Pandas: Werken met tijdreeksen

Dr. Cornelis Stal

April 28, 2022

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import requests
```

1 Werken met tijdreeksen

Tot zover hebben we gezien hoe we data kunnen importeren en verwerken als tekst, gehele getallen en decimale getallen. We hebben terloops ook al een keer een conversie uitgevoerd van een numerieke waarde naar een jaartal, maar verder geen aandacht besteed aan de mogelijkheden van tijd. In deze tutorial zullen we verder ingaan op het werken met tijdreeksen.

> 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 gebruik maken van waterstandmetingen die beschikbaar gemaakt worden door de Vlaamse overheid via Waterinfo.be. We zullen werken met data van 3 meetstations. De bijbehorende URLs geven toegang tot de real-time data gemeten aan het corresponderende station:
>
> - Prosperpolder/Zeeschelde (56088010, ZES01A-1066);
> - Wintam Sluis Afwaarts DVW tij/Zeeschelde (56055010, BS-WIN-AFW-1095);
> - Sluis Merelbeke (102372010, OW006-AFW-1073).

De data worden opgehaald met de `requests`-bibliotheek met parameters die in de documentatie die op de website beschikbaar is.

[ ]:
```
def getData(loc):
    url = 'https://www.waterinfo.be/tsmhic/KiWIS/KiWIS'
    params = {'service': 'kisters', 'type': 'queryServices',
            'request': 'getTimeSeriesValues', 'datasource': '4',
            'format': 'json', 'period': 'P3D', 'ts_id': loc[1],
            'to': '2022-01-13T00:00:+01:00'}
    r = requests.get(url, params=params)
    tide = pd.DataFrame(data=r.json()[0]['data'], columns=['datetime', 'tide'])
    tide['location'] = loc[0]
    return tide

locList = [['properpolder', '56088010'], ['wintam', '56055010'],
    ['merelbeke', '102372010']]
```

