Geo Data analytics with Python: Geopandas

Option GIS-Python

hes·so business.

Jean-Paul Calbimonte

School of Management

Bachelor of Science HES-SO (BSc) in Business Information Technology

swissuniversities

> Geopandas

What is Geopandas? http://geopandas.org/

- Make working with **geospatial data** in python easier.
- Combines the capabilities of **pandas** and **shapely**.
- Provides **geospatial operations** in pandas
- High-level interface to **multiple geometries** to shapely.
- Enables to easily do operations in **python** that would otherwise require a spatial database such as **PostGIS**.

> Geopandas data structures

- GeoPandas implements two main data structures, a **GeoSeries** and a **GeoDataFrame**.

- Subclasses of pandas Series and DataFrame, respectively.

- 3 basic classes of geometric objects (actually shapely objects):
  - Points / Multi-Points
  - Lines / Multi-Lines
  - Polygons / Multi-Polygons

> GeoSeries

- Implements nearly all of the attributes and methods of Shapely objects.
- When applied to a GeoSeries, they will apply elementwise to all geometries in the series.
- Binary operations can be applied between two GeoSeries
- Binary operations can also be applied to a single geometry.

  - **area**: shape area
  - **bounds**: tuple of max and min coordinates on each axis
  - **total_bounds**: tuple of max and min coordinates on each axis for entire GeoSeries
  - **geom_type**: type of geometry.
  - **is_valid**: tests if coordinates make a shape that is reasonable geometric shape.
  - **distance**(other): returns Series with minimum distance from each entry to other
  - **centroid**: returns GeoSeries of centroids
  - **representative_point**(): returns GeoSeries of points that are guaranteed to be within each geometry.
  - **to_crs**(): change coordinate reference system.
  - **plot**(): plot GeoSeries.
  - **geom_almost_equals**(other): is shape almost the same as other
  - **contains**(other): is shape contained within other
  - **intersects**(other): does shape intersect other

> GeoDataFrame

- Tabular data structure that contains a **GeoSeries**.
- Always has **one** GeoSeries column that holds a special status.
- This GeoSeries is referred to as the GeoDataFrame's "**geometry**".
- Spatial methods applied to a GeoDataFrame always act on the "geometry" column.
- The "geometry" column can be accessed through the geometry attribute (**gdf.geometry**),
- Name of the geometry column can be found by typing **gdf.geometry.name**.
- A GeoDataFrame may also contain other columns with geometrical (shapely) objects, but **only one column** can be the active geometry at a time.
- To change which column is the active geometry column: **set_geometry** method.

> Shapefile

- Popular geospatial vector data format for geographic information systems (GIS)
- Developed and regulated by Esri
- (Mostly) open specification for data interoperability among Esri and other GIS
- Shapefile format can spatially describe vector features: points, lines, and polygons,
- Each item usually has attributes that describe it.

