Das sdata-Format

December 8, 2020

Ingolf Lepenies, lepy@mailbox.org

1 Das sdata-Format

1.1 Ein Beispiel zur Ablage einer Tabelle im sdata-Format

Dieses Beispiel zeigt die Ablage einer Tabelle mit zwei Spalten im `sdata`-Format.

[1]:
```
import pandas as pd
df = pd.DataFrame({"column_a": [1.1, 2.1, 3.5],
                   "column_b": [2.4, 5.2, 2.2]}, index=[0, 'max_b', 2])
df.index.name = "index"
df
```

[1]:

index	column_a	column_b
0	1.1	2.4
max_b	2.1	5.2
2	3.5	2.2

sdata steht für **strukturierte Daten**. `sdata` ist eine unter der MIT-Lizenz veröffentlichte open-source Python-Bibiothek, welche das sdata-Format unterstützt. Installiert wird die Bibliothek mittels `pip install sdata`. Die Dokumentation wird unter sdata.readthedocs.io veröffentlicht.

[2]:
```
import sdata
print("sdata v{}".format(sdata.__version__))
```

```
sdata v0.8.4
```

1.2 Die Komponenten des sdata-Formates

Das sdata-Datenformat `Data` besteht aus drei Komponenten: den Daten (hier eine **Tabelle** `table`), den **Metadaten** `metadata` (`name`, `uuid`, ...) und einer **Beschreibung** der Daten `description`.

[3]:
```
data = sdata.Data(name="my data name", table=df, description="my data␣
 ↪description")
print("data:            \t {0}".format(type(data)))
```

```
print("data.metadata:   \t {0}".format(type(data.metadata)))
print("data.table:      \t {0}".format(type(data.table)))
print("data.description:\t {0}".format(type(data.description)))
```

```
data:                   <class 'sdata.Data'>
data.metadata:          <class 'sdata.metadata.Metadata'>
data.table:             <class 'pandas.core.frame.DataFrame'>
data.description:       <class 'str'>
```

1.2.1 sdata.Data.metadata

Jeder Datensatz `Data` benötigt einen Namen `name`. Die vom User vorgegebene Identifikation des Datensatzes ist aber mit einer hohen Wahrscheinlichkeit nicht eindeutig, da i.d.R sehr kurze Bezeichnungen gewählt werden.

[4]:
```
data = sdata.Data(name="basic example")
print("data.name:\t '{0.name}'".format(data))
```

```
data.name:      'basic example'
```

Zur Identifikation eines Datensatzes ist eine möglichst eindeutige Bezeichnung hilfreich. Üblicherweise wird hierzu ein sogenannter Universally Unique Identifier [`uuid`](https://de.wikipedia.org/wiki/Universally_Unique_Identifier) verwendet. Diese Merkmale eines Datensatzes werden im sdata-Format in den Metadaten gespeichert. Diese `uuid` wird automatisch bei jeder Instanziierung eines `sdata.Data`-Objektes generiert.

[5]:
```
data = sdata.Data(uuid="8b1e85eded1241eb854be3365bcf9884")
print("data.uuid:\t '{0.uuid}'".format(data))
```

```
data.uuid:      '8b1e85eded1241eb854be3365bcf9884'
```

Die `uuid` kann aber auch basierend auf einem - möglichst eindeutigen - Names generiert werden.

[6]:
```
my_uuid = sdata.uuid_from_str("Das ist ein möglichst eindeutiger Name für die␣
 ↪Daten")
data = sdata.Data(uuid=my_uuid)
print("data.uuid:\t '{0.uuid}'".format(data))
```

```
data.uuid:      '06f8c76b037c3636a40246f024e87574'
```

Die Komponente `metadata` hat also mindestenz zwei Attribute `name` und `uuid`. Die einfachste Form eines Attributes stellt ein key-value-Tupel dar, d.h. eine Verknüpfung einer Bedeutung mit einer Ausprägung, z.B. `Augenfarbe="blau"` oder `name="basic example"`.

