Pandas

Pandas ist eine Bibliothek, die auf NumPy aufbaut. Generell wird Pandas benutzt um eine effiziente DataFrame-Implementierung durch zuführen und diese zu Bearbeiten.

Pandas wird von **Pan**el-**Da**ta abgeleitet. Es arbeitet insbesondere mit 2 Datentypen:

- **Series**: Ein Data Array mit einem benannten Index.
- **DataFrame**: Eine "Matrix", welche benannte Indizes und Spalten besitzt.

Da Pandas auf NumPy aufbaut, benötigt man die Bibliotheken von Pandas wie auch von NumPy.

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

Out[1]:
```
'1.2.4'
```

In [2]:
```
# pandas Series object as single array
pd.Series([0.25, 0.5, 0.75, 1.0])
```

Out[2]:
```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

In [3]:
```
# pandas DataFras objecs from a matrix
pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]))
```

Out[3]:

	0	1	2
0	1	2	3

	0	1	2
1	4	5	6

Series

Zuerst bestimmen wir die labels = ['A', 'B', 'C'] die wir einer Liste, Array und Dictionary zuordnen:

- li = [11, 22, 33]
- arr = np.array([11, 22, 33])
- dic = {'A': 11, 'B': 22,'C': 33}

In [8]:
```
li = [11, 22, 33]
arr = np.array([11, 22, 33])
dic = {'A': [2, 3], 'B': 22,'C': 33}
```

In [13]:
```
# create Series with data = li
labels = ['A', 'B', 'C']
pd.Series(data = li, index = labels)
```

Out[13]:
```
A    11
B    22
C    33
Name: Test, dtype: int64
```

In [18]:
```
# include index = labels with arr
labels = ['A', 'B', 'C']
pd.Series(arr, index = labels)
```

Out[18]:
```
A    11
B    22
C    33
dtype: int32
```

In [11]:
```
# include dic
pd.Series(dic)
```

Out[11]:
```
A    [2, 3]
B        22
C        33
dtype: object
```

In [19]:
```
# include dic + index  !inices must be the same!
pd.Series(dic, index = labels)
```

Out[19]:
```
A    [2, 3]
B        22
C        33
dtype: object
```

DataFrame

Hierfür bauen wir eine Matrix data = np.random.randint(-50, 50, (6,5)) und setzen zuvor

np.random.seed(42). Zudem bennen wir die columns = ['blue', 'green', 'orange', 'yellow', 'purple'] und die index = ['A', 'B', 'C', 'D', 'E', 'F'] .

In [21]:
```
# create matrix
np.random.seed(42)
data = np.random.randint(-50, 50, (6,5))
data
```

Out[21]:
```
array([[  1,  42, -36,  21,  10],
       [-30,  32,  36,  24,  24],
       [ 37,  49, -27, -48, -29],
       [  2, -49,  37, -21, -13],
       [-49,  13,   9, -30, -18],
       [ 25,   7, -29,  38,  -2]])
```

In [22]:
```
# columns and index
columns = ['blue', 'green', 'orange', 'yellow', 'purple']
index = ['A', 'B', 'C', 'D', 'E', 'F']
```

In [23]:
```
pd.DataFrame(data, index, columns)
```

Out[23]:

	blue	green	orange	yellow	purple
A	1	42	-36	21	10
B	-30	32	36	24	24
C	37	49	-27	-48	-29
D	2	-49	37	-21	-13
E	-49	13	9	-30	-18
F	25	7	-29	38	-2

In [24]:
```
# assign value to object df
df = pd.DataFrame(data, index, columns)
df
```

Out[24]:

	blue	green	orange	yellow	purple
A	1	42	-36	21	10
B	-30	32	36	24	24
C	37	49	-27	-48	-29
D	2	-49	37	-21	-13
E	-49	13	9	-30	-18
F	25	7	-29	38	-2

Informationen gewinnen

In [25]:
```
# show first rows (default = 5)
df.head(2)
```

Out[25]:

	blue	green	orange	yellow	purple
A	1	42	-36	21	10
B	-30	32	36	24	24

In [26]:
```
# show last rows (default = 5)
df.tail(3)
```

Out[26]:

	blue	green	orange	yellow	purple
D	2	-49	37	-21	-13
E	-49	13	9	-30	-18
F	25	7	-29	38	-2

In [29]:
```
# list of column names
df.columns
```

Out[29]:
```
Index(['blue', 'green', 'orange', 'yellow', 'purple'], dtype='object')
```

In [33]:
```
# show 1 columns: only  bracket
df['blue']
```

Out[33]:
```
A     1
B   -30
C    37
D     2
E   -49
F    25
Name: blue, dtype: int32
```

In [34]:
```
# show 2 columns: put the columns in a list! + bracket
df[['blue', 'yellow']]
```

Out[34]:

	blue	yellow
A	1	21
B	-30	24
C	37	-48
D	2	-21
E	-49	-30
F	25	38

In [35]:
```
# list of index
df.index
```

Out[35]:
```
Index(['A', 'B', 'C', 'D', 'E', 'F'], dtype='object')
```

In [38]:
```
# list of values only for Series - columns
df['blue'].unique()
```

Out[38]:
```
array([  1, -30,  37,   2, -49,  25])
```

In [42]:
```
# another possible to write the method
df.blue.unique()
```

Out[42]:
```
array([  1, -30,  37,   2, -49,  25])
```

In [39]:
```
# regarding rows add .loc for string and .iloc for integer indizes
df.loc['A'].unique()
```

Out[39]:
```
array([  1,  42, -36,  21,  10])
```

In [40]:
```
# using integer to grab the row
df.iloc[1]
```

Out[40]:
```
blue     -30
green     32
orange    36
yellow    24
purple    24
Name: B, dtype: int32
```

In [41]:
```
# number of unique values
df.nunique()
```

Out[41]:
```
blue      6
green     6
orange    6
yellow    6
purple    6
dtype: int64
```

Spalte: axis = 1

Reihe: axis = 0

In [44]:
```
# number of unique values for row by using axis = 1
df.nunique(axis = 1)
```

Out[44]:
```
A    5
B    4
C    5
D    5
```

