19 DataFrame

Im vorigen Kapitel haben wir gesehen, dass der Datentyp `Series` logisch gesehen einer Spalte mit Index einer Excel-Tabelle entspricht. In diesem Kapitel geht es nun um den Datentyp `DataFrame`, den man sich nun wie eine komplette Excel-Tabelle vorstellen kann. Man kann also sagen, dass dieser Datentyp auf Tabellen basiert. Ein `DataFrame` besteht aus einer geordneten Sequenz von Spalten. Jede Spalte besteht aus einem eindeutigen Datentyp – wie eine `Series` –, aber verschiedene Spalten können verschiedene Typen haben. So könnte beispielsweise eine Spalte Verkaufszahlen als Float-Zahlen enthalten, während eine andere die zugehörigen Jahreszahlen als Integer-Zahlen enthalten könnte.

	A	B	C
1	Country	City	Population
2	England	London	8615246
3	Germany	Berlin	3562166
4	Spain	Madrid	3165235
5	Italy	Rome	2874038
6	France	Paris	2273305
7	Austria	Vienna	1805681
8	Romania	Bucharest	1803425
9	Germany	Hamburg	1760433
10	Hungary	Budapest	1754000
11	Poland	Warsaw	1740119
12	Spain	Barcelona	1602386
13	Germany	Munich	1493900
14	Italy	Milan	1350680

Bild 19.1 Spreadsheet und DataFrames

19.1 Zusammenhang zu Series

Ein DataFrame hat einen Zeilen- und einen Spaltenindex. Es ist wie ein Dictionary aus Series mit einem normalen Index. Jede Series wird über einen Index, d.h. Namen der Spalte angesprochen. Wir demonstrieren diesen Zusammenhang im folgenden Beispiel, in dem drei Series-Objekte definiert und zu einem `DataFrame` zusammengebaut werden:

```
import pandas as pd

years = range(2014, 2018)

shop1 = pd.Series([2409.14, 2941.01, 3496.83, 3119.55], index=years)
shop2 = pd.Series([1203.45, 3441.62, 3007.83, 3619.53], index=years)
shop3 = pd.Series([3412.12, 3491.16, 3457.19, 1963.10], index=years)
```

Was passiert, wenn diese „shop“-Series-Objekte konkateniert werden? Pandas stellt eine concat()-Funktion für diesen Zweck zur Verfügung:

```
pd.concat([shop1, shop2, shop3])
```

Ausgabe:

```
2014    2409.14
2015    2941.01
2016    3496.83
2017    3119.55
2014    1203.45
2015    3441.62
2016    3007.83
2017    3619.53
2014    3412.12
2015    3491.16
2016    3457.19
2017    1963.10
dtype: float64
```

Das Ergebnis ist wohl nicht das, was wir erwartet haben. Der Grund dafür ist, dass concat() für den axis-Parameter 0 verwendet. Probieren wir es mit „axis=1“:

```
shops_df = pd.concat([shop1, shop2, shop3], axis=1)
print(shops_df)
```

Ausgabe:

```
             0        1        2
2014  2409.14  1203.45  3412.12
2015  2941.01  3441.62  3491.16
2016  3496.83  3007.83  3457.19
2017  3119.55  3619.53  1963.10
```

Die Frage ist, von welchem Datentyp das Ergebnis ist:

```
print(type(shops_df))
```

Ausgabe:

```
<class 'pandas.core.frame.DataFrame'>
```

Das bedeutet, dass wir Series-Objekte durch Konkatenierung in DataFrame-Objekte wandeln können!

19.2 Manipulation der Spaltennamen

Wenn wir uns das eben erzeugte DataFrame anschauen, stört uns, dass die Spaltennamen durch wenig aussagekräftige Nummern bezeichnet werden, also 0, 1 und 2.

```
shops_df.columns
```

Ausgabe:

```
RangeIndex(start=0, stop=3, step=1)
```

```
shops_df.columns.values
```

Ausgabe:

```
array([0, 1, 2])
```

Wir sehen, dass das DataFrame-Objekt eine Property `columns` zur Verfügung stellt, um auf die Spalten zuzugreifen. Nehmen wir nun an, dass sich unsere Shops in Zürich, Winterthur

und Freiburg befinden. Dann wäre es sicherlich von Vorteil, diese Namen auch als Spaltennamen zu verwenden. Dazu müssen wir unsere Spaltennamen umbenennen, was wir über die Property `columns` bewerkstelligen können:

```
cities = ["Zürich", "Winterthur", "Freiburg"]
shops_df.columns = cities
print(shops_df)
```