[7]:
```
attribute1 = sdata.Attribute("Augenfarbe", "blau")
attribute1
```

[7]:
```
(Attr'Augenfarbe':blau(str))
```

Ein `Attribut` (Eigenschaft) des Datenobjektes hat im sdata-Format die Felder

- `name` … Name des Attributes
- `value` … Wert des Attributes
- `dtype` … Datentyp des Attributwertes `values` (default=`str`)
- `unit` … physikalische Einheit des Attributes (*optional*)
- `description` … Beschreibung des Attributes (*optional*)
- `label` … optionales Label des Attributes, z.B. für Plotzwecke

[8]:
```
attribute2 = sdata.Attribute(name="answer",
                             value=42,
                             dtype="int",
                             unit="-",
                             description="""The Answer to the Ultimate Question␣
 ↪of Life, The Universe, and Everything""",
                             label="Die Antwort")
attribute2.to_dict()
```

[8]:
```
{'name': 'answer',
 'value': 42,
 'unit': '-',
 'dtype': 'int',
 'description': 'The Answer to the Ultimate Question of Life, The Universe, and
Everything',
 'label': 'Die Antwort'}
```

Attribute werden in den Metadaten gespeichert.

[9]:
```
metadata = sdata.Metadata()
metadata.add(attribute1)
metadata.add(attribute2)
print(metadata)
metadata.df
```

```
(Metadata'N.N.':3 ['sdata_version', 'Augenfarbe', 'answer'])
```

[9]:

key	name	value	dtype	unit	description	label
sdata_version	sdata_version	0.8.4	str	-		
Augenfarbe	Augenfarbe	blau	str	-		
answer	answer	42	int	-	The Answer to the Ultimate Question of Life, T…	Die Antwort

Ein Attribut - auch Eigenschaft genannt - stellt demnach ein Merkmal des Datenobjektes dar. Nüt-

zlich ist auch eine Angabe einer physikalischen Größe, welche i.d.R. Einheiten behaftet ist. Exemplarisch ist hier ein Attribut `Temperatur` mit der Ausprägung `25.4°C` aufgeführt. Die physikalische Einheit wird im Attributfeld `unit` abgelegt (`unit="degC"`) und der Zahlenwert im Feld `value=25.4`. Da eine physikalische Größe aus dem Produkt einer (reelen) Zahl und einer physikalischen Einheit besteht, sieht das sdata-Attribut-Format auch ein Attributfeld `dtype` zur Definition des Datentypes für den Attributwert `value` vor. Ferner kann jedes Attribut im Feld `description` genauer beschrieben werden. Optional kann auch ein Label für Plot-Zwecke angegeben werden.

[10]:
```
data = sdata.Data(name="basic example",␣
 ↪uuid="38b26864e7794f5182d38459bab85842", table=df)
data.metadata.add("Temperatur",
                  value=25.4,
                  dtype="float",
                  unit="degC",
                  description="Temperatur",
                  label="Temperatur T [°C]")
data.metadata.df
```

[10]:

key	name	value	dtype	unit	description	label
sdata_version	sdata_version	0.8.4	str	-		
name	name	basic example	str	-		
uuid	uuid	38b26864e7794f5182d38459bab85842	str	-		
Temperatur	Temperatur	25.4	float	degC	Temperatur	Temperatur T [°C]

1.2.2 sdata.Data.table

Das eigentliche Datenobjekt `table` ist als `pandas.DataFrame` repräsentiert, d.h. jede Zelle der Tabelle ist durch ein Tupel (index, column) indiziert. Jede Spalte (`column`) ist durch den Spaltennamen (hier z.B. `time` oder `temperature`) indizierbar. Jede Zeile (`row`) ist durch den `index` indiziert, wobei der `index` auch alphanumerisch sein kann (z.B. 'max_temp').

[11]:
```
df = pd.DataFrame({"time": [1.1, 2.1, 3.5],
                   "temperature": [2.4, 5.2, 2.2]}, index=[0, 'max_temp', 2])
df.index.name = "index"
df
```

[11]:

index	time	temperature
0	1.1	2.4

```
max_temp    2.1          5.2
2           3.5          2.2
```

1.2.3 sdata.Data.decription

Die Beschreibung `description` ist vom Typ `str`.

[12]:
```
data.description = "Messergebnis Temperatur."
print(data.description)
```

```
Messergebnis Temperatur.
```

Es ist möglich eine vereinfachte Auszeichnungssprache wie Markdown verwendbar. Beispielhaft ist hier Datenbeschreibung mit Überschriften und Formel aufgeführt.

