GeoPandas (visualisatie): Choropleten

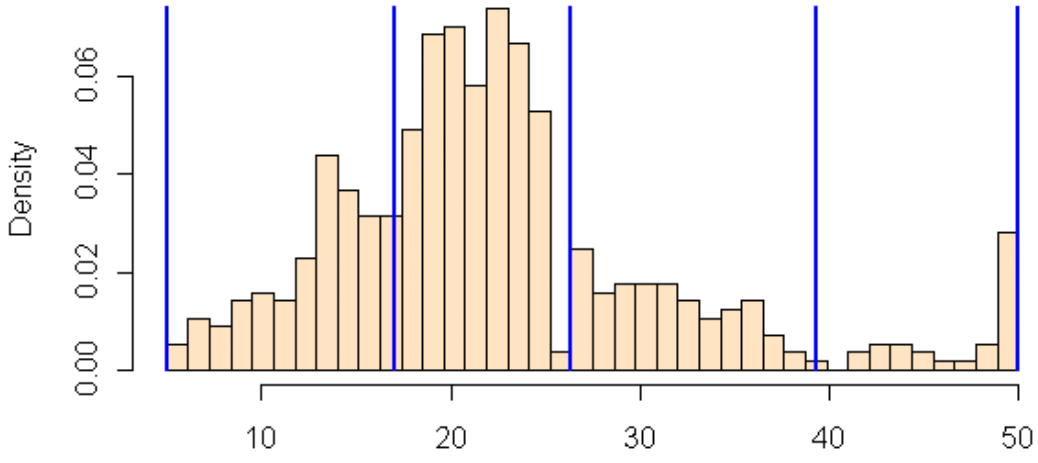
Dr. Cornelis Stal

April 28, 2022

1 Choropleet classificatie schema's van PySAL voor datavisualisaties met GeoPandas

PySAL is een bibliotheek voor ruimtelijke data-analyse, waarbinnen verschillende paketten opgenomen zijn voor met snelle algoritmes voor verschillende doelstellingen. Deze betreffen onder meer exploratieve data-anayse, analyse van ruimtelijke ongelijkheid, netwerkanalyse, studie van tijd-ruimtelijke patronen, enzovoorts.

Deze bibliotheek wordt gebruikt wanneer we bijvoorbeeld vanuit GeoPandas plots willen maken waar kleuren moeten overeenstemmen met een bepaalde klasse van numerieke waarde (choroploeetkaarten). Er zijn veel manieren om dergerlijke continue numerieke waarden te gaan classificeren en in te delen in zogenaamde *bins*. Binnen PySAL zijn er verschillende methoden bneschikbaar om dergelijke indelingen te maken, waarbij de distributie en het aantal *bins* een grote rol spelen.



Als we bijvoorbeeld een kaart willen maken van de gemiddelde jaartemperatuur voor 20 landen, waarbij de temperatuur gelegen is tussen 5°C en 25°C, kunnen we de volgende technieken gebruiken om deze data in te delen in 4 *bins*:

- Kwantielen:
  - Verdeelt de data in gelijke aantallen landen, dus telkens 20/4=5 landen per *bin*.
- Gelijke intervallen
  - Verdeelt de data in gelijke intervallen, dus telkens (25-5)/4=5°C per *bin*.
- Natural Breaks (Fischer Jenks)
  - Dit algoritme zal proberen de data op te splitsen in clusters met min of meer gelijkmatige en natuurlijke spreiding. Het aantal *bins* is afhankelijk van het aantal observaties binnen een gegeven interval.

We starten met het importeren van de vereiste bibliotheken en stellen de afmetingen van de figuren in:

[ ]:
```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
import mapclassify

plt.rcParams['figure.figsize'] = [15, 5]
```

1.1 Visualisatie van Vlaamse bevolkingsdichtheid

We hernemen de data met Vlaamse gemeentegrenzen en bevolkingsstatistieken, en koppelen beide datasets. Op basis van deze data berekenen we de bevolkingsdichtheid (aantal inwoners per hectare)

