PROCESSING VECTOR DATA

Master in Environmental Management of Mountains Areas

ADVANCED GEOMATICS

Andrea Antonello - Free University of Bolzano

March - June 2024

READING AND WRITING GIS STUFF

QGIS, through the GDAL library supplies drivers to read and write the most common GIS vector formats. The preferred format between geospatial professionals is the geopackage.

Main reasons for not using SHP (http://switchfromshapefile.org):

- no definition of the coordinate reference system used
- multifile
- 10 characters attribute names
- 255 attribute fields
- unknown character set
- 2GB size limit

NATURAL EARTH

Natural Earth is a public domain map dataset available at 1:10m, 1:50m and 1:110 million scales and is is supported by NACIS (North American Cartographic Information Society) and many volunteers. It is free for use in any type of project.

For our tests we need the layers:

- cultural: ne_50m_admin_0_countries
- cultural: ne_50m_populated_places
- physical: ne_10m_rivers_lake_centerlines_scale_rank

You can download the complete dataset in a geopackages from here:

https://naciscdn.org/naturalearth/packages/natural_earth_vector.gpkg

LOAD THE LIFESAVER MAP

```
osm = HMap.get_osm_layer() 1
HMap.add_layer(osm) 2
```

1. get the OSM layer
2. add the layer to the current map

REMOVE MAPS

When you run the code several times, you will end up with several layers duplicated in the map. Here is how to remove them:

```
HMap.remove_layers_by_name(["OpenStreetMap", "other map"])
```

The name used is the one visible in the layer view.

READING AN EXISTING GPKG LAYER

```
geopackagePath = folder + "natural_earth_vector.gpkg"
countriesName = "ne_50m_admin_0_countries"
countriesLayer = HVectorLayer.open(geopackagePath, countriesName) 1

print("Schema (first 4 fields):")
counter = 0
for name, type in countriesLayer.fields.items():
    counter = counter + 1
    if counter < 5:
        print("\t", name, "of type", type)
```

1. to access the layer, we use the **open** method of the **HVectorLayer** class, passing the database path and the layer name

The layer object has various attributes that help investigating the layer properties:

```
crs = countriesLayer.prjcode
print("Projection: ", crs)
print("Spatial extent: ", countriesLayer.bbox())
print("Feature count: ", countriesLayer.size())
```

Print out the attributes of the "Italy" feature:

```
print("Attributes for Italy:")
countriesFeatures = countriesLayer.features() 1
nameIndex = countriesLayer.field_index("NAME") 2
fieldNames = countriesLayer.field_names 3
for feature in countriesFeatures:
    if feature.attributes[nameIndex] == 'Italy': 4
        geom = feature.geometry # get the geometry
        print("GEOM:", geom.asWkt()[:50] + "...")
        count = 0
        for index, attribute in enumerate(feature.attributes):
            print(fieldNames[index] + ":", attribute)
            count += 1
            if count > 5:
                print("...")
                break
```

1. get the features iterator
2. get the index of the field "NAME"
3. get all fields names
4. attributes are accesse via their index

FILTERS

FILTERS USING EXPRESSIONS

QGIS supplies a long list of functions that can be used to create expressions, that can be both used to create new fields or to filter data. A complete list of functions is available here.

```
expression = "NAME like 'I%' and POP_EST > 30000000"    1
features = countriesLayer.features(expression)    2
count = 0
for feature in features:
    print(feature.attributes[nameIndex])
    count+=1
print("Feature count with filter: ", count)
```

1. filter features that have the name starting with 'I' and a population major then 30.000.000
2. apply the filter by passing it to the **features** method.

BBOX FILTER

A BBOX filter can be create using a QgsRectangle. This can be used for example to find cities within a "radius" of 200 km (~ 2 degrees) from Trento:

```
lon = 11.119982
lat = 46.080428
point = HPoint(lon, lat)
buffer = point.buffer(2)
citiesLayer = HVectorLayer.open(geopackagePath, citiesName)
HMap.add_layer(citiesLayer)

citiyNameIndex = citiesLayer.field_index("NAME")
print("\napply bbox filter on features")
aoi = buffer.bbox()
count = 0
for feature in citiesLayer.features(bbox=aoi): (1)
    print(feature.attributes[citiyNameIndex])
    count += 1
print("Count =", count)
```

