Einführung in das Programmieren

'Dataframes' in Python

Christoph Draxler, Vanessa Reichel
draxler@phonetik.uni-muenchen.de
vanessa.reichel@campus.lmu.de

12. Juni 2024

Motivation

Wir kennen nun die wesentlichen Daten- und Kontrollstrukturen in Python:

- Variablen für den Zugriff auf Werte
- Funktionen als benannte Abläufe von Befehlen
- Fallunterscheidungen für verschiedene Abläufe in Programmen
- Schleifen zum wiederholten Ausführen von Befehlen
- `List` und `Dictionary` für viele Werte unter einem Namen
- Dateien zum dauerhaften Speichern von Daten
- reguläre Ausdrücke zum Suchen in Texten

Für allgemeine Datenanalysen benötigen wir nun spezielle Datenstrukturen, z. B. Tabellen.

Motivation

Häufig möchte man Werte zueinander in Beziehung setzen:

- Datum und Höchst- bzw. Tiefsttemperatur
- Matrikelnummer, Name und Klausurergebnis
- Versuchspersonennummer, Geschlecht, Alter und Grundfrequenz
- Artikel, Gewicht, Preis
- ...

Das ist mit den Python Datentypen `List` oder `Dictionary`, weil sie eindimensional sind, nicht so einfach möglich. Es gibt aber eine Library namens `pandas`, die Tabellen – ähnlich wie Dataframes in R – implementiert.

Beispiel Kassenzettel

Library `pandas` installieren

pandas ist ein schnelles, leistungsstarkes, flexibles und einfach zu bedienendes Open-Source-Werkzeug zur Datenanalyse und -manipulation, das auf der Programmiersprache Python aufbaut.

Einige Libraries werden bereits mit der Standardinstallation von Python installiert, andere müssen explizit hinzugefügt werden. Zu diesen zählt auch die Library `pandas`.
Informationen dazu findet man unter

https://pandas.pydata.org/

In VisualCode Studio kann man unten im Terminal den Befehl

```
pip install pandas
```

eingeben. `pip` ist ein Programm, das Python-Libraries aus dem Netz lädt und auf dem lokalen Rechner installiert.

DataFrames

`DataFrames` aus der Library `pandas` sind zweidimensionale Listen oder *Tabellen*. Sie haben

- lückenlos durchnummerierte Zeilen
- benannte Spalten

Um `DataFrames` zu verwenden, muss man die Library `pandas` importieren:

```
import pandas as pd
```

Mit 'as pd' definiert man 'pd' als verkürzte Schreibweise für 'pandas' im Code.

Kassenzettel in Python

Einen Kassenzettel wie auf dem Foto (auf Folie 4) kann man in Python wie folgt darstellen:

```
kassenzettel = pd.DataFrame({
    "Artikel" : ["Goldbären", "Wine Gums", "Clementinen"],
    "Gewicht" : [350, 300, 1000],
    "Preis" : [1.19, 0.89, 2.99],
    "Steuer" : ["B", "B", "B"]}
)
```

Die Funktion `DataFrame()` erwartet als Argument ein `Dictionary` mit folgender Struktur:

- die Schlüssel enthalten die Spaltennamen,
- einem Schlüssel ist eine `Liste` zugeordnet – das sind die Werte der entsprechenden Spalte..

Die Länge aller Spalten-Listen muss gleich sein.

DataFrames aus bestehenden Listen

Natürlich kann man DataFrames auch aus schon bestehenden Listen erstellen:

```
artikel = ["Goldbären", "Wine Gums", "Clementinen"]
gewicht = [350, 300, 1000]
preis = [1.19, 0.89, 2.99]
steuer = ["B", "B", "B"]

kassenzettel = pd.DataFrame({
    "Artikel" : artikel,
    "Gewicht" : gewicht,
    "Preis"   : preis,
    "Steuer"  : steuer})
```

Das ist für Listen, deren Elemente unabhängig voneinander berechnet werden, oder für Listen mit sehr vielen Elementen sehr praktisch.

DataFrames einlesen bzw. schreiben

Die `pandas`-Library unterstützt eine Vielzahl von Schreib- und Lesebefehlen speziell für `DataFrames`:

`read_csv()` komma- oder tabulartorgetrennte Textdateien

`read_excel()` Excel-Dateien

`read_json()` JavaScript Objekt-Dateien

`read_sql()` SQL-Datenbankabfragen

`read_html()` HTML Webseiten

Analog gibt es auch `to_csv()`, ... Befehle zum Schreiben der entsprechenden Dateien.

Kassenzettel in Python

```
print(kassenzettel)
       Artikel  Gewicht  Preis Steuer
0    Goldbären      350   1.19      B
1    Wine Gums      300   0.89      B
2  Clementinen     1000   2.99      B
```

`kassenzettel` enthält drei Zeilen und vier Spalten. Die Zeilen sind durchnummeriert, die Spalten sind mit 'Artikel', 'Gewicht', 'Preis' bzw. 'Steuer' bezeichnet. Die Eigenschaft `shape` gibt die Anzahl Zeilen und Spalten eines DataFrames als Tupel wieder: `kassenzettel.shape` ergibt `(3, 4)`.