Ausgabe:

```
      Zürich  Winterthur  Freiburg
2014  2409.14     1203.45   3412.12
2015  2941.01     3441.62   3491.16
2016  3496.83     3007.83   3457.19
2017  3119.55     3619.53   1963.10
```

Andererseits wäre eine Umbenennung in unserem Fall überhaupt nicht notwendig gewesen, wenn die Series bereits entsprechend benamt gewesen wären. Wir zeigen dies in folgendem Fall:

```
shop1.name = "Zürich"
shop2.name = "Winterthur"
shop3.name = "Freiburg"
shops_df2 = pd.concat([shop1, shop2, shop3], axis=1)
print(shops_df2)
```

Ausgabe:

```
      Zürich  Winterthur  Freiburg
2014  2409.14     1203.45   3412.12
2015  2941.01     3441.62   3491.16
2016  3496.83     3007.83   3457.19
2017  3119.55     3619.53   1963.10
```

19.3 Zugriff auf Spalten

Auf die Spalten eines DataFrames können wir einfach durch Indizierung zugreifen:

```
print(shops_df["Zürich"])
```

Ausgabe:

```
2014    2409.14
2015    2941.01
2016    3496.83
2017    3119.55
Name: Zürich, dtype: float64
```

Jede einzelne der Spalten entsprach ursprünglich einer Series und entspricht auch immer noch einer Series. Dies können wir sehen, wenn wir uns den Typ anschauen:

```
print(type(shops_df["Zürich"]))
```

Ausgabe:

```
<class 'pandas.core.series.Series'>
```

Pandas bietet noch eine syntaktisch deutlich einfachere Methode, auf die Spalten zuzugreifen. Die Spaltennamen wurden dazu als Properties implementiert, und dies bedeutet,

dass man einfach den Spaltennamen mittels Punkt an das DataFrame anhängen kann, um die entsprechende Spalte anzusprechen.

```
print(shops_df.Zürich)
```

Ausgabe:

```
2014    2409.14
2015    2941.01
2016    3496.83
2017    3119.55
Name: Zürich, dtype: float64
```

19.4 DataFrames aus Dictionaries

Die Zugriffe auf die Spalten eines DataFrames erinnern an den Zugriff bei Dictionaries. Spaltenname wäre der Key, und die Werte der Spalte entsprechen dann den Werten eines Dictionary. So ist es nicht verwunderlich, dass man aus folgendem Dictionary `cities` auf direkte Art ein DataFrame erzeugen kann:

```
cities = {"name": ["London", "Berlin", "Madrid", "Rome",
                   "Paris", "Vienna", "Bucharest", "Hamburg",
                   "Budapest", "Warsaw", "Barcelona",
                   "Munich", "Milan"],
          "population": [8615246, 3562166, 3165235, 2874038,
                         2273305, 1805681, 1803425, 1760433,
                         1754000, 1740119, 1602386, 1493900,
                         1350680],
          "country": ["England", "Germany", "Spain", "Italy",
                      "France", "Austria", "Romania",
                      "Germany", "Hungary", "Poland", "Spain",
                      "Germany", "Italy"]}

city_frame = pd.DataFrame(cities)
print(city_frame)
```

Ausgabe:

```
         name  population  country
0      London     8615246  England
1      Berlin     3562166  Germany
2      Madrid     3165235    Spain
3        Rome     2874038    Italy
4       Paris     2273305   France
5      Vienna     1805681  Austria
6   Bucharest     1803425  Romania
7     Hamburg     1760433  Germany
8    Budapest     1754000  Hungary
9      Warsaw     1740119   Poland
10  Barcelona     1602386    Spain
11     Munich     1493900  Germany
12      Milan     1350680    Italy
```

19.5 Index ändern

Bei der Erzeugung des DataFrames `city_frame` wurde automatisch der Index 0,1,2,... erzeugt. Bei der Erzeugung können wir aber auch einen eigenen Index verwenden:

```
ordinals = ["first", "second", "third", "fourth",
            "fifth", "sixth", "seventh", "eigth",
            "ninth", "tenth", "eleventh", "twelvth",
            "thirteenth"]
city_frame = pd.DataFrame(cities, index=ordinals)
print(city_frame)
```