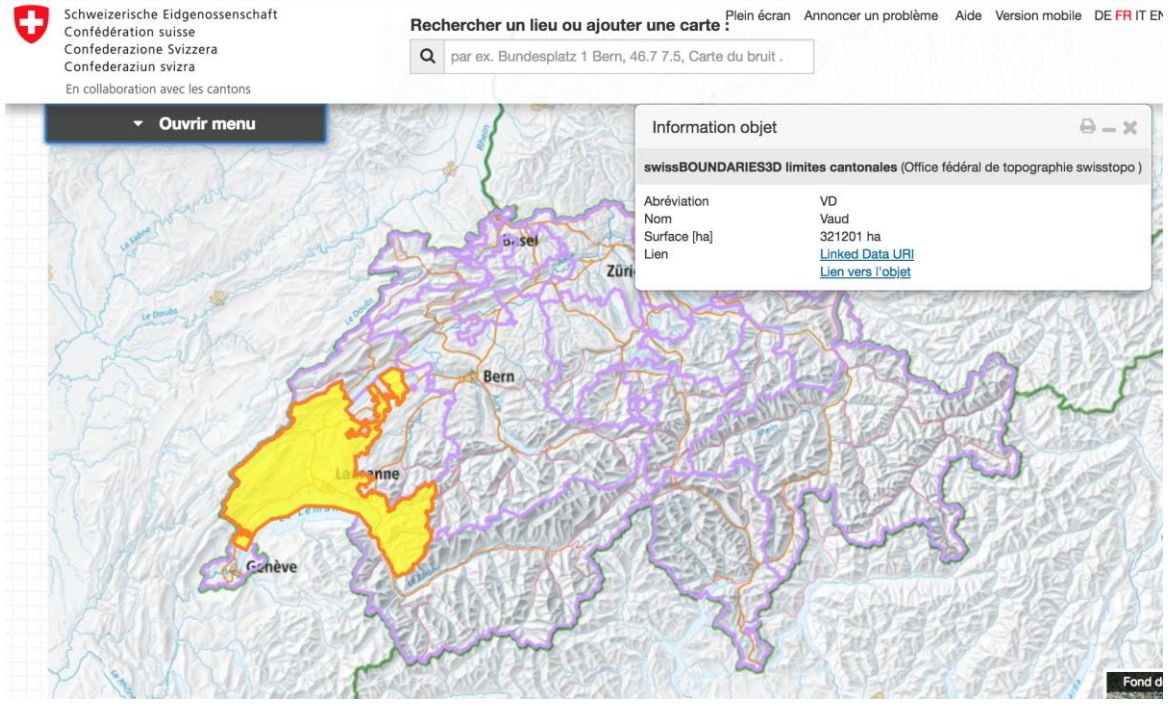
- .shp — shape format; the feature geometry itself
- .shx — shape index format; a positional index of the geometry to allow seeking forwards and backwards quickly
- .dbf — attribute format; columnar attributes for each shape,
- .prj — projection description, using a WKT of the CRS
- .sbn and .sbx — a spatial index of the features
- .fbn and .fbx — a spatial index of the features that are read-only
- .ain and .aih — an attribute index of the active fields in a table
- .ixs — a geocoding index for read-write datasets
- .mxs — a geocoding index for read-write datasets (ODB format)
- .atx — an attribute index for the .dbf file in the form of *shapefile*.*columnname*.atx
- .shp.xml — geospatial metadata in XML format
- .cpg — used to specify the code page (only for .dbf) for identifying the character encoding to be used
- .qix — an alternative quadtree spatial index used by MapServer and GDAL/OGR software

https://www.swisstopo.admin.ch/en/geodata/landscape/boundaries3d.html#links

> Loading Shapefiles

Load shapefile as a Table with spatial objects in it

https://map.geo.admin.ch/ Lots of official Swiss maps

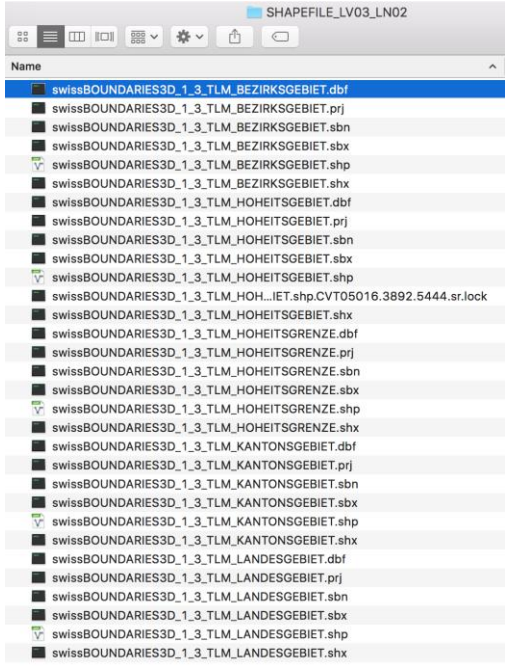


swissBOUNDARIES3D: Swiss limits

http://data.geo.admin.ch/ch.swisstopo.swissboundaries3d-kanton-flaeche.fill/data.zip

> Loading Shapefiles

Lots of files inside the shapefile zip:



5 different layers:

borders (multi lines)