```
E    5
F    5
dtype: int64
```

In [45]:
```
# transpose() DataFrame
df.transpose()
```

Out[45]:

	A	B	C	D	E	F
blue	1	-30	37	2	-49	25
green	42	32	49	-49	13	7
orange	-36	36	-27	37	9	-29
yellow	21	24	-48	-21	-30	38
purple	10	24	-29	-13	-18	-2

In [50]:
```
# or by .T
df = df.T
df
```

Out[50]:

	blue	green	orange	yellow	purple
A	1	42	-36	21	10
B	-30	32	36	24	24
C	37	49	-27	-48	-29
D	2	-49	37	-21	-13
E	-49	13	9	-30	-18
F	25	7	-29	38	-2

Zusammenfassende Methoden

In [54]:
```
# info
df.info
```

Out[54]:
```
<bound method DataFrame.info of    blue  green  orange  yellow  purple
A     1     42     -36      21      10
B   -30     32      36      24      24
C    37     49     -27     -48     -29
D     2    -49      37     -21     -13
E   -49     13       9     -30     -18
F    25      7     -29      38      -2>
```

In [55]:
```
# info()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6 entries, A to F
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
```

```
---  ------  --------------  -----
 0   blue    6 non-null      int32
 1   green   6 non-null      int32
 2   orange  6 non-null      int32
 3   yellow  6 non-null      int32
 4   purple  6 non-null      int32
dtypes: int32(5)
```

In [51]:
```
# describe()
df.describe()
```

Out[51]:

	blue	green	orange	yellow	purple
count	6.000000	6.000000	6.000000	6.000000	6.000000
mean	-2.333333	15.666667	-1.666667	-2.666667	-4.666667
std	32.457151	35.595880	33.452454	34.823364	19.407902
min	-49.000000	-49.000000	-36.000000	-48.000000	-29.000000
25%	-22.250000	8.500000	-28.500000	-27.750000	-16.750000
50%	1.500000	22.500000	-9.000000	0.000000	-7.500000
75%	19.250000	39.500000	29.250000	23.250000	7.000000
max	37.000000	49.000000	37.000000	38.000000	24.000000

In [56]:
```
df.dtypes
```

Out[56]:
```
blue      int32
green     int32
orange    int32
yellow    int32
purple    int32
dtype: object
```

Slicing

In [57]:
```
# select column
df['green']
```

Out[57]:
```
A    42
B    32
C    49
D   -49
E    13
F     7
Name: green, dtype: int32
```

In [58]:
```
# select row
df.loc['E']
```

Out[58]:
```
blue     -49
green     13
orange     9
yellow   -30
```

```
purple   -18
Name: E, dtype: int32
```

In [59]:
```
# select row with integer index
df.iloc[1]
```

Out[59]:
```
blue     -30
green     32
orange    36
yellow    24
purple    24
Name: B, dtype: int32
```

In [60]:
```
# select single value
df.loc['E']['green']
```

Out[60]: 13

In [61]:
```
# select part of DataFrame by [[ ]] - only rows
df.loc[['A', 'C']]
```

Out[61]:

	blue	green	orange	yellow	purple
A	1	42	-36	21	10
C	37	49	-27	-48	-29

In [62]:
```
# select part of Data Frame ba [[row], [columns]]
df.loc[['A', 'B'], ['green', 'purple']]
```

Out[62]:

	green	purple
A	42	10
B	32	24

In [63]:
```
# slicing with figures
df[1:3]
```

Out[63]:

	blue	green	orange	yellow	purple
B	-30	32	36	24	24
C	37	49	-27	-48	-29

Operatoren

In [64]:
```
df
```

Out[64]:

	blue	green	orange	yellow	purple

	blue	green	orange	yellow	purple
A	1	42	-36	21	10
B	-30	32	36	24	24
C	37	49	-27	-48	-29
D	2	-49	37	-21	-13
E	-49	13	9	-30	-18

In [65]:
```
# operation to one column
df['orange'] = df['orange'] * (-1)
df
```

Out[65]:

	blue	green	orange	yellow	purple
A	1	42	36	21	10
B	-30	32	-36	24	24
C	37	49	27	-48	-29
D	2	-49	-37	-21	-13
E	-49	13	-9	-30	-18
F	25	7	29	38	-2

In [66]:
```
# add new column
df['new_color'] = df['blue'] + df['green']
df
```

Out[66]:

	blue	green	orange	yellow	purple	new_color
A	1	42	36	21	10	43
B	-30	32	-36	24	24	2
C	37	49	27	-48	-29	86
D	2	-49	-37	-21	-13	-47
E	-49	13	-9	-30	-18	-36
F	25	7	29	38	-2	32

In [67]:
```
# operation to the complete df
df = abs(df)
df
```

Out[67]:

	blue	green	orange	yellow	purple	new_color
A	1	42	36	21	10	43
B	30	32	36	24	24	2

	blue	green	orange	yellow	purple	new_color
C	37	49	27	48	29	86
D	2	49	37	21	13	47
E	49	13	9	30	18	36

In [68]:
```
# unique()
df['orange'].unique()
```

Out[68]:
```
array([36, 27, 37,  9, 29])
```

In [69]:
```
# nunique()
df['orange'].nunique()
```

Out[69]:
```
5
```

In [70]:
```
# value_counts()
df['orange'].value_counts()
```

Out[70]:
```
36    2
9     1
29    1
27    1
37    1
Name: orange, dtype: int64
```

In [71]:
```
# drop_duplicates() - removes rows
df['orange'].drop_duplicates()
```

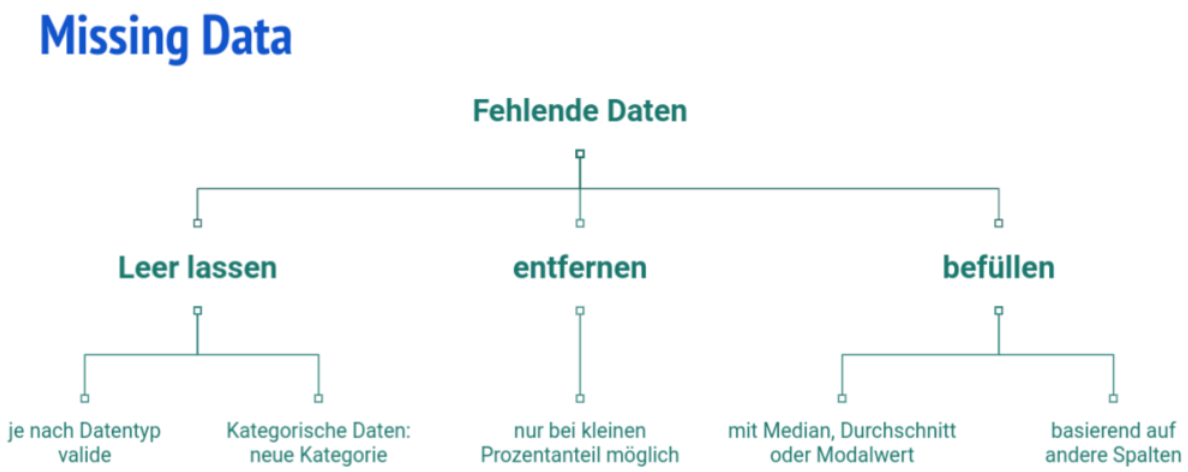
Out[71]:
```
A    36
C    27
D    37
E     9
F    29
Name: orange, dtype: int32
```

In [72]:
```
# save in new column
df['red'] = df.yellow - df.green + df.purple
df
```

Out[72]:

	blue	green	orange	yellow	purple	new_color	red
A	1	42	36	21	10	43	-11
B	30	32	36	24	24	2	16
C	37	49	27	48	29	86	28
D	2	49	37	21	13	47	-15
E	49	13	9	30	18	36	35
F	25	7	29	38	2	32	33

Fehlende Daten



In [73]:
```
# function to create some NaN's in the df
def create_nan(df, n):