Ausgabe:

```
                 name  population  country
first          London     8615246  England
second         Berlin     3562166  Germany
third          Madrid     3165235    Spain
fourth           Rome     2874038    Italy
fifth           Paris     2273305   France
sixth          Vienna     1805681  Austria
seventh     Bucharest     1803425  Romania
eigth         Hamburg     1760433  Germany
ninth        Budapest     1754000  Hungary
tenth          Warsaw     1740119   Poland
eleventh    Barcelona     1602386    Spain
twelvth        Munich     1493900  Germany
thirteenth      Milan     1350680    Italy
```

19.5.1 Umsortierung der Spalten

Die Reihenfolge der Spalten kann zum Zeitpunkt der Erstellung des DataFrames explizit festgelegt werden. Dazu dient der Schlüsselwortparameter `columns`:

```
city_frame = pd.DataFrame(cities,
                          columns = ["name",
                                     "country",
                                     "population"])
print(city_frame)
```

Ausgabe:

```
          name  country  population
0       London  England     8615246
1       Berlin  Germany     3562166
2       Madrid    Spain     3165235
3         Rome    Italy     2874038
4        Paris   France     2273305
5       Vienna  Austria     1805681
6    Bucharest  Romania     1803425
7      Hamburg  Germany     1760433
8     Budapest  Hungary     1754000
9       Warsaw   Poland     1740119
10  Barcelona    Spain     1602386
11     Munich  Germany     1493900
12      Milan    Italy     1350680
```

Im Folgenden ändern wir sowohl die Spaltenreihenfolge als auch die Indexreihenfolge mit der Funktion `reindex`:

```
city_frame.reindex(index=[0, 2, 4, 6,  8, 10, 12, 1, 3, 5, 7, 9, 11],
                   columns=['country', 'name', 'population'])
```

Ausgabe:

```
    country       name  population
0   England     London     8615246
2     Spain     Madrid     3165235
4    France      Paris     2273305
6   Romania  Bucharest     1803425
8   Hungary   Budapest     1754000
10    Spain  Barcelona     1602386
12    Italy      Milan     1350680
1   Germany     Berlin     3562166
3     Italy       Rome     2874038
5   Austria     Vienna     1805681
7   Germany    Hamburg     1760433
9    Poland     Warsaw     1740119
11  Germany     Munich     1493900
```

Jetzt wollen wir die Spalten umbenennen. Dafür verwenden wir die DataFrame-Methode `rename`. Die Methode unterstützt folgende Konventionen:

- (index=index_mapper, columns=columns_mapper, ...)
- (mapper, axis={'index', 'columns'}, ...)

Wir benennen nun im folgenden Beispiel die Spalten unseres DataFrames in rumänische Bezeichnungen um. Den Parameter `inplace` setzen wir auf `True`, damit das DataFrame-Objekt direkt geändert und kein neues erzeugt wird. Der Default-Wert für den Parameter `inplace` ist `False`.

```
city_frame.rename(columns={"name":"Nume",
                           "country":"țară",
                           "population":"populație"},
                  inplace=True)
print(city_frame)
```

Ausgabe:

```
         Nume     țară  populație
0      London  England    8615246
1      Berlin  Germany    3562166
2      Madrid    Spain    3165235
3        Rome    Italy    2874038
4       Paris   France    2273305
5      Vienna  Austria    1805681
6   Bucharest  Romania    1803425
7     Hamburg  Germany    1760433
8    Budapest  Hungary    1754000
9      Warsaw   Poland    1740119
10  Barcelona    Spain    1602386
11     Munich  Germany    1493900
12      Milan    Italy    1350680
```

19.5.2 Spalte in Index umfunktionieren

Der Index des vorigen Beispiels ist nicht besonders informationsträchtig. Im Prinzip zählt er lediglich die Zeilen auf. Man könnte sich auch beispielsweise den Ländernamen als Index wünschen. Nun zeigen wir, wie man direkt bei der Erzeugung eines DataFrames aus einem Dictionary den Index definieren kann. Dazu weisen wir dem Schlüsselwortparameter `index` den Wert von `cities['country']` zu.

```
city_frame = pd.DataFrame(cities,
                          columns=['name', 'population'],
                          index=cities['country'])

print(city_frame)
```