	Géométrie	Description
TLM_HOHEITSGRENZE	Polyligne	Limites administratives (frontières nationale, cantonale, de district, communale)
TLM_HOHEITSGEBIET	Polygone	Unités administratives de base (communes)
TLM_BEZIRKSGEBIET	Polygone	Territoires des districts
TLM_KANTONSGEBIET	Polygone	Territoires des cantons
TLM_LANDESGEBIET	Polygone	Territoires des pays

municipality polygons

districts polygons

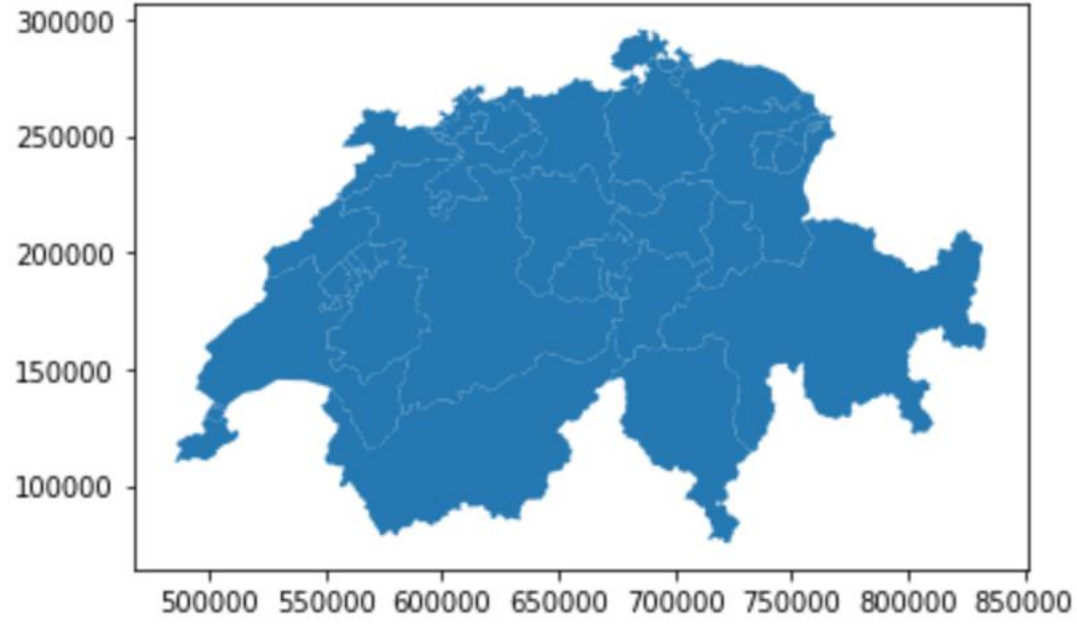
canton polygons

country polygons

> Reading files & plot

```
import geopandas as gpd
fp="../data/cantons/swissBOUNDARIES3D_1_3_TLM_KANTONSGEBIET.shp"

data = gpd.read_file(fp)
type(data)
geopandas.geodataframe.GeoDataFrame
```

```
data.plot()
<matplotlib.axes._subplots.AxesSu
bplot at 0x117deadd0>

matplotlib inline
data.plot()
```



> Exploring GeoDataFrame

```
data[2:5]
```

ERKUNFT	HERKUNFT_J	...	OBJEKTART	REVISION_Q	ICC	KANTONSNUM	SEE_FLAECH	KANTONSFLA	KT_TEIL	NAME	EINWOHNERZ	geometry
AV	2019	...	Kanton	2018_Aufbau	CH	23	1060.0	522463.0	0	Valais	341463.0	POLYGON Z ((679715.0710697878 153452.154104013...
AV	2019	...	Kanton	2018_Aufbau	CH	22	39097.0	321201.0	1	Vaud	793129.0	POLYGON Z ((549756.7944861775 189303.15992634 ...
AV	2019	...	Kanton	2018_Aufbau	CH	21	7147.0	281216.0	0	Ticino	353709.0	POLYGON Z ((679715.0710697878 153452.154104013...

```
data.head(1)
```

NFT	HERKUNFT_J	...	OBJEKTART	REVISION_Q	ICC	KANTONSNUM	SEE_FLAECH	KANTONSFLA	KT_TEIL	NAME	EINWOHNERZ	geometry
AV	2019	...	Kanton	2018_Aufbau	CH	18	NaN	710530.0	0	Graubünden	197888.0	POLYGON Z ((709776.0697413046 185645.950723568...