(1) filter by bbox passing the bbox object to the **features** method

EXACT GEOMETRY FILTER

In pyQGIS there is no way to create an exact geometry filter. Therefore, to have an exact filter, two steps are necessary. Apply the bbox filter to the datasource (important for remote databases), and then check for intersection on the resulting features. The extension do that transparently for you:

```
print("\napply geometry filter on features")
coutn = 0
for feature in citiesLayer.features(geometryfilter=buffer): (1)
    print(feature.attributes[citiyNameIndex])
    count += 1
print("Count =", count)
```

(1) filter by geometry passing the geometry object to the **features** method

CREATE AN IN-MEMORY VECTOR LAYER

One good way to start when creating new data, is the creation of a memory layer. This won't write any data on disk, until you tell it to do so.

The first thing to define when creating a new dataset, is its schema, i.e. its fields and datatypes (possible types are string, integer, double).

This is naturally done using a dictionary:

```
fields = {
    "id": "Integer",
    "name": "String",
}
```

The in-memory layer is then created with the pre-defined schema and, apart of a name, the needed geometry type and crs:

```
just2citiesLayer = HVectorLayer.new("test", "Point", "EPSG:4326", fields)
```

features can be created adding the geometry and the attributes, which need to be in the creation order of the schema:

```
just2citiesLayer.add_feature(HPoint(-122.42, 37.78), [1, "San Francisco"])
just2citiesLayer.add_feature(HPoint(-73.98, 40.47), [2, "New York"])
```

CREATE A NEW GEOPACKAGE

Once we have a memory layer, we can dump it to any supported GIS format:

```
1 path = folder + "test.gpkg"
2 error = just2citiesLayer.dump_to_gpkg(path, overwrite=True) 1
3 if(error):
4     print(error) 2
5
6 HMap.add_layer(just2citiesLayer)
```

1. dump the layer to a geopackage
2. print out any error, if it happened

ONE WITH MORE ATTRIBUTES

```
fields = {
    "name": "String",
    "population": "Integer",
    "lat": "double",
    "lon": "double"
}
oneCityMoreAttributes = HVectorLayer.new("test2", "Point", "EPSG:4326", fields)
oneCityMoreAttributes.add_feature(HPoint(-73.98, 40.47), \
                                  ["New York", 19040000, 40.47, -73.98])
path = folder + "test2.gpkg"
error = oneCityMoreAttributes.dump_to_gpkg(path, overwrite=True)
if(error):
    print(error)
```

STYLE

SEE IT WITH STYLE

To apply a filter directly to a layer, the **subset_filter** method can be used. Let's load the cities layer, visualizing only cities fo Italy.

```
citiesLayer.subset_filter("SOV0NAME='Italy'") 1
```

1 filter out Italian cities

POINTS SHAPE

To create a point style, a Marker can be used, as well as fill and stroke objects:

```
pointStyle = HMarker("square", 4, 45) + \ 1
             HFill("255,0,0,128") + \  2
             HStroke("black", 0.5)  3

citiesLayer.set_style(pointStyle) 4
```

1. well known mark name (ex. circle, square, cross, rectangle, diamond, triangle, star, arrow, filled_arrowhead, x), size and rotation
2. Red, Green, Blue, Alpha values for the fill… or named colors
3. defines the stroke using named color and a width
4. apply the style to the layer

Which should lead to this:

LABELS

Adding a default label is as simple as

```
field = "NAME"
pointStyle += HLabel(field, yoffset = -5) + HHalo("red", 1)
```

Which should now look like:

It is also possible to use more options:

```
labelProperties = {
            "font": "Arial",
            "color": 'black',
            "size": 10,
            "field": field,
            "xoffset": 0.0,
            "yoffset": -5
        }
pointStyle += HLabel(**labelProperties) + HHalo("white", 1)
```

USING EXPRESSIONS IN LABELS

It is possible to use expressions in labels. For example conditions using an if construct of the format:

```
if( condition, result if true, result if false)
```

Here we use **concat** to join different fields and strings. To visualize in the label the population only for cities with more than 1M:

```
field = "if(POP_MAX>1000000,concat(NAME,'(',round(POP_MAX/1000000,1),')'),NAME)"
```

The available functions can be found in the QGIS label properties editor.