[ ]:
```
refgem = gpd.read_file('data/refgem_2018.shp')
pop = pd.read_csv('data/inwoners_2020.csv')
refgem.NISCODE = refgem.NISCODE.astype(int)
refgem = refgem.merge(pop, left_on='NISCODE', right_on='NIS')
refgem['DICHTHEID'] = refgem['INWONERS'] / (refgem['OPPERVL'] / 10000)

print('Observaties, Attributen:',refgem.shape)
refgem.head()
```

Voordat we overgaan tot het cartografische voorstellen van de bevolkingsdichtheid, is het goed te kijken naar de distrubitie van deze data. We gebruiken hiervoor een histogram:

[ ]:
```
refgem['DICHTHEID'].hist(bins=20)
plt.xlabel('DICHTHEID\nAantal inwoners per hectate per gemeente')
plt.ylabel('Aantal gemeenten')
plt.title('Distributie van de bevolkingsdichtheid in Vlaanderen')
plt.show()
```

Laten we eens kijken wat er gebeurt als we deze data plotten zonder een classificatie-schema.

[ ]:
```
refgem.plot(column='DICHTHEID', cmap='OrRd', edgecolor='gray', legend=True)
```

Alle 300 Vlaamse gemeenten zijn zijn ingekleurd volgens een wit-naar-rood gradiënt. Voor het menselijk oog zal het moeilijk zijn om verschillende gemeenten van elkaar te onderscheiden. Het

verschil tussen bijvoorbeeld Oudsbergen (2.02 inwoners per hectare) en Peer (1.88 inwoners per hectare) is bijvoorbeeld visueel niet (of toch moeilijk) te maken.

We zullen daarom werken met kleuren-*bins*.

> **Opmerking:** de waarden voor Oudsbergen en Peer verkrijgen we met behulp van `refgem.loc[refgem['NAAM'] == 'Oudsbergen']` en `refgem.loc[refgem['NAAM'] == 'Peer']`

1.2 Dataclassificatie

Het `scheme`-argument van de `GeoPandas.plot()`-methode laat toe om een bepaalde methode te kiezen voor numerieke data. In sommige gevallen is het gebruik van de `mapclassify`-bibliotheek vereist. De volgende methoden zijn beschikbaar:

- `BoxPlot`;
- `EqualInterval`;
- `FisherJenks`;
- `FisherJenksSampled`;
- `HeadTailBreaks`;
- `JenksCaspall`;
- `JenksCaspallForced`;
- `JenksCaspallSampled`;
- `MaxP`;
- `MaximumBreaks`;
- `NaturalBreaks`;
- `Quantiles`;
- `Percentiles`;
- `StdMean`;
- `UserDefined`.

Een aantal van deze methoden zijn ook terug te vinden in Desktop GIS omgevingen, zoals QGIS en de visualisatie van 'Graduated Symbols'.

Sommige methoden laten toe om expliciete ondergrenzen of bovengrenzen te definieren die niet noodzakelijkerwijze dezelfde zijn als de minima en maxima van een dataset. Dit is vooral handig wanneer de distributie van de data zeer schreef is.

Kwantielen

Met behulp van kwantielen kunnen aantrekkelijke kaarten gemaakt worden die een gelijk aantal waarnemingen in elke klasse plaatsen: als we bijvoorbeeld 30 provincies en 6 gegevensklassen hebt, hebben we 5 provincies in elke klasse. Het probleem met kwantielen is dat we klassen kunnen krijgen met heel verschillende numerieke bereiken (bijvoorbeeld 1-4, 4-9, 9-250).

Als we de data opdelen in 3 klassen geeft dit een duidelijk zicht op de verstedelijking in Vlaanderen:

[ ]:
```
refgem.plot(column='DICHTHEID', scheme='quantiles', k=3, cmap='OrRd',␣
↪edgecolor='gray', legend=True)
```

Het opdeling in twee klassen maakt de opdeling tussen stedelijk- en landelijk gebied nog beter zichtbaar:

[ ]:
```
refgem.plot(column='DICHTHEID', scheme='quantiles', k=2, cmap='OrRd',␣
↪edgecolor='gray', legend=True)
```

Gelijke intervallen

Bij gelijke intervallen worden de gegevens verdeelt in gelijke grootteklassen (bijv. 0-10, 10-20, 20-30, enz.). Dit werkt het beste op gegevens die over het algemeen over het hele bereik zijn verspreid.

