

Geodatenanalyse 2

Termin: Big Data 1 - Modul 1

Umgang mit Datum und Zeit

Ca. 20-30 Minuten

Inhalt

- Das Problem mit Zeitreihen
- Datums- und Zeitformate
- Python's eingebaute Datums- und Zeitfunktionalität
- Der Datums-/Zeit Datentyp in NumPy
- Operationen mit Datum oder Zeit
- Abbildungen mit Datum/Zeit-Achsen

In [1]:

```
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
```

Was ist das Problem mit Zeitreihen?

Die Zahlensysteme des Datums

- Ein Datum besteht aus einem komplizierten System:
 - das Jahr hat 365 Tage, mit Schaltjahr 366 (alle 4 Jahre)
 - Die Tage variieren (28-31 Tage pro Monat)
 - Wochentage folgen dem 7-er System
 - Monate basieren auf der Basis von 12

Die Zahlensysteme der Zeit

- Die Zeit besteht aus einem gemischten System:
 - Sekunden und Minuten auf der Basis von 60
 - Stunden auf der Basis von 24

Welcher Speichertyp?

- Normale Datentypen wie z.B. *integer* oder *floats* werden generell für Berechnungen verwendet

- Man kann ein Datum und eine Zeit als *string* darstellen (z.B., "12.04.2021 12:33:44")
- Aber: Mit strings kann man keine Operationen (z.B. größer/kleiner Vergleiche) oder Berechnungen (z.B. Zeitdifferenz) durchführen

Fazit: Für die Zeitreihenanalyse brauchen wir einen eigenen Datentyp für Datum und Zeit!

```
In [2]: datetime = "19.04.2021 14:10:10"
type(datetime)
```

```
Out[2]: str
```

Was macht Excel?

Interne Umwandlung in eine Fließkommazahl (*float*):

	A	B
1	19/04/2021 06:00	44305.25
2	19/04/2021 12:00	44305.50
3	19/04/2021 18:00	44305.75
4		

- Die Ganzzahl ergibt die Tage (seit 1. Januar 1900)
- Die Fliesskommazahl ergibt die Zeiteinheit innerhalb des Tages

Fazit: Diese Strategie macht es etwas einfacher, aber nimmt nicht die Schwierigkeit aus dem Kalender!

Datum/Zeit-Formate

Das ISO 8601 Datums- und Zeitformat

Quelle: [Wikipedia](#)

Variable	Darstellung	Werte	Erläuterung
J	JJJJ	0000...9999	Jahr <ul style="list-style-type: none"> • kann auch negativ oder Null sein • kann nach Konvention verlängert werden
M	MM	01...12	Monat
w	ww	01...53	Woche des Jahres
T	T	1...7	Tag der Woche, Montag bis Sonntag
	TT	01...31	Tag des Monats
	TTT	001...366	Tag des Jahres
h	hh	00...24	Stunde, 24 nur in 24:00 als Endzeit
m	mm	00...59	Minute
s	ss	00...60	Sekunde, 60 nur als Schaltsekunde
f	f	(0...9)+	dezimale Bruchteile, in der Regel von Sekunden beliebiger Genauigkeit

Mehr dazu in auf der [ISO Webseite](#)

Weitere wichtige Datum/Zeit-Formate

- Das internationale Format (Jahr zuerst): YYYY-mm-dd HH:MM:SS
- Das internationale Format (Tag zuerst): dd/mm/YYYY HH:MM:SS
- Das USA Format (Monat zuerst): mm/dd/YYYY HH:MM:SS (ACHTUNG: Verwechslungsgefahr!)
- Das deutsche Format: dd.mm.YYYY HH:MM:SS

ACHTUNG: Daten haben oft ganz unterschiedliche und zum Teil auch vermischte Formate!

Python's eingebaute Datums- und Zeitfunktionalität

- Das *datetime*-Modul liefert Klassen zur Manipulation von Datums- und Zeitangaben.
- Während die Datums- und Zeitarithmetik unterstützt wird, liegt der Schwerpunkt der Implementierung auf der effizienten Attributextraktion für die Ausgabeformatierung und -manipulation.

