DataFrame (1/3)

Ein einfaches Anlegen eines `DataFrame` mit 2 Spalten:

```
1 data = [[1, 'a'], [2, 'b'] , [ 3, 'c']]
2 df = pd.DataFrame(data, columns = ['zahlen', 'buchstaben'])
3 print (df)
```

Ergibt...

```
   zahlen buchstaben
0       1          a
1       2          b
2       3          c
```

Es kann natürlich wieder ein Index vergeben werden:

```
1 df = pd.DataFrame(data, columns = ['zahlen', 'buchstaben'],
2         index = ['z1', 'z2', 'z3'])
```

Ergibt...

```
    zahlen buchstaben
z1       1          a
z2       2          b
z3       3          c
```

Holen der Zeilenindices:

```
1 df.index
```

DataFrame (2/3)

Erzeugen eines `DataFrame` aus einem Dictionary:

```
data = {'Name':['Hubert', 'Herbert', 'Franz'],
        'Age':[28,42, 12]}
df= pd.DataFrame(data)
```

Erzeugen aus `Series` von Daten (mit Verknüpfung über den Index):

```
d = {'spalte1' : pd.Series([1, 2, 3], index=['x', 'y', 'z']),
        'spalte2' : pd.Series([1, 2, 3, 4], index=['x', 'y', 'z', 'a'])}
df = pd.DataFrame(d)
```

Ergibt...

	spalte1	spalte2
a	NaN	4
x	1.0	1
y	2.0	2
z	3.0	3

Zugriff auf die Spalte über den Namen (und über Zeilenposition):

```
print (df['spalte1']) # Nan 1.0 2.0 3.0
print (df['spalte1'][1]) # 1.0
```

Hinzufügen einer Neuen Spalte geht auch über den Namen

```
df['spalte3'] = pd.Series([100,200,300],index=['a','x','y'])
```

Löschen einer Spalte mit `del`:

```
del df['spalte1']
```

DataFrame (3/3)

Auswahl der Zeilen über den Indexnamen mit `loc`:

```
print(df.loc['a'])
print(df.loc[['a','x']]) # Beide Zeilen werden ausgewählt
```

Auswahl der Zeilen über die Position (en) mit `iloc`:

```
print(df.iloc[0]) # Auswahl der ersten Zeile
print(df.iloc[0:2]) # Auswahl der Zeilen 1 und 2 (: Operator wie gewohnt verwenden)
```

Zeilen hinzufügen mit `append`:

```
df2 = pd.DataFrame([[70., 80.], [71., 81.]] , columns=['spalte2', 'spalte3'])
df = df.append(df2) # Die Spalten werden gematched
```

Löschen von Spalten über den Index mit `drop`:

```
df = df.drop(df.index[0]) # Zuerst den Zeilennamen an der Stelle 0 holen
df = df.drop(df.index[1:3]) # Auch hier kann der : - Operator angewandt werden
df = df = df.drop(columns='Gemnr') # Geht auch mit Spaltennamen
```

Anzahl aller Zellen:

```
print('size:', df.size)
```

Übertragung in ein normales Array:

```
print('values:', df.values)
```

Anzeigen der ersten und letzten Zeilen (optional mit Angabe der Anzahl):

```
print(df.head()) # Die ersten Zeilen
print(df.tail(4)) # Die letzten 4 Zeilen
```

Panel

Werden wir nicht verwenden, deshalb hier nur Grundlegendes
Ist ab Python 3.7 auch nicht mehr im Sprachumfang von `Pandas` zu finden, man verwendet `xarray`

Importieren und Exportieren von Daten

Es gibt hier für alle gängigen Datentypen Importfunktionen. hier die wichtigsten:

- `pd.read_table()`: Die allgemeinste Form für das Einlesen von Zeichengetrennten Datensätzen
- `pd.read_csv()`: Diese ist fast identlisch zur obigen Methode, ist also nur ein Wrapper, der `pd.read_table()` aufruft
- d = pd.read_excel('data/bev_meld.xlsx'): Excel Daten können auch direkt eingelesen werden. Per default wird das erste Worksheet eingelesen, möchte man ein anderes setzt man den Parameter `sheet_name=1` für das 2. Sheet.

Der Export ist ähnlich einfach:

- `df.to_csv('bar.txt', sep='')`
  Weitere Parameter unter:
  `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html`
- `df.to_excel('bar.xls', sheet_name='Sheet1').`
  Weitere Parameter unter:
  `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_excel.html`
- `df.to_json`
- `df.to_html`
- `df.to_latex`
- `df.to_sql`: Schreibt in eine Datenbank unter Verwendung von `SQLAlchemy`

Sehr viele Formate für Import/Export werden unterstütutzt, hier ein Überblick:
`https://pandas.pydata.org/docs/reference/io.html`

Erstes Abfragen eines Dataframes

Grundsätzlicher Überblick über die Daten:

- `df.describe()`: Ein Grobüberblick über den Datensatz (inkl. einfacher Statistiken)
- `df.head(n)`: Zeige die ersten n-Zeilen (default: n=5)
- `df.tail(n)`: Zeige die letzten n-Zeilen.

Auswahl von Zeilen und Spalten:

- `print(df[3:]`: Auswahl von Zeilen wie gewohnt (hier: Zeile 3 bis zum Ende)
- `df.Spaltenname`: Zugriff Über den Spaltennamen möglich
- `df[['Spalte1', 'Spalte2']]`: Es können auch mehrere Spalten ausgewählt werden
- `df.iloc[:,3:]`: Über den Index kann man gleich mehrere Spalten auswählen (Alle Spalten ab dem Index 3)
- `df.iloc[:,[3,5,7]]`: Man kann auch eine Liste von Spalten angeben. Hier: 3, 5, 7
- `df.loc[:, df.columns != 'Bezirk']`: Alle Spalten ausser z.B. `Bezirk`.

Auswahl von Zeilen anhand von Kriterium:

- `df.loc[df.Bezirk == 'IL']`: Auswahl einzelner Zeilen (hier nur die wo Bezirk IL ist)
- `df.loc[df['Bezirk'].isin(['IL','IM'])]`: So kann man mehrere Bezirke auswählen

Zusammenfassen von Daten mit `groupby`

Die Methode groupby kann man als Vorbereitung für die Gruppierung verstehen. Mit dem erstellten können dann unterschiedliche Funktionen angewandt werden (wie z.B. `sum`):

```
1 bew_sum = df.groupby('Bezirk').sum() # es werden alle verbleibenenden
    Spalten in den Gruppen aufsummiert.
```

Möchte man nur bestimmte Spalten aufsummieren, muss man den Datensatz vorher entsprechend herrichten