> **Opmerking:** vermijd gelijke intervallen als de gegevens naar één kant scheef staan of als er uitschieters aanwezig zijn. Dit zal in onderstaand voorbeeld duidelijk worden.

[ ]:
```
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize = [15, 10])
refgem.plot(ax=ax1, column='DICHTHEID', scheme='equal_interval', k=4,
    cmap='OrRd', legend=True, edgecolor='gray')
refgem.plot(ax=ax2, column='DICHTHEID', scheme='equal_interval', k=12,
    cmap='OrRd', legend=True, edgecolor='gray')
ax1.set_axis_off()
ax1.set_title('Bevolkingsdichtheid met 4 klassen')
ax2.set_axis_off()
ax2.set_title('Bevolkingsdichtheid met 12 klassen')
plt.show()
```

Natural breaks

'Natural breals' is een soort "optimaal" classificatieschema dat klasse-onderbrekingen vindt die de variantie binnen de klassen minimaliseren en verschillen tussen klassen maximaliseren. Een nadeel van deze benadering is dat elke dataset een unieke classificatie-oplossing genereert, en als we kaarten tussen kaarten moeten vergelijken, zoals in een atlas of een reeks (bijv. één kaart elk voor 1980, 1990, 2000), willen we vaak één enkel schema dat op alle kaarten kan worden toegepast.

[ ]:
```
refgem.plot(column='DICHTHEID', scheme='natural_breaks', k=3, cmap='OrRd',␣
↪edgecolor='gray', legend=True)
```

Boxplot

Tot slot demonstreren we de werking van de `Boxplot`-classifier:

[ ]:
```
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize = [15, 10])
refgem.plot(ax=ax1, column='DICHTHEID', scheme='quantiles', k=6,
    cmap='OrRd', legend=True, edgecolor='gray')
refgem.plot(ax=ax2, column='DICHTHEID', scheme='boxplot',
    cmap='OrRd', legend=True, edgecolor='gray')
ax1.set_axis_off()
ax1.set_title('Bevolkingsdichtheid volgens interkwartielafstanden')
ax2.set_axis_off()
ax2.set_title('Bevolkingsdichtheid volgens boxplot opdeling')
plt.show()
```

1.3 Evaluatie van de classificatie

Verdeling van de klassen bestuderen

Op basis van de histogram van de data en enkele hierboven gedemonstreerde kaartvoorbeelden bleek al dat we voor de juiste indeling van de aanwezige waarden een goed inzicht nodig hebben in de distirbutie van de data. Hernemen we de `natural_breaks`-classifier met 4 klassen. De grenswaarden van deze opdeling kunnen we eenvoudig raadplegen:

[ ]:
```
ax = refgem.plot(column='DICHTHEID', scheme='NaturalBreaks', k=4, cmap='OrRd',␣
↪edgecolor='gray', legend=True)
labels = [t.get_text() for t in ax.get_legend().get_texts()]
labels
```

De ongelijke verdeling van gemeenten binnen deze klassen is visueel wel zichtbaar, maar we kunnen ook het aantal elementen per klasse raadpleten:

[ ]:
```
nb = mapclassify.NaturalBreaks(refgem.DICHTHEID, k=4)
nb
```

Een evenwichtige verdeling over de verschillende klasse krijgen we uiteraard wel met de `BoxPlot`-classifier:

[ ]:
```
bp = mapclassify.BoxPlot(refgem.DICHTHEID)
bp
```

Eigen klassen aanmaken

Naast het gebruik van enkele voorgedefineerde methodes kunnen we ook onze eigen klassenindeling aanmaken. Het `mapclassify.UserDefined`-object neemt nietvoor twee attributen, namelijk de data zelf, alsook een lijst met waarden van bovengrenzen:

[ ]:
```
bins = [5, 50]
data = refgem['DICHTHEID']
ud = mapclassify.UserDefined(data, bins)
ud.plot(refgem)
```

De verdeling van de bovenstaande klassificatie is ziet er als volgt uit:

[ ]:
```
ud
```

1.4 Andere classificatieschema's in PySAL en categorische data

Geopandas bevat alleen de meest gebruikte classificaties in PySAL. Om de andere te gebruiken, moeten we ze als extra kolommen aan de GeoDataFrame toevoegen.

