Merge und Join

by Woche 14

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In diesem Kapitel werden wir uns mit dem Zusammenführen von Daten, also von mehreren DataFrames beschäftigen. Dazu lernen wir die Funktionen `pd.concat()`, `pd.merge()` und `pd.join()` kennen.

Concat

Die Funktion `pd.concat()` (*concatinate*; verketten) ermöglicht es, zwei oder mehr DataFrames sozusagen schlichtweg aufeinander bzw. nebeneinander *zu kleben*. Standardmäßig können wir einfach DataFrames als Liste übergeben und sie werden aufeinander *geklebt/gestapelt*.

```
df1 = pd.DataFrame({
  'SpA': [1, 2],
  'SpB': [10, 11]
})

df1
```

```
   SpA  SpB
0    1   10
1    2   11
```

```
df2 = pd.DataFrame({
  'SpA': [3, 4],
  'SpB': [12, 13]
})

df2
```

```
   SpA  SpB
0    3   12
1    4   13
```

```
#



pd.concat([df1, df2])
```

```
   SpA  SpB
0    1   10
1    2   11
0    3   12
1    4   13
```

Es sei darauf hingewiesen, dass die Indizes der DataFrames beibehalten werden. Auch sie werden also einfach *aufeinander geklebt*. So kommt es, dass wir oben nun zwei Zeilen mit Index 0 und 1 haben. Natürlich kann man die Indices weiterhin mit `.iloc()` und der Zeilennummer 0-3 ansprechen, aber die Indexlabel, auf welche man mit `.loc()` zugreift, sind nun doppelt vorhanden. Um dies zu verhinden, könnten wir nachträglich `.index.reset()` anwenden oder aber direkt innerhalb von `pd.concat()` das Argument `ignore_index=True` setzen.

Ebenfalls nützlich ist das Argument `keys=` mit dem wir dafür sorgen können, dass auch nach dem Zusammenführen der DataFrames noch erkennbar ist, aus welchem DataFrame die Daten stammen. Diese Keys werden allerdings mal wieder standardmäßig als MultiIndex gesetzt. Falls dies nicht gewünscht ist, müsste wiederum `.reset_index()` angehängt werden.

```
pd.concat(
  [df1, df2],
  ignore_index=True
  )
```

```
   SpA  SpB
0    1   10
1    2   11
2    3   12
3    4   13
```

```
pd.concat(
  [df1, df2],
  keys=['df1', 'df2']
  )
```

```
       SpA  SpB
df1 0    1   10
    1    2   11
df2 0    3   12
    1    4   13
```

```
(
  pd.concat(
    [df1, df2],
    keys=['df1', 'df2']
    )
    .reset_index()
    .rename(columns={
    'level_0': 'Quelle',
    'level_1': 'Quelle_idx'
    })
)
```

```
  Quelle  Quelle_idx  SpA  SpB
0    df1           0    1   10
1    df1           1    2   11
2    df2           0    3   12
3    df2           1    4   13
```

Man kann `pd.concat()` übrigens auch nutzen, falls die DataFrames unterschiedliche Spalten haben. In diesem Fall werden die fehlenden Spalten einfach mit `NaN` aufgefüllt.

```
df3 = pd.DataFrame({
  'SpA': [5, 6],
  'SpC': [14, 15]
})

df3
```

```
   SpA  SpC
0    5   14
1    6   15
```

```
df4 = pd.DataFrame({
  'SpD': [16, 17]
})


df4
```

```
   SpD
0   16
1   17
```

```
pd.concat(
  [df1, df2, df3, df4]
  )
```

```
   SpA   SpB   SpC   SpD
0  1.0  10.0   NaN   NaN
1  2.0  11.0   NaN   NaN
0  3.0  12.0   NaN   NaN
1  4.0  13.0   NaN   NaN
0  5.0   NaN  14.0   NaN
1  6.0   NaN  15.0   NaN
0  NaN   NaN   NaN  16.0
1  NaN   NaN   NaN  17.0
```

Und schließlich können wir die DataFrames auch nebeneinander statt aufeinander *kleben*. Dazu müssen wir das Argument `axis=` setzen. Standardmäßig ist `axis=0` (`axis='rows'` oder `axis='index'` bewirken dasselbe), was bedeutet, dass die DataFrames aufeinander gestapelt werden. Mit `axis=1` (oder `axis='columns'`) werden die DataFrames nebeneinander geklebt.

```
pd.concat(
  [df1, df3],
  axis='rows'
  )
```

```
   SpA   SpB   SpC
0    1  10.0   NaN
1    2  11.0   NaN
0    5   NaN  14.0
1    6   NaN  15.0
```

```
pd.concat(
  [df1, df3],
  axis='columns'
  )
```

```
   SpA  SpB  SpA  SpC
0    1   10    5   14
1    2   11    6   15
```

Merge/Join

In der Praxis muss man häufig Daten zusammenführen, die nicht einfach nur aufeinander gestapelt oder nebeneinander geklebt werden können. Stattdessen müssen sie anhand von gemeinsamen Spaltenwerten/Indices zusammengeführt werden. Diese Art des Zusammenführens wird nicht nur in Python als *Merge* oder *Join* bezeichnet. Tatsächlich gibt es sowohl `pd.merge()` als auch `pd.join()` in Pandas. Wir werden uns hier auf `pd.merge()` konzentrieren, da es flexibler ist und mehr Optionen bietet, wohingegen `pd.join()` nur eine spezielle Form des Mergens, also ein Shortcut für einen bestimmten Fall, ist. Abgesehen von diesen beiden expliziten Funktionen werden die Ausdrücke “Zwei DataFrames mergen” und “Zwei DataFrames joinen” oft synonym verwendet.

Gehen wir erstmal davon aus, dass es zwei Dataframes (`df_x` & `df_y`) gibt, die eine gemeinsame Spalte (`id`) haben. Außer dieser gemeinsamen Spalte haben sie jeweils noch eine weitere Spalte (`x` bzw. `y`), mit Daten die der jeweils andere Dataframe nicht hat. Davon ausgehend, dass zumindest einige ids in beiden Dataframes vorkommen, können wir die Dataframes anhand der gemeinsamen Spalte zusammenführen. In jedem Fall ginge dies mit

- `pd.merge(df_x, df_y, on='id', how='...')` oder
- `df_x.merge(df_y, on='id', how='...')`,

allerdings gibt es dabei überraschend viele Möglichkeiten wie genau (`how=`) dies passieren soll.

Inner Join

Die erste hier erwähnte Art des Mergens/Joinens sei der *inner join*. Mit `how='inner'` werden nur die ids beibehalten, die in beiden Dataframes vorkommen.

```
df_x = pd.DataFrame({
  'id': [1, 2, 3],
  'x': ['x1','x2','x3']
})

df_x
```

```
   id   x
0   1  x1
```

```
1   2  x2
2   3  x3
```

```
df_y = pd.DataFrame({
  'id': [1, 2, 4],
  'y': ['y1','y2','y4']
})

df_y
```

```
   id   y
0   1  y1
1   2  y2
2   4  y4
```

```
pd.merge(
  df_x, df_y,
  on='id',
  how='inner'
  )
#
```

```
   id   x   y
0   1  x1  y1
1   2  x2  y2
```

Left Join

Beim *left join* werden mit `how='left'` alle ids aus dem linken Dataframe (`df_x`) beibehalten, während diejenigen aus dem rechten Dataframe (`df_y`) die nicht im linken vorkommen, mit `NaN` aufgefüllt werden. In diesem Beispiel wird auch gezeigt, dass ggf. in einem Dataframe eine id mehrfach vorkommen kann. In diesem Fall wird die Zeile aus dem linken Dataframe für jede Zeile aus dem rechten Dataframe dupliziert.

```
df_x = pd.DataFrame({
  'id': [1, 2, 3],
  'x': ['x1','x2','x3']
})

df_x
```

```
   id   x
0   1  x1
1   2  x2
2   3  x3
```

```
df_y = pd.DataFrame({
  'id': [1, 2, 4, 2],
  'y': ['y1','y2','y4','y5']
})

df_y
```

```
   id   y
0   1  y1
1   2  y2
2   4  y4
3   2  y5
```

```
pd.merge(
  df_x, df_y,
  on='id',
  how='left'
  )
#
```

```
   id   x    y
0   1  x1   y1
1   2  x2   y2
2   2  x2   y5
3   3  x3  NaN
```

Right Join

Analog zum *left join* werden beim *right join* mit `how='right'` alle ids aus dem rechten Dataframe (`df_y`) beibehalten, während diejenigen aus dem linken Dataframe (`df_x`) die nicht im rechten vorkommen, mit `NaN` aufgefüllt werden.

```
df_x = pd.DataFrame({
  'id': [1, 2, 3],
  'x': ['x1','x2','x3']
})

df_x
```

```
   id   x
0   1  x1
1   2  x2
2   3  x3
```

```
df_y = pd.DataFrame({
  'id': [1, 2, 4],
  'y': ['y1','y2','y4']
})

df_y
```

```
   id   y
0   1  y1
1   2  y2
2   4  y4
```

```
pd.merge(
  df_x, df_y,
  on='id',
  how='right'
  )
#
```

```
   id    x   y
0   1   x1  y1
1   2   x2  y2
2   4  NaN  y4
```

Outer Join

Beim *outer join* (auch *full join* genannt) werden mit `how='outer'` alle ids aus beiden Dataframes beibehalten. Diejenigen, die in nur einem der beiden Dataframes vorkommen, werden mit `NaN` aufgefüllt.

```
df_x = pd.DataFrame({
  'id': [1, 2, 3],
  'x': ['x1','x2','x3']
})

df_x
```

```
   id   x
0   1  x1
1   2  x2
2   3  x3
```

```
df_y = pd.DataFrame({
  'id': [1, 2, 4],
  'y': ['y1','y2','y4']
})

df_y
```

```
   id   y
0   1  y1
1   2  y2
2   4  y4
```

```
pd.merge(
  df_x, df_y,
  on='id',
  how='outer'
  )
#
```

```
   id    x    y
0   1   x1   y1
1   2   x2   y2
2   3   x3  NaN
3   4  NaN   y4
```

ID-Spalten mit unterschiedlichen Namen

Bisher haben wir angenommen, dass die gemeinsame Spalte in beiden Dataframes den gleichen Namen hat, sodass wir diesen namen in `on=` übergeben können. Dies ist jedoch in der Praxis öfter mal nicht der Fall. Obwohl es sich also um dieselben IDs handelt, heißen die Spalten in den Dataframes unterschiedlich.

```
df_x = pd.DataFrame({
  'id_x': [1, 2, 3],
  'x': ['x1','x2','x3']
})

df_x
```

```
   id_x   x
0     1  x1
1     2  x2
2     3  x3
```

```
df_y = pd.DataFrame({
  'id_y': [1, 2, 4],
  'y': ['y1','y2','y4']
})

df_y
```

```
   id_y   y
0     1  y1
1     2  y2
2     4  y4
```

Zum Glück ist es aber auch nicht unbedingt notwendig, sodass wir also nicht die Spalten vorher umbenennen müssen. Mit `left_on=` und `right_on=` können wir explizit angeben, welche Spalten in den Dataframes zusammengeführt werden sollen. Allerdings sind dann ggf. doch noch Schritte in der Nachbearbeitung notwendig, um die Spalten zu bereinigen.

```
pd.merge(
  df_x, df_y,
  how='left',
  left_on='id_x',
  right_on='id_y'
  )


#
```

```
   id_x   x  id_y    y
0     1  x1   1.0   y1
1     2  x2   2.0   y2
2     3  x3   NaN  NaN
```

```
(
  df_x
  .merge(df_y,
    how='left',
    left_on='id_x',
```

```
    right_on='id_y')
  .drop(columns=['id_y'])
  .rename(columns={'id_x': 'id'})
)
```

```
   id   x    y
0   1  x1   y1
1   2  x2   y2
2   3  x3  NaN
```

Hier noch ein Tipp: Zum Beispiel bei diesem outer join sollten wir nicht denselben Ansatz zur Nachbereitung der Daten wie gerade anwenden (also `id_y` löschen und `id_x` umbenennen), sondern entweder doch die Spalten vorher umbenennen oder aber im Nachgang `.combine_first()` nutzen:

```
new_df = pd.merge(
  df_x, df_y,
  how='outer',
  left_on='id_x',
  right_on='id_y'
  )

new_df
```

```
   id_x    x  id_y    y
0   1.0   x1   1.0   y1
1   2.0   x2   2.0   y2
2   3.0   x3   NaN  NaN
3   NaN  NaN   4.0   y4
```

```
new_df['id'] = (
  new_df['id_x']
  .combine_first(new_df['id_y'])
  )

new_df = new_df[['id', 'x', 'y']]

new_df
```

```
    id    x    y
0  1.0   x1   y1
1  2.0   x2   y2
```

```
2  3.0   x3  NaN
3  4.0  NaN   y4
```

Mehrere gemeinsame Spalten

Bisher haben wir angenommen, dass es nur eine gemeinsame Spalte gibt. In der Praxis kann es aber auch vorkommen, dass es die Kombination aus mehreren Spalten ist, die die Daten eindeutig identifiziert. In diesem Fall können wir einfach eine Liste von Spaltennamen übergeben.

```
df_x = pd.DataFrame({
  'Vorname': ['Max', 'Loki', 'Loki'],
  'Nachname': ['Meier', 'Meier', 'Müller'],
  'Gehalt': [1000, 2000, 3000]
})

df_x
```

```
  Vorname Nachname  Gehalt
0     Max    Meier    1000
1    Loki    Meier    2000
2    Loki   Müller    3000
```

```
df_y = pd.DataFrame({
  'Vorname': ['Ute', 'Loki', 'Loki'],
  'Nachname': ['Meier', 'Meier', 'Schmidt'],
  'Rente': [500, 1000, 1500]
})

df_y
```

```
  Vorname Nachname  Rente
0     Ute    Meier    500
1    Loki    Meier   1000
2    Loki  Schmidt   1500
```

```
pd.merge(
  df_x, df_y,
  how='outer',
  on=['Vorname', 'Nachname']
  )
```

```
  Vorname Nachname  Gehalt   Rente
0    Loki    Meier  2000.0  1000.0
1    Loki   Müller  3000.0     NaN
2    Loki  Schmidt     NaN  1500.0
3     Max    Meier  1000.0     NaN
4     Ute    Meier     NaN   500.0
```

`join()` vs. `merge()`

Zum Abschluss sei noch erwähnt, dass `join()` ein Shortcut für `merge()` ist, wenn die gemeinsame Spalte der Dataframes der Index ist.

```
df_x = pd.DataFrame({
  'x': ['x1','x2','x3']},
  index=['Nord', 'Ost', 'West'])

df_x
```

```
       x
Nord  x1
Ost   x2
West  x3
```

```
df_y = pd.DataFrame({
  'y': ['y1','y2','y3','y4']},
  index=['Nord','Ost','Süd','West'])

df_y
```

```
       y
Nord  y1
Ost   y2
Süd   y3
West  y4
```

Tatsächlich kann man mit `pd.merge()` nämlich auch Dataframes mergen, die Indizes als gemeinsame Spalte haben. Dazu muss man `left_index=True` und `right_index=True` setzen anstatt auf spezifische Spaltennamen zu referenzieren. Anstatt dies zu tun kann aber eben auch `pd.join()` genutzt werden.

```
df_x.merge(
  df_y,
```

```
  left_index=True,
  right_index=True,
  how='outer'
  )
```

```
        x   y
Nord   x1  y1
Ost    x2  y2
Süd   NaN  y3
West   x3  y4
```

```
df_x.join(df_y, how='outer')



#
```

```
        x   y
Nord   x1  y1
Ost    x2  y2
Süd   NaN  y3
West   x3  y4
```

Weitere Ressourcen

- How to combine DataFrames in Pandas | Merge, Join, Concat, & Append

Übungen

Bringe die Informationen aus den folgenden beiden Datensätzen so zusammen, dass die Start_Saison und Tore in einer gemeinsamen Tabelle stehen und ausschließlich die Spieler, die in beiden Datensätzen vorkommen, angezeigt werden.

```
df1 = pd.DataFrame({
  'Start_Saison': [2008, 2011, 2008, 2014, 2010, 2021],
  'Vorname': ['Thomas', 'Manuel', 'Toni', 'Toni', 'David', 'David'],
  'Nachname': ['Müller', 'Neuer', 'Kroos', 'Kroos', 'Alaba', 'Alaba'],
  'Verein': ['Bayern', 'Bayern', 'Bayern', 'Madrid', 'Bayern', 'Madrid']
})

df2 = pd.DataFrame({
  'Verein': ['Bayern', 'Bayern', 'Madrid', 'Bayern', 'Madrid'],
```

```
  'Vorname': ['Thomas', 'Toni', 'Toni', 'David', 'David'],
  'Nachname': ['Müller', 'Kroos', 'Kroos', 'Alaba', 'Alaba'],
  'Tore': [149, 13, 22, 22, 3]
})
```

- (A) Geschafft