2 Messergebnis Temperatur.

2.1 subheader

a remarkable text

Bullet list:

- aaa
  - aaa.b
- bbb

Numbered list:

1. foo
2. bar

$f(x) = \frac{1}{2}\sin(x)$

code:

```
name="basic example"
```

A Link.

[13]:
```
data.description = r"""# Messergebnis Temperatur
## subheader

a remarkable text

Bullet list:

- aaa
    - aaa.b
- bbb
```

```
Numbered list:

1. foo
1. bar

$f(x) = \frac{1}{2}\sin(x)$

code:

    name = "basic example"

A [Link](https://github.com/lepy/sdata)."""
```

[14]:
```
print(data.description)
```

```
# Messergebnis Temperatur
## subheader

a remarkable text

Bullet list:

- aaa
    - aaa.b
- bbb


Numbered list:

1. foo
1. bar

$f(x) = \frac{1}{2}\sin(x)$

code:

    name = "basic example"

A [Link](https://github.com/lepy/sdata).
```

2.2 Beispiel einer *Temperaturmessung-001*

Die Daten bestehen aus einer Tabelle mit zwei Spalten für die Zeit und eine gemessene Temperatur.

[15]:
```
df = pd.DataFrame({"time": [1.1, 2.1, 3.5],
                   "temperature": [2.4, 5.2, 2.2]},
                  index=[0, 'max_temp', 2])
df
```

[15]:

	time	temperature
0	1.1	2.4
max_temp	2.1	5.2
2	3.5	2.2

Das `sdata.Data`-Objekt wird mit einen Names und einer aus dem Namen generierten `uuid` versehen. Weiterhin wird für jede Spalte der Tabelle ein Attribut definiert, welche u.a. die physikalische Einheit der Spaltenwerte abbildet.

[16]:
```
data_name = "Temperaturmessung-001"
data = sdata.Data(name=data_name,
                  uuid=sdata.uuid_from_str(data_name),
                  table=df,
                  description="Messergebnis Temperatur")
data.metadata.add("time",
                  value=None,
                  dtype="float",
                  unit="s",
                  description="Zeitachse",
                  label="time $t$")
data.metadata.add("temperature",
                  value=None,
                  dtype="float",
                  unit="°C",
                  description="Zeitachse",
                  label="temperature $T$")
data.describe()
```

[16]:

	0
metadata	5
table_rows	3
table_columns	2
description	23

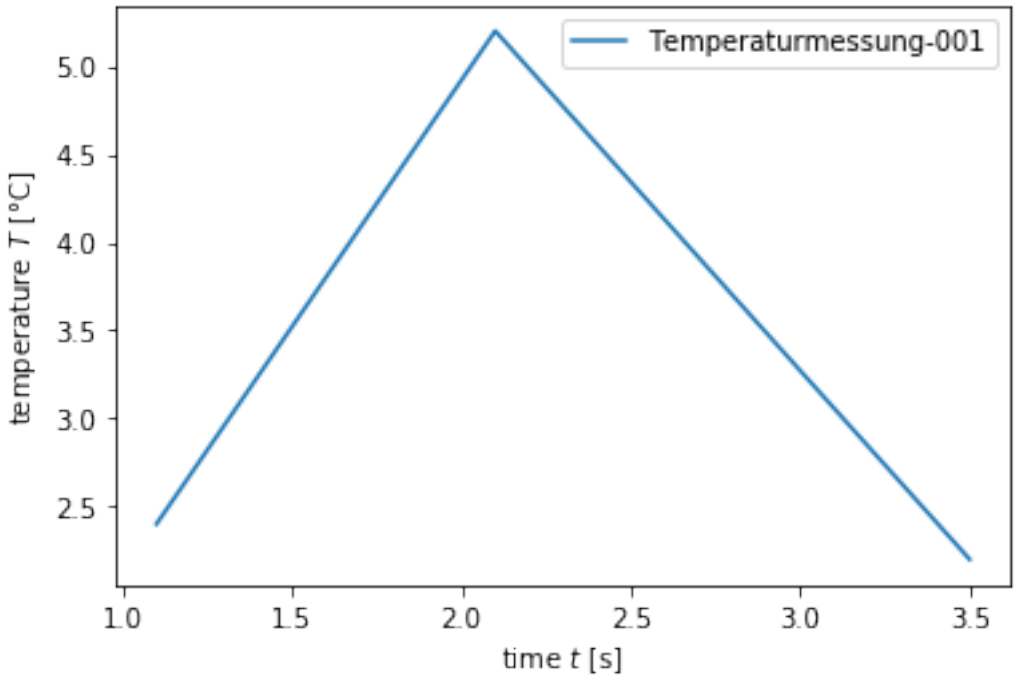
[18]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()