> Het max-p-algoritme bepaalt het aantal regio's (p) endogeen op basis van een reeks gebieden, een matrix van attributen op elk gebied en een drempelwaarde voor de ondergrens. Deze ondergrens definieert de minimumgrens die een variabele voor elke regio moet bereiken; een beperking kan bijvoorbeeld de minimale populatie zijn die elke regio

moet hebben. max-p legt verder een contiguïteitsbeperking op voor de gebieden binnen regio's.

[ ]:
```
def max_p(values, k):
    """
    Gegeven is een lijst met waarden en `k` bins,
    geeft een lijst met nummers terug met de Maximum P *bin*.
    """
    binning = mapclassify.MaxP(values, k=k)
    return binning.yb

refgem['Max_P'] = max_p(refgem['DICHTHEID'].values, k=5)
refgem[['NAAM_x', 'Max_P']].head()
```

[ ]:
```
refgem.plot(column='Max_P', cmap='OrRd', edgecolor='gray', categorical=True,␣
 ↪legend=True)
```

We kunnen de legende verder opmaken met behulp van het `legend_kwds`-argument. De parameters die aan dit argument meegegeven kunnen worden zijn dezelfde als voor de legende bij een standaard `matplotlib.pyplot`-object.

> Voor meer informatie over de argumenten voor de legende en het `legend_kwds`-argument verwijzen we naar de documentatie van matplotlib.

[ ]:
```
refgem['SHORTNIS'] = refgem['NIS'].astype(str).str[:1]
replaceProv = {'1': 'Antwerpen', '2': 'Vlaams-Brabant',
    '3': 'West-Vlaanderen', '4': 'Oost-Vlaanderen', '7': 'Limburg'}
refgem['PROV'] = refgem['SHORTNIS'].replace(replaceProv)
ax = refgem.plot(column='PROV', categorical=True, legend=True,
    legend_kwds={'loc': 'center left', 'bbox_to_anchor':(1,0.5),
        'title': 'Vlaamse provincies'})
```

1.5 Numerieke labels opmaken

Een belangrijke parameter voor het `legend_kwds`-argument is de `fmt`-parameter. Hiermee wordt het formaat van de numerieke labels opgemaakt:

[ ]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, ncols=1, sharex=True, sharey=True,␣
 ↪figsize = [15, 10])
refgem.plot(ax=ax1, column='DICHTHEID', scheme='QUANTILES', k=4,
    cmap='YlOrRd', legend=True, edgecolor='gray',
    legend_kwds={'loc': 'center left', 'bbox_to_anchor':(1,0.5),
        'title': 'Standaard opmaak'})
refgem.plot(ax=ax2, column='DICHTHEID', scheme='QUANTILES', k=4,
    cmap='YlOrRd', legend=True, edgecolor='gray',
    legend_kwds={'loc': 'center left', 'bbox_to_anchor':(1,0.5),
         'fmt':"{:.0f}", 'title': 'Zonder cijfers na de komma'})
refgem.plot(ax=ax3, column='DICHTHEID', scheme='QUANTILES', k=4,
```

```
    cmap='YlOrRd', legend=True, edgecolor='gray',
    legend_kwds={'loc': 'center left', 'bbox_to_anchor':(1,0.5),
         'fmt':"{:.4f}", 'title': '4 cijfersn a de komma'})
ax1.set_axis_off()
ax2.set_axis_off()
ax3.set_axis_off()
```

Eigen labels aanmaken Tot slot geven we nog de mogelijkheid om zelf labels te definiëren, ook wanneer we geen gebruik maken van categorische variabelen. De `labels`-parameter binnen het `legend_kwds`-argument laat toe om dergelijke eigen waarden mee te geven in een lijst. Het is hierbij uiteraard van belang dat het aantal elementen in de lijst gelijk is aan het aantal klassen van de 'classifier', wat dus (meestal) overeen komt met de waarde `k`:

[ ]:
```
refgem.plot(column='DICHTHEID', scheme='quantiles', k=5,
    cmap='YlOrRd', legend=True, edgecolor='gray',
    legend_kwds={'loc': 'center left', 'bbox_to_anchor':(1,0.5), 'labels':
 ↪['laag', '...', 'medium', '...', 'hoog']})
```