

# Integrating Machine Learning into Geographic Research

*Python Refresh and GeoPandas*

# Python Data Types

A few data types that we often use:

- Number (integer and float)
- String
- List
- Dictionary

# Numbers

- Integers
  - Whole number, i.e. no decimals
  - e.g. 34
- Floats
  - Decimal point
  - e.g. 34.8307

# String

- To create a *string variable*, assign a sequence of characters (text) to it

```
>>> mytext = "hotspot maps are cool."  
>>> print(mytext)  
hotspot maps are cool.
```

# Lists

- A list is a data structure in Python
- It is an ordered sequence of elements.
- Each element or value that is inside of a list is called an **item**.
- Lists are defined by having values between **square brackets** [ ].

```
my_list =  
[ 'Geography', 'Sociology', 'Environmental Science' ]
```

```
my_list = [ 516 , 170 , 373 ]
```

# Working with List

- Python lists have an index positioning system; index start with 0

```
>>> crimes = ["arson", "burglary", "robbery"]  
>>> crimes[1] 'burglary'
```

- There are many list methods

```
>>> crimes.append("homicide")  
>>> crimes.remove("arson")  
>>> crimes  
['burglary', 'robbery', 'homicide']
```

# Dictionaries

- **Dictionary** is Python's built-in data structure that links a **key** to a **value**
- These **key-value pairs** provide a useful way to store data in Python.
- Typically used to hold data that are related, such as the information contained in an ID or a user profile, dictionaries are constructed with curly braces on either side { }
- An example dictionary looks like:  

```
person_a = { 'name': 'Yingjie Hu', 'prof': True,  
'course_teaching': 503 }
```
- In addition to the curly braces, there are also colons (:) separating keys and values throughout the dictionary.

# Dictionaries

- Because dictionaries offer key-value pairs for storing data, they can be important elements in your Python program.
- We retrieve **values** by the **keys**.

```
>> print(person_a[ 'name' ] )  
Yingjie Hu
```

- The important thing to remember with dictionaries is the **key - value** relationship.



# Code reuse

- In most cases, we don't build a program from scratch but try to reuse existing code as much as possible
- We can reuse the code in functions, modules, packages
- You probably also heard a lot about the term “libraries”, and we will talk about it as well

# Python Function

- A Python function is generally a small piece of code (typically within 20 lines). There are many built-in python functions that complete various tasks
- You can also define your own function, if you expect that you are going to use certain steps of operations again and again in your project

# Python Function

E.g., you want to make a customized geographic visualization, and you want to apply such a visualization method again and again

```
def awesome_geo_vis(geo_data, width,  
height):  
    code line 1  
    code line 2  
    ...
```

# Python Function

- Defining a function
  - Function blocks begin with the keyword ***def*** followed by the function name and parentheses ( ).
  - Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
  - The code block within every function starts with a colon (:) and is indented.
  - The optional statement ***return*** [expression] exits a function and passes back an expression to the caller.

# Python Function

- A very simple example:

```
def print_msg(univ, major):  
    print("I'm from {0}, and my major is {1}".format(univ, major))  
  
print_msg("UB", "GIS")
```