    ''' Input: df - DataFrame
               n - number of NaNs

        Output: df - DataFrame

        This function creates a DataFrame including some NaN values.

        '''

    np.random.seed(3)

    for i in range(n):
        row_ind = np.random.randint(0, df.shape[0]) # 'high' exclusive!
        col_ind = np.random.randint(0, df.shape[1])

        df.iloc[row_ind,col_ind] = np.nan    # replace np.nan in df

    return df
```

In [74]:
```
# build a df with NaN by using function create_nan()
df_nan = create_nan(df, 4)
df_nan
```

Out[74]:

	blue	green	orange	yellow	purple	new_color	red
A	NaN	42	36	21.0	10	NaN	-11
B	30.0	32	36	NaN	24	2.0	16
C	NaN	49	27	48.0	29	86.0	28
D	2.0	49	37	21.0	13	47.0	-15
E	49.0	13	9	30.0	18	36.0	35

	blue	green	orange	yellow	purple	new color	red

In [75]:
```
# dropna(), default axis = 0, deletion of rows
df_nan.dropna()
```

Out[75]:

	blue	green	orange	yellow	purple	new_color	red
D	2.0	49	37	21.0	13	47.0	-15
E	49.0	13	9	30.0	18	36.0	35
F	25.0	7	29	38.0	2	32.0	33

In [76]:
```
# dropna(), deletion columns
df_nan.dropna(axis = 1)
```

Out[76]:

	green	orange	purple	red
A	42	36	10	-11
B	32	36	24	16
C	49	27	29	28
D	49	37	13	-15
E	13	9	18	35
F	7	29	2	33

In [77]:
```
df_nan
```

Out[77]:

	blue	green	orange	yellow	purple	new_color	red
A	NaN	42	36	21.0	10	NaN	-11
B	30.0	32	36	NaN	24	2.0	16
C	NaN	49	27	48.0	29	86.0	28
D	2.0	49	37	21.0	13	47.0	-15
E	49.0	13	9	30.0	18	36.0	35
F	25.0	7	29	38.0	2	32.0	33

In [78]:
```
# dropna() with threshold - how many non NaNs are still in
df_nan.dropna(thresh = 4)
```

Out[78]:

	blue	green	orange	yellow	purple	new_color	red
A	NaN	42	36	21.0	10	NaN	-11
B	30.0	32	36	NaN	24	2.0	16
C	NaN	49	27	48.0	29	86.0	28

	blue	green	orange	yellow	purple	new_color	red
D	2.0	49	37	21.0	13	47.0	-15
E	49.0	13	9	30.0	18	36.0	35

In [79]:
```
# dropna() with threshold related to 75 %
df_nan.dropna(thresh= len(df_nan.columns)*0.75)
```

Out[79]:

	blue	green	orange	yellow	purple	new_color	red
B	30.0	32	36	NaN	24	2.0	16
C	NaN	49	27	48.0	29	86.0	28
D	2.0	49	37	21.0	13	47.0	-15
E	49.0	13	9	30.0	18	36.0	35
F	25.0	7	29	38.0	2	32.0	33

In [80]:
```
len(df_nan.columns)*0.75
```

Out[80]:
```
5.25
```

Importieren

Das befüllen von NaN's wird auch importieren genannt

In [81]:
```
# fillna() with value = 0
df_nan.fillna(0)
```

Out[81]:

	blue	green	orange	yellow	purple	new_color	red
A	0.0	42	36	21.0	10	0.0	-11
B	30.0	32	36	0.0	24	2.0	16
C	0.0	49	27	48.0	29	86.0	28
D	2.0	49	37	21.0	13	47.0	-15
E	49.0	13	9	30.0	18	36.0	35
F	25.0	7	29	38.0	2	32.0	33

In [82]:
```
# fillna() mean of column 'blue'
df_nan['blue'].fillna(df_nan['blue'].mean())
```

Out[82]:
```
A    26.5
B    30.0
C    26.5
D     2.0
E    49.0
```