Ausgabe:

```
               name  population
England      London     8615246
Germany      Berlin     3562166
Spain        Madrid     3165235
Italy          Rome     2874038
France        Paris     2273305
Austria      Vienna     1805681
Romania   Bucharest     1803425
Germany     Hamburg     1760433
Hungary    Budapest     1754000
Poland       Warsaw     1740119
Spain     Barcelona     1602386
Germany      Munich     1493900
Italy         Milan     1350680
```

Wie sieht es aber aus, wenn ein DataFrame bereits existiert und wir den Index neu setzen wollen bzw. müssen? Zu diesem Zweck steht die Methode `set_index` zur Verfügung. Mit ihr lässt sich beispielsweise eine Spalte in einen Index wandeln:

```
city_frame = pd.DataFrame(cities)
city_frame2 = city_frame.set_index("country")
print(city_frame2)
```

Ausgabe:

```
               name  population
country
England      London     8615246
Germany      Berlin     3562166
Spain        Madrid     3165235
Italy          Rome     2874038
France        Paris     2273305
Austria      Vienna     1805681
Romania   Bucharest     1803425
Germany     Hamburg     1760433
Hungary    Budapest     1754000
Poland       Warsaw     1740119
Spain     Barcelona     1602386
Germany      Munich     1493900
Italy         Milan     1350680
```

Im vorherigen Beispiel haben wir gesehen, dass die Methode `set_index` ein neues DataFrame-Objekt liefert und nicht das originale Objekt verändert. Möchte man kein neues DataFrame erzeugen, sondern das bestehende direkt mit einem neuen Index ver-

sehen, so kann man den Parameter `inplace` auf `True` setzen. Dadurch wird dann das originale Objekt direkt verändert:

```
city_frame = pd.DataFrame(cities)
city_frame.set_index("country", inplace=True)
print(city_frame)
```

Ausgabe:

```
                name  population
country
England       London     8615246
Germany       Berlin     3562166
Spain         Madrid     3165235
Italy           Rome     2874038
France         Paris     2273305
Austria       Vienna     1805681
Romania    Bucharest     1803425
Germany      Hamburg     1760433
Hungary     Budapest     1754000
Poland        Warsaw     1740119
Spain      Barcelona     1602386
Germany       Munich     1493900
Italy          Milan     1350680
```

19.6 Selektion von Zeilen

Bis jetzt haben wir die DataFrame-Objekte über die Spalten indiziert, d.h. wir haben nur auf Spalten zugegriffen. Nun möchten wir demonstrieren, wie wir auch selektiv auf die Zeilen zugreifen können. Dazu verwenden wir die Locators `loc` und `iloc`.

Im ersten Beispiel erzeugen wir ein DataFrame, das nur aus den Zeilen besteht, in denen wir den Index „Germany“ haben:

```
city_frame = pd.DataFrame(cities,
                          columns=("name", "population"),
                          index=cities["country"])
print(city_frame.loc["Germany"])
```

Ausgabe:

```
             name  population
Germany    Berlin     3562166
Germany   Hamburg     1760433
Germany    Munich     1493900
```

Will man mehrere Indexwerte angeben, übergibt man diese als Liste an `loc`:

```
print(city_frame.loc[["Germany", "France"]])
```

Ausgabe:

```
             name  population
Germany    Berlin     3562166
Germany   Hamburg     1760433
Germany    Munich     1493900
France      Paris     2273305
```

Nun wählen wir alle Zeilen aus, in denen in einer Spalte eine Bedingung erfüllt ist, wenn also in unserem Beispiel die Bevölkerungsanzahl größer als zwei Millionen ist:

```
print(city_frame.loc[city_frame.population > 2000000])
```

Ausgabe:

```
            name  population
England  London     8615246
Germany  Berlin     3562166
Spain    Madrid     3165235
Italy      Rome     2874038
France    Paris     2273305
```

19.7 Summen und kumulative Summen

Mit der Methode `sum` kann man die Summe von allen Spalten eines DataTypes berechnen, was aber in unserem Fall wenig Sinn macht:

```
print(city_frame.sum())
```

Bei den Bevölkerungszahlen ergibt die Summation mehr Sinn.

```
city_frame["population"].sum()
```

