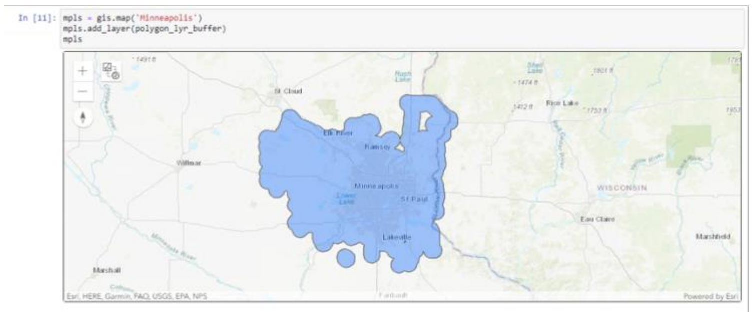
<u>Results / Results Verification</u>



a.



b.



```
In [11]: mpls = gis.map('Minneapolis')
         mpls.add_layer(polygon_lyr_buffer)
         mpls
```

c.

Fig. 14: ***a. visualization of the "bikeways" buffer output created in ArcPro; b. visualization of buffer output created in ArcPro Jupyter Notebooks; c. visualization of buffer output created in ArcOnline Notebooks***

In the previous section, and thru the flowmap in **Figure 2.,** the processes for buffering a network dataset in each of the three ArcGIS environments have been detailed. In this section, I will summarize the main differences between each of the methods.

The most obvious difference of using the ArcPro instead of either of the Jupyter Notebooks is the GUI. With ArcPro, the user must click actions like "create a folder connection" and route to the data via a file-explorer interface. The user does not necessarily need to see what the function code looks like on the back-end, but can instead click a series of actions on pop-up windows to perform an analysis (eg. Clicking the "Toolbox" icon, and then clicking "buffer"). Although the user has to manually enter in the parameter/argument values, much like you would in Python, there is still more of a user-friendly interface here, where the user can click drop-down menus and view "recommended" inputs for the functions.

Using ArcPro Jupyter Notebooks, the user must do more work on the front-end importing the needed modules and hard-coding the directory pathes to where their data is stored. However, because the user imports "arcpy" when working in this environment, we are essentially using the same exact functions as what's downloaded in the ArcPro package. In fact, one could literally go to the "history" after running a function in ArcPro, and copy to clipboard the exact code that was used to perform the function, paste it in the notebook, and it would run exactly the same. The ArcPro Jupyter Notebooks even provide you with the same run messages after you run a function, including warning flags (see **Fig. 5** & **Fig. 8**).

Jupyter Notebooks in ArcOnline are a weird in-between between the GUI offered in ArcPro and the hard-coding required in the ArcPro Jupyter Notebooks. The user may search for data layers online and in their personal repositories, and then click an interactive button for a code-block to appear. The user may also search for functions such as "buffer" in a GUI window beside the notebook, click the desired function, and have its associated code appear in a code-cell. A major difference between this environment and the other two is that you are working *solely online*, so you cannot access files from your computer unless you upload them to the cloud. A major difference between the ArcOnline and ArcPro Jupyter Notebook functionalities is the packages you need to import in order to use them – with the ArcPro Notebooks, you are importing arcpy, whereas with ArcOnline, you are using GIS. This difference changes the syntax of the code you write, and changes what possibilities of what you may do. Though in all three of these environments, if you get confused, you may easily pull up the documentation for the function you are using (in ArcPro, via clicking the "?"; in the Notebook environments, by either clicking Shift+Tab in-line, or by writing out the function, followed by a "?", and then running the cell.