```
F    25.0
```

In [84]:
```
# control that NaN's are not in this mean calculation
(30 + 2 + 49 + 25) / 4
```

Out[84]:
```
26.5
```

In [85]:
```
# fillna() median of column 'yellow'
df_nan['yellow'].fillna(df_nan['yellow'].median())
```

Out[85]:
```
A    21.0
B    30.0
C    48.0
D    21.0
E    30.0
F    38.0
Name: yellow, dtype: float64
```

Datei Laden und Speicher

Häufig werden Datei im csv Format (comma separated values), jedoch können viele Formate geladen werden:

- **CSV**: read_csv // to_csv
- **JSON**: read_json // to_json
- **HTML**: read_html // to_html
- **MS Excel**: read_excel // to_excel
- und weitere

Wichtig ist hier, die Angabe des korrekten Pfades und der Name der Datei. Daher am Besten erst mit 'pwd' den derzeitigen Verzeichnis kontrollieren und mit *TAB*-Taste vorangehen oder auf die Datei im Verzeichnis mittels *linker Maustase* den Pfad kopieren und einfügen.

In [86]:
```
pwd
```

Out[86]:
```
'C:\\Users\\alfa\\anaconda3\\00_Trainingsunterlagen\\Alfa\\03_pandas_intro'
```

In [87]:
```
# read dataframe
df_mac = pd.read_csv('bigmac_old.csv')
df_mac.head()
```

Out[87]:

	date	currency_code	name	local_price	dollar_ex	dollar_price
0	2000-04-01	ARS	Argentina	2.5	1.0	2.5
1	2000-04-01	AUD	Australia	2.59	1.68	15.416.666.666.666.600
2	2000-04-01	BRL	Brazil	2.95	1.79	164.804.469.273.743
3	2000-04-01	CAD	Canada	2.85	1.47	193.877.551.020.408
4	2000-04-01	CHF	Switzerland	5.9	1.7	34.705.882.352.941.200

Data Description of Big Mac

Source: calmcode.io // originally found in the economist.

Purpose: Der Datensatz beinhaltet Preise über mehrere Jahre über den Big Mac von Mac Donalds in verschiedenen Ländern. Die Idee ist, da dieses Produkt über die ganze Welt serviert wird, können die Daten einen interessanten Indicator für die Wirtschaft bilden.

Contents:

Name	Descretion
date	The date of the measurement.
currency_code	International code for the currency in the country.
name	Name of the country.
local_price	Price of a bigmac in the local currency.
dollar_ex	The dollar exchange rate.
dollar_price	the price of a bigmac translated back to dollars.

Kennenlernen des Datensatzes

In [88]:
```
# get info()
df_mac.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1330 entries, 0 to 1329
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   date           1330 non-null   object
 1   currency_code  1330 non-null   object
 2   name           1330 non-null   object
 3   local_price    1330 non-null   object
 4   dollar_ex      1330 non-null   object
 5   dollar_price   1330 non-null   object
dtypes: object(6)
memory usage: 62.5+ KB
```

In [89]:
```
# check type of column date
type(df_mac.date[15])
```

Out[89]: str

In [90]:
```
# transform column into date format
# Date Time Conversion
from datetime import datetime as dt

# Applying the datetime conversion
df_mac.date = pd.to_datetime(df_mac['date']).dt.date
df_mac.head()
```

Out[90]:

	date	currency_code	name	local_price	dollar_ex	dollar_price
0	2000-04-01	ARS	Argentina	2.5	1.0	2.5
1	2000-04-01	AUD	Australia	2.59	1.68	15.416.666.666.666.600
2	2000-04-01	BRL	Brazil	2.95	1.79	164.804.469.273.743
3	2000-04-01	CAD	Canada	2.85	1.47	193.877.551.020.408
4	2000-04-01	CHF	Switzerland	5.9	1.7	34.705.882.352.941.200

In [91]:
```
# recheck type of column date
type(df_mac.date[15])
```

Out[91]:
```
datetime.date
```

In [93]:
```
# just show only local_price by sort_values
df_mac.local_price.sort_values()
```

Out[93]:
```
1199     01.05
1190     01.05
1134     01.05
1143     01.05
196      03.06
         ...
840       99.0
633       99.0
417      99.39
985     9900.0
1028    9900.0
Name: local_price, Length: 1330, dtype: object
```

In [53]:
```
# create new column which counts the the dots in local_price
df_mac['count_dots'] = [ i.count('.') for i in df_mac.local_price]
df_mac.head()
```

Out[53]:

	date	currency_code	name	local_price	dollar_ex	dollar_price	count_dots
0	2000-04-01	ARS	Argentina	2.5	1.0	2.5	1
1	2000-04-01	AUD	Australia	2.59	1.68	15.416.666.666.666.600	1
2	2000-04-01	BRL	Brazil	2.95	1.79	164.804.469.273.743	1
3	2000-04-01	CAD	Canada	2.85	1.47	193.877.551.020.408	1
4	2000-04-01	CHF	Switzerland	5.9	1.7	34.705.882.352.941.200	1

In [54]:
```
# show df only with more than one dot
df_mac[(df_mac.count_dots > 1) == True]
```

Out[54]:

	date	currency_code	name	local_price	dollar_ex	dollar_
242	2006-05-01	EUR	Euro area	29.395.735.289.999.900	780.365.992	37.669.164.970.47
322	2007-06-01	EUR	Euro area	3.057.482.443	7.337.564.659.999.990	41.668.899.48
416	2009-07-01	PEN	Peru	8.056.000.000.000.000	3.0274	26.610.292.660.36
457	2010-01-01	PEN	Peru	8.056.000.000.000.000	2.8705	28.064.797.073.68
484	2010-07-01	EUR	Euro area	33.800.310.710.000.000	78.015.291	433.252.382.66
510	2010-07-01	USD	United States	3.733.333.333	1.0	3.733.33
525	2011-07-01	EUR	Euro area	3.437.660.401	697.520.315	492.840.183.58
567	2012-01-01	EUR	Euro area	349.245.637	788.239.467	443.070.477.46
608	2012-07-01	EUR	Euro area	35.834.822.410.000.000	8.248.443.109.999.990	43.444.346.929.61
628	2012-07-01	SEK	Sweden	3.997.301.987	6.9777	572.868.135.20
649	2013-01-01	EUR	Euro area	35.948.495.719.999.900	7.369.468.290.000.000	4.878.031.128.62
670	2013-01-01	SEK	Sweden	4.055.930.138	6.3492	638.809.635.54
676	2013-01-01	USD	United States	4.367.395.833	1.0	4.367.39
691	2013-07-01	EUR	Euro area	36.238.704.889.999.900	777.756.173	465.939.148.38
718	2013-07-01	USD	United States	4.556.666.667	1.0	4.556.66
733	2014-01-01	EUR	Euro area	3.657.962.724	737.218.475	496.184.353.49
760	2014-01-01	USD	United States	4.624.166.667	1.0	4.624.16
775	2014-07-01	EUR	Euro area	367.923.829	7.427.489.140.000.000	495.354.246.99

Glücklicherweise handelt es sich nicht um sehr viele Ausreißer. Daher betrachten wir die Daten etwas näher, um evtl. einen Algorithmus anzuwenden, die Daten etwas zu bereinigen.

Es lässt sich erahnen, dass bei den USA das cutten auf das Format X.XX sinnvoll sein könnte. Bei den anderen Preisen lässt sich jedoch kein richtiges System erahnen.

Es handelt sich um 18 Datensätze von 1.330 Datensätze insgesamt, also 1,4 % von der Gesamtmenge. Daher werden die Daten gelöscht.

In [55]:
```
# only take the dataset with one dot or lesser
df_mac = df_mac[(df_mac.count_dots <= 1) == True]

