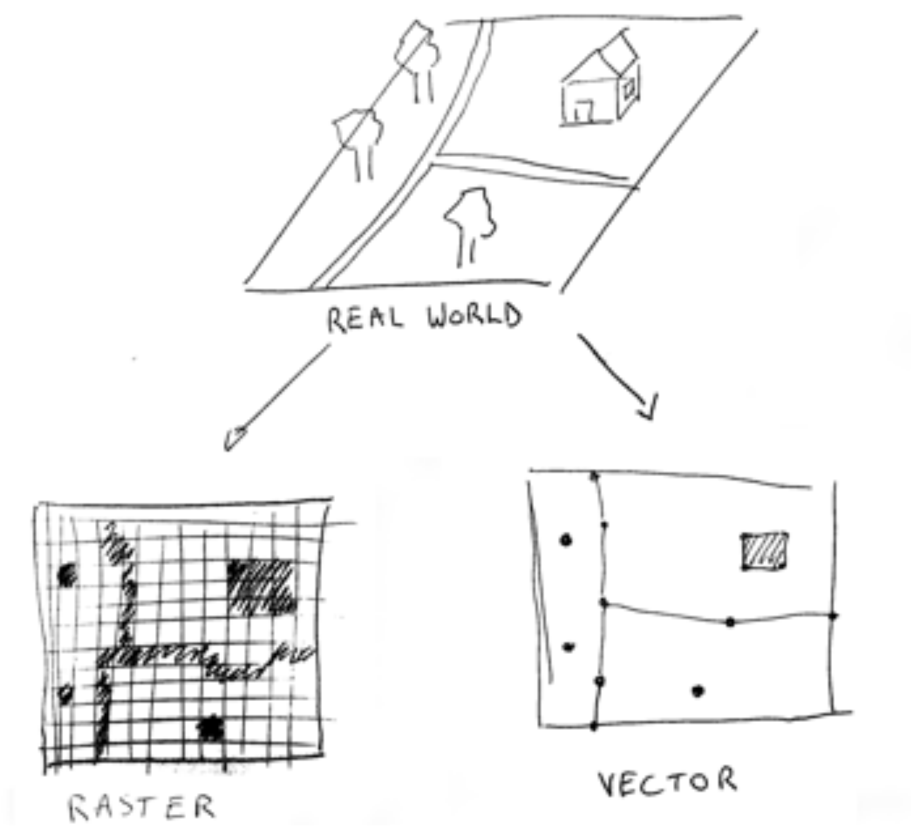
Working with Geospatial Data in Python
by Joris Van den Bossche

data for which a specific location is associated with each record
every observation has a location and can be put on a map
allows us to look at spatial relationships between the data

In GIS geographical information sciences there are two data models for how we record the world
first model is a 'raster'
raster which encodes the world as a continuous surface represented by a grid
example pixels of an image
real life examples - satelite images or altitude data
second model represents the world as a collection of discrete objects using points, lines, and polygons
second model is vector data




```
# Read the restaurants csv file
restaurants = pd.read_csv("paris_restaurants.csv")
```

```
# Import contextily
import contextily

# A figure of all restaurants with background
fig, ax = plt.subplots()
ax.plot(restaurants['x'], restaurants['y'], 'o', markersize=1)
contextily.add_basemap(ax)
plt.show()
```

Geopandas
all features of normal pandas DataFrames but with unique spatial capabilities
multiple points is not easily represented with a CSV file
geospatial data is better represented with special file formats such as GeoJSON, GeoPackage, Shapefiles, amongst others
GeoPanas is specifically designed to work with this data

```
import geopandas
countries = geopandas.read_file(‘countries.geojson’)
countries.head()
```

	name	continent	gdp	geometry
0	Afghanistan	Asia	64080.0	POLYGON ((61.21 35.65, 62.23 35...
1	Angola	Africa	189000.0	MULTIPOLYGON (((23.90 -11.72, 2...
2	Albania	Europe	33900.0	POLYGON ((21.02 40.84, 21.00 40...
4	Argentina	South America	879400.0	MULTIPOLYGON (((-66.96 -54.90, ...
5	Armenia	Asia	26300.0	POLYGON ((43.58 41.09, 44.97 41...

this is a GeoDataFrame
always has a ‘geometry’ column showing the geospatial features
countries.geometry > attribute pulls a GeoSeries of the geometry column and its associated points and polygons
can pull some nice data from this series
example - area
countries.geometry.area

Simple example

```
# Import GeoPandas
import geopandas

# Read the Paris districts dataset
districts = geopandas.read_file('paris_districts.gpkg')

# Inspect the first rows
```

```
print(districts.head())

# Make a quick visualization of the districts
districts.plot()
plt.show()

# Check what kind of object districts is
print(type(districts))

# Check the type of the geometry attribute
print(type(districts.geometry))

# Inspect the first rows of the geometry
print(districts.geometry.head())

# Inspect the area of the districts
print(districts.geometry.area)

# Read the restaurants csv file into a DataFrame
df = pd.read_csv("paris_restaurants.csv")

# Convert it to a GeoDataFrame
restaurants = geopandas.GeoDataFrame(df,
geometry=geopandas.points_from_xy(df.x, df.y))

# Inspect the first rows of the restaurants GeoDataFrame
print(restaurants.head())

# Make a plot of the restaurants
ax = restaurants.plot(markersize=1)
import contextily
contextily.add_basemap(ax)
plt.show()
```

Exploring and visualizing spatial data
just like pandas can filter data
boolean style
example
countries[‘continent’] == ‘Africa’
also called a boolean mask
countries_africa = countries[countries[‘continent’] == ‘Africa’]
countries_africa.plot()
output of plotting subset > map of countries of Africa