Ausgabe:

```
33800614
```

Mit `cumsum` berechnen wir die kumulative Summe:

```
x = city_frame["population"].cumsum()
print(x)
```

Ausgabe:

```
England     8615246
Germany    12177412
Spain      15342647
Italy      18216685
France     20489990
Austria    22295671
Romania    24099096
Germany    25859529
Hungary    27613529
Poland     29353648
Spain      30956034
Germany    32449934
Italy      33800614
Name: population, dtype: int64
```

19.8 Spaltenwerte ersetzen

Das eben berechnete 'x' ist ein Series-Objekt mit der kumulativen Summe. Diese Series können wir der `population`-Spalte zuweisen und ersetzen damit die alten Werte. Im Fol-

genden nutzen wir die Methode `head`, die nur die ersten fünf Zeilen ausgibt, da dies zur Veranschaulichung des Prinzips genügt:

```
city_frame["population"] = x
print(city_frame.head())
```

Ausgabe:

```
            name  population
England   London     8615246
Germany   Berlin    12177412
Spain     Madrid    15342647
Italy       Rome    18216685
France     Paris    20489990
```

Anstelle die Werte in der `population`-Spalte komplett durch die kumulativen Summen zu ersetzen, wollen wir die neuen Werte als neue zusätzliche Spalte `cum_population` dem ursprünglichen DataFrame anfügen:

```
city_frame = pd.DataFrame(cities,
                          columns=["country",
                                   "population",
                                   "cum_population"],
                          index=cities["name"])

print(city_frame.head())
```

Ausgabe:

```
         country  population cum_population
London   England     8615246            NaN
Berlin   Germany     3562166            NaN
Madrid     Spain     3165235            NaN
Rome       Italy     2874038            NaN
Paris     France     2273305            NaN
```

Die neue Spalte `cum_population` enthält nur NaN-Werte, weil noch keine Daten zur Verfügung gestellt wurden.

Nun weisen wir die kumulativen Summen dieser neuen Spalte zu:

```
city_frame["cum_population"] = city_frame["population"].cumsum()
print(city_frame.head())
```

Ausgabe:

```
         country  population  cum_population
London   England     8615246         8615246
Berlin   Germany     3562166        12177412
Madrid     Spain     3165235        15342647
Rome       Italy     2874038        18216685
Paris     France     2273305        20489990
```

Bei der Erstellung eines DataFrame-Objektes aus einem Dictionary können auch Spalten angegeben werden, die nicht im Dictionary enthalten sind. In diesem Fall werden die Werte ebenfalls auf NaN gesetzt:

```
city_frame = pd.DataFrame(cities,
                          columns=["country",
                                   "area",
                                   "population"],
                          index=cities["name"])
print(city_frame.head())
```

Ausgabe:

```
         country area  population
London   England  NaN     8615246
Berlin   Germany  NaN     3562166
Madrid     Spain  NaN     3165235
Rome       Italy  NaN     2874038
Paris     France  NaN     2273305
```

In einem weiteren Schritt kann man dann die Werte für die Fläche in Form einer Liste bzw. eines Arrays an die Spalte `area` zuweisen:

```
# Flächen in qkm:
area = [1572, 891.85, 605.77, 1285,
        105.4, 414.6, 228, 755,
        525.2, 517, 101.9, 310.4,
        181.8]

city_frame["area"] = area
print(city_frame.head())
```

Ausgabe:

```
         country     area  population
London   England  1572.00     8615246
Berlin   Germany   891.85     3562166
Madrid     Spain   605.77     3165235
Rome       Italy  1285.00     2874038
Paris     France   105.40     2273305
```

19.9 Sortierung

DataFrames lassen sich anhand von bestimmten Kriterien sortieren. Im folgenden Beispiel sortieren wir den Inhalt des DataFrame-Objekts anhand der area-Werte in absteigender Größe:

```
city_frame = city_frame.sort_values(by="area", ascending=False)
print(city_frame)
```

Ausgabe:

```
           country     area  population
London     England  1572.00     8615246
Rome         Italy  1285.00     2874038
Berlin     Germany   891.85     3562166
Hamburg    Germany   755.00     1760433
Madrid       Spain   605.77     3165235
Budapest   Hungary   525.20     1754000
Warsaw      Poland   517.00     1740119
Vienna     Austria   414.60     1805681
Munich     Germany   310.40     1493900
Bucharest  Romania   228.00     1803425
Milan        Italy   181.80     1350680
Paris       France   105.40     2273305
Barcelona    Spain   101.90     1602386
```