# control of the deletion of 18 datasets
print('Number of datasets: ', df_mac.shape[0],'\nNumber of columns:  ', df_mac
```

```
Number of datasets:  1312
Number of columns:   7
```

In [56]:
```
# try again to change the datatype of 'local_price' into float
# transform local_price from str into float
df_mac['local_price'] = df_mac['local_price'].astype(float)
type(df_mac.local_price[15])
```

Out[56]: numpy.float64

looking now to the next column 'dollar_ex' to transform them into datatype float df_mac['dollar_ex'] = df_mac['dollar_ex'].astype(float) type(df_mac.dollar_ex[15])

Wir bekommen die selbe Fehlermeldung wie zuvor bei der Spale 'local_price'. Daher wenden wir das selbe Vorgehen an wie bei dieser Spalte. Die zuvor benutzte Spalte '

In [57]:
```
# using the column 'count_dots' for counting the dots in column 'dollar_ex'
df_mac['count_dots'] = [ i.count('.') for i in df_mac.dollar_ex]
df_mac.head()
```

Out[57]:

	date	currency_code	name	local_price	dollar_ex	dollar_price	count_dots
0	2000-04-01	ARS	Argentina	2.50	1.0	2.5	1
1	2000-04-01	AUD	Australia	2.59	1.68	15.416.666.666.666.600	1
2	2000-04-01	BRL	Brazil	2.95	1.79	164.804.469.273.743	1
3	2000-04-01	CAD	Canada	2.85	1.47	193.877.551.020.408	1
4	2000-04-01	CHF	Switzerland	5.90	1.7	34.705.882.352.941.200	1

In [58]:
```
# show df only with more than one dot
df_mac[(df_mac.count_dots > 1) == True]
```

Out[58]:

	date	currency_code	name	local_price	dollar_ex	dollar_price	c
9	2000-04-01	EUR	Euro area	2.56	1.075.268.817	238.080.000.045.235	
10	2000-04-01	GBP	Britain	1.90	632.911.392	30.020.000.019.212.800	
37	2001-04-01	EUR	Euro area	2.57	1.136.363.636	226.160.000.072.371	
38	2001-04-01	GBP	Britain	1.99	699.300.699	284.570.000.122.365	

	date	currency_code	name	local_price	dollar_ex	dollar_price	c
65	2002-04-01	EUR	Euro area	2.67	1.123.595.506	237.629.999.919.206	
...	...	...	...	...	...	...	
1295	2020-01-14	HUF	Hungary	900.00	2.987.502	30.125.502.844.851.600	
1297	2020-01-14	INR	India	188.00	7.087.815	265.243.943.302.696	
1299	2020-01-14	JOD	Jordan	2.30	7.090.000.000.000.000	324.400.564.174.894	
1310	2020-01-14	NZD	New Zealand	6.50	15.135.462.388.376	429.454.999.999.999	
1329	2020-01-14	ZAR	South Africa	31.00	14.390.999.999.999.900	215.412.410.534.362	

Wir haben nun 193 von 1312 Datensätze, bei denen dieser Fehler gefunden wurde. Dieses sind knappe 15 % und daher doch etwas viel um einfach so zu löschen. Daher schauen wir, ob wir einen Durchschnittswert für einzelnen Regionen in Erwägung ziehen können.

In [59]:
```
# show the values 'name' for all dataset with more the one dot
df_mac[(df_mac.count_dots > 1) == True].name.unique()
```

Out[59]:
```
array(['Euro area', 'Britain', 'Hungary', 'Russia', 'Saudi Arabia',
       'Australia', 'Colombia', 'Czech Republic', 'New Zealand', 'Mexico',
       'Egypt', 'Malaysia', 'South Korea', 'South Africa', 'Taiwan',
       'India', 'Peru', 'Ukraine', 'Argentina', 'Canada', 'China',
       'Philippines', 'Moldova', 'Jordan'], dtype=object)
```

In [60]:
```
# looking first at 'Euro area'
df_mac[(df_mac.name == 'Euro area') == True]
```

Out[60]:

	date	currency_code	name	local_price	dollar_ex	dollar_price	coun
9	2000-04-01	EUR	Euro area	2.560	1.075.268.817	238.080.000.045.235	
37	2001-04-01	EUR	Euro area	2.570	1.136.363.636	226.160.000.072.371	
65	2002-04-01	EUR	Euro area	2.670	1.123.595.506	237.629.999.919.206	
98	2003-04-01	EUR	Euro area	2.710	9.090.909.090.000.000	29.810.000.002.981	
131	2004-05-01	EUR	Euro area	2.740	833.333.333	32.880.000.013.152	
171	2005-06-01	EUR	Euro area	2.920	814.929.509	358.313.199.823.029	
208	2006-01-01	EUR	Euro area	2.910	82.815.735	35.138.249.995.607.600	

	date	currency_code	name	local_price	dollar_ex	dollar_price	coun
282	2007-01-01	EUR	Euro area	2.940	771.813.376	380.921.099.765.962	
362	2008-06-01	EUR	Euro area	336.856	630.497.147	5.342.704.588.003.460	
402	2009-07-01	EUR	Euro area	3.310	7.168.972.690.000.000	46.171.189.975.617	
443	2010-01-01	EUR	Euro area	3.360	693.649.638	484.394.399.698.368	
818	2015-01-01	EUR	Euro area	3.680	8.630.734.040.000.000	426.383.200.194.175	
861	2015-07-01	EUR	Euro area	3.700	9.127.002.240.000.000	405.390.499.827.466	
904	2016-01-01	EUR	Euro area	3.720	9.302.325.579.999.990	39.990.000.005.998.400	
947	2016-07-01	EUR	Euro area	3.820	908.306.463	420.562.899.815.015	
990	2017-01-01	EUR	Euro area	3.880	955.246.692	406.177.800.194.884	