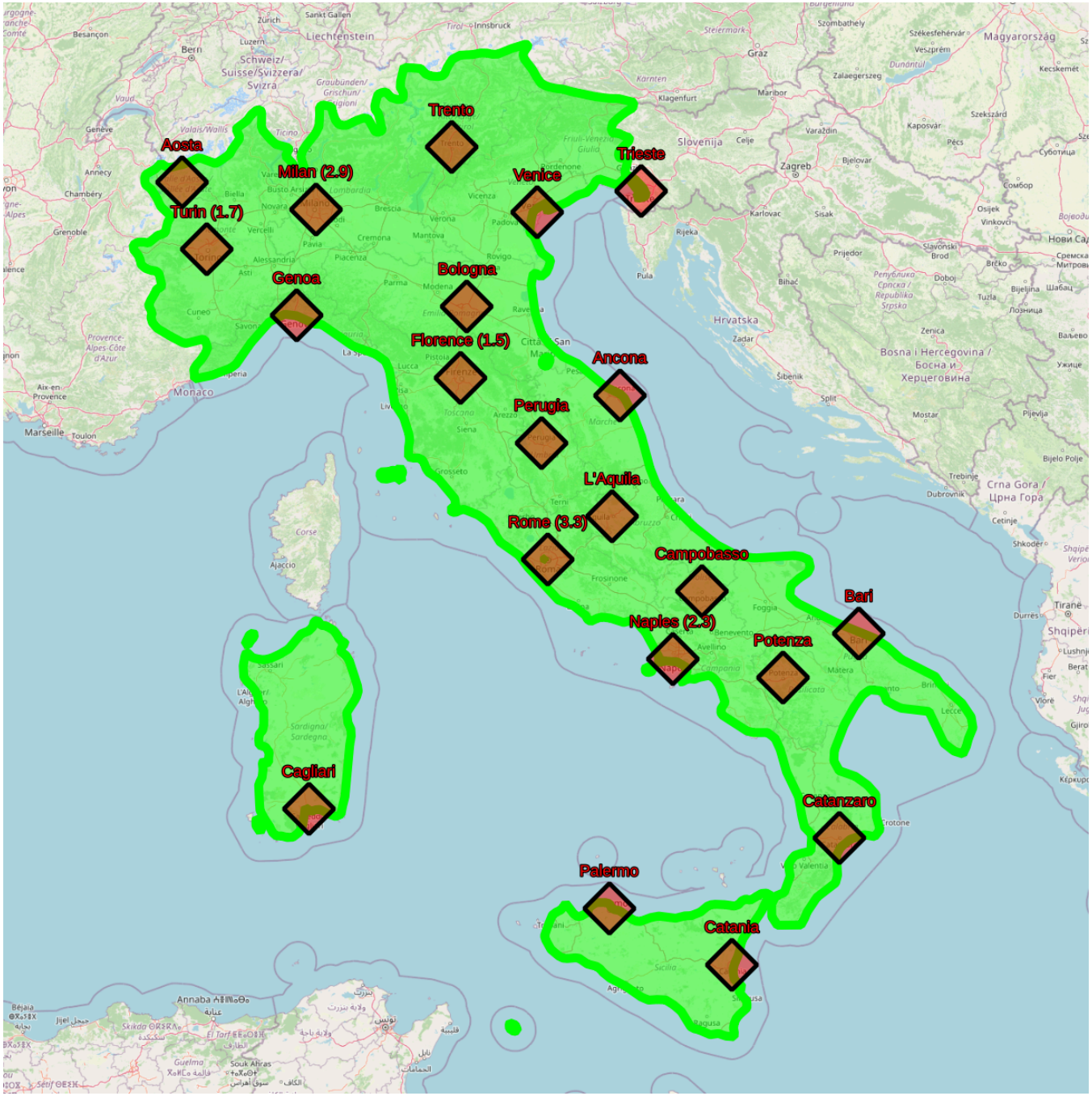
SIMPLE POLYGONS STYLE

When it comes to polygons it is mostly about fill and outline stroke:

```
countriesLayer.subset_filter("NAME='Italy'")
polygonStyle = HFill('0,255,0,128') + HStroke('0,255,0,255', 2) (1)
countriesLayer.set_style(polygonStyle)
```

(1) define a fill with opacity and a stroke.