> Geometry basic operations

```
print(data['geometry'].head(1).exterior)

0    LINEARRING Z (709776.0697413046 185645.9507235...dtype: object
```

```
data['geometry'].head(1).area

0    7.105217e+09dtype: float64
```

> Data Selection

```
selection=data.loc[data['NAME']=='Vaud']
selection
```

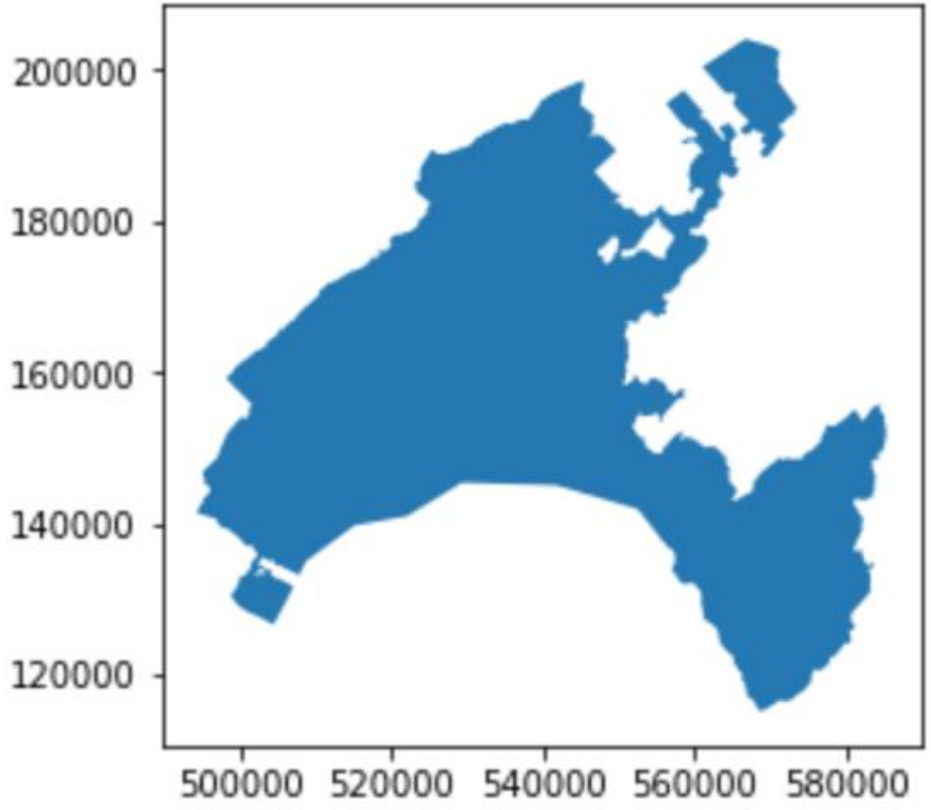
RKUNFT	HERKUNFT_J	...	OBJEKTART	REVISION_Q	ICC	KANTONSNUM	SEE_FLAECH	KANTONSFLA	KT_TEIL	NAME	EINWOHNERZ	geometry
AV	2019	...	Kanton	2018_Aufbau	CH	22	39097.0	321201.0	1	Vaud	793129.0	POLYGON Z ((549756.7944861775 189303.15992634 ...
AV	2015	...	Kanton	2018_Aufbau	CH	22	NaN	NaN	2	Vaud	NaN	POLYGON Z ((569696.1611532553 203238.821218004...

> Iterating over GeoDataFrame

```
for idx,row in selection.iterrows():
    area=row['geometry'].area
    print("The geometry area at {0} is {1}".format(idx,area))

The geometry area at 3 is 3118536618.9
The geometry area at 26 is 93463290.3216
```