Mehr Informationen im [Python Handbuch](#)

Eigenschaften des *date*-Objekts:

```
datetime.date(year, month, day)
```

```
In [3]:  
date = dt.date.today()  
date
```

```
Out[3]: datetime.date(2023, 4, 24)
```

```
In [4]:  
time = dt.time(hour=14, minute=15, second=16)  
time
```

```
Out[4]: datetime.time(14, 15, 16)
```

Syntax des *datetime*-Objekts:

```
datetime.datetime(year, month, day, hour=0, minute=0, second=0,  
microsecond=0, tzinfo=None)
```

```
In [5]:  
date = dt.datetime.today()  
date
```

```
Out[5]: datetime.datetime(2023, 4, 24, 18, 47, 24, 769974)
```

Import von vorformatierten Daten

- Oftmals müssen vorformatierte Daten in Python importiert werden
- Dafür gibt es die Funktion `strptime()` (p steht für *parsing*)

In [6]:

```
datum = "19.04.2021 14:33:12"
print(type(datum))

dt.datetime.strptime(datum, "%d.%m.%Y %H:%M:%S")

<class 'str'>
```

Out[6]: `datetime.datetime(2021, 4, 19, 14, 33, 12)`

Hier werden Schlüsselzeichen benötigt, welche den Zahlen ihren Platz zuweisen:

Directive	Description	Example Output
%a	Weekday as locale's abbreviated name.	Sun, Mon, ..., Sat (en_US) So, Mo, ..., Sa (de_DE)
%A	Weekday as locale's full name.	Sunday, Monday, ..., Saturday (en_US) Sonntag, Montag, ..., Samstag (de_DE)
%w	Weekday as a decimal number, where 0 is Sunday and 6 is Saturday.	0, 1, 2, 3, 4, 5, 6
%d	Day of the month as a zero-padded decimal number.	01, 02, ..., 31
%b	Month as locale's abbreviated name.	Jan, Feb, ..., Dec (en_US) Jan, Feb, ..., Dez (de_DE)
%B	Month as locale's full name.	January, February, ..., December (en_US) Januar, Februar, ..., Dezember (de_DE)
%m	Month as a zero-padded decimal number.	01, 02 ... 12
%y	Year without century as a zero-padded decimal number.	01, 02, ... 99
%Y	Year with century as a decimal number.	0001, 0002, ... , 9999
%H	Hour (24-hour clock) as a zero-padded decimal number.	01, 02, ... , 23
%I	Hour (12-hour clock) as a zero-padded decimal number.	01, 02, ... , 12
%p	Locale's equivalent of either AM or PM.	AM, PM (en_US) am, pm (de_DE)
%M	Minute as a zero-padded decimal number.	01, 02, ... , 59
%S	Second as a zero-padded decimal number.	01, 02, ... , 59
%f	Microsecond as a decimal number, zero-padded on the left.	000000, 000001, ..., 999999 Not applicable with time module.
%z	UTC offset in the form ±HHMM[SS] (empty string if the object is naive).	(empty), +0000, -0400, +1030
%Z	Time zone name (empty string if the object is naive).	(empty), UTC, IST, CST

Export in ein formatiertes Format

- Andersrum müssen Daten aus Python exportiert werden
- Dafür gibt es die Funktion `strftime()` (f steht für *formatting*)

In [7]:

```
date = dt.datetime.now()
string = date.strftime("%Y-%m-%d %H:%M:%S")
print(string)
print(type(string))
```

2023-04-24 18:47:24
<class 'str'>

Datums- und/oder Zeitrechnungen

- Manchmal muss ein Datum und/oder eine Zeit auch berechnet werden
- Dazu benötigt es ein eigenes Objekt

Eigenschaften des *timedelta*-Objekts:

```
datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0,
minutes=0, hours=0, weeks=0)¶
```

```
In [8]: new_date = date + dt.timedelta(days=53)
new_date
```

```
Out[8]: datetime.datetime(2023, 6, 16, 18, 47, 24, 789906)
```

Beispiel: Countdown bis zum nächsten Neujahr

```
In [9]: countdown = dt.datetime(2024, 1, 1, hour=0, minute=0, second=0) - dt.datetime.now()
countdown
```

```
Out[9]: datetime.timedelta(days=251, seconds=18755, microseconds=195144)
```

Frage: Was ist mit Zeitreihen ...?

Dafür brauchen wir wieder einen anderen Datentyp!