In terms of results verification, a very coarse-grained analysis would be that all three methods "appeared" to produce the same result (see **Figure 14**). And that is what should be expected, because we used the same source data (bikeways) in each procedure, along with the same parameters (3.5 mile buffer, rounded line edges, full, dissolve, etc.). Another indication that the same results were achieved through both ArcPro environments is that the same warning flags were brought up from each (see **Fig. 5** & **Fig. 8**). However, to fully assess the interoperability of these three output layers, we would need to look at the attributes of each layer (precise total buffer area, etc.). Although I did not do this for this lab (as I figured the purpose was to mainly demonstrate the *processes* of buffering a dataset in each ArcGIS environment), I am fairly confident all three layers would have identical attributes, as all were created using the same(?) ArcPro engine.

Discussion and Conclusion

All three of the ArcGIS environments covered in this lab report each have their own advantages and disadvantages, depending on the task you wish to complete with them. If you are looking to perform spatial analyses that is visualization-heavy, or if you need to zoom in/out between scales, and analyze the attribute tables of shapefile features, your best bet may be ArcPro's robust visualization GUI. However, if you need to perform a string of data manipulation functions on a massive amount of data, but you already know *which* specific functions you need to execute, using a Jupyter Notebook would likely be the most efficient choice. Depending on whether your data is on your computer or in the cloud (or if you even have ArcPro installed on your computer), you may choose ArcOnline Notebooks over ArcPro Notebooks, and vice versa. I personally am more comfortable with the arcpy package, and am frequently working with data that I have local copies of…so I often turn to Jupyter Notebooks within ArcPro (or, launching Notebooks via the command line, after designating the "arcgispro (3)" conda environment). However, working in the Jupyter Notebooks within ArcPro seems to offer some powerful possibilities, like being able to visualize the layers you've created immediately. Additionally, the ArcOnline Notebooks may allow you the advantage of mapping data on the fly without downloading it, and easily sharing your findings with others (who may have varying levels of experience and local software)

GitHub

Having worked with GitHub last semester in Advanced Geocomputing, I thought it would be a cake-walk to configure it. And although I did already understand the fundamentals of how Git/ version controlling works, I had been using Git Bash through a Windows virtual machine. This time around, I decided I would set configure git through the Mac "Terminal" this time instead of Windows (what I had been used to), and I ran into a considerable amount of unexpected problems.

The first of these problems was trying to figure out via context clues (web searches) that "bash" ***IS*** the mac terminal, and that you do not need Homebrew or anything to download "bash." The second major obstacle I encountered was figuring out why I couldn't "cd" (change directory) into my folders…eventually realizing that you need a ~special syntax~ for folders that contain spaces in the names. I finally figured out that you may use "tab" to auto-complete the folder names, though I'm still quite confused how the forward and backslashes relate to calling a folder whose name contains a space.

Another point of confusion for me was figuring out what the ~$b file was that randomly appeared as an "untracked file" when I did a "git status." After some investigation, I learned that those files are "lock files," which are created when you have certain files open, and disappear after you close them.

An even larger point of confusion for me was GitHub's new double authentication password requirement. I had originally used GitHub where you just had one account password, and would enter it in to verify your account when committing/pushing a change. It took me an embarrassingly large amount of time to figure out how to use a double authentication program like 1Password, and how such an app could communicate with GitHub and generate temporary, 6-digit "key" passwords that only last for 30 seconds.

Additionally, I also managed to "diverge from main" when trying to set up my repo (I think I had made changes that were untracked using the GitHub website, and then made different untracked changes on the cloned folder on my computer). I couldn't remember how I successfully pulled/merged the two repos into one, but I definitely internalized the importance of tracking changes (in one place).

Finally, a mistake I learned from was deleting a file from my repo manually, and assuming when I staged the repo for commit, the manual deletion would be tracked as a "change." I later learned that best-practice is to use "git rm" (remove) on the file first thru bash, before you go and delete it manually from your computer.

References

Minnesota Department of Transportation. Shapefile. St. Paul, MN: Metropolitan Council, 2007
https://gisdata.mn.gov/dataset/us-mn-state-metc-trans-bikeways

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	**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	**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	**27**
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	**16**
		100	**95**