Nehmen wir an, dass wir lediglich die Flächenwerte von London, Hamburg und Milan hätten. Die areas-Werte befinden sich in einem Series-Objekt mit den korrekten Indizes. Die Zuweisung funktioniert ebenfalls:

```
city_frame = pd.DataFrame(cities,
                          columns=["country",
                                   "area",
                                   "population"],
                          index=cities["name"])

some_areas = pd.Series([1572, 755, 181.8],
                       index=['London', 'Hamburg', 'Milan'])

city_frame['area'] = some_areas
print(city_frame)
```

Ausgabe:

```
           country    area  population
London     England  1572.0     8615246
Berlin     Germany     NaN     3562166
Madrid       Spain     NaN     3165235
Rome         Italy     NaN     2874038
Paris       France     NaN     2273305
Vienna     Austria     NaN     1805681
Bucharest  Romania     NaN     1803425
Hamburg    Germany   755.0     1760433
Budapest   Hungary     NaN     1754000
Warsaw      Poland     NaN     1740119
Barcelona    Spain     NaN     1602386
Munich     Germany     NaN     1493900
Milan        Italy   181.8     1350680
```

19.10 Spalten einfügen

In vorherigen Beispielen haben wir Spalten bei der Erstellung des DataFrames hinzugefügt. Es ist jedoch oft notwendig, Spalten direkt in ein bereits bestehendes DataFrame einzufügen.

```
city_frame = pd.DataFrame(cities,
                          columns = ["country",
                                     "population"],
                          index = cities["name"])


city_frame.insert(loc = 1,
                  column = 'area',
                  value = area)
print(city_frame)
```

Ausgabe:

```
           country     area  population
London     England  1572.00     8615246
Berlin     Germany   891.85     3562166
Madrid       Spain   605.77     3165235
```

```
Rome         Italy  1285.00     2874038
Paris       France   105.40     2273305
Vienna     Austria   414.60     1805681
Bucharest  Romania   228.00     1803425
Hamburg    Germany   755.00     1760433
Budapest   Hungary   525.20     1754000
Warsaw      Poland   517.00     1740119
Barcelona    Spain   101.90     1602386
Munich     Germany   310.40     1493900
Milan        Italy   181.80     1350680
```

19.11 DataFrame und verschachtelte Dictionaries

Verschachtelte Dictionaries können ebenfalls an ein DataFrame übergeben werden. Die Indizes des äußeren Dictionarys entsprechen den Spalten, und die inneren Schlüssel der Dictionaries entsprechen den Indizes der einzelnen Zeilen:

```
growth = {"Switzerland": {"2010": 3.0,
                          "2011": 1.8,
                          "2012": 1.1,
                          "2013": 1.9},
          "Germany": {"2010": 4.1,
                      "2011": 3.6,
                      "2012": 0.4,
                      "2013": 0.1},
          "France": {"2010": 2.0,
                     "2011": 2.1,
                     "2012": 0.3,
                     "2013": 0.3},
          "Greece": {"2010": -5.4,
                     "2011": -8.9,
                     "2012": -6.6,
                     "2013": -3.3},
          "Italy": {"2010": 1.7,
                    "2011": 0.6,
                    "2012": -2.3,
                    "2013": -1.9}
          }
growth_frame = pd.DataFrame(growth)
print(growth_frame)
```

Ausgabe:

```
      Switzerland  Germany  France  Greece  Italy
2010          3.0      4.1     2.0    -5.4    1.7
2011          1.8      3.6     2.1    -8.9    0.6
2012          1.1      0.4     0.3    -6.6   -2.3
2013          1.9      0.1     0.3    -3.3   -1.9
```

Sie möchten vielleicht die Jahre als Spalten und die Länder als Zeilen? Eine Vertauschung von Index und Spalten ist mittels `transpose` ganz einfach zu realisieren:

```
print(growth_frame.transpose())
```

Ausgabe:

```
              2010  2011  2012  2013
Switzerland    3.0   1.8   1.1   1.9
Germany        4.1   3.6   0.4   0.1
France         2.0   2.1   0.3   0.3
Greece        -5.4  -8.9  -6.6  -3.3
Italy          1.7   0.6  -2.3  -1.9
```