Der *datetime64[ns]* Datentyp in NumPy

- Das Datums-/Zeitobjekt von NumPy ist speziell für Zeitreihen entwickelt
- Damit lassen sich Vektoren bearbeiten

Mehr Informationen im [NumPy Handbuch](#)

```
In [10]: date = np.datetime64('2021-04-12')
print(date)

dates = np.array(['2007-07-13', '2006-01-13', '2010-08-13'], dtype='datetime64')
print(dates)
```

```
2021-04-12
['2007-07-13' '2006-01-13' '2010-08-13']
```

Rechnen mit Datum und Zeit

```
In [11]: np.datetime64('2021-07-19') - np.datetime64('2021-04-19')
```

```
Out[11]: numpy.timedelta64(91, 'D')
```

Was ist *timedelta64*?

Ein Datenobjekt für Datums- und Zeitdifferenzen!

Beispielrechnung:

```
In [12]: np.datetime64('2021-07-19') + np.timedelta64(45, 'D') + np.timedelta64(13, 'h')
```

```
Out[12]: numpy.datetime64('2021-09-02T13','h')
```

```
In [13]: datetime = np.datetime64('2021-07-19T12:33:35.456')
datetime
```

```
Out[13]: numpy.datetime64('2021-07-19T12:33:35.456')
```

```
In [14]: datetime - np.timedelta64(23, 'D') + np.timedelta64(11, 'h') - np.timedelta64(34,
```

```
Out[14]: numpy.datetime64('2021-06-26T22:59:35.456')
```

Zeitreihen erstellen

- Leider bietet *datetime* und *NumPy* nur begrenzte Funktionalität für das Erstellen von Zeitreihen
- Dazu braucht man dann Schleifen oder Iteratoren

```
In [15]: base = np.datetime64('2021-01-01')
array = np.array(base)
for i in range(365):
    array = np.append(array, base + np.timedelta64(i + 1, 'D'))

array
```

```
Out[15]: array(['2021-01-01', '2021-01-02', '2021-01-03', '2021-01-04',
               '2021-01-05', '2021-01-06', '2021-01-07', '2021-01-08',
               '2021-01-09', '2021-01-10', '2021-01-11', '2021-01-12',
               '2021-01-13', '2021-01-14', '2021-01-15', '2021-01-16',
               '2021-01-17', '2021-01-18', '2021-01-19', '2021-01-20',
               '2021-01-21', '2021-01-22', '2021-01-23', '2021-01-24',
               '2021-01-25', '2021-01-26', '2021-01-27', '2021-01-28',
               '2021-01-29', '2021-01-30', '2021-01-31', '2021-02-01',
               '2021-02-02', '2021-02-03', '2021-02-04', '2021-02-05',
               '2021-02-06', '2021-02-07', '2021-02-08', '2021-02-09',
               '2021-02-10', '2021-02-11', '2021-02-12', '2021-02-13',
               '2021-02-14', '2021-02-15', '2021-02-16', '2021-02-17',
               '2021-02-18', '2021-02-19', '2021-02-20', '2021-02-21',
               '2021-02-22', '2021-02-23', '2021-02-24', '2021-02-25',
               '2021-02-26', '2021-02-27', '2021-02-28', '2021-03-01',
               '2021-03-02', '2021-03-03', '2021-03-04', '2021-03-05',
               '2021-03-06', '2021-03-07', '2021-03-08', '2021-03-09',
               '2021-03-10', '2021-03-11', '2021-03-12', '2021-03-13',
               '2021-03-14', '2021-03-15', '2021-03-16', '2021-03-17',
               '2021-03-18', '2021-03-19', '2021-03-20', '2021-03-21',
               '2021-03-22', '2021-03-23', '2021-03-24', '2021-03-25',
               '2021-03-26', '2021-03-27', '2021-03-28', '2021-03-29',
               '2021-03-30', '2021-03-31', '2021-04-01', '2021-04-02'],
```