pay attention to the order in which layers are added to the map


Trento
Aosta
Milan (2.9)
Venice
Trieste
Turin (1.7)
Bologna
Genoa
Florence (1.5)
Ancona
Perugia
L'Aquila
Rome (3.8)
Campobasso
Bari
Naples (2.3)
Potenza
Cagliari
Catanzaro
Palermo
Catania

SIMPLE LINE STYLE WITH LABELLING

The rivers layer doesn’t have an attribute to extract the rivers in Italy. We will need an intersection filter. The **sub_layer** function allows to extract a sublayer based on a geometry filter. It is also possible to define which fields of the original layer to keep.

```
riversItalyLayer = riversLayer.sub_layer(italyFeatures[0].geometry, \
    "rivers_italy", ['scalerank', 'name'])

HMap.add_layer(riversItalyLayer)
```

Let’s apply some basic style:

```
riversStyle = HStroke('0,0,255,255', 2)
riversItalyLayer.set_style(riversStyle)
```

Our map should now look like:

Let’s add some labels:

```
labelProperties = {
            "font": "Arial",
            "color": 'red',
            "size": 14,
            "field": 'name',
            "along_line": True,   1
            "bold": True,
            "italic": False
        }
labelStyle = HLabel(**labelProperties) + HHalo("white", 1)
riversStyle = HStroke('0,0,255,255', 2) + labelStyle
```

1. In case of lines it is possible to make the label follow its direction.

ADVANCED STYLE FOR RIVERS

It is possible to apply thematic styling:

```
ranges = [
    [0, 0],
    [1, 5],
    [6, 8],
    [8, 9],
    [10, 11]
]
styles = [
    HStroke('0,0,255,255', 7),
    HStroke('0,0,255,255', 5),
    HStroke('0,0,255,255', 3),
    HStroke('0,0,255,255', 2),
    HStroke('0,0,255,255', 1),
]
riversItalyLayer.set_graduated_style('scalerank', ranges, styles, labelStyle)
```

The **set_graduated_style** function needs the field name to act on, the value ranges, the colors for said ranges and the optional label style.


Trento
Aosta
Trieste
Milan (2.9)
Venice
Turin (1.7)
Bologna
Genoa
Florence (1.5)
Ancona
Perugia
L'Aquila

PRINTING DATA TO IMAGE AND PAPER

QGIS allows the creation of print layouts via scripting. This is an interesting way to export data, since it is possible to add elements as legend, scalebar or labels to the exported map.

CREATE A LAYOUT

Create a printer object, passing the iface object:

```
printer = HPrinter(iface)
```

ADD THE MAP CANVAS

```
mapProperties = {
    "x": 5,  (1)
    "y": 25,
    "width": 285,
    "height": 180,
    "frame": True, (2)
    "extent": [10, 44, 12, 46]  (3)
}
printer.add_map(**mapProperties)
```

1. define paper position and size in mm
2. add a frame around the map and then add the map object to the layout
3. set the world extent the map should cover.

EXPORT THE GENERATED MAP TO PDF

```
path = "/home/hydrologis/TMP/UNIBZ/AG/test.pdf"
printer.dump_to_pdf(path)
```

To export to image:

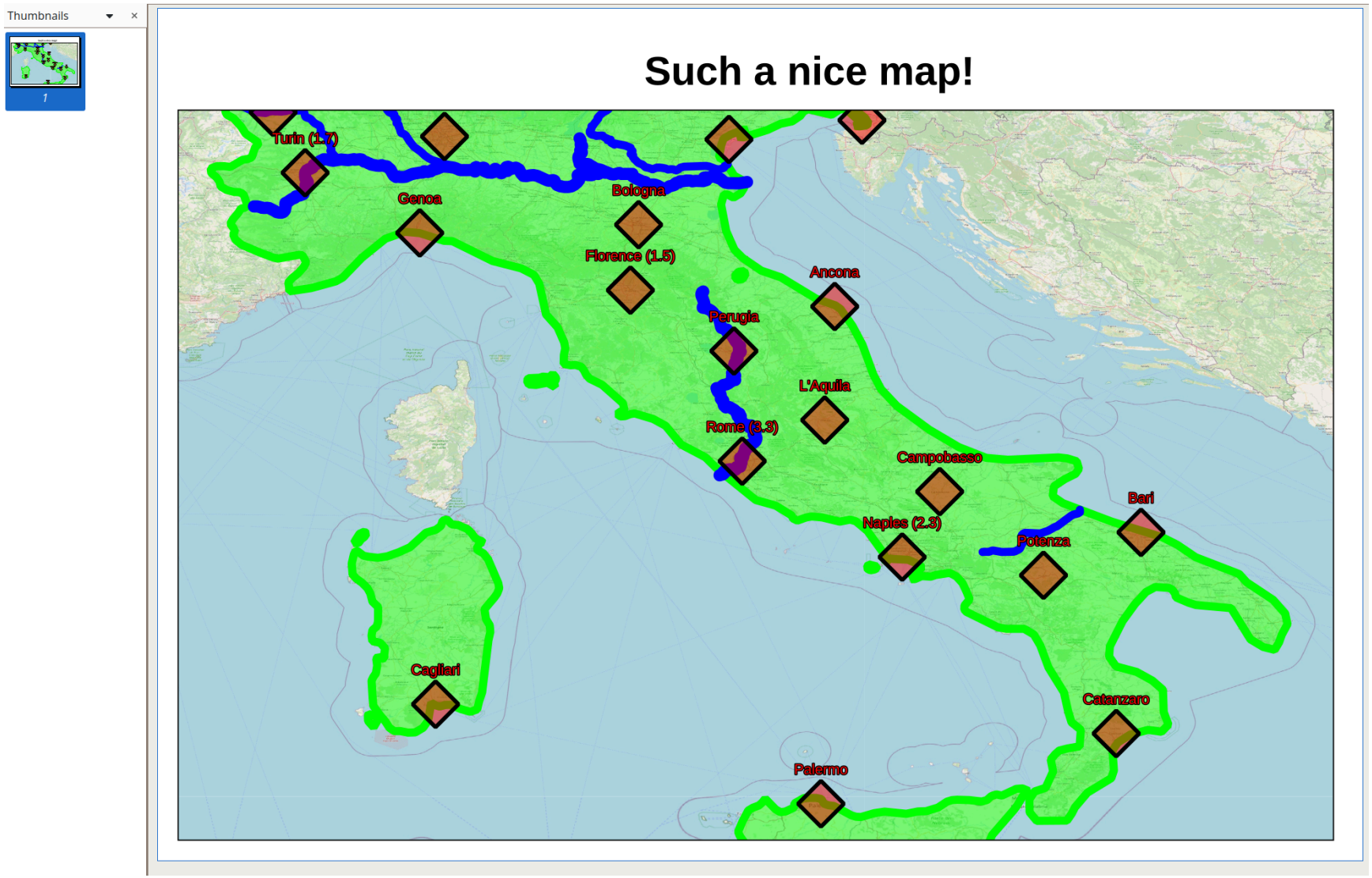
```
path = "/home/hydrologis/TMP/UNIBZ/AG/test.png"
printer.dump_to_image(path)
```

The result should look like:

ADD A TITLE LABEL

```
labelProperties = {
    "x": 120,
    "y": 10,
    "text": "Such a nice map!",
    "font_size": 28,
    "bold": True,
    "italic": False
}
printer.add_label(**labelProperties)
```