```
tide = getData(locList[0])
tide = tide.append(getData(locList[1]))
tide = tide.append(getData(locList[2]))
tide.head()
```

De kolom `datetime` bevat tekstwaarden waarop we nog geen handelingen uit kunnen voeren die we zouden mogen verwachten op een tijdsaanduiding. In de volgende paragraaf gaan we bekijken hoe we de conversie uit kunnen voeren en wekne handelingen we op tijdobjecten uit kunnen voeren:

[ ]:
```
tide.info()
```

We controleren ook nog eens of er daadwerkelijk data van 3 stations is ingeladen:

[ ]:
```
tide['location'].unique()
```

1.1 Het `pandas` `datetime`-objecttype

De ingeladen van de kolom `datetime` resulteerde dus in een aantal tekstwaarden, maar een conversie naar een tijdsaanduiding is vereist om verder te kunnen werken:

[ ]:
```
tide["datetime"] = pd.to_datetime(tide["datetime"])
tide["datetime"].head()
```

In eerste instatie worden tijdsaanduidingen zonder verdere handelingen altijd ingeladen als het meest geschikte datatype (meestal string). Hierdoor zijn operaties die specifiek zijn voor tijdsaanduidingen niet mogelijk, zoals het ophalen van een jaar, dag van de week, … Door gebruik te maken van de `to_datetime()`-functie zal `pandas` de strings interpreteren en converteren naar een daadwerkelijke tijdsaanduiding (zoals `datetime64[ns, UTC]` of `datetime64[ns, pytz.FixedOffset(60)]` zoals in ons geval). In `pandas` worden tijdsobjecten op gelijkaardige manier aangemaakt als objecten uit de standaard `datetime.datetime` bibliotheek, namelijk via `pandas.Timestamp`.

> **Opmerking:** aangezien tijdsaanduidingen vaak vervat zitten in datasets, zijn in `pandas` verschillende de input-functies (zoals `pandas.read_csv()` en `pandas.read_json()`) een optie geïmplementeerd waarmee een bepaalde kolomen direct al omgezet kan worden naar `datetime`-objecten. We maken hiervoor gebruik van de `parse_dates`-parameter. Bovenstaande code had dan als volgt geweest:
>
> ```
> tide = pd.DataFrame(data=r.json()[0]['data'], columns=['datetime',
> 'tide'], parse_dates=["datetime"])
> ```

Vraag stelt zich nu wel wat dergelijke `pandas.Timestamp`-objecten zo nuttig maakt. We zullen dit illustreren aan de hand van enkele voorbeelden, te starten met het zoeken naar het eerste en laatste meetmoment:

[ ]:
```
tide["datetime"].min(), tide["datetime"].max()
```

Technisch gezien hadden we bovenstaande opvraging ook uit kunnen voeren op `string`-objecten. Dankzij `pandas.Timestamp` kunnen we echter ook berekeningen op deze waarden uitvoeren. Zo

kunnnen we deze resultaten daadwerkelijk interpreteren als tijdsaanduingen, en bijvoorbeeld het tijdsverschil tussen beide waarden berekenen:

[ ]:
```
tide["datetime"].max() - tide["datetime"].min()
```

Het resultaat van bovenstaande is een `pandas.Timedelta`-object dat vergelijkbaar is met een `datetime.timedelta`-objecxt uit de standaard Python bibliotheek. Dit object definieerd een zekere tijdsduur.

> **Gebruikshandleiding:** de verschillende concepten die betrekking hebben op tijd worden nader omschreven in de sectie over 'time related concepts' in de handleiding.

Met een `pandas.Timestamp`-object kunnen we bepaalde elementen uit de tijdsaanduiding ophalen:

[ ]:
```
tide["hour"] = tide["datetime"].dt.hour
tide.head()
```

Dankzij het gebruik van `pandas.Timestamp`-objecten voor tijdsaanduiding kunnen we vele tijdgerelateerde eigenschappen bevragen met `pandas`, zoals de maand (`month`), het jaar (`year`), de dag van het jaar (`weekofyear`), het seizoen (`quarter`), … Al deze eigenschappen zijn toegankleijk via de `dt`-'accessor'.

> **Gebruikshandleiding:** een overzicht van de beschikbare dataeigenschappen wordt gegeven de volgende overzichtstabel voor datum- en tijdscomponenten. Voor meer informatie over de `dt`-'accessor' om met de eigenschappen van tijdsobjecten te werken kan hier in een afzonderlijke sectie van de handleiding gevonden worden.

We wensen nu de gemiddelde waterstand per uur te krijgen voor iedere meetlocatie:

[ ]:
```
tide.groupby([tide["datetime"].dt.hour, "location"])["tide"].mean()
```

Ter herinnering verwijzen we nog eens terug naar de tutorial over het berekenen van statistieken voor informati over het `split-apply-combine`-patroon dat we gebruiken voor de `groupby()`-methode.

In bovenstaande voorbeeld willen we een bepaalde statistiek berekenen (bijvoorbeeld de gemiddelde waterstand uit de kolom `tide`) voor **ieder uur** en voor **iedere meetlocatie**. Om te groeperen voor ieder uur gebruiken we uit het `pandas Timestamp`-object de `datetime`-eigenschap `hour` (voor een 24-uurs dagindeling). Deze eigenschap is toegankelijk via de `dt`-'accessor'. Het groeperen over zowel de uren als de locaties resulteert in een gemiddelde voor iedere unieke combinatie.