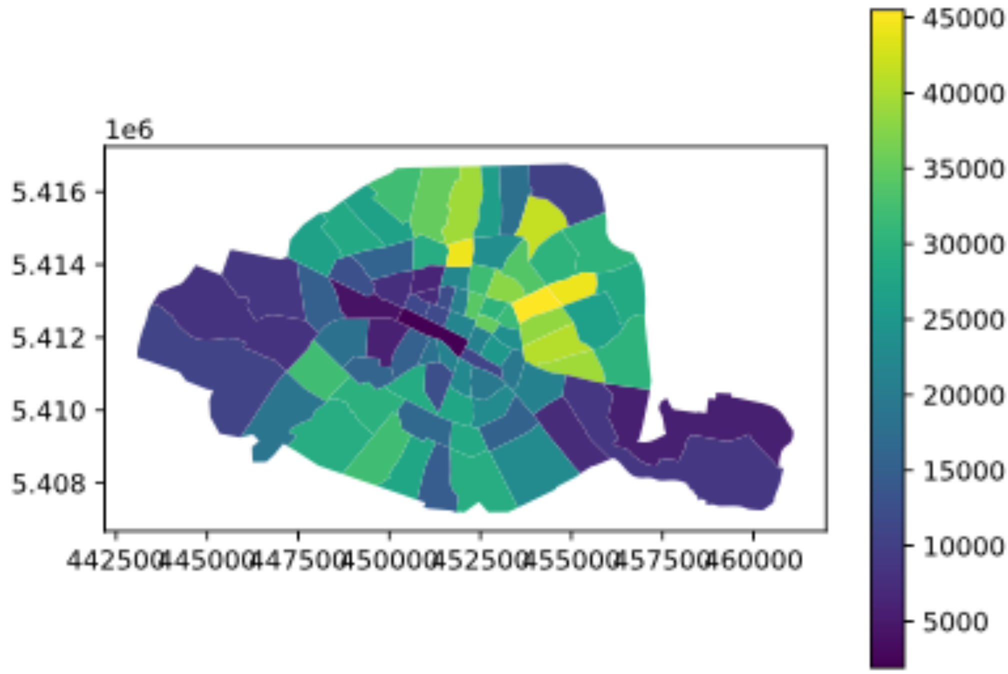
Adjusting the color - uniform color
```
countries.plot(color=‘red’) > output all countries in red
```

Adjusting the color - based on attribute colors
```
countries.plot(column=‘gdp_per_cap’)
```

Multi-layered plot
```
fig, ax = plt.subplots(figsize=(12,6))
countries.plot(ax=ax)
cities.plot(ax=ax, color=‘red’, markersize=10)
ax.set_axis_off() #this removes the box with ticks and labels around the figure
```

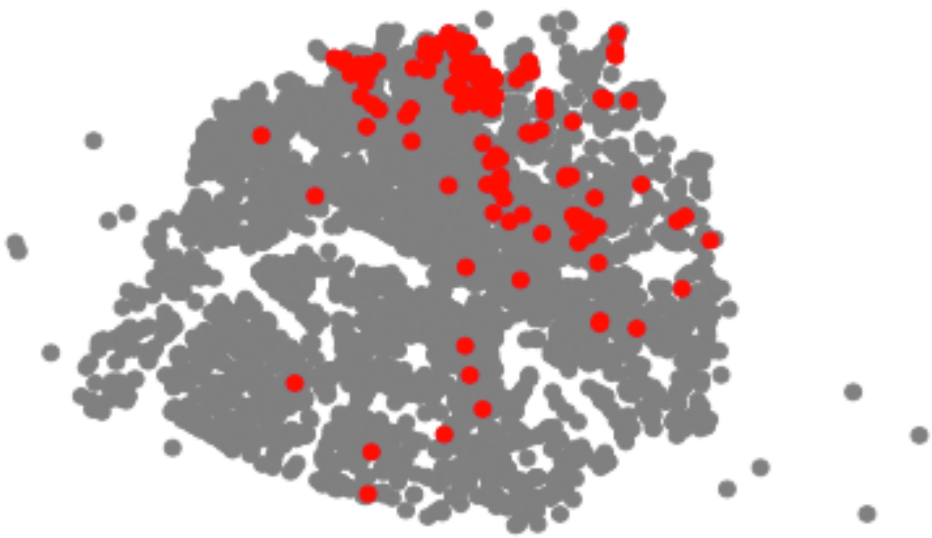
```
# Inspect the first rows of the districts dataset
print(districts.head())

# Inspect the area of the districts
print(districts.geometry.area)

# Add a population density column
districts['population_density'] = districts['population'] / districts.geometry.area *
10**6

# Make a plot of the districts colored by the population density
districts.plot(column='population_density', legend=True)
plt.show()
```

output>




```
# Load the restaurants dataset
restaurants = geopandas.read_file("paris_restaurants.geosjon")

# Take a subset of the African restaurants
african_restaurants = restaurants[restaurants['type'] == 'African restaurant']

# Make a multi-layered plot
fig, ax = plt.subplots(figsize=(10, 10))
restaurants.plot(ax=ax, color='grey')
african_restaurants.plot(ax=ax, color='red')
# Remove the box, ticks and labels
ax.set_axis_off()
plt.show()
```

output>



the power of geospatial data, shows us that a majority of the African restaurants are to the north

Shapely geometries and spatial relationships
scalar geometry values
cities = geopandas.read_file('ne_110m_populated_places.shp')
cities.head()

	name	geometry
0	Vatican City	POINT (12.45338654497177 41.90328217996012)
1	San Marino	POINT (12.44177015780014 43.936095834768)
2	Vaduz	POINT (9.516669472907267 47.13372377429357)
3	Lobamba	POINT (31.19999710971274 -26.46666746135247)
4	Luxembourg	POINT (6.130002806227083 49.61166037912108)

brussels = cities.loc[170, 'geometry']

