Pandas: Werken met tekst

Dr. Cornelis Stal

April 28, 2022

[ ]:
```
import pandas as pd
```

1 Werken met tekst?

Deze tutorial is een vertaling van de *Pandas Tutorial* op https://pandas.pydata.org/pandas-docs/stable/getting_started/.

Data: voor deze tutorial zullen we gebruikmaken van de jaarlijkse vastgoedcijfers die bijgehouden en beschikbaar gemaakt worden door Statbel via deze link. We richten ons hierbij meer bepaald op de cijfers van verkoop van onroerende goederen (N) per jaar 2010-2021 voor de individuele gemeenten. Aanvullend zullen we wat handelingen uitvoeren op een manueel toe te voegen set van basisgegevens over de Vlaamse provincies.

Hier hier om de data te downloaden.

[ ]:
```
statbel = 'https://statbel.fgov.be/sites/default/files/files/\
    documents/Bouwen%20%26%20wonen/2.1%20Vastgoedprijzen/NL_immo_jaar.xlsx'

vastgoed = pd.read_excel(statbel, sheet_name='Per gemeente', skiprows=2,
    usecols=['refnis', 'lokaliteit', 'jaar', 'aantal transacties',
    'mediaan prijs(€)', 'eerste kwartiel prijs(€)','derde kwartiel prijs(€)'])
vastgoed['refnis'] = vastgoed['refnis'].astype(str)
vastgoed.head()

prov = pd.DataFrame({
    'hoofdstad': ['Brugge', 'Gent', 'Antwerpen', 'Leuven', 'Hasselt'],
    'provincie': ['West-Vlaanderen, West-Flanders',
    'Oost-Vlaanderen, East-Flanders', 'Antwerpen, Antwerp',
    'Vlaams-Brabant, Flemish-Brabant', 'Limburg, Limburg'],
'provnis': ['3', '4', '1', '2', '7']})
```

1.1 Methodes voor teskst op kolommen

De plaatsnamen staan hier in hoofdletters. Laten we nu eens enkel de eerste letter (of eerste letter na een witruimte) met een hoofdletter laten beginnen:

[ ]:
```
vastgoed["lokaliteit"] = vastgoed["lokaliteit"].str.capitalize()
vastgoed["lokaliteit"]
```

Om deze handeling uit te voeren selecteren we eerst de gewenste kolom (`lokaliteit`). Aan deze selectie voegen we de `str`-'accessor' toe. Tot slot voeren we de `capitelize()`-methode uit. Zoals bij statistische or rekenkundige handelingen worden rijen element per element verwerkt.

Zoals bij `datetime`-objecten in een tijdreeks, waar methodes aan kunnen worden gesproken met de `dt`-'accessor', zijn een groot aantal specialistische methodes voor tekst beschikbaar via de `str`-'accessor'. Over het algemeen komen de namen van deze methodes overeen met de ingebouwde methodes voor tekst in Python. Grote verschil met deze ingebouwde methodes is dat teksten bij `pandas` niet uitgevoerd worden op enkele elementen, maar element per element worden uitgevoerd voor hele collecties in kolommen.

We voegen een nieuwe kolom `provnis` toe aan de vastgoed `DataFrame`, waarbij het eerste karakter uit de kolom `refnis` wordt opgeslagen. Deze nieuwe waarde is een aanduiding van de provincie:

[ ]:
```
vastgoed['provnis'] = vastgoed['refnis'].str[0]
vastgoed.head()
```

[ ]:
```
prov["provincie"].str.split(",")
```

Door gebruik te maken van de `Series.str.split()`-methode kriojgen we een lijst terug met in dit geval twee elementen: de Nederlandstalige naam van iedere Vlaamse provincie en de Engelstalige naam van iedere provincie. In bovenstaande voorbeeld werden beide zaken oorspronkelijk gescheiden door een komma. We hebben nu dus de elementen voor en na de komma gescheiden.

[ ]:
```
prov["provincie"] = prov["provincie"].str.split(",").str.get(0)
prov.head()
```

Zoals we gezien hebben bij de algemene tutorials over Python, kunnen individuele elementen in een lijst aanspreken door middel van de index. Wanneer we dit wensen te doen met een `Series`-object, maken we opnieuw gebruik van de `str`-'accessor. Vervolgens spreken we individuele elementen aan met de `get()`-methode. In ons geval zijn we enkel geïnteresseerd in de Nederlandstalige naam van een provincie, dus geven we een index `0` mee aan de `Series.str.get()`-methode. Merk bij bovenstaande code op dat we meerdere handelingen in één enkele regel code verwerkt hebben.

> **Gebruikshandleiding:** meer informatie over de extractie van delen van tekst kan gevonden worden over de sectie rond het splitsen en vervangen van strings in de handleiding.

1.2 Samenvoegen van de tabellen

Het samenvoegen van twee tabellen is reeds behandeld in de tutorial over het combineren van tabellen. Deze operatie voeren we nogmaals uit met onze demodata:

[ ]:
```
merged = pd.merge(vastgoed, prov, how='left', on='provnis')
merged.head()
```

Deze koppeling wordt uitgevoerd op basis van corresponderende waarden in de kolom `provnis`. Een gemeenschappelijke naam van het koppelveld is echter niet vereist. Veronderstellen we nu dat we enkel geïnteresseerd zijn in de verschillende statistieken voor de provinciehoofdsteden:

```
[ ]: merged = pd.merge(vastgoed, prov, how='right',
         left_on='lokaliteit', right_on='hoofdstad')
     merged.head()
```

We koppelen de hoofdsteden en provincienamen (rechts) aan de `DataFrame` met vastgoed-data. Vermits iedere provinciehoofdstad voorzien zal moeten worden van de corresponderende statistieken, voeren we een 'right join' uit. De kolom met plaatsnamen in de vastgoed-dataset (`left_on='lokaliteit'`) en de kolom met de namen van de provinciehoofdsteden (`right_on='hoofdstad'`) worden gebruikt als koppelvelden.

