Python Grundlagen

Julian Huber & Matthias Panny

Scientific Computing II

🎯 Lernziele

- Studierende können Daten in das tidy data Format bringen
- Studierende können Daten mit pandas auswerten

Tidy Data

> Happy families are all alike; every unhappy family is unhappy in its own way
> — *Leo Tolstoy*

> Like families, tidy datasets are all alike but every messy dataset is messy in its own way. Tidy datasets provide a standardized way to link the structure of a dataset (its physical layout) with its semantics (its meaning).
> — *Hadley Wickham*

Idee von tidy data

- Viele Funktionen erwarten die gleichen ordentliche Datenstrukturen as input
- Entspricht der Struktur von Datenbanken (Codd's 3. Normalform)
- H. Wickham, "Tidy Data", J. Stat. Soft., vol. 59, no. 10, pp. 1–23, Sep. 2014.

Tidy Data

Beispiel

Robot_ID	Timestamp	Sensor_Type	Sensor_Value	Unit
1	2023-09-27 08:00:00	Temperature	25.3	°C
1	2023-09-27 08:00:15	Temperature	25.5	°C
1	2023-09-27 08:00:30	Humidity	45.2	%
2	2023-09-27 08:00:00	Temperature	24.8	°C
2	2023-09-27 08:00:15	Temperature	24.9	°C
2	2023-09-27 08:00:30	Humidity	46.0	%
3	2023-09-27 08:00:00	Temperature	25.0	°C
3	2023-09-27 08:00:15	Temperature	25.1	°C
3	2023-09-27 08:00:30	Humidity	45.8	%

Tidy Data - Regeln

- Jede Variable steht in einer eigenen Spalte
- Jede Beobachtung/jeder Fall steht in einer eigenen Zeile
- Jeder einzelne Wert steht in einer Zelle

Diskussion

- Inwiefern macht es Sinn den Zeitstempel (inkl. Datum und Uhrzeit) in eine Spalte zu setzen?

Tidy Data

Untidy Data: Mehr als ein Wert pro Zelle

Robot_ID	Sensor_Data
1	{"Timestamp": "2023-09-27 08:00:00", "Type": "Temperature", "Value": 25.3}
1	{"Timestamp": "2023-09-27 08:00:15", "Type": "Temperature", "Value": 25.5}
1	{"Timestamp": "2023-09-27 08:00:30", "Type": "Humidity", "Value": 45.2}
2	{"Timestamp": "2023-09-27 08:00:00", "Type": "Temperature", "Value": 24.8}
2	{"Timestamp": "2023-09-27 08:00:15", "Type": "Temperature", "Value": 24.9}
2	{"Timestamp": "2023-09-27 08:00:30", "Type": "Humidity", "Value": 46.0}
3	{"Timestamp": "2023-09-27 08:00:00", "Type": "Temperature", "Value": 25.0}
3	{"Timestamp": "2023-09-27 08:00:15", "Type": "Temperature", "Value": 25.1}
3	{"Timestamp": "2023-09-27 08:00:30", "Type": "Humidity", "Value": 45.8}

Tidy Data

- Wie alle Regeln gibt es auch Ausnahmen
- Vorteile einer Spalte für `Temperature_C` und `Humidity_%`:
  - Platzsparend
- Vorteile `Sensor_Type`, `Sensor_Value`, `Unit`:
  - Flexibler z.B: bei verschiedenen Einheiten
  - Einfacher zu verarbeiten
  - Kann leichter in erste Form umgewandelt werden als umgekehrt

Untidy Data?