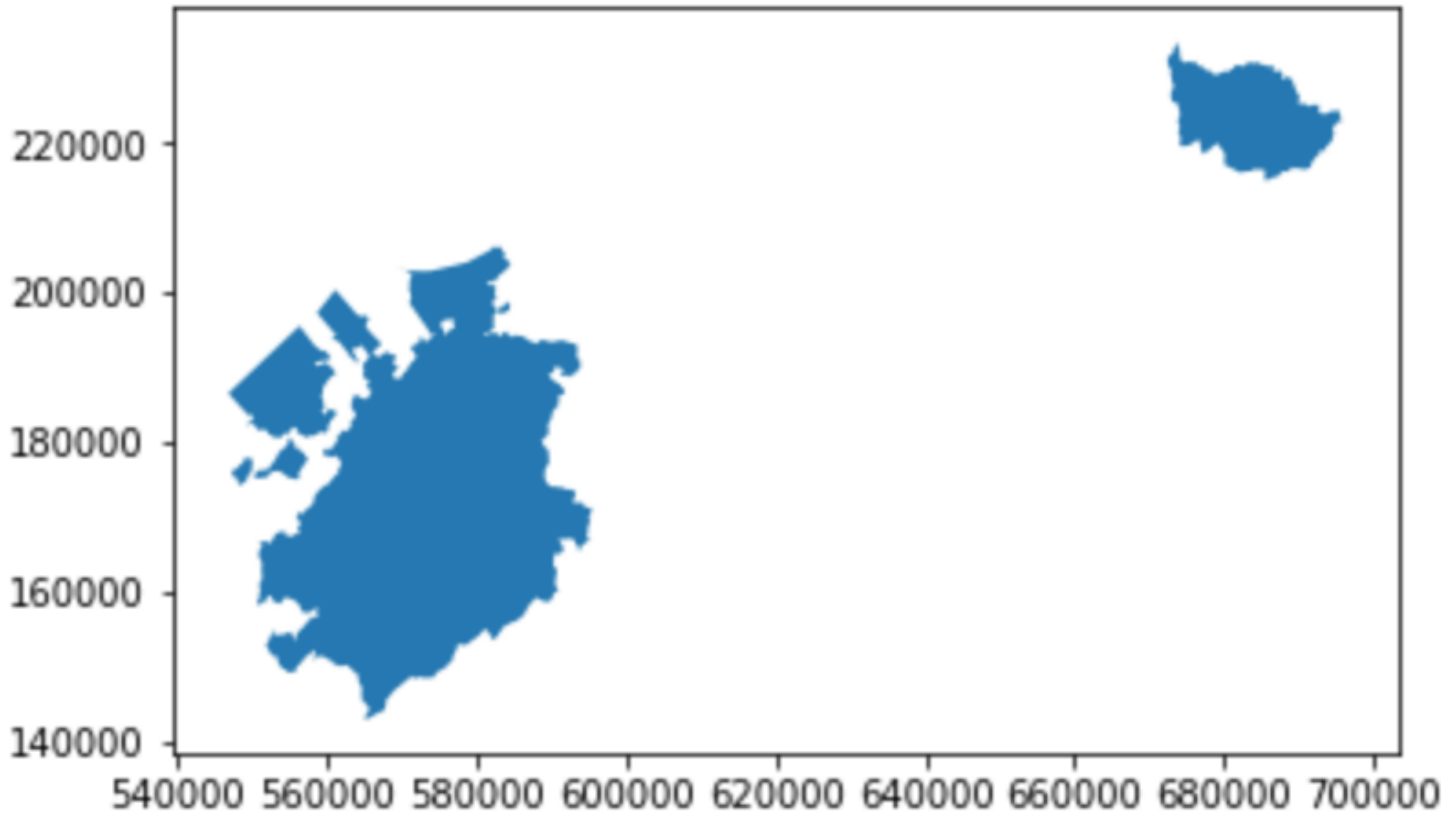
> Data filtering

```
selection=data.loc[data['NAME']=='Vaud']
selection.plot()
```



> Data filtering

```
zugAndFribourg=data.loc[(data['NAME']=='Zug')
                        |(data['NAME']=='Fribourg')]
zugAndFribourg.plot()
```



> Iterating and spatial operations

```
for idx,row in zugAndFribourg.iterrows():
    dist=row['geometry'].distance(data.loc[6]['geometry'])
    print("distance from {0} to {1} is {2}"
                  .format(data.loc[8]['NAME'],row['NAME'],dist))
```

```
distance from Luzern to Fribourg is 89285.8176542
distance from Luzern to Zug is 0.0
distance from Luzern to Fribourg is 120366.375973
distance from Luzern to Fribourg is 128313.159856
distance from Luzern to Fribourg is 135083.448233
distance from Luzern to Fribourg is 95400.1243037
distance from Luzern to Fribourg is 118589.427963
```

