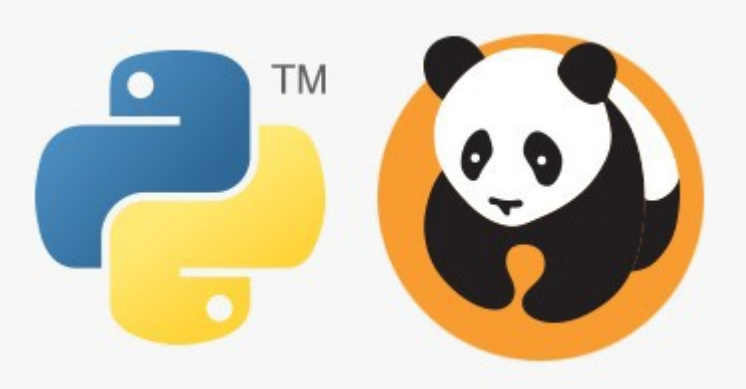
Pandas - Fortsetzung -



Bildquelle: www.medium.com

In diesem Notebook werden die apply Funktion besprochen. Des weiteren werden wir concaten, mergen und appenden von Dataframes durchführen.

In [1]:
```
# import of package
import pandas as pd
import numpy as np
```

read_csv()

In [2]:
```
# read dataframe
df_birth = pd.read_csv('birthdays.csv')
df_birth.head()
```

Out[2]:

	Unnamed: 0	state	year	month	day	date	wday	births
0	1	AK	1969	1	1	1969-01-01	Wed	14
1	2	AL	1969	1	1	1969-01-01	Wed	174
2	3	AR	1969	1	1	1969-01-01	Wed	78
3	4	AZ	1969	1	1	1969-01-01	Wed	84
4	5	CA	1969	1	1	1969-01-01	Wed	824

In [8]:
```
df_birth = df_birth.rename(columns = {'Unnamed: 0': 'Number'})
df_birth.head(3)
```

Out[8]:

	Number	state	year	month	day	date	wday	births
0	1	AK	1969	1	1	1969-01-01	Wed	14
1	2	AL	1969	1	1	1969-01-01	Wed	174
2	3	AR	1969	1	1	1969-01-01	Wed	78

In [3]:

```
# read dataframe in subfolder 'data'
df_fish = pd.read_csv('data/fish.csv')
df_fish.head()
```

Out[3]:

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

In [4]:

```
# read dataframe in another directory using '../' to jump level up
df_OL_rio = pd.read_csv( '../data/OL_Rio_de_Janeiro.csv')
df_OL_rio.head()
```

Out[4]:

	id	name	nationality	sex	dob	height	weight	sport	gold	silver	bronze
0	736041664	A Jesus Garcia	ESP	male	10/17/69	1.72	64.0	athletics	0	0	0
1	532037425	A Lam Shin	KOR	female	9/23/86	1.68	56.0	fencing	0	0	0
2	435962603	Aaron Brown	CAN	male	5/27/92	1.98	79.0	athletics	0	0	1
3	521041435	Aaron Cook	MDA	male	1/2/91	1.83	80.0	taekwondo	0	0	0
4	33922579	Aaron Gate	NZL	male	11/26/90	1.81	71.0	cycling	0	0	0

In [5]:

```
# read dataframe in another directory using '../' to jump level up
df_wine = pd.read_csv('../data/UCI/Wine/winequality-red.csv')
df_wine.head()
```

Out[5]:

	fixed acidity;"volatile acidity";"citric acid";"residual sugar";"chlorides";"free sulfur dioxide";"total sulfur dioxide";"density";"pH";"sulphates";"alcohol";"quality"
0	7.4;0.7;0;1.9;0.076;11;34;0.9978;3.51;0.56;9.4;5
1	7.8;0.88;0;2.6;0.098;25;67;0.9968;3.2;0.68;9.8;5

	fixed acidity;"volatile acidity";"citric acid";"residual sugar";"chlorides";"free sulfur dioxide";"total sulfur dioxide";"density";"pH";"sulphates";"alcohol";"quality"
2	7.8;0.76;0.04;2.3;0.092;15;54;0.997;3.26;0.65;...
3	11.2;0.28;0.56;1.9;0.075;17;60;0.998;3.16;0.58...