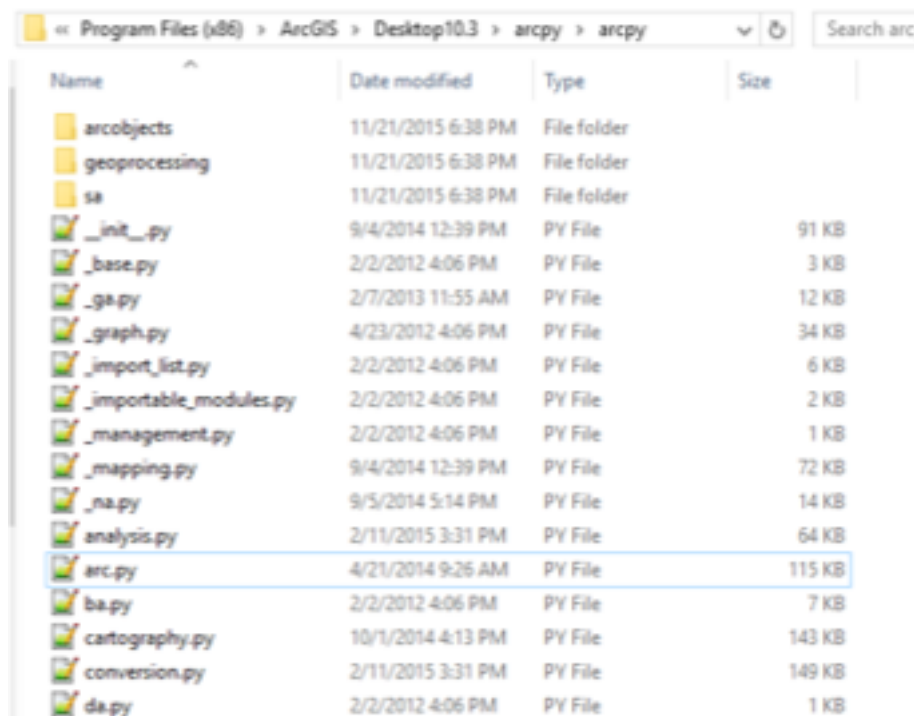
I'm from UB, and my major is GIS

# Python Module

- Python function allows you to reuse typically a few lines of code. What if you (or someone else) have created multiple functions in a Python file? That's a Python module!
- A Python module is simple a \*.py file that allows you to use the functions in that file
- You can import an entire module  
`>>> import time`
- The functions in the module can be called using a .  
`<module>.<function>`
- You can also import just one function of a module  
`from <module> import <function>`

# Python Package

- While a Python module is a file, a Python package is a folder that contains multiple modules (\*.py files) and a special file called “\_\_init\_\_.py”.



« Program Files (x86) > ArcGIS > Desktop10.3 > arcpy > arcpy

Name	Date modified	Type	Size
arcobjects	11/21/2015 6:38 PM	File folder	
geoprocessing	11/21/2015 6:38 PM	File folder	
sa	11/21/2015 6:38 PM	File folder	
__init__.py	9/4/2014 12:39 PM	PY File	91 KB
_base.py	2/2/2012 4:06 PM	PY File	3 KB
_ga.py	2/7/2013 11:55 AM	PY File	12 KB
_graph.py	4/23/2012 4:06 PM	PY File	34 KB
_import_list.py	2/2/2012 4:06 PM	PY File	6 KB
_importable_modules.py	2/2/2012 4:06 PM	PY File	2 KB
_management.py	2/2/2012 4:06 PM	PY File	1 KB
_mapping.py	9/4/2014 12:39 PM	PY File	72 KB
_na.py	9/5/2014 5:14 PM	PY File	14 KB
_analysis.py	2/11/2015 3:31 PM	PY File	64 KB
arc.py	4/21/2014 9:26 AM	PY File	115 KB
ba.py	2/2/2012 4:06 PM	PY File	7 KB
cartography.py	10/1/2014 4:13 PM	PY File	143 KB
conversion.py	2/11/2015 3:31 PM	PY File	149 KB
da.py	2/2/2012 4:06 PM	PY File	1 KB

# Importing a package

- To use the code in a package, you can import individual modules of the package

- You can use:

```
import <package>.<module>
```

```
import GISTools.classes  
import GISTools.functions  
import GISTools.analysis
```

- You can also do:

```
from <package> import <module>
```

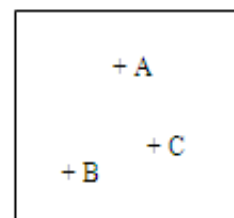


# Library

- While Python module and package are differentiated by how they are stored on a computer (files or folders), *library* is more of a concept used to refer to a group of scripts that serves a particular purpose
- E.g., scikit-learn is a Python machine learning library
- A library is more of an abstract term. In most cases, a library is a package but a very simple library can also be a module

# GeoPandas

- A free and open source Python library for working with vector data
- Work well for building an automatic workflow

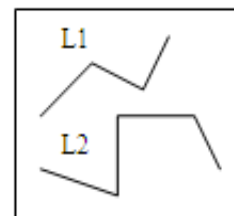


Points and their coordinates

A: (15, 7)

B: (9, 27)

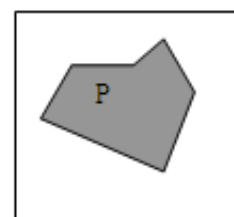
C: (19, 21)



Lines and their coordinates

L1: (3,16) (10,8) (17,14) (25,3)

L2: (3, 37) (13, 40) (13,16) (30,16) (33,35)



The polygon and its coordinates

P: (10,10) (20,10) (25,3) (30,13) (19, 37) (3,17)

# GeoPandas

- Before we explore **GeoPandas**, we need to understand **Pandas**.
- **Pandas** stands for Python Data Analysis Library; It provides “fast, flexible, and expressive data structures designed to make working with structured data both easy and intuitive.”
- Essentially, Pandas makes it easy for us to analyze data in python.
- The backbone of Pandas are **DataFrame** and **Series**

# Pandas

- A **DataFrame** is like a spreadsheet. It is a two dimensional data structure, with rows representing each data record and columns representing attributes
- A **Series** is like one column of a spreadsheet. It is a one dimensional data structure
- By putting our dataset into this DataFrame structure, we can work with the data easily.