Shapely package
provides point, linestring, and polygon objects
allows us to manipulate and analyze geometric objects

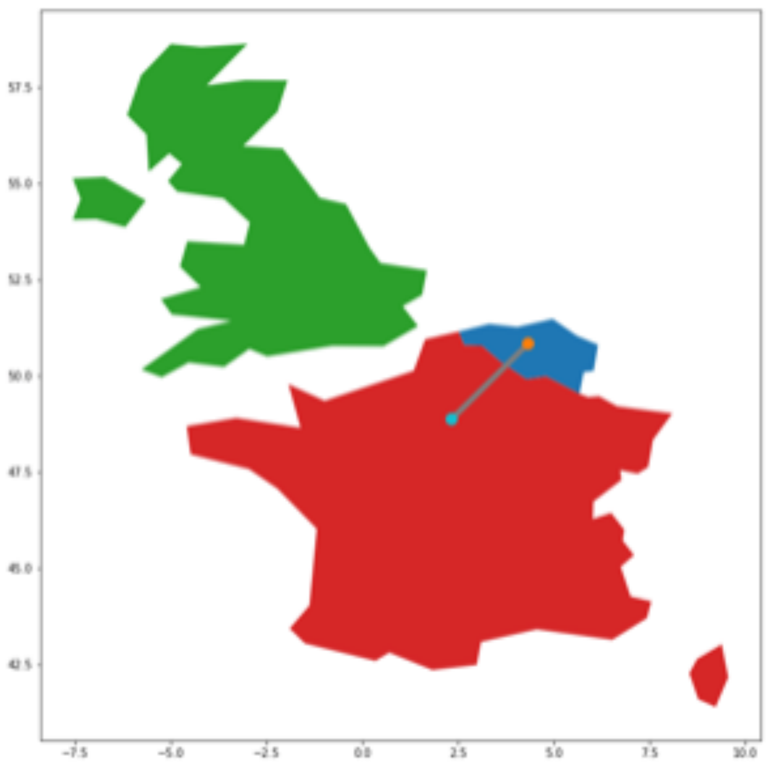
Accessing geometry objects form GeoDataFrame
belguim = countries.loc[countries['name'] == 'Belgium', 'geometry'].squeeze()

Manually create an object with Shapely
from shapely.geometry import Point
p = Point(1, 2)

objects have area attributes as well
belgium.area

distance between two geometries attribute
brussels.distance(paris)

Spatial relationships
shapely does not have a method to visualize multiple geometries
we have to put them into a GeoSeries prior to plotting
example
geopandas.GeoSeries([belgium, france, uk, paris, brussels, line]).plot()



shapely is smart and finds the relationship between point paris and brussels and places the line

contains() method
boolean response
allow you to see if something is within something like a point within a polygon
belgium.contains(brussels)
output > True

other useful methods
.within()
.touches()
.intersects()

example
```
# Construct a point object for the Eiffel Tower
eiffel_tower = Point(255422.6, 6250868.9)

# Accessing the Montparnasse geometry (Polygon) and restaurant
district_montparnasse = districts.loc[52, 'geometry']
resto = restaurants.loc[956, 'geometry']

# Is the Eiffel Tower located within the Montparnasse district?
print(eiffel_tower.within(district_montparnasse))

# Does the Montparnasse district contains the restaurant?
print(district_montparnasse.contains(resto))

# The distance between the Eiffel Tower and the restaurant?
print(eiffel_tower.distance(resto))
```

Element-wise spatial relationship methods
can use methods on entire GeoSeries
example
cities.within(france)
output > boolean response for each observation of the Series

Filtering by spatial relation
```
cities[cities.within(france)]
#pulls only the cities in France
```
output>

	name	geometry
10	Monaco	POINT (7.406913173465057 43.73964568785249)
13	Andorra	POINT (1.51648596050552 42.5000014435459)
235	Paris	POINT (2.33138946713035 48.86863878981461)

Another example of filtering by spatial relation
we add rivers dataset
which countries does the Amazon river flow?
amazon = rivers[rivers[‘name’] == ‘Amazonas’].geometry.squeeze() #squeeze()
useful to convert a one-row DF to a Series
mask = countries.intersects(amazon)
countries[mask]
output>

```
         name      continent                                          geometry
22     Brazil  South America  POLYGON ((-57.63 -30.22, -56.29 -28....
35   Colombia  South America  POLYGON ((-66.88 1.25, -67.07 1.13, ...
124      Peru  South America  POLYGON ((-69.53 -10.95, -68.67 -12....
```

example
# The distance from each restaurant to the Eiffel Tower
dist_eiffel = restaurants.distance(eiffel_tower)

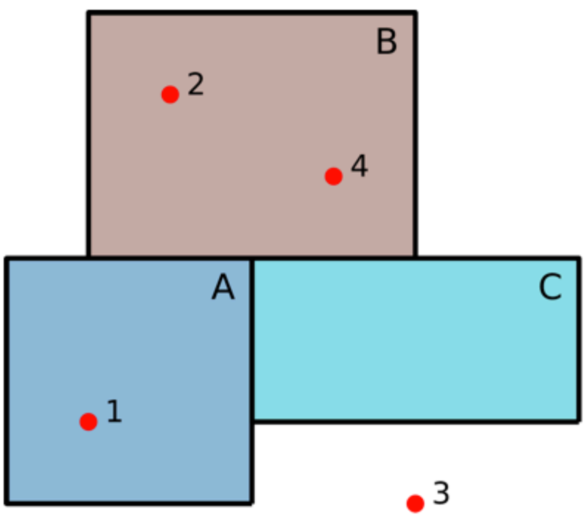
# The distance to the closest restaurant
print(dist_eiffel.min())