'2021-04-03', '2021-04-04', '2021-04-05', '2021-04-06',
'2021-04-07', '2021-04-08', '2021-04-09', '2021-04-10',
'2021-04-11', '2021-04-12', '2021-04-13', '2021-04-14',
'2021-04-15', '2021-04-16', '2021-04-17', '2021-04-18',
'2021-04-19', '2021-04-20', '2021-04-21', '2021-04-22',
'2021-04-23', '2021-04-24', '2021-04-25', '2021-04-26',
'2021-04-27', '2021-04-28', '2021-04-29', '2021-04-30',
'2021-05-01', '2021-05-02', '2021-05-03', '2021-05-04',
'2021-05-05', '2021-05-06', '2021-05-07', '2021-05-08',
'2021-05-09', '2021-05-10', '2021-05-11', '2021-05-12',
'2021-05-13', '2021-05-14', '2021-05-15', '2021-05-16',
'2021-05-17', '2021-05-18', '2021-05-19', '2021-05-20',
'2021-05-21', '2021-05-22', '2021-05-23', '2021-05-24',
'2021-05-25', '2021-05-26', '2021-05-27', '2021-05-28',
'2021-05-29', '2021-05-30', '2021-05-31', '2021-06-01',
'2021-06-02', '2021-06-03', '2021-06-04', '2021-06-05',
'2021-06-06', '2021-06-07', '2021-06-08', '2021-06-09',
'2021-06-10', '2021-06-11', '2021-06-12', '2021-06-13',
'2021-06-14', '2021-06-15', '2021-06-16', '2021-06-17',
'2021-06-18', '2021-06-19', '2021-06-20', '2021-06-21',
'2021-06-22', '2021-06-23', '2021-06-24', '2021-06-25',
'2021-06-26', '2021-06-27', '2021-06-28', '2021-06-29',
'2021-06-30', '2021-07-01', '2021-07-02', '2021-07-03',
'2021-07-04', '2021-07-05', '2021-07-06', '2021-07-07',
'2021-07-08', '2021-07-09', '2021-07-10', '2021-07-11',
'2021-07-12', '2021-07-13', '2021-07-14', '2021-07-15',
'2021-07-16', '2021-07-17', '2021-07-18', '2021-07-19',
'2021-07-20', '2021-07-21', '2021-07-22', '2021-07-23',
'2021-07-24', '2021-07-25', '2021-07-26', '2021-07-27',
'2021-07-28', '2021-07-29', '2021-07-30', '2021-07-31',
'2021-08-01', '2021-08-02', '2021-08-03', '2021-08-04',
'2021-08-05', '2021-08-06', '2021-08-07', '2021-08-08',
'2021-08-09', '2021-08-10', '2021-08-11', '2021-08-12',
'2021-08-13', '2021-08-14', '2021-08-15', '2021-08-16',
'2021-08-17', '2021-08-18', '2021-08-19', '2021-08-20',
'2021-08-21', '2021-08-22', '2021-08-23', '2021-08-24',
'2021-08-25', '2021-08-26', '2021-08-27', '2021-08-28',
'2021-08-29', '2021-08-30', '2021-08-31', '2021-09-01',
'2021-09-02', '2021-09-03', '2021-09-04', '2021-09-05',
'2021-09-06', '2021-09-07', '2021-09-08', '2021-09-09',
'2021-09-10', '2021-09-11', '2021-09-12', '2021-09-13',
'2021-09-14', '2021-09-15', '2021-09-16', '2021-09-17',
'2021-09-18', '2021-09-19', '2021-09-20', '2021-09-21',
'2021-09-22', '2021-09-23', '2021-09-24', '2021-09-25',
'2021-09-26', '2021-09-27', '2021-09-28', '2021-09-29',
'2021-09-30', '2021-10-01', '2021-10-02', '2021-10-03',
'2021-10-04', '2021-10-05', '2021-10-06', '2021-10-07',
'2021-10-08', '2021-10-09', '2021-10-10', '2021-10-11',
'2021-10-12', '2021-10-13', '2021-10-14', '2021-10-15',
'2021-10-16', '2021-10-17', '2021-10-18', '2021-10-19',
'2021-10-20', '2021-10-21', '2021-10-22', '2021-10-23',
'2021-10-24', '2021-10-25', '2021-10-26', '2021-10-27',
'2021-10-28', '2021-10-29', '2021-10-30', '2021-10-31',
'2021-11-01', '2021-11-02', '2021-11-03', '2021-11-04',
'2021-11-05', '2021-11-06', '2021-11-07', '2021-11-08',
'2021-11-09', '2021-11-10', '2021-11-11', '2021-11-12',
'2021-11-13', '2021-11-14', '2021-11-15', '2021-11-16',
'2021-11-17', '2021-11-18', '2021-11-19', '2021-11-20',
'2021-11-21', '2021-11-22', '2021-11-23', '2021-11-24',
'2021-11-25', '2021-11-26', '2021-11-27', '2021-11-28',
'2021-11-29', '2021-11-30', '2021-12-01', '2021-12-02',
'2021-12-03', '2021-12-04', '2021-12-05', '2021-12-06',
'2021-12-07', '2021-12-08', '2021-12-09', '2021-12-10',
'2021-12-11', '2021-12-12', '2021-12-13', '2021-12-14',
'2021-12-15', '2021-12-16', '2021-12-17', '2021-12-18',
'2021-12-19', '2021-12-20', '2021-12-21', '2021-12-22',

```
'2021-12-23', '2021-12-24', '2021-12-25', '2021-12-26',
'2021-12-27', '2021-12-28', '2021-12-29', '2021-12-30'
```