More info: <https://pypi.org/project/pandas/>

# Pandas

## Creating a new dataframe using Pandas

Approach 1: First create a 2D array, and then assign column names

```
>> import pandas as pd  
>> data = [['Alex',10],['Bob',12],['Clarke',13]]  
>> df = pd.DataFrame(data,columns=['Name','Age'])  
>> print(df)
```

	Name	Age
0	Alex	10
1	Bob	12
2	Clarke	13

# Pandas

## Creating a new dataframe using Pandas

Approach 2: First create a dictionary, and then create a df

```
>> import pandas as pd

>> data = { 'Name': ['Alex', 'Bob',
                    'Clarke'], 'Age': [10, 12, 13] }

>> df = pd.DataFrame(data)

>> print(df)
```

	Name	Age
0	Alex	10
1	Bob	12
2	Clarke	13

# Pandas

## Creating a new dataframe using Pandas

Typically, you will not create a dataframe from scratch, but will read data from a file, e.g., data.csv

```
df = pd.read_csv('file_demo.csv',header=None)
df.columns = ["Course Number","Course Title"]
df
```

# Pandas

## Retrieving values from a dataframe:

### Getting a row

```
>> df.iloc[0]
```

### Getting a column

```
>> df['name']
```

### Getting the value of a cell

```
>> df['name'][1] #column first
```

```
>> df.iloc[0]['name'] # row first
```



# Pandas

## Some other commonly used dataframe functions

- Your data is often big (e.g., hundreds of thousands of rows), and you typically don't want to show all data at once

```
>> df.head()
```

- Your data is often big, you may want only a subset of your data that meet a certain condition

```
>> df[df['population'] > 100000]
```

- Your data is often big, you may want to know how large is your data

```
>> df.shape
```

# Pandas

- Pandas is powerful data science library. It has many many other functions that allow us to play with data easily
- The way we should learn how to use Pandas (and many other Python libraries) is NOT to first spend two months learning all of its functions, and then start to use it for your research project (you will probably forget a lot at that point)
- What we should do is to quickly learn the basics of a library that is sufficient for us to get started, and then use it immediately in our project!
- Sure, you will encounter something that you don't know. You should then search online to find the answer (Learn as you go!) For example, try to find out “how to sort a column in Python Pandas”

# GeoPandas

- **GeoPandas:** A library that is built on Pandas for geospatial data
  - It combines the capabilities of Pandas, Shapely, and a number of other packages, and makes them into a unified framework for working with vector data
  - GeoPandas enables you to easily do some common GIS operations in python that would otherwise require a professional GIS software
- Instead of **Series** and **DataFrames**, GeoPandas uses **GeoSeries** and **GeoDataFrames** as foundational data structures.
- As you can likely guess, these two data structures work the same as they did in pandas, but with additional geospatial functionality

# GeoPandas

- **GeoDataFrame** is like a dataframe, except it will have a column that represents the spatial footprints of geographic features
- For a **GeoDataFrame**, each row is a geographic feature, and each column contains an attribute of this geographic feature. Among these columns, one column contains the geometry representation (e.g., points, lines, and polygons), and this column is in the type of **GeoSeries**
- Other columns in a **GeoDataFrame** that contain regular numeric and string values are in the type of **Series**

# GeoPandas

- To use Geopandas, you will need to first install it in Google Colab
- Note that Google Colab provides a **standard** machine learning and data science environment, but it may not provide every library you need, especially specialized libraries like Geopandas
- To install it on Google Colab

```
>> !pip install geopandas
```

# GeoPandas

- Using GeoPandas to read a shapefile. The shapefile becomes a GeoDataFrame

```
>> import geopandas as gpd
```

```
>> world = gpd.read_file('world.shp')
```

```
>> world.head()
```

	pop_est	continent	name	iso_a3	dp_md_est	geometry
0	920938	Oceania	Fiji	FJI	8374.0	POLYGON ((180 -16.0671, 180 -16.55...
1	53950935	Africa	Tanzania	TZA	150600.0	POLYGON ((33.9037 -0.950...
2	603253	Africa	W. Sahara	ESH	906.5	POLYGON ((-8.665 27.656425...
...						

```
>> type(world)
```

```
>> type(world['geometry'])
```