> Iterating and spatial operations

```
for idx,row in zugAndFribourg.iterrows():
  dist=row['geometry'].centroid.distance(data.loc[6]['geometry'].centroid)
  print("distance from {0} to {1} is {2}"
                .format(data.loc[8]['NAME'],row['NAME'],dist))
```

```
distance from Luzern to Fribourg is 141967.4230203907
distance from Luzern to Zug is 29767.045955095706
distance from Luzern to Fribourg is 151613.55850258234
distance from Luzern to Fribourg is 156247.23860257282
distance from Luzern to Fribourg is 161859.94375018188
distance from Luzern to Fribourg is 120833.9616898067
distance from Luzern to Fribourg is 143423.81018197164
```

> Creating new GeoDataFrame

```
newData=gpd.GeoDataFrame()
newData['geometry']=None

data['center']=None

for idx,row in data.iterrows():
    center=row['geometry'].centroid
        data.loc[idx,'center']=center
```

```
df=gpd.GeoDataFrame(data['center'])

conversion={'center':'geometry'}
df.rename(columns=conversion)
```

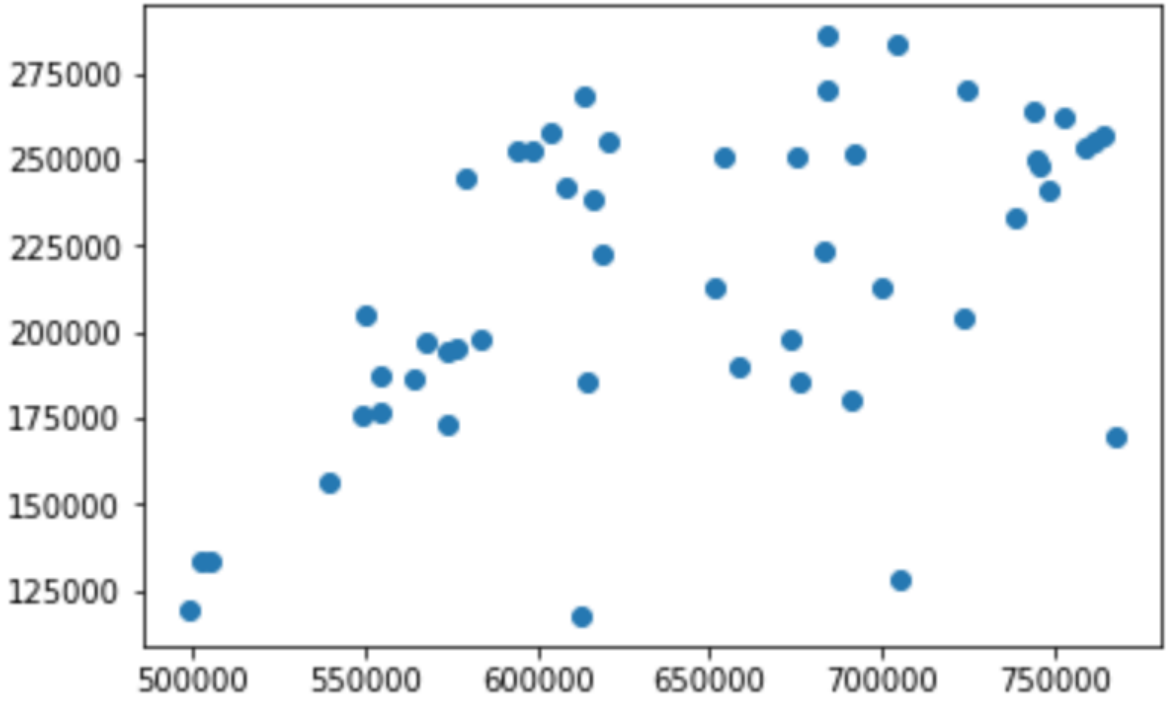
geometry
POLYGON Z ((679715.0710697878 153452.154104013...
POLYGON Z ((549756.7944861775 189303.15992634 ...
POLYGON Z ((679715.0710697878 153452.154104013...