In [6]:
```
# Use of differen delimiter & read dataframe in another directory using '../'
df_wine = pd.read_csv('../data/UCI/Wine/winequality-red.csv', delimiter = ';')
df_wine.head()
```

Out[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	qu
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	

Data Description of Birthdays

Source: calmcode.io // originally found in an R package.

Purpose: Der Datensatz enthält Geburtstagsdaten über die United States pro State. Das Ziel ist zu versuchen und zu entdecken von interessanten Mustern in den Daten.

Contents:

Name	Descretion
state	Der US State indem derjenige geboren ist.
year	Das Jahr.
month	Der Monat.
day	Der Tag des Monats.
date	Das Datum.
wday	Der Wochentag.
births	Anzahl der Geburtstagskinder an diesem Tag

In [9]:
```
# delete column Unnamed
df_birth = df_birth.drop('Number', axis = 1)
df_birth.head()
```

Out[9]:

	state	year	month	day	date	wday	births
0	AK	1969	1	1	1969-01-01	Wed	14

	state	year	month	day	date	wday	births
1	AL	1969	1	1	1969-01-01	Wed	174
2	AR	1969	1	1	1969-01-01	Wed	78
3	AZ	1969	1	1	1969-01-01	Wed	84

Flower Brackets

Bildquelle: vecteezy.com

Ohne Klammern

Ohne Klammern werden Objekte aufgerufen, z.B. Spalten einer Tabelle, Variablen, ...

In [10]:
```
a = 7
a
```

Out[10]: 7

In [11]:
```
df_fish.Species
```

Out[11]:
```
0      Bream
1      Bream
2      Bream
3      Bream
4      Bream
       ...
154    Smelt
155    Smelt
156    Smelt
157    Smelt
158    Smelt
Name: Species, Length: 159, dtype: object
```

Runde Klammern () - parentheses

In runden Klammern wird der Datentyp Tupel deklariert. Mit Runden Klammer werden Methoden wie len(), read_csv(), ... aufgerufen. In den Klammern können Argumente übergeben werden. Es kann auch das Objekt vorangeschrieben werden (z.B. auch die Bibliotheken), wenn das Objekt einen Bezug zu der Methode hat. Es können mehrere Methoden hintereinander getrennt durch einen Punkt angewendet werden, falls das Object die Methode verarbeitet oder die zweite Methode wird als Argument übergeben. Zudem können mittels Operationen mehrere Methoden verknüpft werden.

In [12]:
```
tupel = (8, 9, 10)
tupel
```

Out[12]: (8, 9, 10)

In [13]:
```
len(df_fish.Species)
```

Out[13]: 159

In [14]:
```
np.max(df_birth.year)
```

Out[14]: 1988

In [16]:
```
str(np.max(df_birth.year))
```

Out[16]: '1988'

In [17]:
```
np.max(df_birth.year) - np.min(df_birth.year)
```

Out[17]: 19

Eckige Klammern [] - brackets / square brackets

In eckigen Klammern wird der Datentyp Listen deklariert. Durch die Eckige Klammer nach einem Objekt, kann auf eine Position zugegriffen werden. Bei einem DataFrame müssen wir .loc bei einem String-Index bzw. .iloc bei einem Integer-Index voranstellen.

In [18]:
```
liste = [5, 6, 8]
string = 'Hallo'
liste[0], string[1]
```

Out[18]: (5, 'a')

In [20]:
```
df_fish.iloc[2]
```

Out[20]:
```
Species      Bream
Weight       340.0
Length1       23.9
Length2       26.5
Length3       31.1
Height     12.3778
Width       4.6961
Name: 2, dtype: object
```

Bei einem DataFram kann dann durch eine zweite Klammer auf Werte zugegriffen werden. Achtung: Durch die erste Klammer erstellen wir eine Pandas Series (df_fish.iloc[2]). Mit der zweiten Klammer greifen wir auf das Objekt von Pandas Series zu und entnehmen den Wert an der Position 4. In der Series beginnt die Positionierung bei 0.

