GeoPandas: Algemene introductie

Dr. Cornelis Stal

April 27, 2022

1 Introductie tot `GeoPandas`

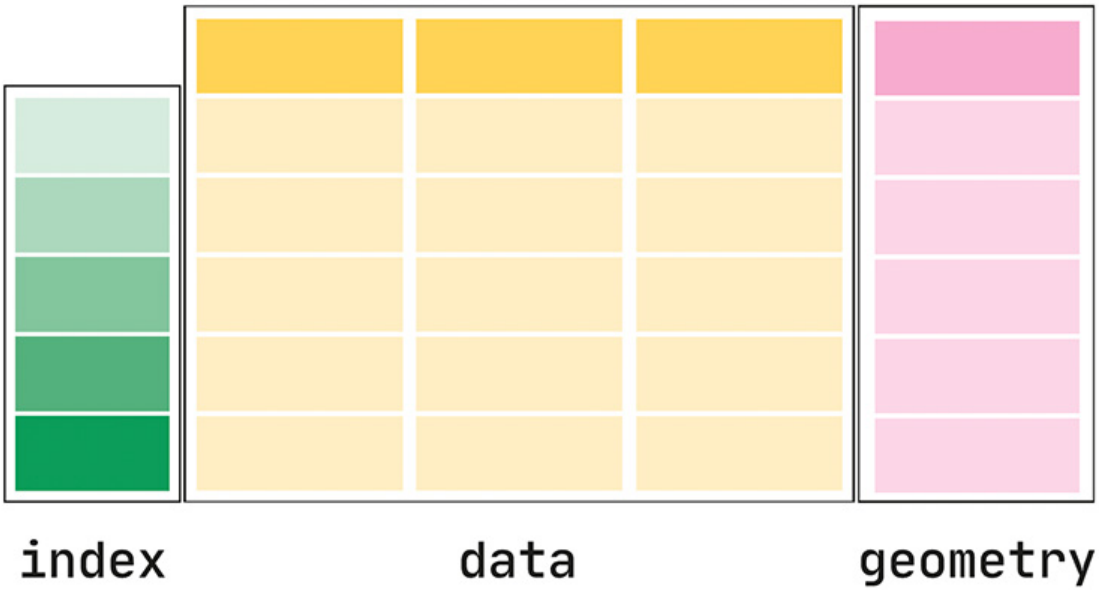
Met deze *notebook* willen we u kennis laten maken met enkel belangrijke concepten en basisfunctionaliteiten van `GeoPandas`. In principe zou u met het doorlopen van dit document al van start moeten kunnen gaan met het gebruik van deze uiterst interessante bibliotheek voor ruimtelijke data in Python. Naast deze algemene introductie zijn een aantal aanvullende *notebooks* aangemaakt die dieper ingaan op enkele handige functies en methoden.

Deze tutorial is een vertaling van de 'getting started guide' op `https://geopandas.org`.

1.1 Concepten

Zoals de naam al doet vermoeden breidt `GeoPandas` de populaire data science-bibliotheek `pandas` uit door ondersteuning voor ruimtelijke gegevens toe te voegen. Kennis over `pandas` is vereist om te kunnen werken met `GeoPandas`, en we verwijzen daarvoor graag door naar onze eerdere *notebooks* of naar de *Getting started guide*.

De basis van de gegevensstructuur binnen `GeoPandas` is de `geopandas.GeoDataFrame`, een subklasse van `pandas.DataFrame`, die geometriekolommen kan opslaan en ruimtelijke bewerkingen kan uitvoeren. De `geopandas.GeoSeries`, een subklasse van `pandas.Series`, handelt de geometrieën af. Zodoende is de `GeoDataFrame` een combinatie van `pandas.Series`, met traditionele gegevens (numeriek, boolean, tekst enz.) en `geopandas.GeoSeries`, met geometrieën (punten, polygonen enz.). We kunnen zoveel kolommen met geometrieën toevoegen als gewenst. In tegenstelling tot bestandsformaten die in reguliere desktop GIS-software gebruikt worden, is hier geen limiet op.



Each `GeoSeries` can contain any geometry type (you can even mix them within a single array) and has a `GeoSeries.crs` attribute, which stores information about the projection (CRS stands for Coordinate Reference System). Therefore, each `GeoSeries` in a `GeoDataFrame` can be in a different projection, allowing you to have, for example, multiple versions (different projections) of the same geometry.