Zugriff auf DataFrames

Auf die Spalten eines DataFrames greift man über den Namen oder eine Liste von Namen zu, auf die Werte der Spalte über einen numerischen Index oder eine Bereichsangabe in eckigen Klammern.

`kassenzettel[['Artikel', 'Preis']]` für die beiden vollständigen Spalten 'Artikel' und 'Preis'

`kassenzettel[['Preis', 'Artikel']]` für die beiden Spalten 'Artikel' und 'Preis' in vertauschter Reihenfolge

`kassenzettel['Artikel'][0]` für den ersten Wert in der Spalte 'Artikel'

`kassenzettel['Artikel'][0:3]` für die ersten drei Einträge der Spalte 'Artikel'

Probieren Sie es aus!

Zugriff über die Position

Auch ein Zugriff über die Zeilen- bzw. Spaltennummer ist möglich. Dazu verwendet man die Eigenschaft `iloc` eines Dataframes:

```
print(kassenzettel.iloc[0:2, 0:3])
```

Achtung: hier darf die Bereichsangabe nicht selber wieder in eckigen Klammern stehen.

Zugriff über den Inhalt

Eine bequeme Notation, um Zeilen über den Inhalt auszuwählen, ist die Verwendung von Vergleichsoperatoren beim Zugriff auf den DataFrame:

```
kassenzettel[kassenzettel['Preis'] < 1.0]
```

gibt alle Zeilen, deren Spalte 'Preis' einen Wert kleiner als 1.0 hat, zurück.

Komplexer Zugriff über den Inhalt

Die Bedingungen zur Auwahl von Zeilen können komplex sein. Dazu müssen die einzelnen Bestandteile geklammert werden, und als logische Operatoren & für UND und | für ODER verwendet werden.

```
kassenzettel[
    (kassenzettel['Preis'] > 1.0) &
    (kassenzettel['Gewicht'] < 1000)
]
```

Zeilenweise iterieren

Um einen Dataframe zeilenweise zu durchlaufen, verwendet man die Funktion `iterrows()`:

```
for index, zeile in kassenzettel.iterrows():
    print(f"{index + 1}: {zeile['Artikel']}")
```

druckt zeilenweise eine Zeilennummer (beginnend bei 1, wegen der Lesbarkeit) und den entsprechenden Artikel des Kassenzettels aus. Die Funktion `iterrows()` gibt sowohl den Index (beginnend bei 0) als auch den Inhalt der Zeile zurück.

Spalte hinzufügen

Um eine Spalte in einem Dataframe hinzuzufügen, reicht es, die Spalte und ihren Inhalt anzugeben, z. B.

```
kassenzettel['Warengruppe'] = ['Süßw.', 'Süßw.', 'Obst']
```

Damit bekommt der Kassenzettel eine neue Spalte mit dem Titel `Warengruppe` und passenden Werten.

Zeile hinzufügen

Um Zeilen zu einem Dataframe hinzuzufügen muss man den Inhalt der Zeilen erst in einen eigenen Dataframe konvertieren und dann mit dem Befehl `concat()` an den bestehenden Dataframe anfügen. Dabei müssen die Spalten des bestehenden Dataframes übernommen werden:

```
zeile = pd.DataFrame(
    columns=kassenzettel.columns,
    data=['Äpfel', '1500', '4,99', 'B'])
kassenzettel=pd.concat(
    [kassenzettel, zeile], ignore_index=True)
```

Mit `ignore_index=True` wird erreicht, dass der Index des verlängerten Dataframes fortgezählt wird.

Aufgaben

Lösen Sie die folgenden Aufgaben:

- Summieren Sie die Preise zum Endbetrag
- Berechnen Sie den Durchschnittspreis aller Artikel
- Berechnen Sie den Kilopreis aller Artikel
- Berechnen Sie die Anzahl Spalten des Dataframes
- Berechnen Sie den MWSt-Betrag des Preises ('B' bedeutet 7%MWSt.)

Vergleich Liste, Dictionary, DataFrame

Wir kennen nun einige Datenstrukturen für viele Werte:

- `List` für einfache Listen
- `Dictionary` für Schlüssel-Wert-Paare
- `DataFrame` für Tabellen

Wozu braucht man diese verschiedenen Datenstrukturen?
Mein Schlüssel zum Verständnis liegt in einer immer 'natürlicheren' Beschreibung der Daten, die wir bearbeiten wollen.