Statt `transpose()` kann man auch einfach die Property-Schreibweise `T` verwenden:

```
print(growth_frame.T)
```

Ausgabe:

```
              2010  2011  2012  2013
Switzerland    3.0   1.8   1.1   1.9
Germany        4.1   3.6   0.4   0.1
France         2.0   2.1   0.3   0.3
Greece        -5.4  -8.9  -6.6  -3.3
Italy          1.7   0.6  -2.3  -1.9
```

```
growth_frame = growth_frame.T
growth_frame2 = growth_frame.reindex(["Switzerland",
                                      "Italy",
                                      "Germany",
                                      "Greece"])
print(growth_frame2)
```

Ausgabe:

```
              2010  2011  2012  2013
Switzerland    3.0   1.8   1.1   1.9
Italy          1.7   0.6  -2.3  -1.9
Germany        4.1   3.6   0.4   0.1
Greece        -5.4  -8.9  -6.6  -3.3
```

19.12 Aufgaben

1. Aufgabe:

Erzeugen Sie ein DataFrame, das wie folgt aussieht:

```
Vienna   country           Austria
         area                414.6
         population        1805681
Hamburg  country           Germany
         area                  755
         population        1760433
Berlin   country           Germany
         area                891.85
         population        3562166
Zürich   country       Switzerland
         area                 87.88
         population         378884
dtype: object
```

2. Aufgabe:

Vertauschen Sie die Indices der vorigen Series.

3. Aufgabe:

Erzeugen Sie ein beliebiges DataFrame mit einem Index, der aus Vornamen besteht, sowie einer Spalte für das Gewicht und einer Spalte für die Körpergröße.

Extrahieren Sie dann alle Zeilen, deren BMI[1] im normalen Bereich liegt, d.h. zwischen 18.5 und 25.

Es gilt:

$$BMI = \frac{g}{h**2}$$

4. Aufgabe:

Geben Sie die Zeilen aus, in deren Namen ein kleines „i“ vorkommt.

5. Aufgabe:

Fügen Sie an den in Aufgabe 3 erzeugten DataFrame eine Zeile mit dem BMI an.

6. Aufgabe:

Geben Sie das in der letzten Aufgabe erzeugte DataFrame absteigend sortiert nach dem BMI aus.

7. Aufgabe:

Geben Sie nun alle Zeilen aus, deren BMI-Werte innerhalb von 18.5 und 23 liegen und deren Vornamen ein „a“ enthalten.

8. Aufgabe:

Erzeugen Sie ein DataFrame mit einem Index, der aus den Monatsnamen besteht, und die Spaltennamen Vornamen entsprechen. Füllen Sie nun die Spalten mit zufälligen ganzen Zahlen zwischen 120 und 200.

9. Aufgabe:

Kehren Sie das eben erzeugte DataFrame um, sodass der Index den Vornamen und die Spaltennamen den Monatsnamen entsprechen.

19.13 Lösungen

Lösung zur 1. Aufgabe:

```
import pandas as pd

cities = ["Vienna", "Vienna", "Vienna",
          "Hamburg", "Hamburg", "Hamburg",
          "Berlin", "Berlin", "Berlin",
          "Zürich", "Zürich", "Zürich"]

data = ["Austria", 414.60,    1805681,
        "Germany", 755.00,    1760433,
        "Germany", 891.85,    3562166,
        "Switzerland", 87.88, 378884]

index = [cities, ["country", "area", "population",
                  "country", "area", "population",
                  "country", "area", "population",
                  "country", "area", "population"]]

city_series = pd.Series(data, index=index)
print(city_series)
```

Ausgabe:

```
Vienna   country           Austria
         area                414.6
         population        1805681
Hamburg  country           Germany
         area                  755
         population        1760433
Berlin   country           Germany
         area                891.85
         population        3562166
Zürich   country       Switzerland
         area                87.88
         population         378884
dtype: object
```

Lösung zur 2. Aufgabe:

```
city_series = city_series.sort_index()

city_series = city_series.swaplevel()
city_series.sort_index(inplace=True)
print(city_series)
```