# Filter the restaurants for closer than 1 km
restaurants_eiffel = restaurants[dist_eiffel < 1000]

# Make a plot of the close-by restaurants
ax = restaurants_eiffel.plot()
geopandas.GeoSeries([eiffel_tower]).plot(ax=ax, color='red')
contextily.add_basemap(ax)
ax.set_axis_off()
plt.show()

ouput>

Spatial Join
transferring attributes from one layer to another based on their spatial relationship
What if we wanted to know which country each city was attached to?
We have the cities and countries dataset
left join example



points	geometry		polygon
1	POINT (2 2)		A
2	POINT (3 6)	←	B
3	POINT (6 1)		nan
4	POINT (5 5)		B

SPATIAL JOIN = *transferring attributes from one layer to another based on their spatial relationship*

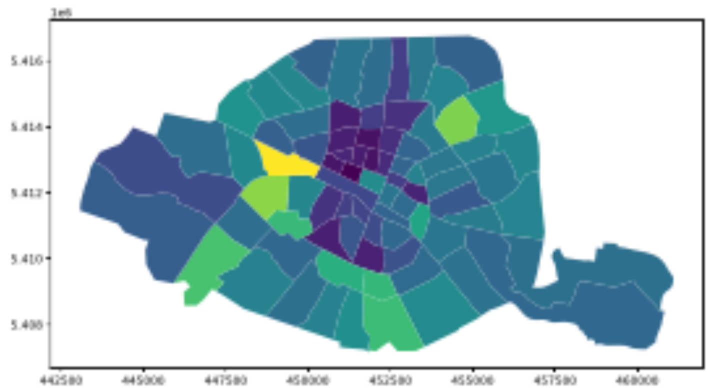
joined = geopandas.sjoin(cities, countries[['name', 'geometry']], op='within')
first argument > specify the geodataframe to which we want to add information
second argument > the geodataframe that contains the information we want to add
third argumet > 'op' specifies which spatial relationship we want to use to match both datasets
**our example, we are checking whether rows in the table on the left (cities) are

‘within’ those in the table on the right (countries)
and joining those where that is the case
*argument order is important here

example

```
# Read the trees and districts data
trees = geopandas.read_file("paris_trees.gpkg")
districts = geopandas.read_file("paris_districts_utm.geojson")

# Spatial join of the trees and districts datasets
joined = geopandas.sjoin(trees, districts, op='within')

# Calculate the number of trees in each district
trees_by_district = joined.groupby('district_name').size()

# Convert the series to a DataFrame and specify column name
trees_by_district = trees_by_district.to_frame(name='n_trees')

# Inspect the result
print(trees_by_district.head())

# Merge the 'districts' and 'trees_by_district' dataframes
districts_trees = pd.merge(districts, trees_by_district, on='district_name')

# Add a column with the tree density
districts_trees['n_trees_per_area'] = districts_trees['n_trees'] /
districts_trees.geometry.area

# Make of map of the districts colored by 'n_trees_per_area'
districts_trees.plot(column='n_trees_per_area')
plt.show()
```

output>



Choropleths
maps onto which an attribute, a non-spatial variable, is displayed
we encode its values by using a color scheme
*hard for human eye to see small differences in color in a continuous scale
to create effective choropleths use this classification scheme:

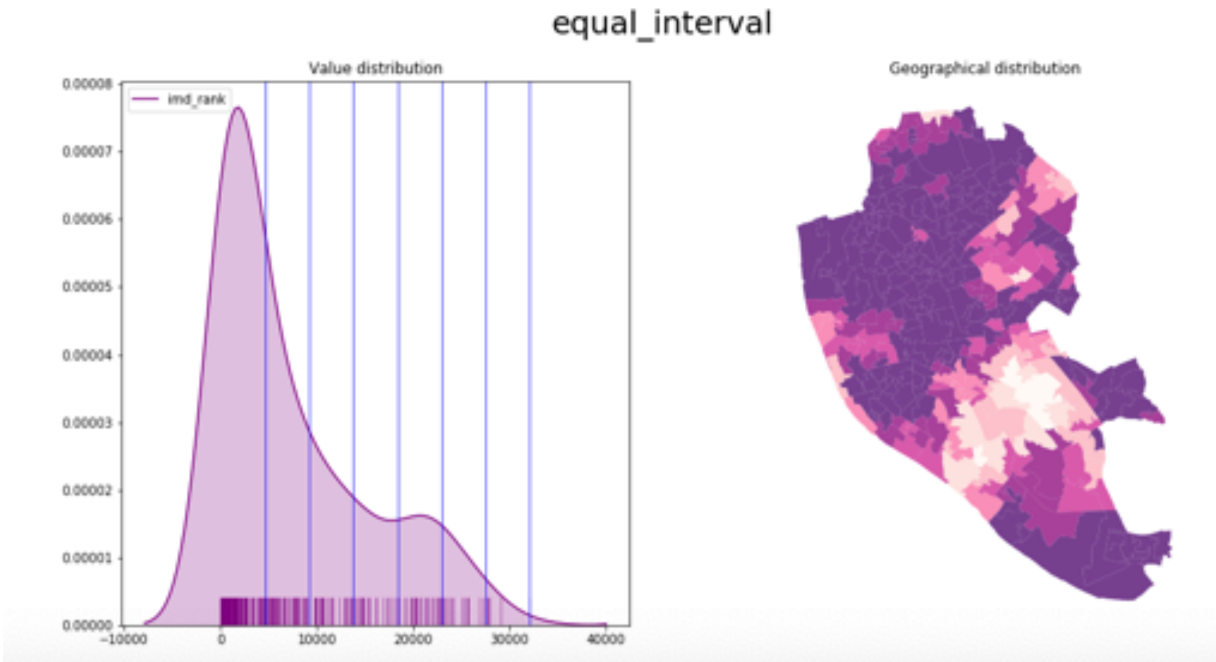
1. number of classes (k)
2. classification algorithm (scheme)
3. color palette (cmap)

example
locations.plot(column='variable', scheme='quantiles', k=7, cmap='viridis')