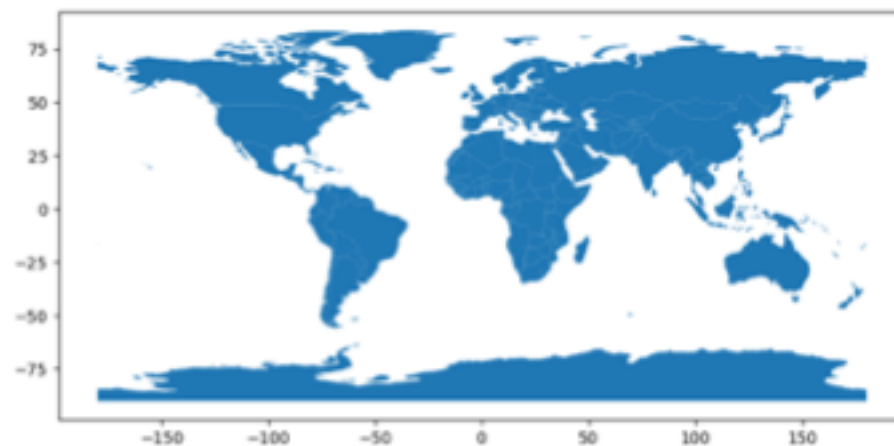
```
>> type(world['name'])
```

# GeoPandas

- Quickly visualize the shapefile

```
   pop_est  continent  name  iso_a3  dp_md_est  geometry
0   920938      Oceania  Fiji    FJI    8374.0  POLYGON ((180 -16.0671, 180 -16.55...
1  53950935        Africa  Tanzania  TZA    150600.0  POLYGON ((33.9037 -0.950...
2   603253        Africa   W. Sahara  ESH      906.5  POLYGON ((-8.665 27.656425...
...
```

```
>> world.plot()
```



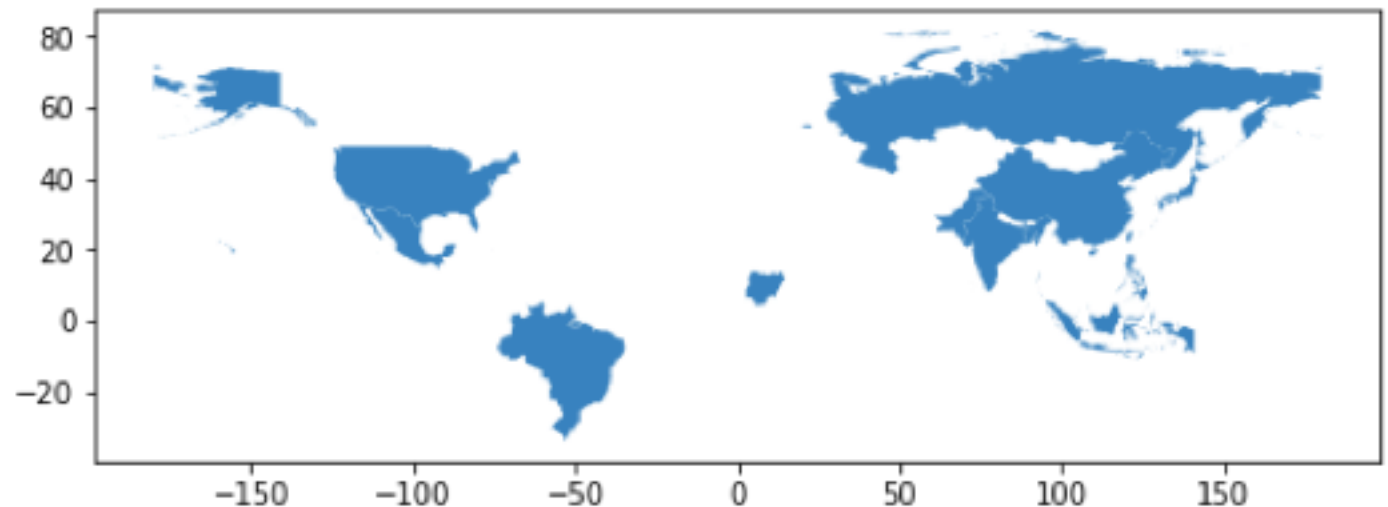
# GeoPandas

- Filter the geographic data by an attribute

	pop_est	continent	name	iso_a3	dp_md_est	geometry
0	920938	Oceania	Fiji	FJI	8374.0	POLYGON ((180 -16.0671, 180 -16.55...
1	53950935	Africa	Tanzania	TZA	150600.0	POLYGON ((33.9037 -0.950...
2	603253	Africa	W. Sahara	ESH	906.5	POLYGON ((-8.665 27.656425...
...						

```
>> major_countries = world[world['pop']>100000000]
```

```
>> major_countries.plot()
```