1033	2017-07-01	EUR	Euro area	3.910	875.695.083	44.650.244.998.577.800	
1076	2018-01-01	EUR	Euro area	3.950	816.826.629	48.357.874.973.246	
1121	2018-07-01	EUR	Euro area	4.040	8.532.059.212.490.940	4.735.082	
1177	2019-01-01	EUR	Euro area	4.050	87.248.614.928.238	4.641.907.499.999.990	
1233	2019-07-09	EUR	Euro area	4.080	892.339.267.389.462	4.572.252	

Wir können hier gut erkennen, dass die Spalte 'dollar_ex' einer kompletten Bearbeitung bedarf. Die Spalte 'dollar_price' lässt sich dann mit local_pric und dollar_ex ermitteln.

Für die nächsten Schritte benötigen wir erstmal nicht die Spalten 'dollar_ex', 'dollar_price' und 'count_dots' nicht. Daher löschen wir diese Spalten.

Wir können hier auch einen Ausreiß mit dem Index 362 erkenn. Hier werden wir den 'local_price' in 3.360 ändern.

In [61]:
```
df_mac = df_mac[(((df_mac.name == 'Euro area') == True) & ((df_mac.local_price
                 ((df_mac.name == 'Euro area') == False)]
df_mac.shape
```

Out[61]: (1311, 7)

In [62]:
```
# call the columns
df_mac.columns
```

Out[62]:
```
Index(['date', 'currency_code', 'name', 'local_price', 'dollar_ex',
       'dollar_price', 'count_dots'],
      dtype='object')
```

In [97]:
```
# slicing the DataFrame
df_mac = df_mac[['date', 'currency_code', 'name', 'local_price']]
df_mac.head()
```

Out[97]:

	date	currency_code	name	local_price
0	2000-04-01	ARS	Argentina	2.5
1	2000-04-01	AUD	Australia	2.59
2	2000-04-01	BRL	Brazil	2.95
3	2000-04-01	CAD	Canada	2.85
4	2000-04-01	CHF	Switzerland	5.9

In [98]:
```
# describe()
df_mac.describe()
```

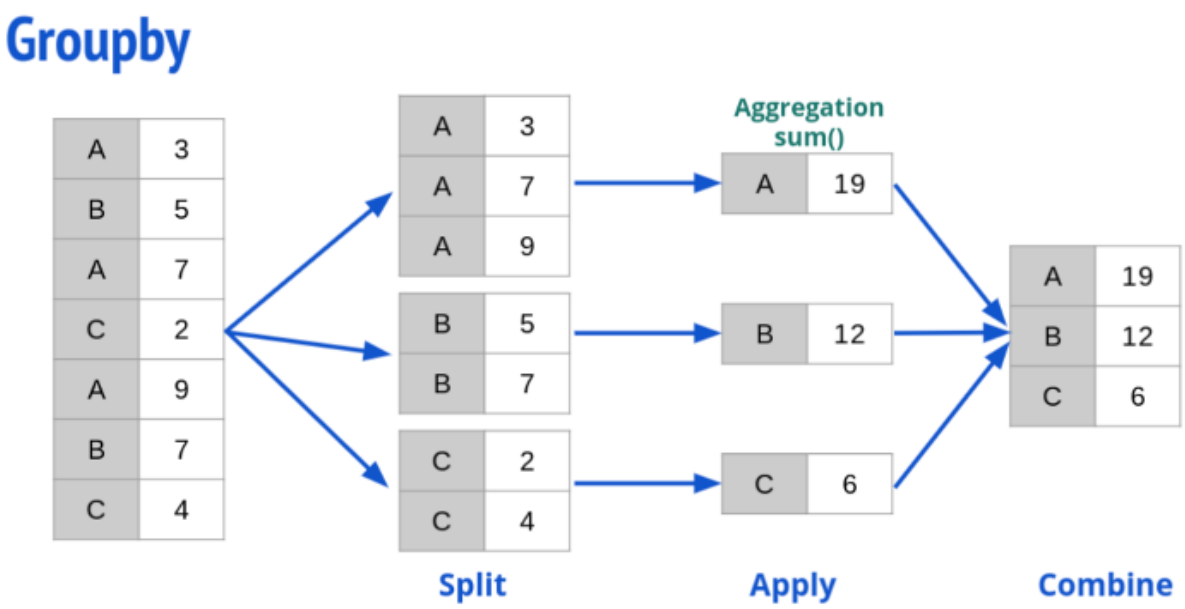
Out[98]:

	date	currency_code	name	local_price
count	1330	1330	1330	1330
unique	32	56	57	541
top	2019-01-01	KRW	Sweden	6.5
freq	56	32	32	26

Die Aussagekraft dieser Auswertung ist nicht sehr vielsagend. Daher sollte man eine Trennung nach der 'currency_code' oder 'name' vornehmen.

groupby()

Prozess: Split - Apply - Combine



Hier können u.a. die Aggregatsfunktionen verwendet werden:

In [92]:
```
# read dataframe
df_birth = pd.read_csv('birthdays.csv')
df_birth.head()
```

Out[92]:

	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 [93]:
```
df_birth.columns
```

Out[93]:
```
Index(['Unnamed: 0', 'state', 'year', 'month', 'day', 'date', 'wday',
       'births'],
```

In [94]:
```
# delete column Unnamed
df_birth = df_birth.drop('Unnamed: 0', axis = 1)
df_birth.head()
```

Out[94]:

	state	year	month	day	date	wday	births
0	AK	1969	1	1	1969-01-01	Wed	14
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
4	CA	1969	1	1	1969-01-01	Wed	824

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 in dem derjenige geboren ist.
year	Das Jahr.
month	Der Monat.
day	Der Tag des Monats.
date	Das Datum.
wday	Der Wochentag.
births	Anzahl der Geburten an dem Tag in dem genannten US State

In [95]:
```
# get information of the dataframe
df_birth.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 372864 entries, 0 to 372863
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   state   372864 non-null  object
 1   year    372864 non-null  int64 
 2   month   372864 non-null  int64 
 3   day     372864 non-null  int64 
 4   date    372864 non-null  object
 5   wday    372864 non-null  object
 6   births  372864 non-null  int64 
dtypes: int64(4), object(3)
memory usage: 19.9+ MB
```

In [96]:
```
# looking at the statistic
df_birth.describe()
```

Out[96]:

	year	month	day	births
count	372864.000000	372864.000000	372864.000000	372864.000000
mean	1978.495350	6.522394	15.741927	189.040878
std	5.767962	3.448818	8.806830	207.460454
min	1969.000000	1.000000	1.000000	1.000000
25%	1973.000000	4.000000	8.000000	52.000000
50%	1978.000000	7.000000	16.000000	129.000000
75%	1983.000000	10.000000	23.000000	223.000000
max	1988.000000	12.000000	31.000000	1779.000000

In [97]:
```
# show the column names
df_birth.columns
```

Out[97]:
```
Index(['state', 'year', 'month', 'day', 'date', 'wday', 'births'], dtype='object')
```

In [103…
```
# create object type pandas from groupby. Save this object in df_day
df_day = df_birth.groupby('day')
```

In [105…
```
# apply aggregation function to the object df_day
df_day.sum().head()
```

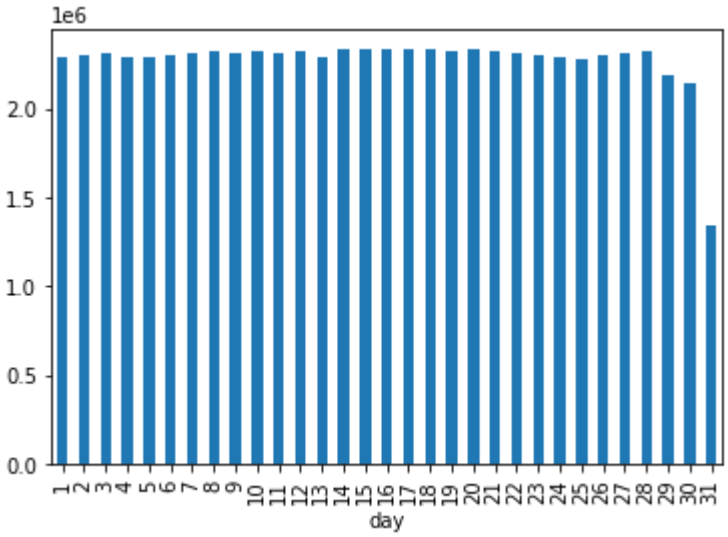
Out[105…

day	year	month	births
1	24216840	79560	2286854

day	year	month	births
2	24216840	79560	2303168
3	24216840	79560	2309087

In [109…