Robot_ID	Timestamp	Temperature_C	Humidity_%
1	2023-09-27 08:00:00	25.3	45.2
1	2023-09-27 08:00:15	25.5	45.1
2	2023-09-27 08:00:00	24.8	46.0
2	2023-09-27 08:00:15	24.9	46.2
3	2023-09-27 08:00:00	25.0	45.8
3	2023-09-27 08:00:15	25.1	45.9

Comma Separated Values (CSV-Files)

- speichert Tabellen als einfache Textdatei
- Namenskonvention `<file_name>.csv`
- kann mit Excel oder jedem Texteditor geöffnet werden
- `README.txt` könnte ergänzt werden, um weitere Informationen (z.B. Einheit der Größen) zu ergänzen
- teilweise hierfür auch zweite Kopfzeile

pandas_data.csv

```
Robot_ID,Timestamp,Sensor_Type,Sensor_Value,Unit
1,2023-09-27 08:00:00,Temperature,25.3,°C
1,2023-09-27 08:00:15,Temperature,25.5,°C
1,2023-09-27 08:00:30,Humidity,45.2,%
2,2023-09-27 08:00:00,Temperature,24.8,°C
2,2023-09-27 08:00:15,Temperature,24.9,°C
2,2023-09-27 08:00:30,Humidity,46.0,%
3,2023-09-27 08:00:00,Temperature,25.0,°C
3,2023-09-27 08:00:15,Temperature,25.1,°C
3,2023-09-27 08:00:30,Humidity,45.8,%
```

Comma Separated Values (CSV-Files)

- In jeder Zeile werden die gleichen Spalten, am besten im gleichen Datentyp erwartet (vgl. `Array`, `Series`)
  - leider nicht immer so (vgl. `"45.8"`)
- Der Delimiter oder Separator ist das Trennzeichen zwischen Spalten (z.B. `,`, `;`, `\t`)
- Je nach Spracheinstellung kann dies zu Problemen mit den Dezimaltrennzeichen führen (`3,14` vs. `3.14`)

`data.csv` - Einige Fehlerquellen

```
Robot_ID,Timestamp,Sensor_Type,Sensor_Value
1;2023-09-27 08:00:00;Temperature,25.3
1;2023-09-27 08:00:15;Temperature,
1;2023-09-27 08:00:30;45.2
2;2023-09-27 08:00:00;Temperature,24.8
2;2023-09-27 08:00:15;Temperature,24.9
2;2023-09-27 08:00:30;Humidity,"46.0"
3;2023-09-27 08:00:00;Temperature,25.0
3;2023-09-27 08:00:15;Temperature,25.1
3;2023-09-27 08:00:30;Humidity,"45.8"
```

Tabellendaten in Python

- Python ist gut geeignet um mit tabellarischen Daten zu arbeiten
- Meistverwendetes package: `pandas` → `pip install pandas`

pandas

- Eigner Datentyp für Tabellen: `DataFrame`
- Viele Funktionen zum Einlesen, Bearbeiten und Auswerten
- Testdaten: pandas_data.csv

```
import pandas as pd

df = pd.read_csv(r"01_Python_Grundlagen\Examples\pandas\pandas_data.csv", sep = ",")
print(df.head())
```

```
    Robot_ID            Timestamp  Sensor_Type  Sensor_Value
0          1  2023-09-27 08:00:00  Temperature          25.3
1          1  2023-09-27 08:00:15  Temperature          25.5
2          1  2023-09-27 08:00:30     Humidity          45.2
3          2  2023-09-27 08:00:00  Temperature          24.8
4          2  2023-09-27 08:00:15  Temperature          24.9
```

Tabellendaten in Python

Index

- Bei der ersten (unbeschrifteten) Spalte handelt es ist um den Index (vgl. Liste)
- Dieser macht jede Zeile (Beobachtung) eindeutig identifizierbar
- Oft ist der Index einfach durchnummeriert (`RangeIndex`)

```
print(df.index)
# RangeIndex(start=0, stop=9, step=1)
```

- Es könnte aber jeder Datentyp in Frage kommen, sofern der Index für jede Zeile eindeutig ist (hier z.B. *nicht* der `Timestamp`)

Adressierung über Index

- Beobachtungen können über den Index mittels `.loc[]` aufgerufen werden

```
print(df.loc[0])
```

```
Robot_ID                          1
Timestamp       2023-09-27 08:00:00
Sensor_Type             Temperature
Sensor_Value                   25.3
Name: 0, dtype: object
```

- `[]` anstelle von `()`, da `loc` keine Funktion, sondern ein Objekt ist und die Notation so `numpy` ähnelt