Elke `GeoSeries` kan elk type geometrie bevatten. we kunnen zelfs geometriën integreren in een *array*. De `GeoSeries` beschikt eveneens over een `GeoSeries.crs`-attribuut, waarin informatie over het coördinaatreferentiesysteem (CRS) opgeslagen wordt. Daarom kan elke `GeoSeries` in een `GeoDataFrame` zich in een andere projectie bevinden, waardoor we bijvoorbeeld meerdere versies (verschillende projecties) van dezelfde geometrie kunnen hebben.

Slechts één `GeoSeries` in een `GeoDataFrame` wordt beschouwd als de "actieve" geometrie, wat betekent dat alle geometrische bewerkingen die op een `GeoDataFrame` worden toegepast op deze "actieve" kolom worden toegepast.

> **Gebruikshandleiding:** voor meer informatie over datastructuren verwijzen we naar de corresponderende sectie in de gebruikershandleiding.

1.2 Bestanden lezen en schrijven

Vooraleer we kunnen werken met `GeoPandas` moeten we uiteraard eerst data inladen.

Bestanden lezen

Wanneer we een bestand hebben dat zowel ruimtelijke als niet-ruimtelijke data bevat (bijvoorbeeld een GeoPackage, GeoJSON, ESRI Shapefile, …), kunnen we deze data inlezen met behulp van `geopandas.read_file()`. Deze methode detecteert automatisch het bestandstype en resulteert in een `GeoDataFrame`.

> **Opmerking:** voor deze demonstratie zullen we gebruik maken van de dataset van statische sectoren in Vlaanderen, die beschikbaar gesteld wordt via het platform Provincie in Cijfers, met verwijzing naar het Belgische Statistiekbureau.

De data zijn beschikbaar via deze link, en we kunnen de rechtstreeks ophalen met een `GET`-*request* uit de `requests`-bibliotheek. Vervolgens importeren we de *response* in een `GeoDataFrame` en inspecteren we het resultaat:

[ ]:
```
import geopandas as gpd
import requests
r = requests.get('https://provincies.incijfers.be/report/GeoJSON/statsec.txt')
statsec = gpd.read_file(r.text)
statsec.head()
```

[ ]:
```
r.text
```

Bestanden schrijven

Om een `GeoDataFrame` weg te schrijven naar een nieuw bestand gebruiken we `GeoDataFrame.to_file()`. Het standaard bestandsformaat waarin data weggeschreven worden is de ESRI Shapefile, maar we kunnen ook andere formaten gebruiken door een waarde toe te kennen aan de `driver`-variabele:

[ ]:
```
statsec.to_file('data/statsec_Vlaanderen.geojson', driver='GeoJSON')
```

> **Gebruikshandleiding:** voor meer informatie over het lezen en schrijven van bestanden verwijzen we naar de corresponderende sectie in de gebruikshandleiding.

1.3 Eenvoudige toegang tot de data en aanverwante methoden

Nu we beschikken over een `GeoDataFrame`-object gaan we dieper in op de geometrische *features* binnen deze dataset. Aangezien er slechts één kolom met geometrieën aanwezig is in de gegevensset, wordt deze kolom automatisch de "actieve" geometrie en worden ruimtelijke methoden die op het `GeoDataFrame` worden gebruikt, toegepast op de `geometry`-kolom.

Oppervlakten berekenen

Om de oppervlakte van elke `Polygon` (of in veel gevallen elke `MultiPolygon`)te berekenen, openen we het attribuut `GeoDataFrame.area`, dat een `pandas.Series` retourneert. Merk op dat `GeoDataFrame.area` gewoon van het type `GeoSeries.area` is, toegepast op de "actieve" geometriekolom.

Maar om de resultaten beter leesbaar te maken, stellen we eerst de identifier in het veld `geoitem` in als index. Dit veld is een unieke *identifier* voor iedere statistische sector. We willen de oppervlaktes overigens berekenen in Ha en niet in m^2 (of in graden^2, zoals we dadelijk zullen zien):

[ ]:
```
statsec = statsec.set_index('geoitem')
statsec['area'] = statsec.area / 1000022
```

De kans is heel groot dat we zojuist de volgende waarschuwing hebben gekregen: `UserWarning: Geometry is in a geographic CRS. Results from 'area' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before this operation.` Wanneer data zonder expliciete vermelding van een CRS worden ingeladen, zal `GeoPandas` ervan uitgaan dat de data in WGS'84 (EPSG:4326) geprojecteerd staan. Op basis

van de bestudering van de data lijkt dit inderdaad correct. We gebruiken vervolgens de methode `to_crs('EPSG:3812')` om de data te transformeren naar het Belgische Lambert '08 CRS.