**necessary information loss
large number of values into a small number of colors
positive is that it make the map more interpretable
we do this by defining a number of classes (k)
need to find the sweet spot to tell enough information in a clear manner
studies show that number should be between 3 and 12

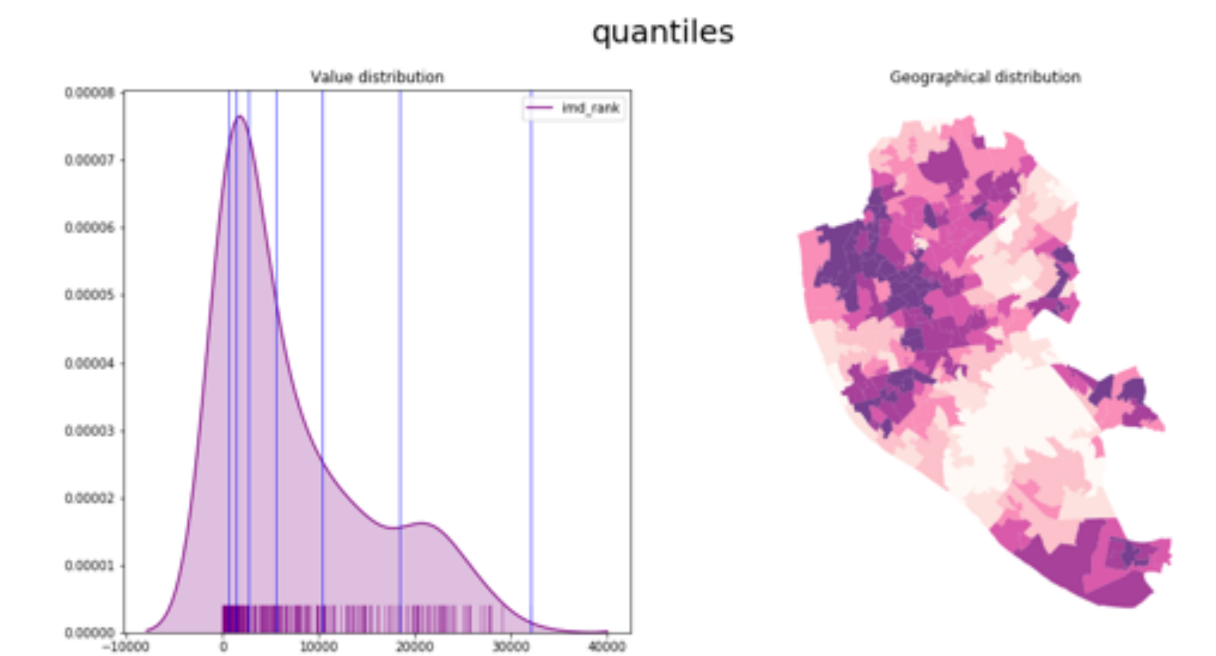
next - how do we allocate every value in our variable into one of the k groups?
two common approaches > equal intervals or quantiles

Equal Intervals
example
locations.plot(column='variable', scheme='equal_interval', k=7, cmap='Purples')
*splits range into equal segments and assigns a different color to each 'bin'



*can see here where a problem may occur if variable is unevenly distributed

Quantiles
classification ranks all the values and allocates the same proportion to each color bin
this balances the number of observations per color
example
locations.plot(column='variable', scheme='quantiles', k=7, cmap='Purples)



Color
scheme is important based on type of variable

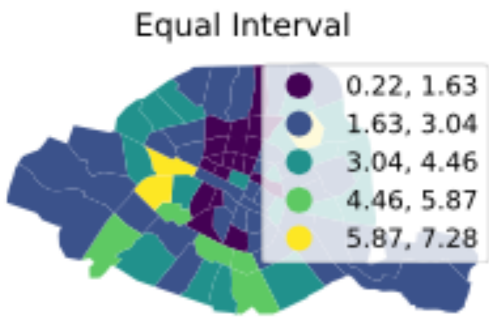
example
Set up figure and subplots
fig, axes = plt.subplots(nrows=2)

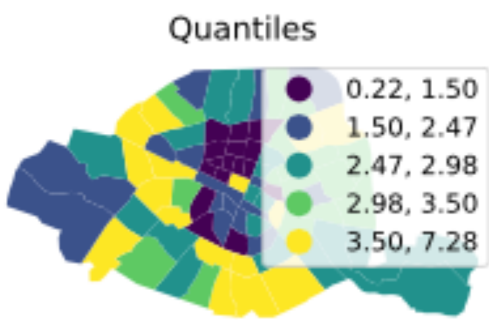
# Plot equal interval map
districts_trees.plot(column='n_trees_per_area', scheme='equal_interval', k=5, legend=True, ax=(axes[0]))
axes[0].set_title('Equal Interval')
axes[0].set_axis_off()

# Plot quantiles map
districts_trees.plot(column='n_trees_per_area', scheme='quantiles', k=5, legend=True, ax=(axes[1]))
axes[1].set_title('Quantiles')
axes[1].set_axis_off()