```
df_day.sum()['births'].plot(kind = 'bar');
```



In [101…

```
# display how many datasets are available per year
df_birth.groupby(['year']).count().tail(2)
```

Out[101…

year	state	month	day	date	wday	births
1987	18615	18615	18615	18615	18615	18615
1988	18666	18666	18666	18666	18666	18666

In [112…

```
# groupby by using two arguements. Putting this into a list!
df_birth.groupby(['state', 'wday']).count()
```

Out[112…

state	wday	year	month	day	date	births
AK	**Fri**	1044	1044	1044	1044	1044
	Mon	1044	1044	1044	1044	1044
	Sat	1044	1044	1044	1044	1044
	Sun	1043	1043	1043	1043	1043
	Thurs	1044	1044	1044	1044	1044
…	**…**	…	…	…	…	…

		year	month	day	date	births
state	**wday**					
WY	**Sat**	1044	1044	1044	1044	1044
	Sun	1043	1043	1043	1043	1043
	Thurs	1044	1044	1044	1044	1044
	Tues	1043	1043	1043	1043	1043
	Wed	1044	1044	1044	1044	1044

In [121…

```
# groupby by using two arguements. reset_index()
df_birth.groupby(['state', 'wday']).count().reset_index()
```

Out[121…

	state	wday	year	month	day	date	births
0	AK	Fri	1044	1044	1044	1044	1044
1	AK	Mon	1044	1044	1044	1044	1044
2	AK	Sat	1044	1044	1044	1044	1044
3	AK	Sun	1043	1043	1043	1043	1043
4	AK	Thurs	1044	1044	1044	1044	1044
…	…	…	…	…	…	…	…
352	WY	Sat	1044	1044	1044	1044	1044
353	WY	Sun	1043	1043	1043	1043	1043
354	WY	Thurs	1044	1044	1044	1044	1044
355	WY	Tues	1043	1043	1043	1043	1043
356	WY	Wed	1044	1044	1044	1044	1044

357 rows × 7 columns

Ist diese Abfrage sinnvoll?

Nein - da an jedem Tag die Auswertung erhoben wurde. Daher wenden wir die Abfrage mittels einem anderem Aggregator an.

In [113…

```
# groupby only shown one single result
df_birth.groupby(['state', 'wday']).sum()['births']
```

Out[113…

```
state  wday
AK     Fri      27240
       Mon      27352
       Sat      23307
       Sun      22321
       Thurs    28291
                ...
WY     Sat      20380
       Sun      19098
       Thurs    22673
```

```
       Tues      23435
       Wed       23218
Name: births, Length: 357, dtype: int64
```

In [116…

```
# groupby in combination with describe()
df_birth.groupby('wday').describe()['births'].T
```

Out[116…

wday	Fri	Mon	Sat	Sun	Thurs	Tues	
count	53275.000000	53242.000000	53280.000000	53240.000000	53291.000000	53238.000000	53298.00(
mean	198.842309	194.808967	170.638363	162.418295	195.811038	203.124235	197.63
std	216.741181	212.710378	186.625299	178.022728	214.212605	220.723691	215.49(
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00(
25%	55.000000	54.000000	48.000000	46.000000	54.000000	56.000000	54.00(
50%	136.000000	132.000000	116.000000	110.000000	134.000000	139.000000	135.00(
75%	234.000000	230.000000	198.000000	189.250000	229.000000	238.000000	231.00(
max	1779.000000	1710.000000	1463.000000	1400.000000	1715.000000	1773.000000	1712.00(

In [117…

```
# determine the type of the groupby method
df_birth_stat = df_birth.groupby('wday').describe()['births'].T
type(df_birth_stat)
```

Out[117…