> **Opmerking:** in deze tutorial werken we op een zeer beperkte tijdreeks. We kunnen deze data dus niet gebruiken om iets zonvols te zeggen over de lange termijn.

We kunnen de gegroupeerde waarden (gemiddelde waarden per uur) ook visualiseren in een grafie:

[ ]:
```
fig, axs = plt.subplots(figsize=(12, 4))
tide.groupby(tide["datetime"].dt.hour)["tide"].mean().plot(ax=axs)
plt.xlabel("Uren sinds het begin van de meting")
plt.ylabel("Hoogte (m, TAW)")
```

1.2 `datetime` als index

In de tutorial over het herstructureren van data hebben we een kruistabel gemaakt met behulp van de `pivot()`-methode. Hiermee konden we data in een tabel herstructureren, waardoor alle hoogtewaarden per locatie in een nieuwe kolom worden verwerkt:

[ ]:
```
tide2 = tide.pivot(index="datetime", columns="location", values="tide")
tide2.head()
```

> **Opmerking:** door het aanmaken van kruistabellen met de `pivot()`-methode zijn de waarden in de kolom `datetime` omgeet tot indices van de tabel. Over het algemeen zullen we kolommen aanduiden als index met behulp van de `set_index()`-methode.

Het gebruik van een kolom met `pandas.Timestamp`-waarden (zoals `DatetimeIndex`) geeft toegang tot enkele zeer krachtige functies. De eigenschappen die we tot zover aan moesten spreken via de `dt`-'accessor' kunnen we nu bijvoorbeeld rechtstreeks aanspreken vanuit de index:

[ ]:
```
tide2.index.hour, tide2.index.minute
```

Bijkomende voordelen zijn de eenvoud waarmee subsets aangemaakt kunnen worden of het gemak waarmee de tijdschaal aangepast kan worden in een grafiek. We kunenn nu bijvoorbeeld heel eenvoudig een plot maken van het gemeten waterpeil voor de locatie `merelbeke` voor 12 januari 2022:

[ ]:
```
tide2['2022-01-12':'2022-01-13']['merelbeke'].plot()
```

De opgegeven string is automatisch geïnterpreteerd als een `datetime`-waarde, resulterend in een selectie van de gehele dataset als een `DatetimeIndex`-object.

> **Gebruikshandleiding:** voor meer informatie over `DatetimeIndex` en het maken van subsets op basis van tijd en met behulp van tekst verwijzen we naar de sectie over tijdreeks-indexing in de handleiding.

1.3 Temporeel interval (frequentie) van een tijdreeks veranderen ('resampling')

Binnen de dataset hebben we te maken met een niet-gelijkmatige temporele resolutie:

- Merelbeke: 1 minuut
- Wintam: 5 minuten
- Prosperpolder: 10 minuten

Laten we de temporele resolutie nu eens gelijkstellen door de data gemeten in Merelbeke en Wintam te reduceren tot de resolutie van in de Prosperpolder, dus alle data met een resolutie van 10 minuten:

[ ]:
```
tideResample = tide2.resample(rule='10T').mean()
tideResample.head()
```

Een zeer krachtig instrument om toe te passen op tijdreeksen is de `resample()`-methode. Wanneer de `datetime` is ingesteld als index van de data, kunnen we met deze methode de frequentie van een gegeven dataset veranderen. In bovenstaand voorbeeld hebben we minuutdata veranderd in een waarde voor iedere 10 minuten.

De `resample()`-methode is qua werking gelijkaardig aan de `groupby()`-operatie:

- Het voorziet in een tijd-gebaseerde groepering op basis van een string. Met deze string bepalen we de nieuwe frequentie, zoals `M` voor maand of `5H` voor 5 uur, …;
- Het vereist een functie die bepaalde waarden statistisch laat samenvoegen, zoals `mean`, `max`, …

  Gebruikshandleiding: een overzicht van de aliassen die gebruikt kunnen worden voor het aanmaken van nieuwe tijdreeksen met een gegeven frequentie wordt gegeven in de sectie over 'offset aliases overview table' in de handleiding.

Nadat we de frequentie vastgelegd hebben voor een bepaalde tijdreeks, kunnen we deze achteraf nog raadplegen met de `freq`-attribuut:

[ ]:
```
tideResample.index.freq
```

Om af te sluiten maken we nog een grafiek met de waterstandsmetingen per uur voor de drie locaties:

[ ]:
```
tideResample.resample("H").mean().plot(style="-o", figsize=(10, 5));
```

Gebruikshandleiding: voor meer informatie over de kracht van de `resampling`-methode verwijzen we naar de gelijknamige sectie in de handleiding.

1.4 Te onthouden:

- Een geldige tekst met een datum en/of tijd kan worden omgezet tot een `datetime`-object met behulp van de `to_datetime`-function, of als onderdeel van de functie waarbij de data worden ingelezen;
- `datetime`-objecten in `pandas` ondersteunen berekeningen, logische operaties en de extractie van tijd-gerelateerde eigenschappen. Hiervoor gebruiken we de `dt`-'accessor';
- Een `DatetimeIndex`-object bevat eveneens dergelijke tijd-gerelateerde eigenschappen en ondersteunt daarnaast de eenvoudige extractie van subsets;
- De `resample()`-methode is een krachtig instrument om de temporele frequentie van een tijdreeks te veranderen.

Gebruikshandleiding: voor meer informatie tijdreeksen verwijzen we naar de handleiding.