# Display maps
plt.show()

output>





Coordinate Reference System (CRS)
most common longitude and latitude
**delineated in Python as (lon, lat)
*good to know > longitude limited to range -180 to 180 and latitude limited to range -90 to 90
going from globe to flat map is called 'projection'
projected coordinates > lon, lat to x, y
projection inevitably causes errors
projection systems attempt to minimize this
examples > Mercator projection and Albers Equal Area projection
CRS is defined by a set of parameters
can be described in different ways
one representation is proj4 string
**most are identified by a number in the EPSG system
also called WGS84
stored in the crs attribute
variable.crs

Transforming to another CRS
can do this with the to_crs() method
example
gdf2 = gdf.to_crs({'proj': 'longlat', 'datum': 'WGS84', 'no_defs': True})
or shortcut by specifying the epsg number
gdf2 = gdf.to_crs(epsg=4326)

Why would you convert?
sources with a different CRS
working with multiple datasets
can convert one crs to the other DF like this:
df2 = df2.to_crs(df1.crs)
another reason
for mapping > distorition of shapes and distances (not all longitudes and latitudes are created equal)
another reason
for distance and area based calculations
geopandas and shapely assume all data is in a 2D cartesian plane
thus calculations will only be correct if you data is properly projected

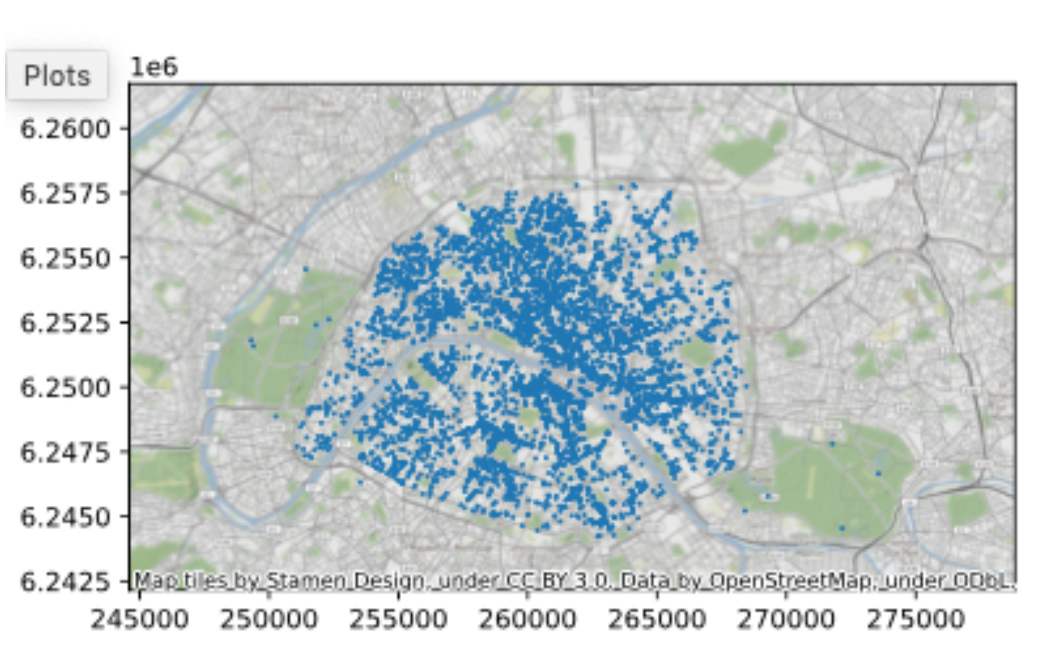
Choosing CRS
depends on field
hard to project whole earth but easier to get accuracy for smaller areas so good CRS's for specific areas
most countries have a standard CRS
two good resources to help pick the best CRS:
-spatialreference.org
-epsg.io

example

```
# Print the CRS information
print(districts.crs)

# Plot the districts dataset
districts.plot()
plt.show()

# Convert the districts to the RGF93 reference system
districts_RGF93 = districts.to_crs(epsg=2154)

# Plot the districts dataset again
districts_RGF93.plot()
plt.show()

# Construct a Point object for the Eiffel Tower
from shapely.geometry import Point
eiffel_tower = Point(2.2945, 48.8584)  # Longtitude, Latitude

# Put the point in a GeoSeries with the correct CRS
s_eiffel_tower = geopandas.GeoSeries([eiffel_tower], crs='EPSG:4326')
```

```
#Convert to other CRS
s_eiffel_tower_projected = s_eiffel_tower.to_crs('epsg:2154')

# Print the projected point
print(s_eiffel_tower_projected)

# Extract the single Point
eiffel_tower = s_eiffel_tower_projected[0]

# Ensure the restaurants use the same CRS
restaurants = restaurants.to_crs('epsg:2154')

# The distance from each restaurant to the Eiffel Tower
dist_eiffel = restaurants.distance(eiffel_tower)

# The distance to the closest restaurant
print(dist_eiffel.min())

# Convert to the Web Mercator projection
restaurants_webmercator = restaurants.to_crs('EPSG:3857')

# Plot the restaurants with a background map
ax = restaurants_webmercator.plot(markersize=1)
contextily.add_basemap(ax)
plt.show()
```

output>



Spatial operations
intersection
imagine two overlapping circles (a and b)
a.intersection(b)
output > new polygon made up of the intersecting area

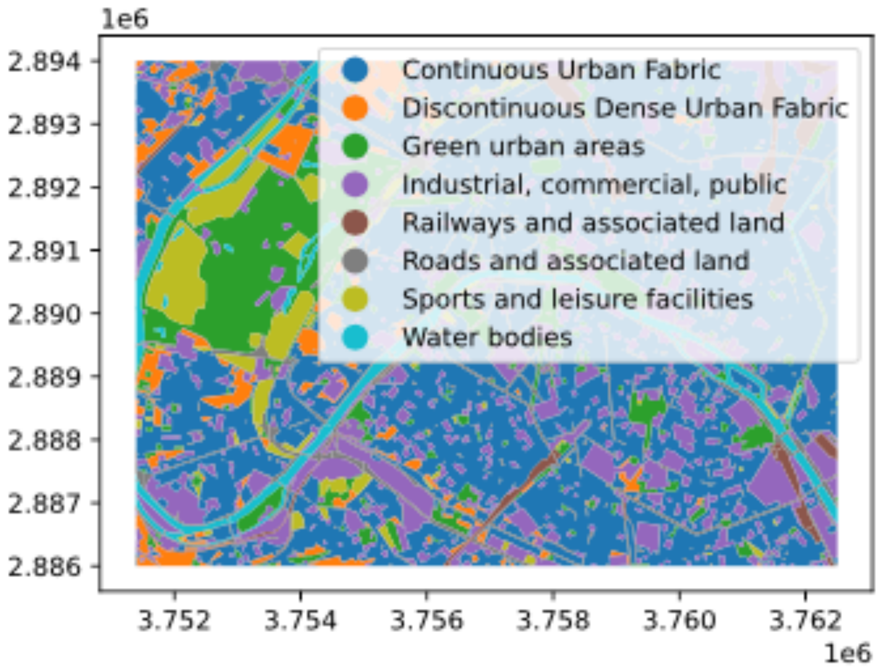
union
a.union(b)
ouput > a new polygon made up of circle a and b including the overlap

difference
a.difference(b)
ouput is the part of a circle that does not intersect with b

```
# Import the land use dataset
land_use = geopandas.read_file('paris_land_use.shp')
print(land_use.head())

# Make a plot of the land use with 'class' as the color
land_use.plot(column='class', legend=True, figsize=(15, 10))
plt.show()

# Add the area as a new column
land_use['area'] = land_use.geometry.area
```

# Calculate the total area for each land use class
total_area = land_use.groupby('class')['area'].sum() / 1000**2
print(total_area)

output>



Overlaying two datasets
example overlay two datasets and only take area where there is overlap
datasets are countries and geologic_regions
geopandas.overlay(countries, geologic_regions, how='intersection')
*difference from intersection method is that this function can handle more than one polygon
secondly overlay() keeps the attribute information of both datasets

example
# Print the first rows of the overlay result
print(combined.head())

Add the area as a column
combined['area'] = combined.area

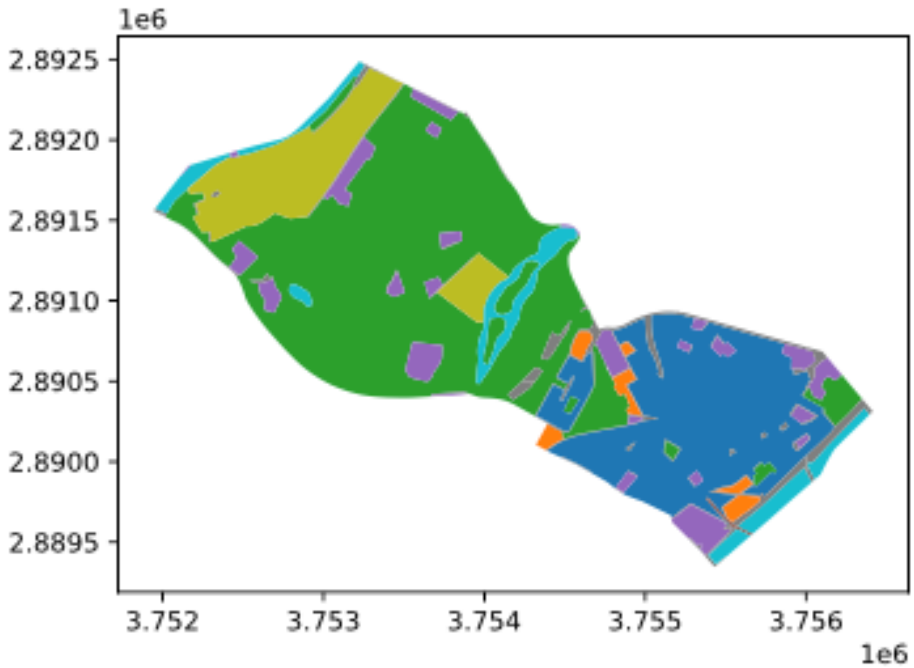
# Take a subset for the Muette district
land_use_muette = combined[combined['district_name'] == 'Muette']

Visualize the land use of the Muette district

land_use_muette.plot(column='class')
plt.show()

# Calculate the total area for each land use class
print(land_use_muette.groupby('class')['area'].sum() / 1000**2)

output>



Geospatial file formats
Most popular is ESRI Shape file
**key factor to remember is that the file consists of multiple files
(.shp, .dbf, .shx, .prj, and more)
make sure to copy all the files
other common ones are GeoJSON and GeoPackage
GeoPandas also has the capability to read data directly from popular spatial databases such as PostGIS

GeoPandas also has the capability to write such files
geodataframe.to_file('mydata.shp', driver='ESRI Shapefile')
*driver argument determines type of file
examples 'GeoJSON' and 'GPKG'

Unary union
convert a series of geometries to a single union geometry
such as taking all the polygons of African countries and turnin them into a defined

polygon that represents the continent
example
GeoSeries.uniary_union

Buffer operation
creates a buffer around a geometry
can place a buffer around any geometry
the buffer is a new polygon
on a GeoSeries the operation will create a buffer for each geometry element-wise
can specify the size with distance argument
example
point.buffer(distance)

example
```
# goma is a Point
print(type(goma))

# Create a buffer of 50km around Goma
goma_buffer = goma.buffer(50000)

# The buffer is a polygon
print(type(goma_buffer))

# Check how many sites are located within the buffer
mask = mining_sites.within(goma_buffer)
print(mask.sum())

# Calculate the area of national park within the buffer
# Calculate the intersection between the national park and the buffer
intersection = national_parks.intersection(goma_buffer)

# Calculate the total area of the intersection
total_intersection_area = intersection.area.sum()

# Print the area in square kilometers
print(total_intersection_area / (1000**2))
```

Nice example
```
# Extract the single polygon for the Kahuzi-Biega National park
kahuzi = national_parks[national_parks['Name'] == "Kahuzi-Biega National
park"].geometry.squeeze()

# Take a subset of the mining sites located within Kahuzi
```

```
sites_kahuzi = mining_sites[mining_sites.geometry.within(kahuzi)]
print(sites_kahuzi)

# Determine in which national park a mining site is located
sites_within_park = geopandas.sjoin(mining_sites, national_parks, op='within',
how='inner')
print(sites_within_park.head())

# The number of mining sites in each national park
print(sites_within_park['Name'].value_counts())
```

Applying custom spatial operations
example - find total river length within 50km of each city?

```
#can be done as so for a single point
area = cairo.buffer(50000)
rivers_within_area = rivers.intersection(area)
print(rivers_within_area.length.sum() / 1000)
#how to apply this to a Series
def river_length(geom, rivers):
        area = geom.buffer(50000)
        rivers_within_area = rivers.intersection(area)
        return rivers_within_area.length.sum() / 1000
#can call this created function for single point
river_length(cairo, rivers=rivers)
#can call on all citis
cities.geometry.apply(river_length, rivers=rivers)
```

Example

```
# Get the geometry of the first row
single_mine = mining_sites.geometry[0]

# Calculate the distance from each national park to this mine
dist = national_parks.distance(single_mine)

# The index of the minimal distance
idx = dist.idxmin()

# Access the name of the corresponding national park
closest_park = national_parks.loc[idx, 'Name']
print(closest_park)

# Define a function that returns the closest national park
def closest_national_park(geom, national_parks):
```

```
    dist = national_parks.distance(geom)
    idx = dist.idxmin()
    closest_park = national_parks.loc[idx, 'Name']
    return closest_park

# Call the function on single_mine
print(closest_national_park(single_mine, national_parks))

# Apply the function to all mining sites
mining_sites['closest_park'] = mining_sites.geometry.apply(closest_national_park, national_parks=national_parks)
print(mining_sites.head())
```

Raster data
represents the world as a grid, where each pixel in that grid takes a continuous or discrete value
example
continious > active rain path
discrete > land types
*raster can have more than one ‘band’
can think of a band as a type of layer
meaning each pixel can have multiple values
ie a blue value, green value, red value

Rasterio package
import rasterio
pythonic interface to the GDAL library
#open a raster file
src = rasterio.open(‘DEM_world.tif’)
gives back metadata
so we can see how many bands
src.count
or how many pixels
src.width, src.height

To read and store actual raster data
array = src.read()
comes in the form of a numpy array

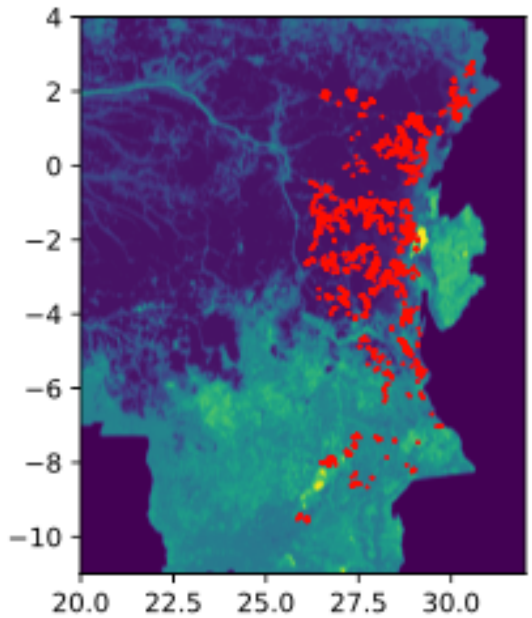
Plotting a raster dataset
import rasterio.plot
raseterio.plo.show(src, cmap=‘terrain’)

rasterstats package
summary statistics
to extract pixel value for points
rasterstats.point_query(geometries, ‘path/to/raster’, interpolation=‘nearest’ | ‘bilinear’)
to extract pixel values for polygons
rasterstats.zonal_stats(geometries, ‘path/to/raster’, stats=[‘min’, ‘mean’, ‘max’])
real example
result = rasterstats.zonal_stats(countries.geometry, ‘DEM_gworld.tif’, stats=[‘mean’])
#need to then assign results to a new column of the DataFrame
countries[‘mean_elevation’] = pd.DataFrame(result)
countries.sort_values(‘mean_elevation’, ascending=False).head()

example

```
# Import the rasterio package
import rasterio

# Open the raster dataset
src = rasterio.open("central_africa_vegetation_map_foraf.tif")

# Import the plotting functionality of rasterio
import rasterio.plot

# Plot the raster layer with the mining sites
ax = rasterio.plot.show(src)
mining_sites.plot(ax=ax, color='red', markersize=1)
plt.show()
```

output>




```
# Import the rasterstats package
import rasterstats

# Extract the nearest value in the raster for all mining sites
vegetation_raster = "central_africa_vegetation_map_foraf.tif"
mining_sites['vegetation'] = rasterstats.point_query(mining_sites.geometry,
vegetation_raster, interpolate='nearest')
print(mining_sites.head())

# Replace numeric vegation types codes with description
mining_sites['vegetation'] = mining_sites['vegetation'].replace(vegetation_types)

# Make a plot indicating the vegetation type
mining_sites.plot(column='vegetation', legend=True)
plt.show()
```