Tabellendaten in Python

Series

- Über eckige Klammern (indexing) direkt lassen sich hingegen die Spalten (Variablen) aufrufen
- In der Ausgabe erkennen wir auch links den `Index` wieder
- Bei der Rückgabe handelt sich um ein `Series`-Objekt mit einem `Index` und einer Sammlung von Daten gleichen Datentyps (`dtype`)

```
print(df["Timestamp"])
```

```
0    2023-09-27 08:00:00
1    2023-09-27 08:00:15
2    2023-09-27 08:00:30
3    2023-09-27 08:00:00
4    2023-09-27 08:00:15
5    2023-09-27 08:00:30
6    2023-09-27 08:00:00
7    2023-09-27 08:00:15
8    2023-09-27 08:00:30
Name: Timestamp, dtype: object
```

Datentypen einer Series

- `int64`
  - Beispiele: `1`, `-42`, `1000`
- `float64`
  - Beispiele: `3.14`, `-0.5`, `2.71828`
- `bool`
  - Beispiele: `True`, `False`
- `object`
  - Vielzahl von Python-Objekte. Auch bei "gemischten" `Series`
  - Beispiele: `"Hallo"`, `None`, `[1, 2, 3]`
- `datetime64`
  - Beispiele: `2023-09-27 08:00:00`, `2022-03-15 12:30:00`
- `timedelta64`
  - Beispiele: `1 day`, `2 hours`, `30 minutes`
- `categorical`
  - Dieser Datentyp wird für kategorische Daten verwendet, die eine begrenzte Anzahl von eindeutigen Werten haben.
  - Beispiele: `"Rot"`, `"Grün"`, `"Blau"`

Beispieldatensatz

	sensor_id	component_status	temp_C	pressure_pa
0	1	Operational	98.2	101325.0
1	2	Faulty	NaN	100984.0
2	3	Operational	3.0	102045.0
3	4	Operational	24.8	102045.0
4	5	Faulty	101.7	NaN

```
# Das Abrufen von Spalten durch den Spaltennamen gibt nur diese Spalte zurück.
first_column = df["sensor_id"]
print(first_column)

0    1
1    2
2    3
3    4
4    5
Name: sensor_id, dtype: int64
```

Aufruf von Methoden

	sensor_id	component_status	temp_C	pressure_pa
0	1	Operational	98.2	101325.0
1	2	Faulty	NaN	100984.0
2	3	Operational	3.0	102045.0
3	4	Operational	24.8	102045.0
4	5	Faulty	101.7	NaN

```
# Funktionen, die für eine Spalte aufgerufen werden und einen Wert zurückgeben
df["component_status"].mode()
# > "Operational"

df["temp_C"].mean()
# > 56.925
```

Auswahl von Daten-Schnitten

	sensor_id	component_status	temp_C	pressure_pa
0	1	Operational	98.2	101325.0
1	2	Faulty	NaN	100984.0
2	3	Operational	3.0	102045.0
3	4	Operational	24.8	102045.0
4	5	Faulty	101.7	NaN

```
# Das Abrufen von Zeilen durch den Indexierungs-Operator
# gibt alle Werte (Spalten) für diese Zeile zurück.
second_row = df.loc[1]
print(second_row)

sensor_id                2
component_status      Faulty
temp_C                   NaN
pressure_pa         100984.0
Name: 1, dtype: object
```

Auswahl von Daten-Schnitten

	sensor_id	component_status	temp_C	pressure_pa
0	1	Operational	98.2	101325.0
1	2	Faulty	NaN	100984.0
2	3	Operational	3.0	102045.0
3	4	Operational	24.8	102045.0
4	5	Faulty	101.7	NaN

```
# Das Abrufen einer Zelle über Index und Spaltennamen
cell = df.loc[1]["component_status"]
print(cell)
#> "Faulty"

# Abrufen einer Zelle über den iloc operator
# (hierbei werden Zeilen und Spalten durchnummeriert)
# cell = df.iloc[1][1] --> funktioniert, aber FutureWarning
cell = df.iloc[1, 1]
print(cell)
#> "Faulty"
```