> **Opmerking:** als de toewijzing van WGS'84 als initieel CRS niet correct is, kunnen we dit 'over-rulen' met de `set_crs('EPSG:<EPSG_CODE/>')`-methode. Meer over `GeoPandas` zal later in deze 'notebook' besproken worden.

[ ]:
```
statsec = statsec.to_crs('EPSG:3812')
statsec['area'] = statsec.area / 10000
statsec['area']
```

Grenzen en centroïdes van polygonen

De grenzen van iedere individuele polygoon kunnen we verkrijgen met behulp van `GeoDataFrame.boundary`. Dit resulteert in een verzameling nieuwe geometriën van het type `LineString`:

[ ]:
```
statsec['boundary'] = statsec.boundary
statsec['boundary']
```

Aangezien we de grenzen verwerkt hebben in de nieuwe kolom **`boundary`**, zijn er nu twee geometriekolommen aanwezig in een en dezelfde `GeoDataFrame`. Op dezelfe manier kunnen we geometriën toevoegen, zoals bijvoorbeeld de buffer van objecten met de `GeoDataFrame.buffer`-methode of de centroïde van een object:

[ ]:
```
statsec['centroid'] = statsec.centroid
statsec['centroid']
```

Afstanden meten

Met de `distance`-methode kunnen we de afstand berekenen tussen punten. In onderstaande voorbeeld berekenen we de afstand van de eerste feature in de databank alle andere punten. Merk hierbij op dat ook de afstand van het eerste punt tot het punt zelf berekend wordt, wat uiteraard resulteert in een afstand van 0 meter:

[ ]:
```
firstPoint = statsec['centroid'].iloc[0]
statsec['distance'] = statsec['centroid'].distance(firstPoint)
statsec['distance']
```

Merk op dat `geopandas.GeoDataFrame` een subklasse is van `pandas.DataFrame`. Dit betekend dat alle functionaliteiten van `pandas` ook beschikbaar zijn op een ruimtelijke dataset. We hoeven ons dus niet te beperken tot louter ruimtelijke bevragingen, maar kunne ook handelingen uitvoeren op de niet-ruimtelijke attributen. Het is zelfs mogelijk om beide zaken tegelijk te gebruiken.

Om bijvoorbeeld de gemiddelde afstand te berekenen van alle zojuist afgeleide afstanden kunnen we de `mean`-methode toepassen op alle waarden in de kolom `distance`:

[ ]:
```
statsec['distance'].mean()
```

1.4 Kaartjes maken

Met `GeoPandas` kunnen we ook heel eenvoudig ruimtelijke data plotten in een kaart. Zelfs zonder nu al zorg te dragen voor de cartografische vormvereisten en de visuele attractiviteit van onze voorstelling, kunnen we makkelijk controlerne hoe onze ruimtelijke data eruit zien, en of ze ruimtelijk correct worden geprojecteerd. Om de "actieve" geometrie te plotten maken we gebruik van de `GeoDataFrame.plot()`-methode.

> **Opmerking:** de plot die we zojuist gemaakt hebben is een `AxesSubplot`-object uit de `matplotlib`-bibliotheek. Meer informatie over deze bibliotheek is hier terug te vinden. In onderstaande code gebruiken we een `pyplot`-subobject om de afmetingen van onze plot aan te passen.

[ ]:
```
# Instelling om figuren wat groter af te beelden
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [15, 5]

statsec.plot()
```

Om een kleur toe te kennen aan onze *features* op basis van een bepaald veld, dienen we de naam van dit veld als attribuut mee te geven aan onze plot. Met `legend=True` kunnen we eveneens voor dit veld een legende toevoegen.

In onderstaand voorbeeld geven we de statistische eenheden weer met een kleur in fucntie van de oppervlakte, zoals verwerkt in het veld `area`:

[ ]:
```
statsec.plot('area', legend=True)
```

We kunnen de data ook projecteren op een interactive *viewer* met passende achtergrond kaart met behulp van de`GeoDataFrame.explore()`-methode. Deze methode werkt op dezelfde manier als de zojuist gebruikte `plot()`-methode, maar resulteert in een interactieve kaart in plaats van een statische afbeelding:

[ ]:
```
statsec.explore("area", legend=False)
```

Met `GeoDataFrame.set_geometry` kunnen we de huidige "actieve" geometrie veranderen. Om in plaats van de polygonen van de statistische sectoren de corresponderende centroïdes te visualiseren als punt-geometriën gebruiken we:

[ ]:
```
statsec = statsec.set_geometry("centroid")
statsec.plot("area", legend=True)
```

We kunnen ook beide geometrische lagen of `GeoSeries` op elkaar visualiseren. Hiervoor projecteren we de data laag per laag door de ene plot te gebruiken als `axis` van de volgende:

[ ]:
```
ax = statsec["geometry"].plot()
statsec["centroid"].plot(ax=ax, color="black")
```

Tot slot stellen we de `GeoSeries` in de kolom `geometry` terug in als onze "actieve" geometrie.

```
[ ]: statsec = statsec.set_geometry("geometry")
```

Gebruikshandleiding: voor meer informatie over het maken van kaarten met GeoPandas verwijzen we naar de corresponderende sectie in de gebruikshandleiding.