# GeoPandas

- Obtaining coordinate reference system (CRS) information

```
>> major_countries.crs
```

```
<Geographic 2D CRS: EPSG:4326>
```

```
Name: WGS 84 Axis Info [ellipsoidal]:
```

- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
- Area of Use: name: World
- bounds: (-180.0, -90.0, 180.0, 90.0) Datum: World Geodetic System 1984
- Ellipsoid: WGS 84 - Prime Meridian: Greenwich

## GeoPandas: Play with the included data of GeoPandas

- So far, we have loaded an external shapefile and explored it a bit
- Geopandas also provides several internal shapefile datasets that allow users to quickly learn the functions
- “Natural Earth” dataset: Natural Earth is a public domain map dataset available in Geopandas
- The datasets available on GeoPandas: `gpd.datasets.available`

# GeoPandas

- Play with Natural Earth data
- **Visualize two layers on the same map:**

```
gpd.datasets.available

ne_world =
gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))

ne_cities =
gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))

fig, ax = plt.subplots(figsize=(10,8))

ne_world.plot(ax=ax, color="white", edgecolor="black")

ne_cities.plot(ax=ax, color="red", markersize=5)
```

# Spatial Analysis

- We can use GeoPandas to do a variety of spatial analysis
- Finding the geometric centers of polygons



# Spatial Analysis

- Finding the geometric centers of polygons

```
nyc = gpd.read_file(gpd.datasets.get_path("nybb"))

fig, axes = plt.subplots(figsize=(8,8))
nyc.plot(ax=axes, color="white", edgecolor="black")

centroids = nyc.geometry.centroid
centroids.plot(ax=axes, marker="o", color='red',
markersize=20)
```

# Spatial Analysis

- Buffer

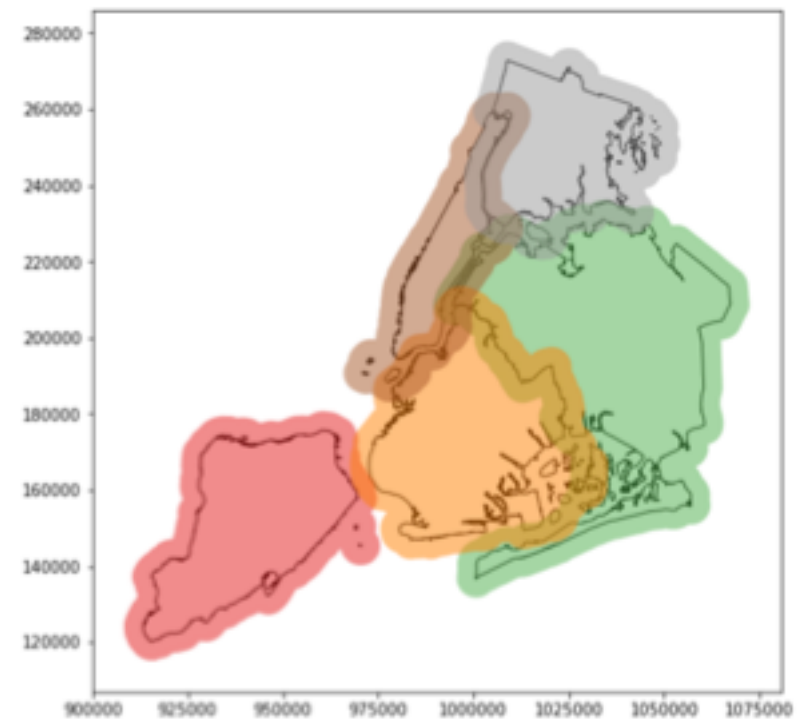
```
nyc_buffer = nyc.geometry.buffer(5000)
```

```
fig, ax = plt.subplots(figsize=(8,8))
```

```
nyc.plot(ax=ax, color="white", edgecolor="black")  
nyc_buffer.plot(ax=ax, cmap=plt.get_cmap("Set1"),  
alpha=0.5)
```

# Spatial Analysis

- Buffer



## Further information

- **GeoPandas**

- Documentation: <http://geopandas.org/>

- **Pandas**

- Documentation: <https://pypi.org/project/pandas/>



# Summary

- GeoPandas: a comprehensive Python library for processing geospatial data
- GeoSeries and GeoDataFrame
- Read and explore spatial data (you can use GeoPandas to write shapefile as well; see its document)
- Spatial analysis
  - Centroid
  - Buffer
  - Attribute join
  - Spatial join
  - Dissolve
  - Overlay