	geometry
0	POINT (767575.0244501946 169561.5515886946)
1	POINT (614277.2987648115 185601.5741448295)
2	POINT (612816.6401953455 117581.908002399)
3	POINT (539303.7917365739 156760.3369243431)
4	POINT (705582.446308137 128055.1089343966)
5	POINT (739058.4421031841 233039.9063328989)
6	POINT (691804.3897484293 252035.1222239229)
7	POINT (573827.5419753399 173065.4281091813)
8	POINT (651010.0992641809 213189.1664508271)

> Setting geometry GeoSeries

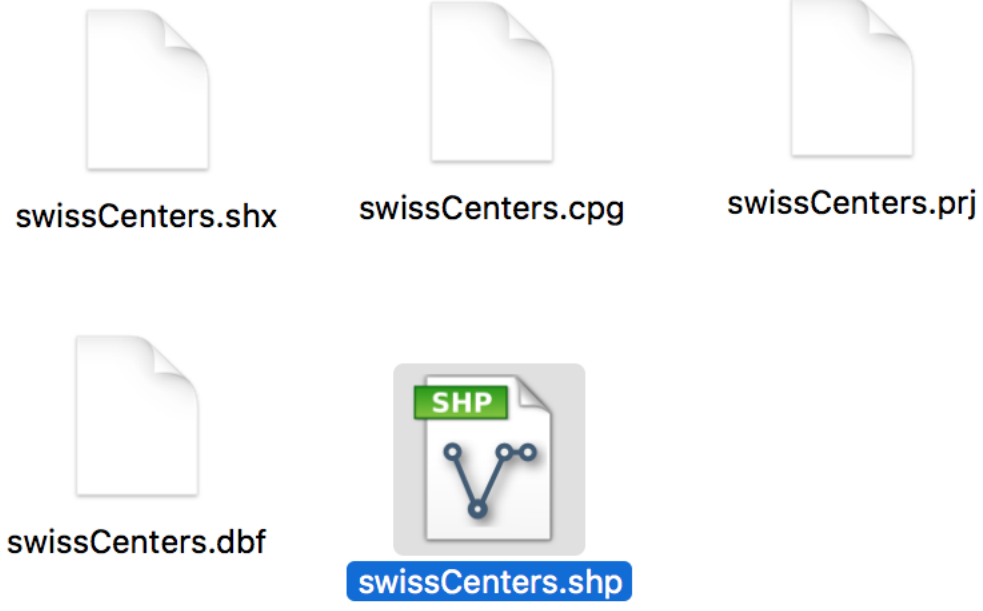
```
df.crs=data.crs
df.crs

{u'ellps': u'bessel', u'k_0': 1, u'lat_0': 46.95240555555556,
u'lon_0': 7.439583333333333, u'no_defs': True, u'proj': u'somerc',
u'units': u'm', u'x_0': 600000, u'y_0': 200000}
```