Op dit resultaat kunnen we eenvoudig verdere selecties maken, bijvoorbeeld om verder in te zoomen op de stad Gent en Brugge:

```
[ ]: merged[merged['provincie'].str.contains('Vlaanderen')]
```

De `Series.str.contains()`-methode controleert in iedere rij in de kolom `provincie` op waarden waarin de string `Vlaanderen` vervat zit en zal een resultaat `True` (`Vlaanderen` is onderdeel van de provincienaam) of `False` (`Vlaanderen` is geen onderdeel van de provincienaam) retourneren. Dit resultaat wordt vervolgens gebruikt om een subset aan te maken via een conditionele (booleaanse) indexering, zoals we gezien hebben in de tutorial over subsets. Vermits we voor 11 jaar data hebben, en vermist Brugge en Gent de hoofdstad zijn van respectievelijk West-Vlaanderen en Oost-Vlaanderen, krijgen we 22 resultaten terug.

> **Opmerking:** er worden zeer krachtige methodes ondersteund voor de extractie van strings, zoals de bovenstaande `Series.str.contains()`-methode. Een ander mooi voorbeeld is de `Series.str.extract()`-methode, waarbij selecties kunnen worden aangemaakt op basis van reguliere expressies. Dit thema ligt echter buiten de inhoud van deze tutorials.
>
> **Opmerking:** het is belangrijk te weten dat, zoals bij eerder besproken expressies op strings, de expressie hoofdlettergevoelig is. In bovenstaande voorbeeld zal bijvoorbeeld `.str.contains('vlaanderen')` geen resultaten terug geven. Het veranderen van hoofdletters naar kleine letters (`lower()`) of van kleine letters naar hoofdletters (`upper()`) kan hierbij helpen. Ook is Python streng op witruimten, waardoor het verwijderen van spaties nodig kan zijn aan het begin of einde van een string (`rstrip()`, `lstrip`, …).

Stel dat we ons de vraag zouden stellen welke provincie de kortste naam heeft:

```
[ ]: prov['provincie'].str.len()
```

Om de kortste naam te krijgen achterhalen we eerst de lengte van iedere provincienaam op basis van de kolom `provincie`. Vanuit de `pandas string`-methodes kunnen we meer bepaald de `Series.str.len()`-functie toepassen om de lengte van iedere individuele naam terug te krijgen (element per element ofwel 'element-wise').

```
[ ]: prov["provincie"].str.len().idxmin()
```

Vervolgens willen we weten welke provincie de kortste naam heeft, en willen we hiervoor de naam verkrijgen. De `idxmin()`-methode laat een dergelijke operatie toe, met dien verstande dat deze

methode werkt op een verzameling gehele getallen. We hoeven hier dus geen gebruik te maken van de `str`-'accessor'.

> **Opmerking:** mochten we geïnteresseerd zijn in de langste naam, zouden we de methode `idxmax()` moeten gebruiken. In bovenstaande geval komt deze waarde twee keer voor, maar enkel het eerste resultaat zal worden gegeven.

[ ]:
```
prov.loc[prov["provincie"].str.len().idxmin(), "provincie"]
```

Gebaseerd op de index van de rij (`4`) en de kolom (`provincie`) kunnen we met de `loc`-operator tot slot een selectie aanmaken. Dit resulteert in een waarde `Limburg`.

We maken nu een nieuwe kolom `prov` aan voor iedere provincie de corresponderende `provnis`-waarde. Deze komt overeen met de eerste waarde van de NIS-code:

[ ]:
```
vastgoed['prov'] = vastgoed['provnis'].replace({
    1: 'Antwerpen', 2: 'Vlaams-Brabant', 3: 'West-Vlaanderen',
    4: 'Oost-Vlaanderen', 7: 'Limburg'})
vastgoed['prov']
```

Vermits de `replace()`-methode geen string-methode is, beschikken we hiermee met een eenvoudige manier om gebruik te maken van vocabulaires of bibliotheken om bepaalde waarden te vertalen. Deze methode vereist een `dictionary` om de vertaling uit te voeren volgens de syntax `{oude_waarde : nieuwe_waarde}`.

> **Opmerking:** het is technisch gezien ook mogelijk om gewoon de `replace()`-methode te gebruiken uit de standaard Python bibliotheek. Ook hiermee kunnen we bepaalde waarden of karakters vervangen. In tegenstelling tot de `replace()`-methode uit de `pandas`-bibliotheek zullen we dan wel iedere mogelijke vervanging regel per regel moeten programmeren:
>
> ```
> vastgoed['prov'] = vastgoed['provnis'].replace('Antwerpen', "1")
> vastgoed['prov'] = vastgoed['provnis'].replace('Vlaams-Brabant', "2")
> ...
> ```
>
> Dit is uiteraard erg omslachtig en zeer gevoelig voor fouten.

1.3 Te onthouden:

- String methoden zijn beschikbaar via de `str`-'accessor';
- String methoden werken element per element ofwel *element-wise* en kunnen worden gebruikt voor conditionele indexering;
- De `replace()`-methode biedt een eenvoudige manier om waardes om te zetten volgens een gegeven `dictionary`.

> **Gebruikshandleiding:** voor meer informatie het werken met tekst verwijzen we naar de handleiding.