1.5 Objecten selecteren op basis van attributen

Het selecteren van objecten binnen een `GeoPandas GeoDataFrame` op basis van attributen werkt op dezelfde manier als dat we dat gewoon zijn met een gewone `Pandas DataFrame`. Zo kunnen we eenvoudig zoeken naar objecten waarvan het veld `naam` overeenkomt met de string `Aaigem`:

```
[ ]: aaigem = statsec[statsec['naam'] == 'Aaigem']
     aaigem
```

Vermits het veld `geoitem` ingesteld staat als index, kunnen we Aaigem ook opvragen met de `loc()`-methode, waarbij de waarde `44021B354` als atribuut meegegeven wordt. Merk hierbij op dat we in dit geval een `GeoSeries`-object terugkrijgen, en geen `GeoDataFrame`:

```
[ ]: aaigem = statsec.loc["44021B354"]
     aaigem
```

Door een tweede attribuut mee te geven aan de `loc()`-methode kunnen we bepaalde attributen van een object opvragen. Wanneer we slechts in één attribuut geïnteresseerd zijn, volstaat opgaven van deze attribuut als een string. Voor meerdere attributen geven we een lijst van strings mee:

```
[ ]: aaigem = statsec.loc['44021B354', ['geometry', 'naam']]
     aaigem
```

Logische expressies op numerieke waarden zijn, naar analogie met `Pandas`, eveneens mogelijk:

```
[ ]: largeSections = statsec[statsec['area'] > 2000]
     largeSections.head()
```

En uiteraard kunnen we selecteren op basis van substrings:

```
[ ]: gent = statsec[statsec.index.str.contains('44021')]
     gent.plot(figsize=[15, 10])
```

1.6 Geometriën aanmaken

Laten we nu eens verder kijken naar de mogelijkheden die `GeoPandas` biedt die specifiek gericht zijn op geometriëen. In een andere tutorial wordt het aanmaken van nieuwe geometriëen behandeld. In deze sectie zullen we ons echter beperken tot de manipulatie van bestaande geometrische objecten.

Convex hull

Om de convexe omhullende te berekenen van een object maken we gebruik van de `GeoDataFrame.convex_hull`-attribuut:

[ ]:
```
gent["convex_hull"] = gent.convex_hull
ax = gent["convex_hull"].plot(alpha=.5, figsize=[15, 10])  # saving the first␣
 ↪plot as an axis and setting alpha (transparency) to 0.5
gent["boundary"].plot(ax=ax, color="white", linewidth=.5)  # passing the first␣
 ↪plot and setting linewitdth to 0.5
```

Buffer

Het berekenen van een buffer rond geometrische objecten is mogelijk met de `GeoDataFrame.buffer()`-methode. Deze en andere methodes worden automatisch uitgevoerd op de momenteel geactiveerde geometrie. Zoals eerder aangegeven kunnen we de actieve geometrie veranderen met de `set_geometry()`-methode. Naast `GeoDataFrame`-objecten kunnen geometrische methoden ook toegepast worden op `GeoSeries`.

Laten we eens een buffer van 5 km berekenen rond de polygonen van Gent en 15 km rond de centroides van de statistische sectoren in Gent. Vervoglens visualiseren we het resultaat:

[ ]:
```
gent["buffered"] = gent.buffer(5000)
gent["buffered_centroid"] = gent["centroid"].buffer(15000)

ax = gent["buffered_centroid"].plot(alpha=.5, color="blue", figsize=[15, 10])
gent["buffered"].plot(ax=ax, color="red", alpha=.25)
gent.plot(ax=ax, color="white", linewidth=.5)
```

> **Opmerking:** de attribuut die aan de `buffer()`-methode gegeven wordt, maar ook aan veel andere geometrische methoden, worden uitgedrukt in het eenhedenstelsel van het CRS. In ons geval (EPSG:3857) is dit in meter.