Datenanalyse mit pandas

Filtern und where-clause

	sensor_id	component_status	temp_C	pressure_pa
0	1	Operational	98.2	101325.0
1	2	Faulty	NaN	100984.0
2	3	Operational	3.0	102045.0
3	4	Operational	24.8	102045.0
4	5	Faulty	101.7	NaN

```
# liefert eine Serie, für alle Sensoren mit Status "Operational" wahr ist
filter1 = df["component_status"] == "Operational"
print(filter1)

0     True
1    False
2     True
3     True
4    False
Name: component_status, dtype: bool
```

Filtern und where-clause

```
# Wir können diese Serie verwenden, um nur nach funktionierenden Sensoren zu filtern.
filtered_df = df.where(filter1)
print(filtered_df)
```

	sensor_id	component_status	temp_C	pressure_pa
0	1.0	Operational	98.2	101325.0
1	NaN	NaN	NaN	NaN
2	3.0	Operational	3.0	102045.0
3	4.0	Operational	24.8	102045.0
4	NaN	NaN	NaN	NaN

Filtern und where-clause

```
# Es gibt auch einen kürzeren Weg, dies zu schreiben
# Nimm df an allen Indexpositionen, bei denen "component_status" "Operational" ist.
filtered_df = df[df["component_status"] == "Operational"]
print(filtered_df)
```

	sensor_id	component_status	temp_C	pressure_pa
0	0	1	Operational	98.2
2	2	3	Operational	3.0
3	3	4	Operational	24.8

- Beachten Sie, dass ein letzter Befehl nicht der üblichen Syntax entspricht: `df[<column_name>]` vs `df[<series_of_bools>]`

Gruppieren

	sensor_id	component_status	temp_C	pressure_pa
0	1	Operational	98.2	101325.0
1	2	Faulty	NaN	100984.0
2	3	Operational	3.0	102045.0
3	4	Operational	24.8	102045.0
4	5	Faulty	101.7	NaN

```
# liefert eine die Durchschnittswerte nach Sensor Status
df.groupby("component_status").mean()
```

component_status	sensor_id	temp_C	pressure_pa
Faulty	3.500000	101.7	100984.0
Operational	2.666667	42.0	101805.0

```
# liefert eine die Durchschnittswerte nach Sensor Status
# und gibt nur die Temperatur zurück
df.groupby("component_status").mean()["temp_C"]
```

Erstellen von neuen Spalten

	sensor_id	component_status	temp_C	pressure_pa
0	1	Operational	98.2	101325.0
1	2	Faulty	NaN	100984.0
2	3	Operational	3.0	102045.0
3	4	Operational	24.8	102045.0
4	5	Faulty	101.7	NaN

```
# Erstellt eine neue Spalte durch Berechnung
df['pressure_hpa'] = df['pressure_pa'] / 100
df
```

	sensor_id	...	pressure_pa	pressure_hpa
0	1	...	101325.0	1013.25
1	2	...	100984.0	1009.84
2	3	...	102045.0	1020.45
3	4	...	102045.0	1020.45
4	5	...	NaN	NaN

inplace-Argumente

- die meisten Methoden auf DataFrames haben als Rückgabe einen neuen DataFrame
  - `new_df = df.where(filter1)`
- `inplace`-Argument erlaubt es, die Änderungen direkt auf dem DataFrame durchzuführen
  - `df.where(filter1, inplace=True)`
  - vgl. `list.append()` verändert die Liste direkt
  - `print(df.where(filter1, inplace=True))` gibt dann `None` zurück

 Aufgabe

Um das Verständnis für pandas weiter zu vertiefen kann das hier verlinkte Notebook bearbeitet werden.

Jupyter Notebook

🏆 Hausübung

- Es wurde ein Windkanalversuch durchgeführt um den Luftwiderstand eines neuentwickelten Motorrades im Modellversuch zu bestimmen.
- Dazu wurde das Modell des Motorrades in den Windkanal gestellt und bei verschiedenen Windgeschwindigkeiten mit einem Kraft-Momenten-Sensor die aus dem Luftwiderstand resultierenden Kräfte und Momente gemessen.

Aufgabe

- Laden & transformieren Sie die gemessenen Daten
- Bestimmen Sie den c_w-Wert des Motorrads
- Visualisieren Sie die Ergebnisse