In [16]:

```
timeseries = np.arange(np.datetime64('2021-01-01 00:00:00'), np.datetime64('2021-12-31 23:59:59'))
len(timeseries)
```

Out[16]: 31449600

Operationen mit Datum oder Zeit

- Für Zeitreihenanalysen benötigt man oft Logik-Operationen
- z.B. Vergleich: Ist ein Datum-/Zeit größer, kleiner oder gleich?
- oder, wie viele Objekte entsprechen einem Kriterium?

Vergleiche

In [17]:

```
reference = np.datetime64('2021-04-12 12:33:00')

boolean = timeseries > reference

new_ts = timeseries[boolean]
new_ts
```

Out[17]: array(['2021-04-12T12:33:01', '2021-04-12T12:33:02',
 '2021-04-12T12:33:03', ..., '2021-12-30T23:59:57',
 '2021-12-30T23:59:58', '2021-12-30T23:59:59'],
 dtype='datetime64[s]')

In [18]:

```
reference = np.datetime64('2021-04-12 12:33:00')

idx = np.where(timeseries == reference)
idx
```

Out[18]: (array([8771580], dtype=int64),)

Abbildungen mit Datum/Zeit

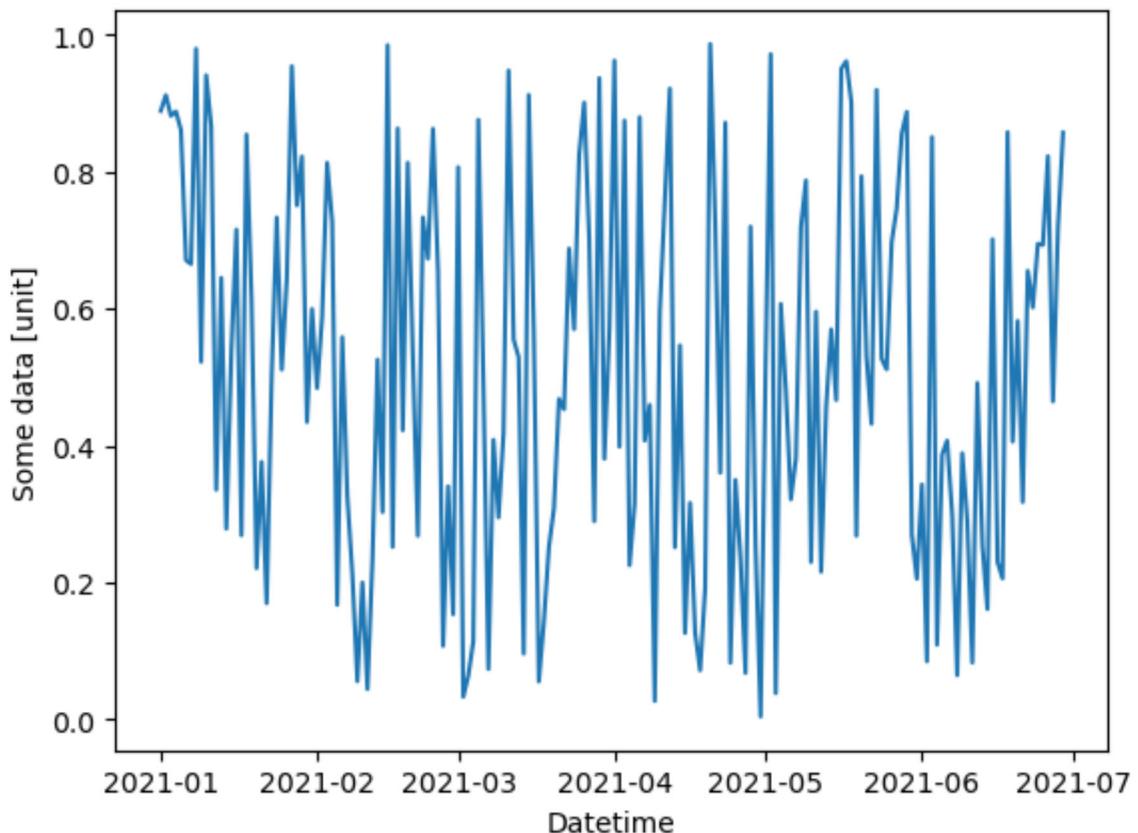
- Abbildungen mit *matplotlib* erkennen automatisch, ob ein Datum/Zeit-Objekt übergeben wurde
- Die Achsenbeschriftung wird auch automatisch übernommen