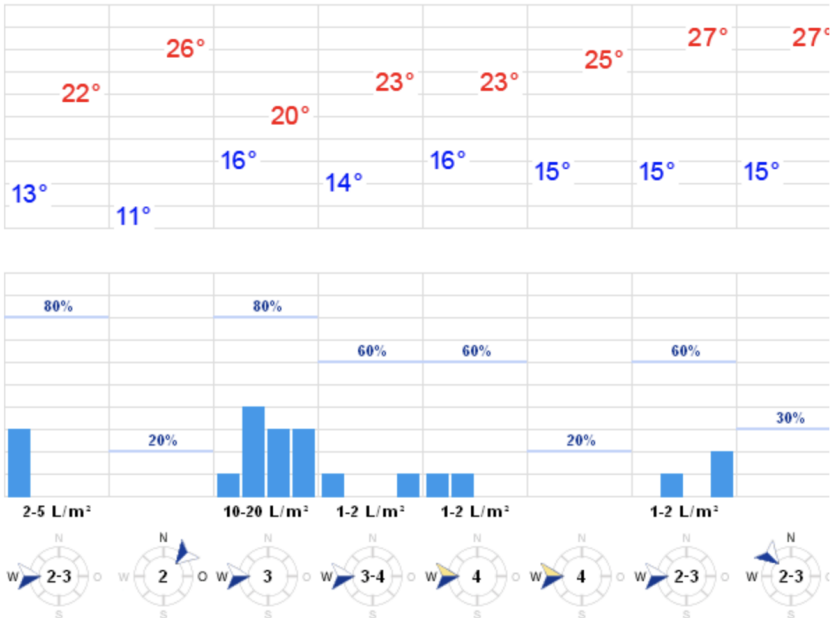
Vergleich Liste, Dictionary, DataFrame

'natürlicher' – was heißt das?

- die verwendeten Begriffe sollen aus dem Anwendungsgebiet kommen
- relevante Beziehungen zwischen den Daten müssen erhalten bleiben
- eine maschinelle Umsetzung muss möglich sein

Datenmodellierung versucht diese Anforderungen in Einklang zu bringen.

Aufgabe: eigenen Dataframe erstellen



Übertragen Sie die Wettergrafik (`wetteronline.de` vom 28.06.2023 für München) in eine Tabelle.

- Speichern Sie diese als Tabulator-getrennte Textdatei und laden Sie die Tabelle anschließend in einen Dataframe, oder
- Legen Sie gleich einen Dataframe an.

Das notwendige Wissen haben Sie nun!

Aufgabe: DataFrame für den Wetterbericht

Hier ein Vorschlag für einen solchen Dataframe:

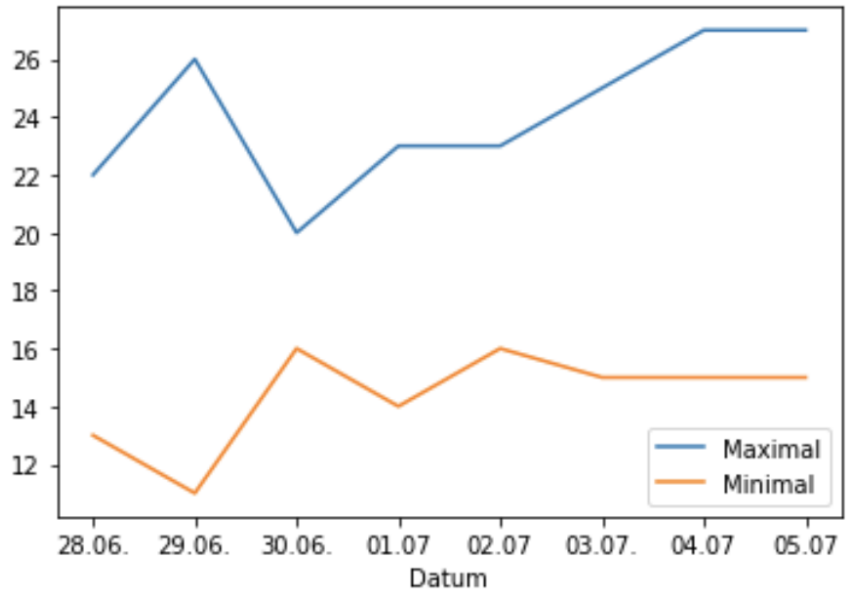
```
maxTemp = [22, 26, 20, 23, 23, 25, 27, 27]
minTemp = [13, 11, 16, 14, 16, 15, 15, 15]
sonnenstunden = [7, 16, 3, 7, 6, 13, 7, 8]
datum = ['28.06.', '29.06.', '30.06.', '01.07',
   '02.07', '03.07.', '04.07', '05.07']
tage = ['Mi', 'Do', 'Fr', 'Sa', 'So', 'Mo', 'Di', 'Mi']
wind = [2, 2, 3, 3, 4, 4, 2, 2]

vorhersage = pd.DataFrame({
   "Tage" : tage,
   "Datum" : datum,
   "Maximal" : maxTemp,
   "Minimal" : minTemp,
   "Sonne" : sonnenstunden},
   "Wind" : wind)
```

Und nun noch zeichnen...

Die Library `pandas` kann auch zeichnen:

```
vorhersage.plot(x="Datum", y=["Maximal", "Minimal"])
```

zeichnet ein Kurvendiagramm der Maximal- und Minimaltemperaturen der Wettervorhersage:



Lesen Sie die Dokumentation von Pandas und entdecken Sie die Vielfalt der grafischen Möglichkeiten!