1.7 Ruimtelijke relaties

Naast bevragingen op basis van attributen laat `GeoPandas` ook die om ruimtelijke bevragingen uit te voegen op basis van ruimtelijke relaties tussen verschillende geometriëen.

In onderstaande voorbeeld zullen we bekijken wat de relatie is tussen de gebufferde zones rond de statistische sectoren in Gent, en de andere sectoren in Vlaanderen. We voegen alle entiteiten binnen Gent samen en berekenen opnieuw de buffer van 5 km:

[ ]:
```
gent = statsec[statsec.index.str.contains('44021')]
gentDissolved = gent.dissolve()['geometry']
gentDissolved = gentDissolved.buffer(5000)

ax = gentDissolved.plot(color="blue", figsize=[15, 10])
gent.plot(ax=ax, color="white")
```

De berekende polygoon is een 'shapely.geometry'-object, zoals alle andere geometrische objecten in een `GeoPandas GeoDataFrame`:

[ ]:
```
gentDissolved.geometry
```

We kunnen nu kijken welke statistische sectoren de gebufferde polygoon van Gent overlappen met de `sjoin()`-methode:

[ ]:
```
gentDissolved_gdf = gpd.GeoDataFrame(geometry=gentDissolved)
intersect = gpd.sjoin(statsec, gentDissolved_gdf, how='inner')
intersect.head()
```

> **Opmergkin:** merk bij het vorige voorbeeld op dat we een 'inner join' hebben toegepast op beide datasets. Vergelijk dit resultaat eens met een 'left' of 'right' join.

De ruimtelije relatie die tussen beide `GeoDataFrame`-objecten geëvalueerd dient te worden wordt ook wel een predicaat genoemd. Deze relatie kan vastgelegd worden door aan de `sjoin()`-methode een waarde mee teg geven voor de `predicate`-parameter. De volgende mogelijkheden zijn beschikbaar:

[ ]:
```
statsec.sindex.valid_query_predicates
```

Om bijvoorbeeld alle zones te selecteren die geheel gelegen zijn binnen de samengevoegde polygoon, wordt de `within` parameter gebruikt:

[ ]:
```
within = gpd.sjoin(statsec, gentDissolved_gdf, how='inner', predicate='within')

ax = intersect.plot(color="blue", figsize=[15, 10])
within.plot(ax=ax, color="red")
```

1.8 Projecties

Iedere `GeoSeries` heeft een eigen coördinaat referentiesysteem (CRS) dat toegankelijk is via `GeoSeries.crs`. Het CRS laat toe om `GeoPandas`-objecten op een coherente manier te positioneren op het aardoppervlak. Vanzelfsprekend is het van belang dat deze positionering gebasseerd is op ondubbelzinnige coördinaten/In sommige gevallen is het CRS geografisch, wat betekent dat de coördinaten in lengte- en breedtegraad zijn. In dergelijke gevallen wordt vaak gebruik gemaakt van WGS'84 (EPSG:4326). Geprojecteerde coördinaatsystemen laten toe om coördinaten uit te drukken met metrische en cartesiaanse waarden. In België wordt hiervoor vaak gebruik gemaakt van Lambert '72 (EPSG:31370) of Lambert '08 (EPSG:3812). Voor webcartografie maken we meestal gebruik van Pseudo-Mercator (EPSG:3857).

Laten we eens kijken in welk CRS de statistische sectoren zich (momenteel) bevinden:

[ ]:
```
statsec.crs
```

Geometriëen worden uitgedrukt in `EPSG:3812` (Lambert '08) met coordianten in meters. We kunnen deze data eenvoudig projecteren in een ander CRS, zoals WGS84 met de `GeoSeries.to_crs()`-methode():

[ ]:
```
statsec = statsec.set_geometry("geometry")
statsec_4326 = statsec.to_crs("EPSG:4326")
statsec_4326.plot()
```

[ ]:
```
statsec_4326.crs
```

De data worden nu inderdaad geprojecteerd in WGS'84. Op basis van eht grid rond de kaart zien we nu ook geografische coördinaten in plaats van metrische geprojecteerde coördinaten.

> **Opmerking:** voor berekeningen die afhankelijk zijn van afstanden of oppervlaktes moeten we altijd gebruik maken van geprojecteerde CRS en geen geografische CRS. `GeoPandas` voert geometrische berekeningen altijd planair uit, terwijl geografische coördinaten eigenlijk sferische coördinaten zijn. Hierdoor zullen resultaten van dergelijke berekeningen niet correct zijn. Indien er toch gebruik gemaakt wordt van geografische coördinaten is een naberekening vereist.