In [19]:

```
timeseries = np.arange(np.datetime64('2021-01-01'), np.datetime64('2021-06-30'))
data = np.random.rand(len(timeseries))

plt.plot(timeseries, data)
plt.xlabel('Datetime')
plt.ylabel('Some data [unit]')

plt.show()
```



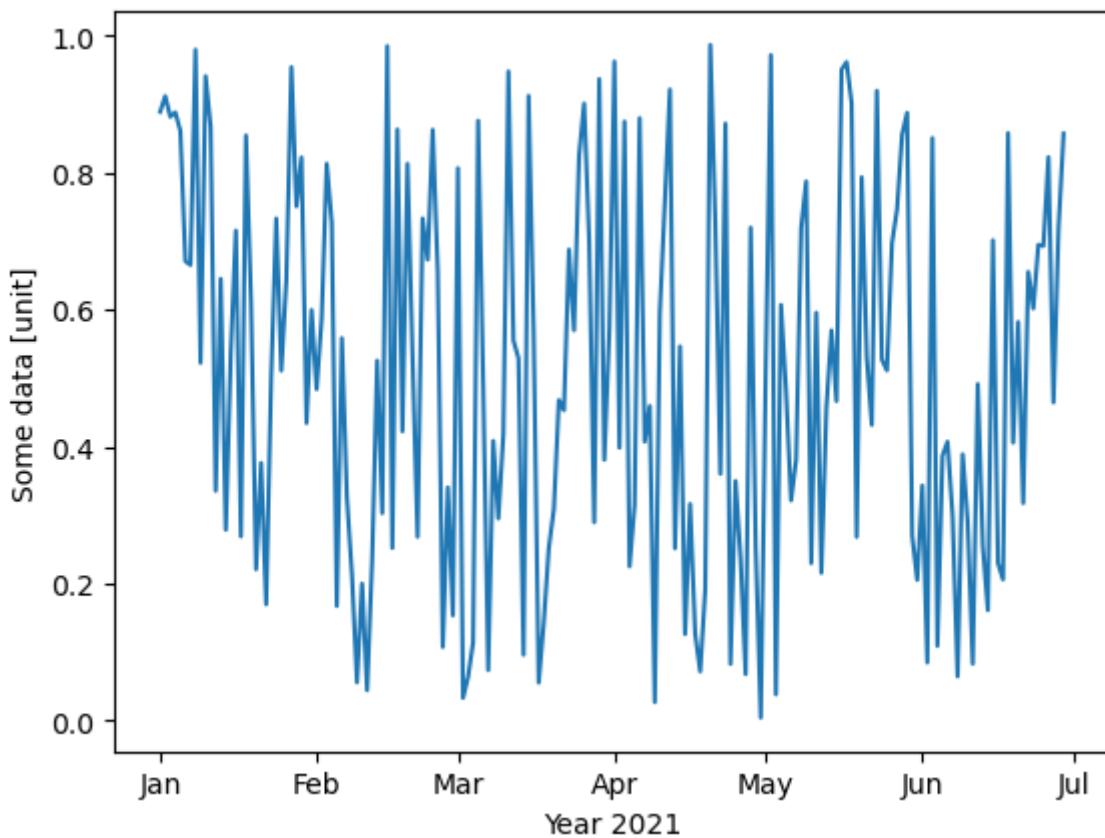
Beispiel: Bei der Achsenbeschriftung muss oder möchte man oft eingreifen

```
In [20]: import matplotlib.dates as mdates
```

```
In [21]: fig, ax = plt.subplots(1)
ax.plot(timeseries, data)
ax.set_xlabel('Year 2021')
ax.set_ylabel('Some data [unit]')

# custom formatting
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b'))

plt.show()
```



ENDE