In [21]:
```
df_fish.iloc[2][4]
```

Out[21]: 31.1

Durch den .loc Anweisung können wir auch mehrere Einträge aus dem DatenFrame auswählen. Hier greift diese jedoch bei [2:4] auf die tatsächlichen Indexe von 2 und 3 und 4.

In [25]:
```
df_fish.loc[2:4]
```

Out[25]:

	Species	Weight	Length1	Length2	Length3	Height	Width
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

In [26]:
```
df_fish.iloc[2:4]
```

Out[26]:

	Species	Weight	Length1	Length2	Length3	Height	Width
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555

Durch df_fish.loc[2:4] haben wir wieder ein Pandas DataFrame erstellt. Hier kann mittels Aufruf des Spaltennamens direkt eine Spalte ausgewählt werden.

In [27]:
```
df_fish.iloc[2:4]['Weight']
```

Out[27]:
```
2    340.0
3    363.0
Name: Weight, dtype: float64
```

Möchte man nicht nur eine Spalte auswählen, muss man die Spaltennamen in eine Liste schreiben. Daher müssen wir dort zwei eckige Klammern einsetzen.

In [28]:
```
df_fish.iloc[2:4][['Species', 'Weight']]
```

Out[28]:

	Species	Weight
2	Bream	340.0
3	Bream	363.0

Geschweifte Klammer {} - braces

Durch die geschweifte Klammer wird der Datentyp Dictionary deklariert. Hierbei wir der Schlüssel mittel ':' Doppelpunkt vom Wert getrennt. Die einzelne Dictionary Einträge werden durch ein Komma getrennt.

Es gibt Methoden, wie z.B. rename(), die als Argument ein Dictionary verarbeiten.

In [29]:
```
dic = {'green': 'yes', 'yellow': 'maybe', 'red': 'no'}
dic
```

Out[29]:
```
{'green': 'yes', 'yellow': 'maybe', 'red': 'no'}
```

In [30]:
```
df_fish = df_fish.rename(columns = {'Length1': 'normal_Length'})
df_fish
```

Out[30]:

	Species	Weight	normal_Length	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340
...	...	...	...	...	...	...	...
154	Smelt	12.2	11.5	12.2	13.4	2.0904	1.3936
155	Smelt	13.4	11.7	12.4	13.5	2.4300	1.2690
156	Smelt	12.2	12.1	13.0	13.8	2.2770	1.2558
157	Smelt	19.7	13.2	14.3	15.2	2.8728	2.0672
158	Smelt	19.9	13.8	15.0	16.2	2.9322	1.8792

159 rows × 7 columns

Boolsche Abfrage für das Slicen

In [31]:
```
df_fish.Species == 'Bream'
```

Out[31]:
```
0       True
1       True
2       True
3       True
4       True
       ...
154    False
155    False
156    False
157    False
158    False
Name: Species, Length: 159, dtype: bool
```

In [33]:
```
df_fish[df_fish.Species == 'Bream'].tail()
```

Out[33]:

	Species	Weight	normal_Length	Length2	Length3	Height	Width
30	Bream	920.0	35.0	38.5	44.1	18.0369	6.3063
31	Bream	955.0	35.0	38.5	44.0	18.0840	6.2920
32	Bream	925.0	36.2	39.5	45.3	18.7542	6.7497
33	Bream	975.0	37.4	41.0	45.9	18.6354	6.7473
34	Bream	950.0	38.0	41.0	46.5	17.6235	6.3705

In [34]:
```
len(df_fish[df_fish.Species == 'Bream'])
```

Out[34]: 35

In [35]:
```
len(df_fish[(df_fish.Species == 'Bream') == True])
```

Out[35]: 35

In [36]:
```
len(df_fish[(df_fish.Species == 'Bream') == False])
```

Out[36]: 124

In [43]:
```
df_fish[(df_fish.Species == 'Bream') == False]['Species'].unique()
```

Out[43]:
```
array(['Roach', 'Whitefish', 'Parkki', 'Perch', 'Pike', 'Smelt'],
      dtype=object)
```

apply Funktion

In [44]:
```
# define a function to year only the year without 19-century
def year_xx(year):
    return year - 1900


def month_name(month):
    if month == 1: return 'jan'
    if month == 2: return 'feb'
    if month == 3: return 'mar'
    if month == 4: return 'apr'
    if month == 5: return 'mai'
    if month == 6: return 'jun'
    if month == 7: return 'jul'
    if month == 8: return 'aug'
    if month == 9: return 'sep'
    if month == 10: return 'oct'
    if month == 11: return 'nov'
    if month == 12: return 'dec'

    else:  return 'hallo'
```

In [47]:
```
year_xx(df_birth['year'])
```

Out[47]:
```
0         69
1         69
2         69
3         69
4         69
          ..
372859    88
372860    88
372861    88
372862    88
372863    88
Name: year, Length: 372864, dtype: int64
```

In [45]:
```
df_birth['year'].apply(year_xx)
```

Out[45]:
```
0         69
1         69
2         69
3         69
4         69
          ..
372859    88
372860    88
372861    88
372862    88
372863    88
Name: year, Length: 372864, dtype: int64
```

In [46]:
```
df_birth['month'].apply(month_name)
```

Out[46]:
```
0         jan
1         jan
2         jan
3         jan
4         jan
         ...
372859    dec
372860    dec
372861    dec
372862    dec
372863    dec
Name: month, Length: 372864, dtype: object
```

List Comprehension

In [48]:
```
# list comprehension by using column df_birth['month']

coll_month = ['jan', 'feb', 'mar', 'apr', 'mai', 'jun', 'jul', 'aug', 'sep',

df_birth['month_name'] = [coll_month[j-1] for j in df_birth['month']]

# check if the code is working correctly
print('values of month_name: ', df_birth.month_name.unique())
df_birth.head()
```

```
values of month_name:  ['jan' 'feb' 'mar' 'apr' 'mai' 'jun' 'jul' 'aug' 'sep'
```

```
'oct' 'nov' 'dec']
```

Out[48]:

	state	year	month	day	date	wday	births	month_name
0	AK	1969	1	1	1969-01-01	Wed	14	jan
1	AL	1969	1	1	1969-01-01	Wed	174	jan
2	AR	1969	1	1	1969-01-01	Wed	78	jan
3	AZ	1969	1	1	1969-01-01	Wed	84	jan
4	CA	1969	1	1	1969-01-01	Wed	824	jan

In [49]:

```
# list comprehension by using range of len(df_birth['month'])
df_birth['age'] = [2021 - df_birth.year[i] for i in range(len(df_birth.year))
# check if the code is working correctly
print('values of age: ', df_birth.age.unique())
df_birth.head()
```

```
values of age:  [52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33]
```

Out[49]:

	state	year	month	day	date	wday	births	month_name	age
0	AK	1969	1	1	1969-01-01	Wed	14	jan	52
1	AL	1969	1	1	1969-01-01	Wed	174	jan	52
2	AR	1969	1	1	1969-01-01	Wed	78	jan	52
3	AZ	1969	1	1	1969-01-01	Wed	84	jan	52
4	CA	1969	1	1	1969-01-01	Wed	824	jan	52

In [50]:

```
# code is working correctly. Now adding if they have birthday befor jul, stay

df_birth['age'] = [(2021 - df_birth.year[i]) if df_birth.month[i] < 7 else (2(
# check if the code is working correctly
print('values of age: ', df_birth.age.unique())
df_birth.head()
```

```
values of age:  [52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 3
2]
```

Out[50]:

	state	year	month	day	date	wday	births	month_name	age
0	AK	1969	1	1	1969-01-01	Wed	14	jan	52
1	AL	1969	1	1	1969-01-01	Wed	174	jan	52
2	AR	1969	1	1	1969-01-01	Wed	78	jan	52
3	AZ	1969	1	1	1969-01-01	Wed	84	jan	52
4	CA	1969	1	1	1969-01-01	Wed	824	jan	52

In [51]:

```
df_birth[df_birth.month > 6].head()
```

Out[51]:

	state	year	month	day	date	wday	births	month_name	age

	state	year	month	day	date	wday	births	month_name	age
9295	AK	1969	7	1	1969-07-01	Tues	14	jul	51
9296	AL	1969	7	1	1969-07-01	Tues	200	jul	51
9297	AR	1969	7	1	1969-07-01	Tues	100	jul	51
9298	AZ	1969	7	1	1969-07-01	Tues	108	jul	51

merge, concate and append

In [52]:
```
#df_1 has only single animals
df_1 = pd.DataFrame({'animals': ['elephant', 'monkey', 'cat', 'dog'],
                     'value': [1, 2, 3, 5]})

#df_2 has doubling of monkey and cat
df_2 = pd.DataFrame({'animals': ['monkey', 'monkey', 'cat', 'cat', 'bird'],
                     'value': [6, 7, 8, 9, 10]})
```

In [53]:
```
df_1
```

Out[53]:

	animals	value
0	elephant	1
1	monkey	2
2	cat	3
3	dog	5

In [54]:
```
df_2
```

Out[54]:

	animals	value
0	monkey	6
1	monkey	7
2	cat	8
3	cat	9
4	bird	10

In [55]:
```
# as we have the same column we can append both table
df_3 = df_1.append(df_2)
df_3
```

Out[55]:

	animals	value
0	elephant	1
1	monkey	2

	animals	value
2	cat	3
3	dog	5
0	monkey	6
1	monkey	7
2	cat	8
3	cat	9

In [56]:
```
# concat can do the same like append
df_concat_1 = pd.concat([df_1, df_2])
df_concat_1
```

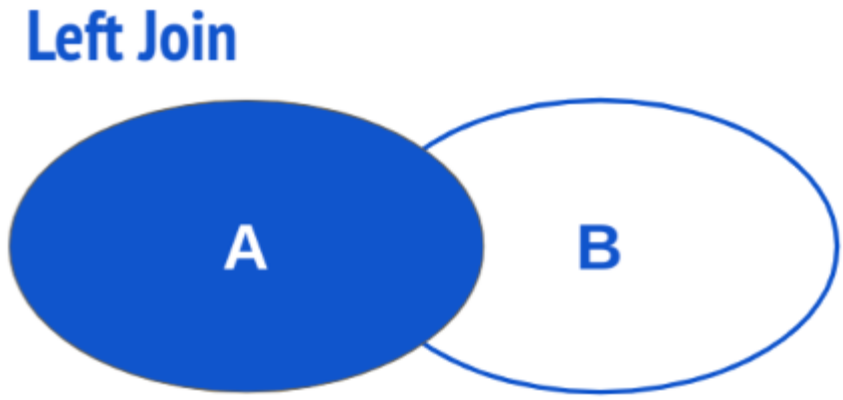
Out[56]:

	animals	value
0	elephant	1
1	monkey	2
2	cat	3
3	dog	5
0	monkey	6
1	monkey	7
2	cat	8
3	cat	9
4	bird	10

In [57]:
```
pd.concat([df_1, df_2], keys= ['df_1', 'df_2'])
```

Out[57]:

		animals	value
df_1	**0**	elephant	1
	1	monkey	2
	2	cat	3
	3	dog	5
df_2	**0**	monkey	6
	1	monkey	7
	2	cat	8
	3	cat	9
	4	bird	10



In [64]:
```
df_2_neu = df_2.drop_duplicates(subset = ['animals'])
df_2_neu
```

Out[64]:

	animals	value
0	monkey	6
2	cat	8
4	bird	10

In [66]:
```
# left join merge
df_2 = df_2.drop_duplicates()
df_left_join = pd.merge(df_1, df_2, how = 'left', on = 'animals', validate =
df_left_join
```

Out[66]:

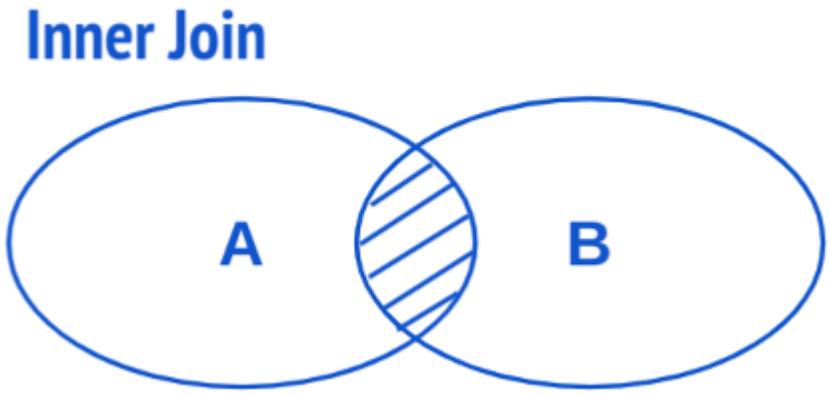
	animals	value_x	value_y
0	elephant	1	NaN
1	monkey	2	6.0
2	monkey	2	7.0
3	cat	3	8.0
4	cat	3	9.0
5	dog	5	NaN

In [69]:
```
df_right_join = pd.merge(df_2_neu, df_1,  how = 'left', on = 'animals', valida
df_right_join
```

Out[69]:

	animals	value_x	value_y
0	monkey	6	2.0
1	cat	8	3.0
2	bird	10	NaN

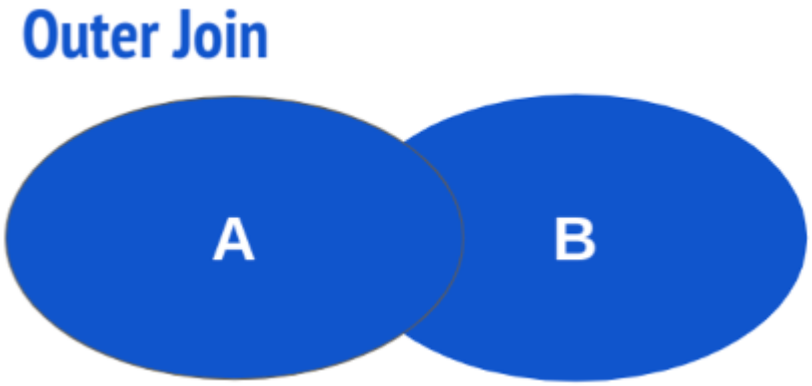
Es wurden alle Werte des linken DataFrames genommen, jedoch wurden 'cat' und 'monkey' nun doppelte Einträgen.



In [72]:
```
# inner join merge
df_inner_join = pd.merge(df_1, df_2, how = 'inner', on = 'animals')
df_inner_join
```

Out[72]:

	animals	value_x	value_y
0	monkey	2	6
1	monkey	2	7
2	cat	3	8
3	cat	3	9



In [73]:
```
# outer join merge
df_outer_join = pd.merge(df_1, df_2, how = 'outer', on = 'animals')
df_outer_join
```

Out[73]:

	animals	value_x	value_y
0	elephant	1.0	NaN
1	monkey	2.0	6.0
2	monkey	2.0	7.0

	animals	value_x	value_y
3	cat	3.0	8.0
4	cat	3.0	9.0
5	dog	5.0	NaN

In [74]:
```
# by using indicator = True
df_ind_join = pd.merge(df_1, df_2, how = 'outer', on = 'animals', indicator =
df_ind_join
```

Out[74]:

	animals	value_x	value_y	_merge
0	elephant	1.0	NaN	left_only
1	monkey	2.0	6.0	both
2	monkey	2.0	7.0	both
3	cat	3.0	8.0	both
4	cat	3.0	9.0	both
5	dog	5.0	NaN	left_only
6	bird	NaN	10.0	right_only

Hinweis:

- mit indicator = True kann dann anschließend nach '_merge' - Wunsch gefiltert werden!
- ab und an ist es notwendig ein validate zu setzten (1:1 oder 1:m)!
- Vorsichtig sein mit Duplicates!
- Hilfreich ist mit df.shape die Struktur zur überprüfen!

In [ ]:
```

```