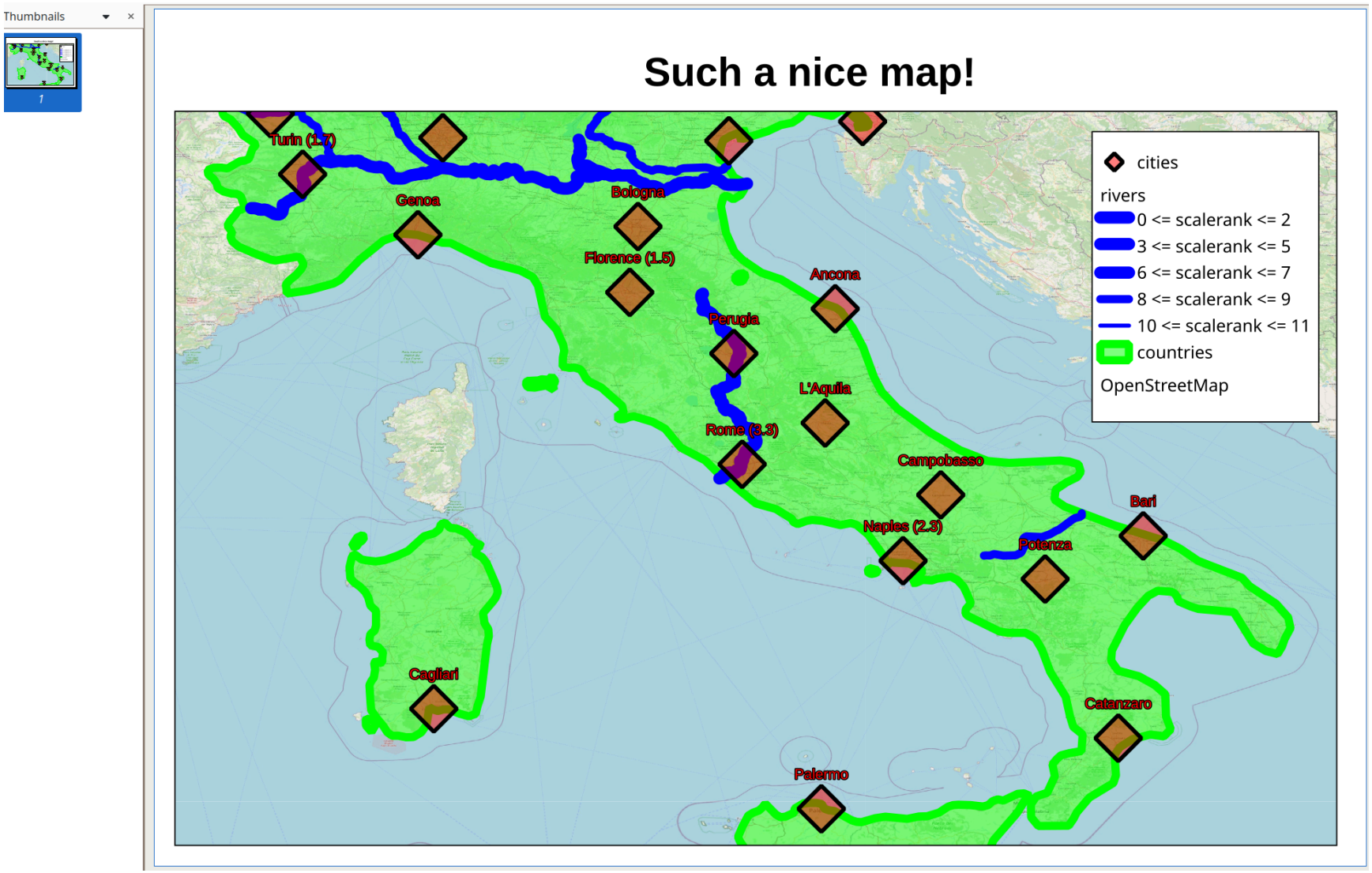
The result should look like:



ADD THE LEGEND

```
legendProperties = {
    "x": 215,
    "y": 30,
    "width": 150,
    "height": 100,
    "frame": True,
    "max_symbol_size": 3
}
printer.add_legend(**legendProperties)
```

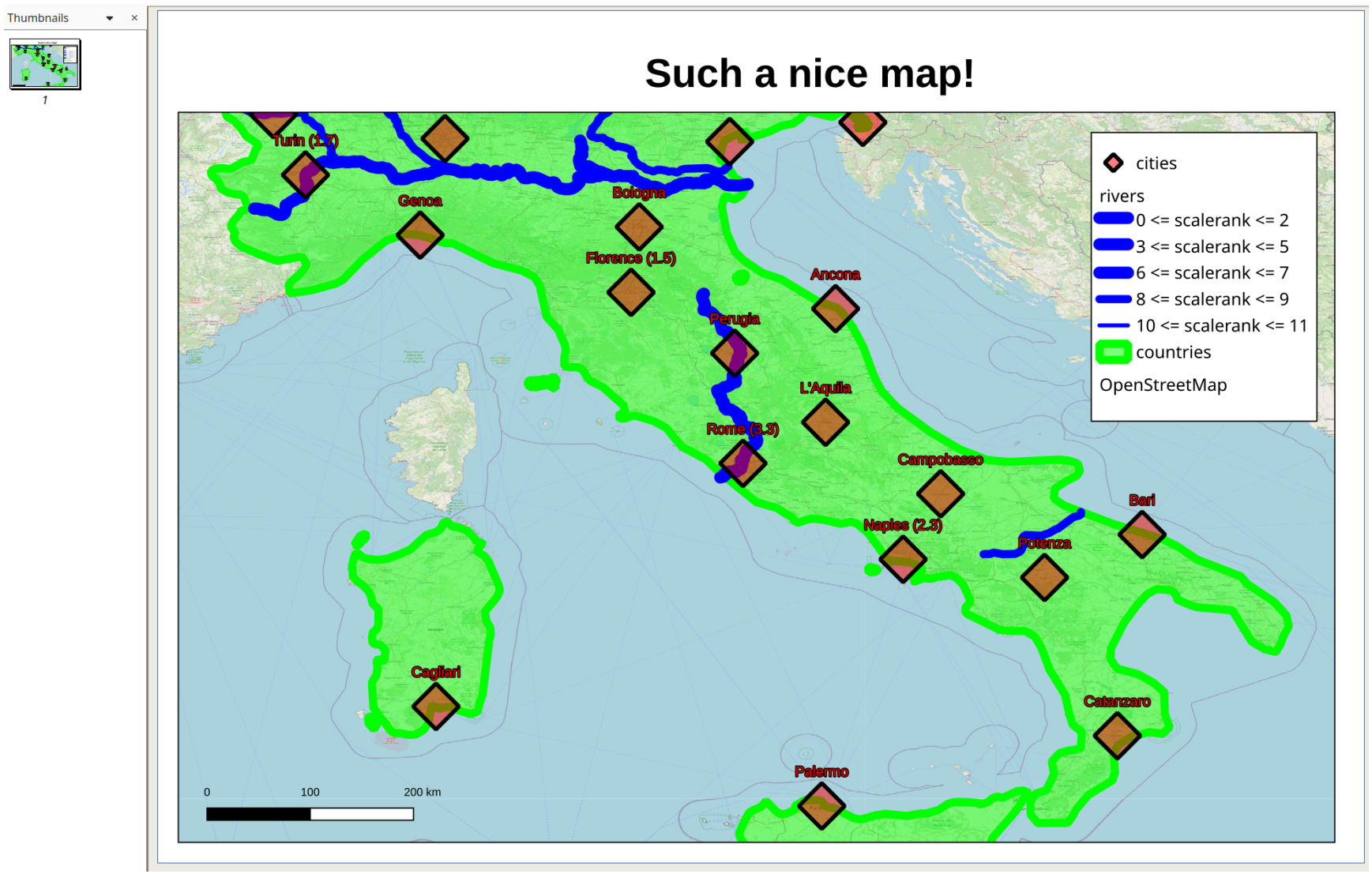
The result should look like:



ADD THE SCALEBAR

```
scalebarProperties = {
    "x": 10,
    "y": 190,
    "units": "km",
    "segments": 4,
    "unit_per_segment": 10,
    "style": "Single Box", # or 'Line Ticks Up'
    "font_size": 12
}
printer.add_scalebar(**scalebarProperties)
```

The result should look like:



EXERCISES

01 - CONVERT THE STATIONS FILE TO GEOPACKAGE

Parse the stations.txt file and create a vector layer with all the columns of the CSV file as attributes. Then save it to geopackage.

02 - FIND CITIES INSIDE FRANCE

Using the Natural Earth datasets, print out the names of the cities that lie inside the French country polygon.

03 - CREATE A GPKG WITH THE COUNTRIES CENTROIDS

Using the Natural Earth datasets, create a new geopackage containing the layer of the centroids of the countries.

Print the names of the countries that do not contain their centroid.

04 - STYLE COUNTRIES BASED ON POPULATION

Using the Natural Earth datasets, create a nice pdf with a map that colors the countries as follows:

- red: population larger than 80000000
- blue: population between 1000000 and 80000000
- green: population smaller than 1000000

```
<license>
  This work is released under Creative Commons Attribution
  Share Alike (CC-BY-SA).
</license>
<sources>
  Much of the knowledge needed to create this training material has
  been produced by the sparkling knights of the
  <a href="http://www.osgeo.org">OSGEO</a> and
  <a href="http://qgis.org">QGIS</a>,
  communities.
  Their websites are filled up with learning material that can be used
  to grow knowledge beyond the boundaries of this lessons
  Another essential source has been the Wikipedia project.
</sources>
<important>
  This work is part of the Advanced Geomatics Course given in 2024 at the
  EMMA Master of the Free University of Bolzano.
</important>
```