Ausgabe:

```
area        Berlin          891.85
            Hamburg            755
            Vienna           414.6
            Zürich           87.88
country     Berlin         Germany
            Hamburg        Germany
            Vienna         Austria
            Zürich     Switzerland
```

```
population  Berlin         3562166
            Hamburg        1760433
            Vienna         1805681
            Zürich          378884
dtype: object
```

Lösung zur 3. Aufgabe:

```
import pandas as pd
persons = { "Name" : ["Henry", "Sarah", "Elke",
                      "Lulu", "Vera", "Toni",
                      "Maria", "Chris"],
            "Größe" : [179, 165, 172, 154, 150,
                       189, 176, 175],
            "Gewicht" : [65, 58, 58, 45, 43, 99, 68, 60]
          }

pdf = pd.DataFrame(persons,
                   columns = ["Gewicht", "Größe"],
                   index=persons["Name"])

bmi = (pdf.Gewicht / ((pdf.Größe/100) ** 2))
bmi_okay = pdf.loc[(20 < bmi) & (bmi < 25)]
print(bmi_okay)
```

Ausgabe:

```
        Gewicht  Größe
Henry       65    179
Sarah       58    165
Maria       68    176
```

Lösung zur 4. Aufgabe:

```
pdf.loc[pdf.index.str.contains("i")]
```

Ausgabe:

```
         Gewicht  Größe
 Toni        99    189
 Maria       68    176
 Chris       60    175
```

Lösung zur 5. Aufgabe:

```
pdf.insert(loc=len(pdf.columns),
           column="BMI",
           value=(pdf.Gewicht / ((pdf.Größe/100) ** 2)))
pdf
```

Ausgabe:

```
         Gewicht  Größe        BMI
 Henry       65    179  20.286508
 Sarah       58    165  21.303949
```

```
Elke        58    172  19.605192
Lulu        45    154  18.974532
Vera        43    150  19.111111
Toni        99    189  27.714790
Maria       68    176  21.952479
Chris       60    175  19.591837
```

Lösung zur 6. Aufgabe:

```
pdf.sort_values(by="BMI", ascending=False)
```

Ausgabe:

```
        Gewicht  Größe        BMI
Toni        99    189  27.714790
Maria       68    176  21.952479
Sarah       58    165  21.303949
Henry       65    179  20.286508
Elke        58    172  19.605192
Chris       60    175  19.591837
Vera        43    150  19.111111
Lulu        45    154  18.974532
```

Lösung zur 7. Aufgabe:

```
bmi_okay = (18.5 < pdf['BMI']) & (pdf['BMI'] < 23.5)
name_contains_a = pdf.index.str.contains('a')
print(pdf.loc[bmi_okay & name_contains_a])
```

Ausgabe:

```
       Gewicht  Größe        BMI
Sarah       58    165  21.303949
Vera        43    150  19.111111
Maria       68    176  21.952479
```

Lösung zur 8. Aufgabe:

```
import numpy as np
import pandas as pd

names = ['Jonas', 'Leon', 'Finn', 'Guido',
         'Lara', "Hannah", "Mila", "Lina"]
index = ["Januar", "Februar", "März",
         "April", "Mai", "Juni",
         "Juli", "August", "September",
         "Oktober", "November", "Dezember"]
df = pd.DataFrame(np.random.randint(120,
                                    200,
                                    size=(len(index),
                                          len(names))),
                  columns = names,
                  index = index)
print(df)
```

Ausgabe:

```
             Jonas  Leon  ...   Mila  Lina
Januar         175   129  ...    173   190
Februar        124   124  ...    171   163
März           196   168  ...    176   192
April          172   183  ...    152   160
Mai            143   167  ...    197   154
Juni           170   159  ...    177   143
Juli           192   134  ...    127   182
August         155   138  ...    167   166
September      172   170  ...    182   136
Oktober        157   196  ...    122   163
November       188   183  ...    134   144
Dezember       193   138  ...    171   180

[12 rows x 8 columns]
```

Lösung zur 9. Aufgabe:

```
new_df = df.transpose()
print(new_df)
```

Ausgabe:

```
          Januar  Februar    ...     November  Dezember
Jonas        175       124    ...          188       193
Leon         129       124    ...          183       138
Finn         188       136    ...          181       180
Guido        187       180    ...          195       130
Lara         132       171    ...          194       166
Hannah       156       157    ...          174       133
Mila         173       171    ...          134       171
Lina         190       163    ...          144       180

[8 rows x 12 columns]
```