x_var = "time"
y_var = "temperature"

x_attr = data.metadata.get(x_var)
y_attr = data.metadata.get(y_var)
```

```
ax.plot(data.df[x_var], data.df[y_var], label=data.name)
ax.legend(loc="best")
ax.set_xlabel("{0.label} [{0.unit}]".format(x_attr))
ax.set_ylabel("{0.label} [{0.unit}]".format(y_attr))
print("plot")
```

```
plot
```



2.2.1 Export

Der Export kann in verschiedene Formate erfolgen.

[19]:
```
import os
filepath_xlsx = os.path.join("/tmp", data.osname + ".xlsx")
data.to_xlsx(filepath=filepath_xlsx)
print("Saved '{0.name}' to '{1}'.".format(data, filepath_xlsx))
```

```
Saved 'Temperaturmessung-001' to '/tmp/temperaturmessung-001.xlsx'.
```

[20]:
```
filepath_json = os.path.join("/tmp", data.osname + ".json")
data.to_json(filepath=filepath_json)
print("Saved '{0.name}' to '{1}'.".format(data, filepath_json))
```

```
Saved 'Temperaturmessung-001' to '/tmp/temperaturmessung-001.json'.
```

[21]:
```
filepath_csv = os.path.join("/tmp", data.osname + ".csv")
data.to_csv(filepath=filepath_csv)
print("Saved '{0.name}' to '{1}'.".format(data, filepath_csv))
```

```
Saved 'Temperaturmessung-001' to '/tmp/temperaturmessung-001.csv'.
```

2.2.2 Import

Der Import kann aus den verschiedene Export-Formate erfolgen.

[22]:
```
filepath_xlsx = os.path.join("/tmp", data.osname + ".xlsx")
data_xlsx = data.from_xlsx(filepath=filepath_xlsx)
data_xlsx
```

[22]:
```
(Data 'Temperaturmessung-001':e13d9387728c375eb98686eacf42b6df)
```

[23]:
```
filepath_json = os.path.join("/tmp", data.osname + ".json")
data_json = data.from_json(filepath=filepath_json)
data_json
```

[23]:
```
(Data 'Temperaturmessung-001':e13d9387728c375eb98686eacf42b6df)
```

Jedes `sdata.Data`-Objekt kann seinen Hash-Wert bestimmen. Dieser Hash-Wert, hier ein `sha3-256`, kann u.a. zum Vergleich der Daten benutzt werden.

[24]:
```
assert data.sha3_256==data_xlsx.sha3_256
assert data.sha3_256==data_json.sha3_256

print(data.sha3_256)
print(data_xlsx.sha3_256)
print(data_json.sha3_256)
```

```
7d681f4302252912cc9ad1c90f5a2b4be37362cfcecfdfe00f8d6ffee6faf48a
7d681f4302252912cc9ad1c90f5a2b4be37362cfcecfdfe00f8d6ffee6faf48a
7d681f4302252912cc9ad1c90f5a2b4be37362cfcecfdfe00f8d6ffee6faf48a
```

2.2.3 Zusammenfassung

Das `sdata`-Format bietet eine einfache Möglichkeit Daten lesbar für Menschen und zugleich maschinenlesbar abzulegen. Dies wird u.a. durch die Import-Export-Formate `xlsx`, `json`, `csv` gewährleistet. Durch die Qualifikation der Massendaten durch Metadaten und einer Beschreibung in Textform ist eine vollständige Beschreibung der Daten möglich. Die open-source Python-Bibliothek `sdata` erlaubt eine einfach Benutzung und Erweiterung des Datenformates.

[ ]:
```

```