```
pandas.core.frame.DataFrame
```

In [118…

```
# sort columns
wdays = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun']
df_birth_stat = df_birth_stat[wdays]
df_birth_stat
```

Out[118…

wday	Mon	Tues	Wed	Thurs	Fri	Sat	
count	53242.000000	53238.000000	53298.000000	53291.000000	53275.000000	53280.000000	53240.00(
mean	194.808967	203.124235	197.634789	195.811038	198.842309	170.638363	162.41
std	212.710378	220.723691	215.490691	214.212605	216.741181	186.625299	178.02
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00(
25%	54.000000	56.000000	54.000000	54.000000	55.000000	48.000000	46.00(
50%	132.000000	139.000000	135.000000	134.000000	136.000000	116.000000	110.00(
75%	230.000000	238.000000	231.000000	229.000000	234.000000	198.000000	189.25(
max	1710.000000	1773.000000	1712.000000	1715.000000	1779.000000	1463.000000	1400.00(

In [119…
```
# sort columns

df_birth_stat = df_birth_stat[['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'S
df_birth_stat
```

Out[119…

wday	Mon	Tues	Wed	Thurs	Fri	Sat	
count	53242.000000	53238.000000	53298.000000	53291.000000	53275.000000	53280.000000	53240.000
mean	194.808967	203.124235	197.634789	195.811038	198.842309	170.638363	162.418
std	212.710378	220.723691	215.490691	214.212605	216.741181	186.625299	178.022
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000
25%	54.000000	56.000000	54.000000	54.000000	55.000000	48.000000	46.000
50%	132.000000	139.000000	135.000000	134.000000	136.000000	116.000000	110.000
75%	230.000000	238.000000	231.000000	229.000000	234.000000	198.000000	189.250
max	1710.000000	1773.000000	1712.000000	1715.000000	1779.000000	1463.000000	1400.000

In [122…
```
df_birth.columns
```

Out[122…
```
Index(['state', 'year', 'month', 'day', 'date', 'wday', 'births'], dtype='obje
ct')
```

In [123…
```
# combining aggregation by using aggregate()
df_birth.groupby('wday').aggregate([min, np.median, max])['births']
```

Out[123…

wday	min	median	max
Fri	1	136	1779
Mon	1	132	1710
Sat	1	116	1463
Sun	1	110	1400
Thurs	1	134	1715
Tues	1	139	1773
Wed	1	135	1712

Pivot

In [124…
```
# pivot_tabel by using aggfunc default 'mean'
# df.pivot_table(value, row, column, aggfunc)
df_birth.pivot_table('births','month', 'wday')
```

Out[124…

wday	Fri	Mon	Sat	Sun	Thurs	Tues	Wed

month							
1	191.381264	188.446924	166.849455	159.222371	187.586275	193.265945	188.416889
2	195.239062	190.584056	169.061493	160.088857	192.310834	197.633575	193.178218
3	194.825332	192.428540	167.628553	159.464860	192.301916	199.345230	193.381684
4	190.213327	187.824925	163.321429	153.494004	188.692570	196.204924	190.050687
5	194.505398	185.087344	164.803268	156.681428	190.964530	197.385027	191.889114
6	197.974176	194.777477	168.657064	160.328929	196.110394	200.868726	196.604868
7	206.738048	200.473293	177.210339	167.990865	205.806972	211.700375	206.562238
8	209.825072	206.208636	180.765367	171.385185	206.824882	214.317068	208.266043
9	214.914286	203.014810	183.010369	174.741995	211.455475	216.383975	212.748979
10	202.133769	197.346034	172.659396	165.477496	198.974223	204.368267	199.515091
11	194.287886	196.132894	167.627563	161.245946	185.082065	202.086403	194.394755
12	193.849822	194.958581	166.079100	158.624521	193.257325	203.265257	196.179739

rename und index

In [125…

```
# renaming of columns
df_birth_stat = df_birth_stat.rename(columns = {'Tues': 'Tue', 'Thurs': 'Thu'
df_birth_stat
```

Out[125…

wday	Mon	Tue	Wed	Thu	Fri	Sat	
count	53242.000000	53238.000000	53298.000000	53291.000000	53275.000000	53280.000000	53240.00(
mean	194.808967	203.124235	197.634789	195.811038	198.842309	170.638363	162.41
std	212.710378	220.723691	215.490691	214.212605	216.741181	186.625299	178.02
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00(
25%	54.000000	56.000000	54.000000	54.000000	55.000000	48.000000	46.00(
50%	132.000000	139.000000	135.000000	134.000000	136.000000	116.000000	110.00(
75%	230.000000	238.000000	231.000000	229.000000	234.000000	198.000000	189.25(
max	1710.000000	1773.000000	1712.000000	1715.000000	1779.000000	1463.000000	1400.00(

In [130…

```
# reset_index()
df_birth_stat = df_birth_stat.reset_index()
df_birth_stat
```

Out[130…

wday	Mon	Tue	Wed	Thu	Fri	Sat	
0	53242.000000	53238.000000	53298.000000	53291.000000	53275.000000	53280.000000	53240.00(
1	194.808967	203.124235	197.634789	195.811038	198.842309	170.638363	162.41

wday	Mon	Tue	Wed	Thu	Fri	Sat	
2	212.710378	220.723691	215.490691	214.212605	216.741181	186.625299	178.022
3	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000
4	54.000000	56.000000	54.000000	54.000000	55.000000	48.000000	46.000
5	132.000000	139.000000	135.000000	134.000000	136.000000	116.000000	110.000
6	230.000000	238.000000	231.000000	229.000000	234.000000	198.000000	189.250

In [131…

```
# set_index()
df_birth_stat = df_birth_stat.set_index('Mon')
df_birth_stat
```

Out[131…

wday Mon	Tue	Wed	Thu	Fri	Sat	Sun
53242.000000	53238.000000	53298.000000	53291.000000	53275.000000	53280.000000	53240.000000
194.808967	203.124235	197.634789	195.811038	198.842309	170.638363	162.418295
212.710378	220.723691	215.490691	214.212605	216.741181	186.625299	178.022728
1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
54.000000	56.000000	54.000000	54.000000	55.000000	48.000000	46.000000
132.000000	139.000000	135.000000	134.000000	136.000000	116.000000	110.000000
230.000000	238.000000	231.000000	229.000000	234.000000	198.000000	189.250000
1710.000000	1773.000000	1712.000000	1715.000000	1779.000000	1463.000000	1400.000000

In [ ]: