JANUARY 11, 2017

INTRODUCTION TO PYTHON AND GEOSPATIAL TECHNOLOGY

TOPIC 3

VINCENT A. DINOTO, JR.
JEFFERSON COMMUNITY AND TECHNICAL COLLEGE
Vince.dinoto@kctcs.edu

Contents

Figures

Assumptions

It is assumed that each geoprocessing tool discussed in this module will be a concept that the learner has functional knowledge. This type of information could have been gained through experiences in the geospatial field or through an introduction to geospatial technology course. If the functionality of a specific tool is unclear, review the introduction to gist website, http://introductiontogist.weebly.com. The complete functionality of the Python commands will be explained in the module.

Programming Mode

As noted in the previous module an IDE (Integrated Development Environment) has two modes of operation. In the immediate mode, when a return key is pressed at the end of the line, the line is executed; unless a looping function is used which requires a double return. In immediate mode, the content of variables are remembered from one line of code to the next. The programming mode will allow the user to write all lines of code, but the execution of the code will not occur until the scripts are saved and then run. A Python script is a text file, with the py extension instead of the normal text extension. When the script has the py extension, the IDE will automatically assume that the script is in the Python format. If a simple text editor is used make sure that the py extension is added when saved. The IDE editor will prompt the user with suggestions of command format while typing.

If the code runs successfully with no errors, in general, the results are displayed in the immediate mode window. If there were any errors associated with the execution of the code, then an error message will appear in the immediate mode window. The error messages can be several lines long and at times, they may be somewhat difficult to fully understand. For example, a simple typographical error can generate long error messages. Most of the error messages will reveal the line where the error is located which is very useful in the troubleshooting process. The author's experience has been that errors are usually typographical and not logical. For example, a missing ending bracket can cause serious errors.

Once the process runs successfully, the file or results will need to be reviewed, generally, in Esri ArcMap Desktop. In some processes when a new file is produced and an error occurs, the file will be created with the appropriate name, but not correctly populated. Once the error is corrected and the script is executed again, a new error might result because the file already exists and the script was not written to overwrite the current files. Thus when a file(s) is created as part of a script, check that no previous files of the same name were created during the testing phase. Delete any files while testing. Make sure that no files are open that are going to be addressed by the code, since this will also cause errors because the file is locked when open, unless it is a relational database.

Functionality

The following concepts will be covered in this module:

- arcpy
- buffer

- env (environment)
- merge
- clip
- tabular join
- spatial join
- geocoding

arcpy

Arcpy is a module, which contains the geoprocessing commands that can be used in a Python script. The arcpy module is included with the installation of Esri ArcMap Desktop and thus not a free library of commands. When using the Python window inside of ArcMap the module is automatically loaded, will be discussed in a future module. When working with the IDE outside of ArcMap the module must be loaded to accomplish the geoprocessing commands. Code created in an IDE can also be used inside of ArcMap without the need for modifications.

The arcpy module should be loaded as one of the first lines in any script. Loading a module is accomplished by using the import statement with the module. **Hint**: if the user loads the arcpy module in immediate mode prior to beginning a script, it will provide suggestions on geoprocessing commands while writing the script. **Note:** all the letters in arcpy are lower case, but in some writings, capitalization is used.

Format

To use tools from the arcpy module, an understanding the structure of the Esri ArcToolbox is very useful. In general, the formatting of a tool:

arcpy.tool_location(properties)

- where tool is the name of the tool
- location is the toolbox which contains the tool
- properties are the inputted parameters for the functionality of the tool.
- For example:

  arcpy.Union_analysis(input feature, output feature, join attributes, tolerance, gaps)

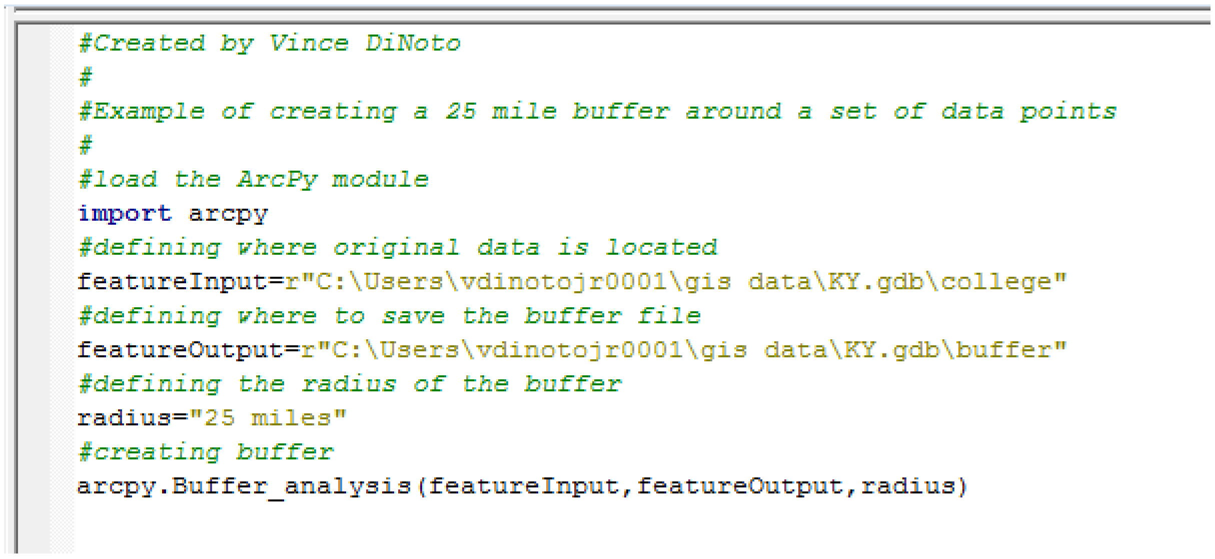
The final three parameters are optional and thus do not have to be included, if one of these commands is used then commas are required so that the proper command is applied. The Union tool is located in the analysis toolbox.

The data symbol to the left will appear at any location in which data is required to complete the technical skill lesson or assignment. Data used in the technical skill lessons are provided so the learner can follow the processes being demonstrated and can be downloaded. All data is provided in a single compressed file folder for the entire course. For technical skill assignments, in general the learner will need to locate data from external sources, in some cases it can be the same data set used in the technical skill lessons.

Example: Buffer

Create the code that will produce a 25-mile buffer around college campuses in Kentucky.

The code in Figure 1 will be discussed line by line, the comments will be omitted and only the functional part of the script will be examined.



```
#Created by Vince DiNoto
#
#Example of creating a 25 mile buffer around a set of data points
#
#load the ArcPy module
import arcpy
#defining where original data is located
featureInput=r"C:\Users\vdinotojr0001\gis data\KY.gdb\college"
#defining where to save the buffer file
featureOutput=r"C:\Users\vdinotojr0001\gis data\KY.gdb\buffer"
#defining the radius of the buffer
radius="25 miles"
#creating buffer
arcpy.Buffer_analysis(featureInput,featureOutput,radius)
```

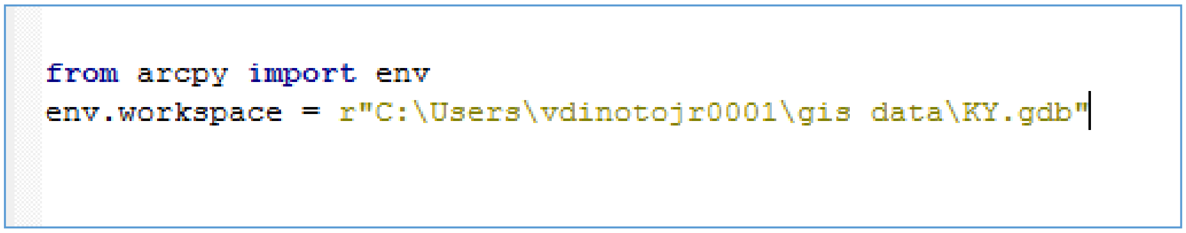
Figure 1: Buffer Script

1. To load the arcpy module, use the **import arcpy** command; this should always be one of the first functional lines of code.
2. A variable called ***featureInput*** has a pathway defined; this is the location of the data. Before the pathway is defined note that the letter 'r' is placed in the script. The 'r' is used to inform the software that standard pathway functions are being used, note it is placed outside of the quotation marks so it is not part of the string. The pathway shown in Figure 1 is to the author's data location and will be different on the learner's computer. The data is coming from a geodatabase.
3. A variable called ***featureOutput*** is used to define the saving pathway, to the same geodatabase.
4. A variable called ***radius*** is defined to be 25 miles. This is a hard coded distance and would require a modification of the code to change the buffer radius.
5. The **buffer** command is located in the analysis toolbox.
6. The three variables are used as the input to the buffer command. The variables could have been directly entered in the command line instead. The same script can be used for other operations just by making minor edits. If the pathways and radius had been directly entered, quotation marks would be required similar as in the defining of the variables.
7. Four additional parameters were not used in the Buffer command.
    a. line_side expects FULL, LEFT, RIGHT, OUTSIDE_ONLY (default value)
    b. line_end_type expects ROUND (default value) FLAT
    c. dissolve_option expects NONE (default value), ALL, LIST
    d. dissolve_field – lists of fields from the input to dissolve the output

e. since each of the parameters were after the first three parameters and they were all ignored they can be omitted in the coding, if one of them was used such as dissolve_option, then leading sets of commas would be required.

The multiple ring buffer is just a single ring buffer used with a loop function.

Environment (env)



```
from arcpy import env
env.workspace = r"C:\Users\vdinotojr0001\gis data\KY.gdb"
```

Figure 2: Environment Statement

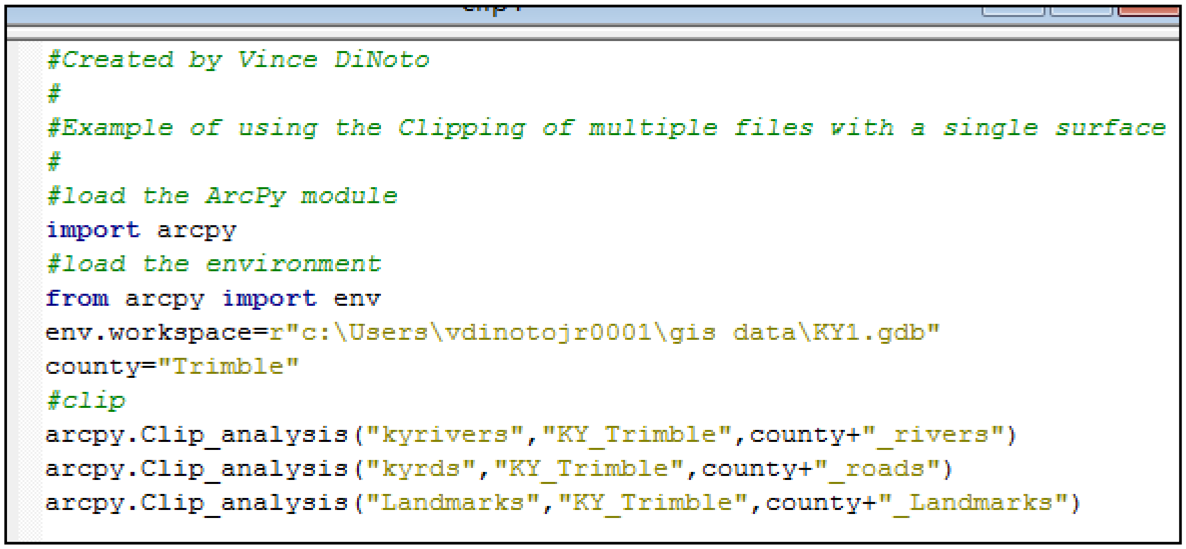
The ***env*** command is part of arcpy, the purpose of the environment statement is to set the environment and thus eliminate the need for individual pathways, used in the buffer example. If in the buffer example the ***env*** command was used, the complete pathway to the input and output would not have needed to be fully specified.

The environment command many times is the second line of functional script after the import of the arcpy module. The format can be seen in Figure 2. It is part of the arcpy module so the ***from*** statement is used in conjunction with the ***import*** statement. The command that is then used from the arcpy env module is workspace. The pathway defined in Figure 2 is to a geodatabase that contains multiple files that will be used. Note: the ‘r’ serves the same function as noted earlier. In the ***clip*** command, the function of the ***env*** statement will be explored.

Clip

The clip tool will use a polygon boundary (clipping file) to clip features that are contained within it. The results of the clipping will be the creation of a new file for the selected region. In this example, multiple shapefiles will be clipped and each will be stored with a new name. This will be accomplished through three individual commands and will not use a looping statement. The format for the clip command is:

arcpy.Clip_analysis(input file, clipping file, output file)



```
#Created by Vince DiNoto
#
#Example of using the Clipping of multiple files with a single surface
#
#load the ArcPy module
import arcpy
#load the environment
from arcpy import env
env.workspace=r"c:\Users\vdinotojr0001\gis data\KY1.gdb"
county="Trimble"
#clip
arcpy.Clip_analysis("kyrivers","KY_Trimble",county+"_rivers")
arcpy.Clip_analysis("kyrds","KY_Trimble",county+"_roads")
arcpy.Clip_analysis("Landmarks","KY_Trimble",county+"_Landmarks")
```

Figure 3: Clipping

1. Import arcpy
2. Import environment from arcpy
3. Defining the workspace which is a geodatabase.
4. Creating a variable called county and defining the string content as Trimble, this is our county of interest. If the script is for a different county then the name would be replaced.
5. All three clipping commands are the same, the first file to be clipped is the kyrivers file. The pathway is only the file name since the workspace had already been defined. The clipping boundary will be the KY_Trimble a polygon feature class file. The output file will be Trimble_rivers, which is based on the use of the variable and the information inside the quotations. The clipping file could have been defined as “KY_”+county.
6. The output files are stored in the same geodatabase as the input file.

Assignment 3.1

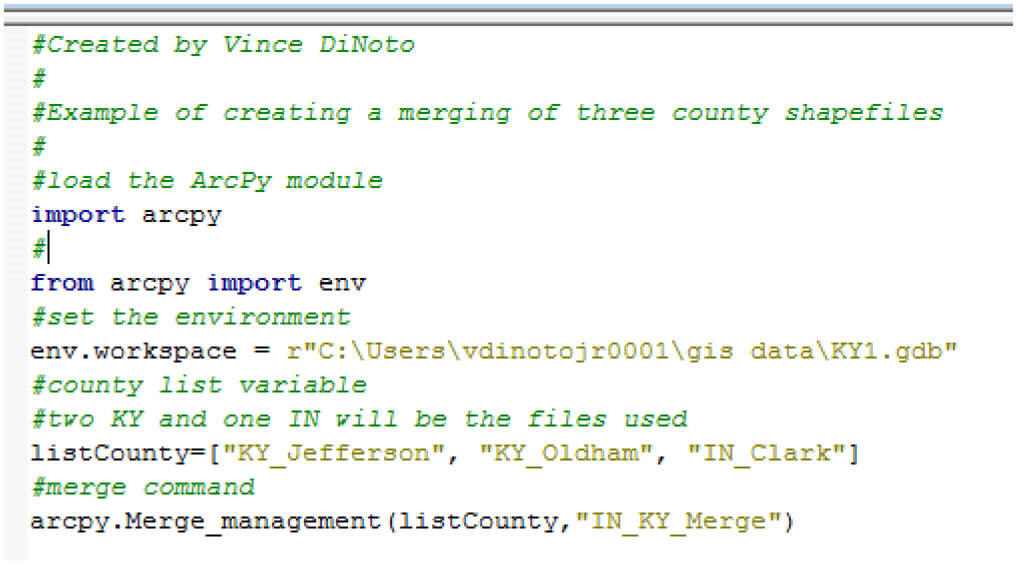
Using the college point file for the entire United State and a state county shapefile. Create a 30 mile radius buffer around each point and clip the created shapefile to your state. Provide a single script to your instructor to accomplish this operation. Provide comments to your instructor and the geodatabase, and a map with the resulting file. The buffer can extend outside of the individual state boundary, but extra points can be awarded if the buffer does not extend beyond the state boundaries.

Merge

Part I

Three contiguous counties will be merged together to create a single regional geography. The counties will have the same attribute information. This operation will be completed using a single Python script.

Comments have been placed in the script in Figure 4. Each line of the script will be explained that has functional code.



```
#Created by Vince DiNoto
#
#Example of creating a merging of three county shapefiles
#
#load the ArcPy module
import arcpy
#
from arcpy import env
#set the environment
env.workspace = r"C:\Users\vdinotojr0001\gis data\KY1.gdb"
#county list variable
#two KY and one IN will be the files used
listCounty=["KY_Jefferson", "KY_Oldham", "IN_Clark"]
#merge command
arcpy.Merge_management(listCounty,"IN_KY_Merge")
```

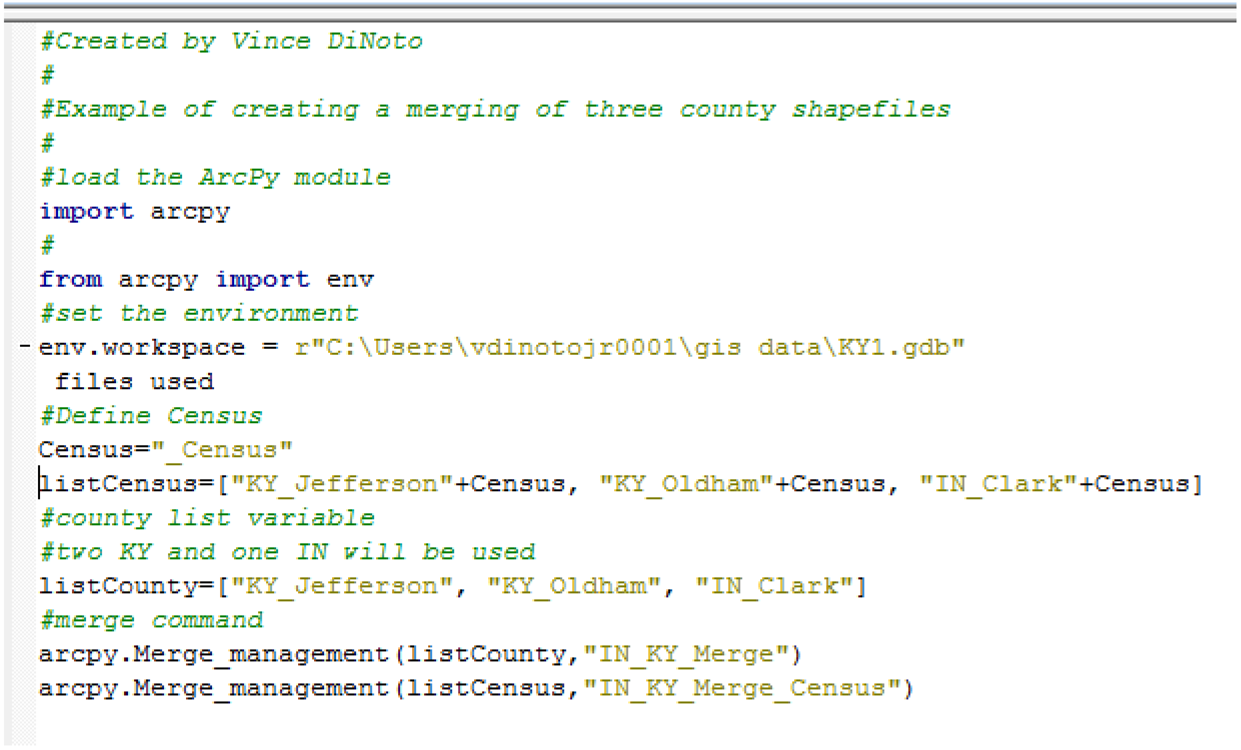
Figure 4: Merging Script

1. The first two lines of the functional code are used to load arcpy and the environment.
2. The location of the geodatabase is set using the workspace location. The ‘r’ allows the standard pathway convention to be used. The geodatabase is located in the author’s ‘gis data’ folder and called KY1.
3. A list variable is created that contains the names of the three counties that are being used. One of the counties is in Indiana and the other two are in Kentucky. Note how descriptive the naming convention of the content and the variables are. The naming convention will allow the data to be easily located.
4. The merge command is in the management toolbox. The first parameter is a list variable, since it is describing those counties which will be merged together. They could have been manually inputted into this function but that is not the recommended way. The second parameter is the name of the output file, which could have been a variable. All the data and the saving location is in the KY1 geodatabase, since the workspace is set.
5. To confirm the operation go to ArcMap or ArcCatalog and make sure the new file has been created and that when displayed or previewed it looks correct. Do not have any files open while running the Python script. Close ArcMap or ArcCatalog before

proceeding to the next step. If errors are generated, after fixing the issue, delete the created file, this will be necessary because no overwrite commands are used in the script.

Part II

In this section of the merge example, will explain how to complete redundant tasks more efficiently. The merge command demonstrated in Part I will be used, but an additional merge statement will be added so that multiple components can be combined together. Thus, both county boundaries and census tracts for the three counties will be merged together as well as creating variable lists. There are multiple ways that a script could be modified to increase



```
#Created by Vince DiNoto
#
#Example of creating a merging of three county shapefiles
#
#load the ArcPy module
import arcpy
#
from arcpy import env
#set the environment
env.workspace = r"C:\Users\vdinotojr0001\gis data\KY1.gdb"
 files used
#Define Census
Census="_Census"
listCensus=["KY_Jefferson"+Census, "KY_Oldham"+Census, "IN_Clark"+Census]
#county list variable
#two KY and one IN will be used
listCounty=["KY_Jefferson", "KY_Oldham", "IN_Clark"]
#merge command
arcpy.Merge_management(listCounty,"IN_KY_Merge")
arcpy.Merge_management(listCensus,"IN_KY_Merge_Census")
```

Figure 5: Merge Example 2

functionality.

1. The first three statements load the arcpy module, sets the environment and the pathway to the geodatabase.
2. A string variable named Census is defined, notice the underscore is used as part of the variable, this is because spaces are not valid characters and cannot be used.
3. A string list variable for the Census Tracts is defined; it uses a mathematical function to combine the county names and the value of the Census variable.
4. Another string list variable is defined which is the names of the county boundary files.

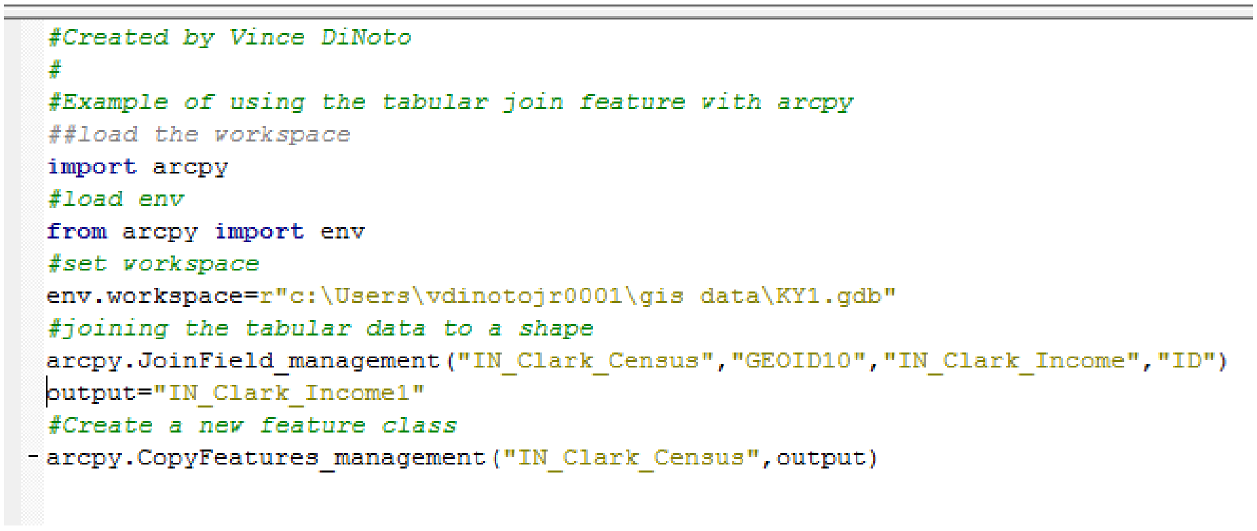
How might the ***listCounty*** and ***Census*** variables been used together to create the ***listCensus*** variable? Proper naming convention makes the process much easier to do and follow.

5. There are two merge statements, one to combine the county boundary files together and the other to combine the census tract boundary files together.

6. Confirm the results of both of these merge commands by using ArcMap and displaying the information. Close ArcMap when the review is completed.

Tabular Join

The Tabular Join process will combine spatial data with a table to create a file at the census tract level that contains information about a specific region and its demographics. For a tabular join to work a field in the spatial data and the table must have the same values and formatting type (i.e. either numbers or text). Formatting the data is critical to this process, in this example prior to executing the joining script; a new column was created to insure that both data sets were in the same data format. The process could have also been done in the Python script, but it was beyond the scope that has been taught up to this point. The tabular data was downloaded from the U.S. Census Bureau initially. There are numerous locations to obtain the spatial data, which also includes the U.S. Census Bureau.



```
#Created by Vince DiNoto
#
#Example of using the tabular join feature with arcpy
##load the workspace
import arcpy
#load env
from arcpy import env
#set workspace
env.workspace=r"c:\Users\vdinotojr0001\gis data\KY1.gdb"
#joining the tabular data to a shape
arcpy.JoinField_management("IN_Clark_Census","GEOID10","IN_Clark_Income","ID")
output="IN_Clark_Income1"
#Create a new feature class
arcpy.CopyFeatures_management("IN_Clark_Census",output)
```

Figure 6: Tabular Join Python Script

1. The first three lines of functional script loads arcpy, the environment and sets the workspace to the KY1 geodatabase.
2. The Join Field command has four variables; notice the command is located in the management toolbox.
    - The first parameter is the name of the spatial data file
    - The second parameter is the field from the spatial data file that will be used to accomplish the tabular join.
    - The third parameter is the name of the table to be joined to the spatial data file.
    - The fourth parameter is the field in the table, which contains the same data as that in the shapefile and is of the same format type.
3. At this point, there is not an output file location. Remember when doing a tabular join in Esri ArcMap that the resulting file must be exported to make the results permanent. If exporting is not completed, then the tabular join will be lost when the project is closed.

4. The **CopyFeatures** command creates a new feature class file, which will contain the spatial demographic information. The command is located in the management toolbox. A string variable named output was created which contains the name of the new output file, this step is not required and the name could have been added directly into the command line. The spatial data file is specified so the command knows where to get the information that will create the final output file (feature class file).
5. The output file name uses the geographical location of the data and the demographic type, which was income. The number one was added since there was already a file in the author's geodatabase with the same name.
6. Open the output file and explore in ArcMap.

Assignment 3.2

Use seven unique counties at the census tract level and two demographic spreadsheets (income and population) for the seven counties.

Create a regional geography of this data. Make sure appropriate naming conventions are used.

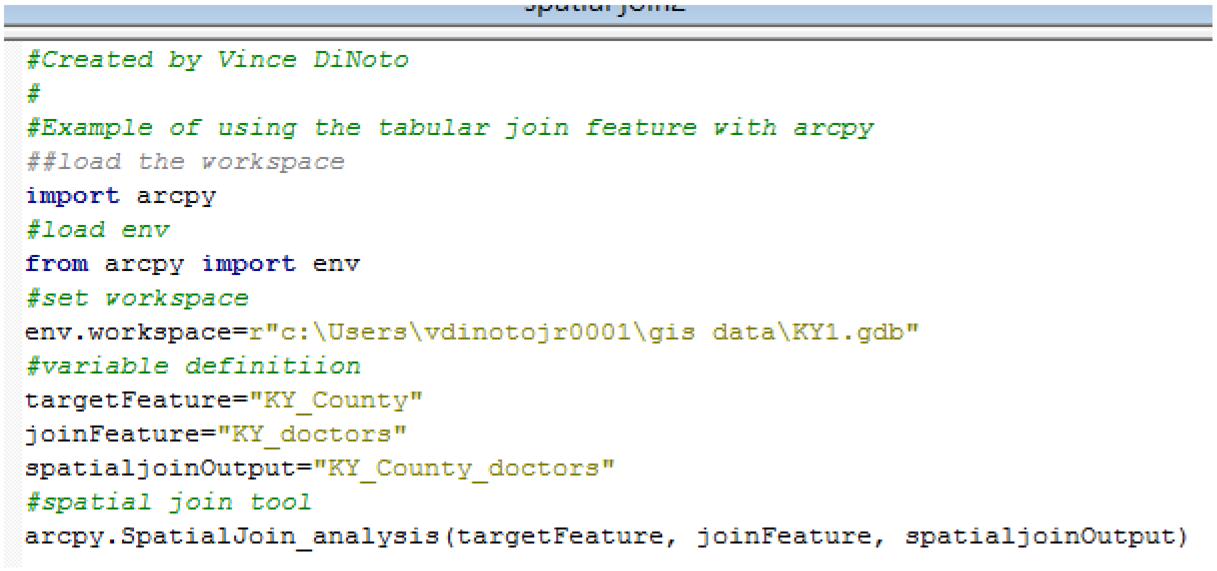
Use a single Python script to complete the process. Tasks that are required will include:

1. Merging the counties into a single feature class file.
2. Joining two different data sets with the merged feature class file.
3. Exporting the demographic feature class file into a new file. The script and the created feature class file should be provided to your instructor.

Spatial Join

A spatial join is the process that can be used to count the number of items contained within a boundary. There are other spatial join processes that can also be performed, but this is the most common use of this command. For example, if the location of students at a college is known, then a spatial join could be used to determine how many students live in each census tract in a county.

In this example, the number of doctors within the state of Kentucky will be determined per county.



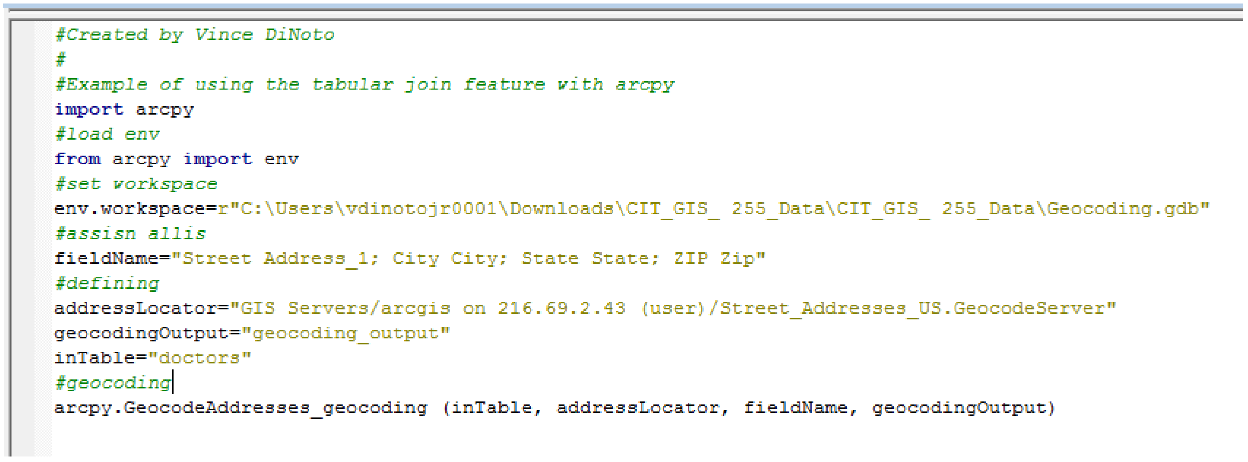
```
#Created by Vince DiNoto
#
#Example of using the tabular join feature with arcpy
##load the workspace
import arcpy
#load env
from arcpy import env
#set workspace
env.workspace=r"c:\Users\vdinotojr0001\gis data\KY1.gdb"
#variable definitiion
targetFeature="KY_County"
joinFeature="KY_doctors"
spatialjoinOutput="KY_County_doctors"
#spatial join tool
arcpy.SpatialJoin_analysis(targetFeature, joinFeature, spatialjoinOutput)
```

Figure 7: Spatial Join Python Script

1. Loading arcpy, setting the environment, and the workspace are set in the first three lines of functional code.
2. The next three lines of functional code defines three variables.
3. The **SpatialJoin** command is located in the analysis toolbox.
    - o The first parameter is the boundary file, for this example it will be a Kentucky County shapefile.
    - o The second parameter is what shapefile will be counted, for this example it is KY_doctors.
    - o The final parameter is the name of the output file, which will contain a new field for the count.
4. Each row of the Kentucky county file represented an individual polygon, which is generally a single county (there are a few exceptions in western Kentucky. Note the output file is created as part of the process.
5. Open in ArcMap and explore the results.

Geocoding

Geocoding is the process of taking street level addresses and creating a positional location. This is accomplished by comparing the addresses to a locator file, which is used to determine approximate positions. When this operation is completed, a new point feature class file will be created. Before geocoding ensure that, the table (Excel, CSV) has only one header role and no special characters in the names on the header row (such as spaces). Once the table has been modified it can be used in the geocoding process, typically the geocoder will be placed on a server class machine such as an Esri ArcGIS Server.



```
#Created by Vince DiNoto
#
#Example of using the tabular join feature with arcpy
import arcpy
#load env
from arcpy import env
#set workspace
env.workspace=r"C:\Users\vdinotojr0001\Downloads\CIT_GIS_ 255_Data\CIT_GIS_ 255_Data\Geocoding.gdb"
#assisn allis
fieldName="Street Address_1; City City; State State; ZIP Zip"
#defining
addressLocator="GIS Servers/arcgis on 216.69.2.43 (user)/Street_Addresses_US.GeocodeServer"
geocodingOutput="geocoding_output"
inTable="doctors"
#geocoding
arcpy.GeocodeAddresses_geocoding (inTable, addressLocator, fieldName, geocodingOutput)
```

Figure 8: Geocoding Python Script

1. The first three parameters in the functional script are comparable to those used in other scripts in this module.
2. The next lines define four different variables and some take a unique format.
   - The ***fieldName*** variable is used to compare the name in the address file (CSV) to that in the locator file. The locator file has a set name for street, city, state and zip code. The address file (CSV) has field names that may differ and should be reviewed in the attribute table. This variable must use the format shown to compare the two files. This comparison variable must be exact for the two files, the first variable in each group is from the locator
   - The ***addressLocator*** file is the pathway to the locator. The locator is on a geospatial server. This variable shows the name of the server and the location of the locator service. **Care must be taken that no typing errors are made**.
   - The third variable is the name of the output file; a more descriptive name would help improve the understanding of the geography.
   - The final defined variable is the name of the input file, which is a table.
3. The geocoding function is called **GeocodeAddresses** and is located in the geocoding toolbox.
4. Open the geocoding_output file in Esri ArcMap Desktop and the locations of the points should be displayed.

Assignment 3.3

Starting with a table containing address data and a single state county shapefile, create a Python script that will generate a single feature class file that will contain the number of data points in each county. This should be accomplished by using the geocoding and spatial join commands. Provide you instructor the Python script and an image of the results.

Case Study Topic 3

In this case study, the learner will demonstrate successful scripting of four geoprocessing commands (spatial join, geocoding, buffering and clipping) and display the final results. The learner will provide the instructor with the script, images of the results, and a written description of what the script is doing. The learner may also wish to provide the instructor a map package. The learner will start with a list of address and the study area boundaries. The result will be a map that counts the number of colleges within each county and a 35-mile buffer around each college. The buffer should not extend beyond the state boundary.