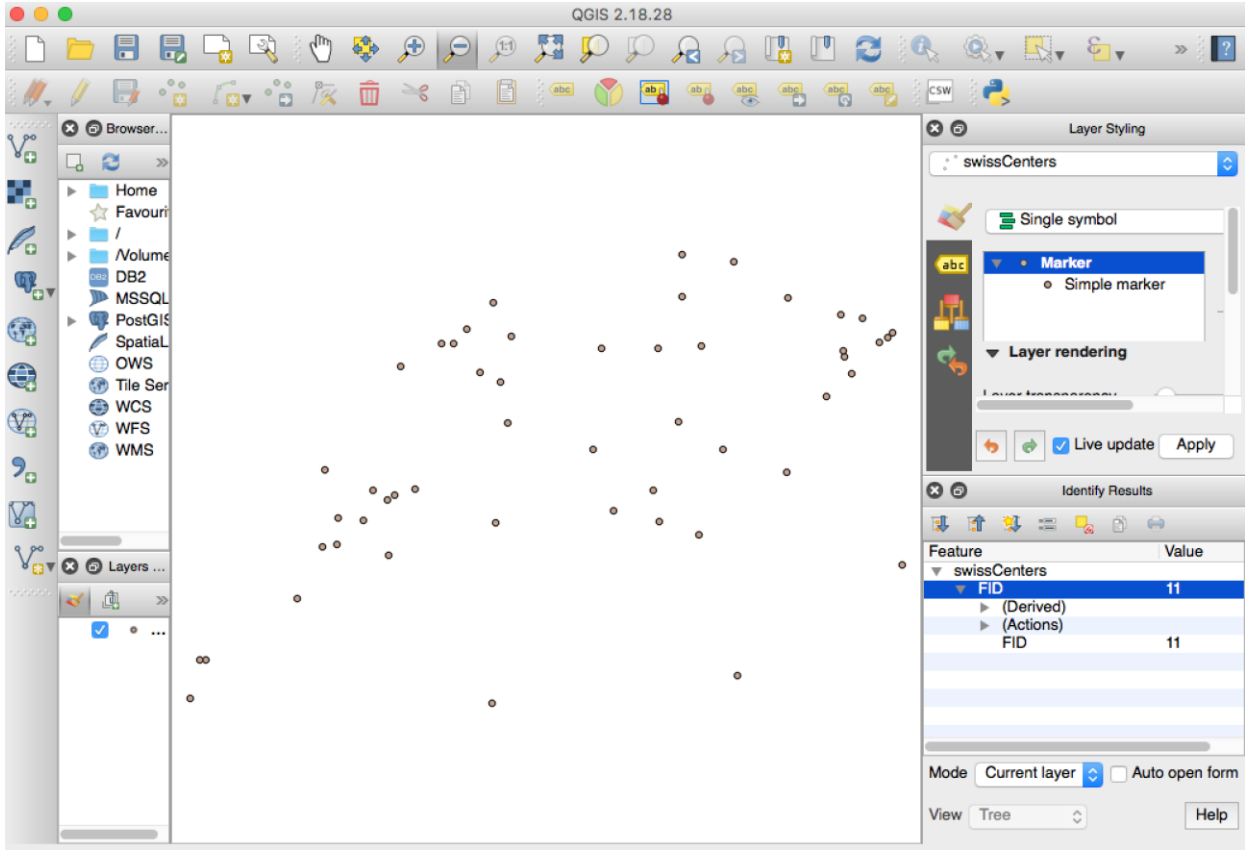
```
df=df.set_geometry('center')
df.plot()
```

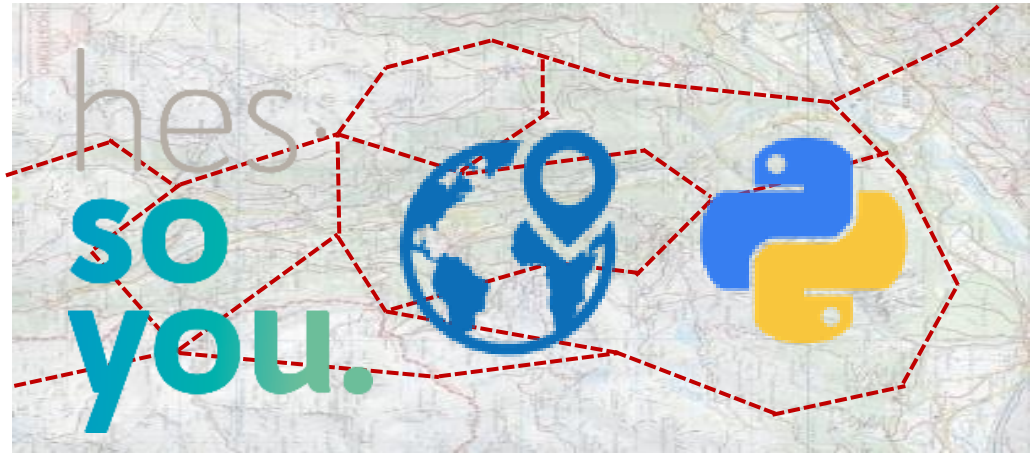


> Writing shapefiles

```
df.to_file('swissCenters.shp')
```







Thank you for your attention.

School of Management
Route de la Plaine 2
3960 Sierre

hevs.ch/heg

swissuniversities