pandas

1 Introduction to pandas dataframes

Pandas is a library providing high-performance, easy-to-use data structures and data analysis tools for Python.

In particulars, it enables the user to work with tabular datastructures. The main keywords used in pandas are tables, known as **Dataframes** and arrays, known as **Series**. Pandas has a huge number of api for file parsing, data manipulation, visualization, data groupby and so on. We will look at only some of the key features with some practical examples.

Some useful reference:

- Pandas 10 minutes intro: http://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html#min
- Pandas cookbook: http://pandas.pydata.org/pandas-docs/stable/user_guide/cookbook.html#cookbook
- Pandas cheat sheet: http://pandas.pydata.org/Pandas_Cheat_Sheet.pdf
- Pandas general user guide: http://pandas.pydata.org/pandas-docs/stable/user_guide/index.html

[1]:
```
# the usual import statement is this one
import pandas as pd
# just in case
import numpy as np
```

2 Dataframes and series

Dataframes are two dimensional datastructure with possibly heterogeneous data. The easiest way to think about them is a table. Each dataframe has a number of rows and columns. The rows are indexed and the columns are labeled.

[2]:
```
# example of dataframe
df = pd.DataFrame({'fruit': ['apple', 'pear', 'banana'],
                   'price': [1.5, 0.9, 1.1]})

df
```

[2]:

	fruit	price
0	apple	1.5

```
1    pear    0.9
2  banana    1.1
```

The columns in a pandas dataframe are known as **series**. A series is like a numpy array with some more pandas api on top of it.

[3]:
```
s = df.fruit
print(s)
print(type(s))
```

```
0     apple
1      pear
2    banana
Name: fruit, dtype: object
<class 'pandas.core.series.Series'>
```

Pandas dataframes and series are built on top of numpy arrays and the underlying datastructur can be always accessed via the `.values` attribute

[4]:
```
df.values
```

[4]:
```
array([['apple', 1.5],
       ['pear', 0.9],
       ['banana', 1.1]], dtype=object)
```

Let's look at some example of I/O file manipulation

3 From files to pandas

Pandas has several functions to import data from one format into a dataframe. Let's look at the most used examples

3.1 From csv to dataframe

`pd.read_csv()` can read csv data into dataframes. There are lot of arguments that can be passed to configure the parsing of the file.

[6]:
```
# let's read a csv from an url:
# https://vincentarelbundock.github.io/Rdatasets/doc/DAAG/frogs.html
csv_url = 'https://vincentarelbundock.github.io/Rdatasets/csv/DAAG/frogs.csv'
df = pd.read_csv(csv_url)
df.head()  # prints only the first rows
```

[6]:

	Unnamed: 0	pres.abs	northing	easting	altitude	distance	NoOfPools	\
0	2	1	115	1047	1500	500	232	
1	3	1	110	1042	1520	250	66	
2	4	1	112	1040	1540	250	32	
3	5	1	109	1033	1590	250	9	

```
4           6         1       109      1032      1590       250         67
```

```
   NoOfSites       avrain   meanmin    meanmax
0          3   155.000000  3.566667  14.000000
1          5   157.666667  3.466667  13.800000
2          5   159.666667  3.400000  13.600000
3          5   165.000000  3.200000  13.166667
4          5   165.000000  3.200000  13.166667
```

It's also possible to read data from a particular format csv/txt/tsv file

[7]:
```
%%writefile csv_example1.csv
first_column;second_column;third_column
'a';12;3
'b1';13;5
```

```
Overwriting csv_example1.csv
```

[8]:
```
df = pd.read_csv('csv_example1.csv',
                 delimiter=';',  # specify a different delimiter
                 quotechar="'") # specify that strings are enclosed by ''
df
```

[8]:
```
  first_column  second_column  third_column
0            a             12             3
1           b1             13             5
```

Pandas can read also data from excel files

[9]:
```
url = 'https://github.com/bharathirajatut/sample-excel-dataset/blob/master/
 ↪airline.xls?raw=true'
df = pd.read_excel(url)
df.head()
```

[9]:
```
   YEAR      Y      W       R      L      K
0  1948  1.214  0.243  0.1454  1.415  0.612
1  1949  1.354  0.260  0.2181  1.384  0.559
2  1950  1.569  0.278  0.3157  1.388  0.573
3  1951  1.948  0.297  0.3940  1.550  0.564
4  1952  2.265  0.310  0.3559  1.802  0.574
```

It has also some nice functions to parse simple html tables from a webpage

[10]:
```
url = 'https://en.wikipedia.org/wiki/Table_(information)#Simple_table'
df_list = pd.read_html(url)
df_list
```

[10]: [0   1
0 NaN This article may require cleanup to meet Wikip...,

```
           0           1    2
0  First name   Last name  Age
1        Tinu    Elejogun   14
2   Blaszczyk  Kostrzewski   25
3        Lily   McGarrett   18
4   Olatunkbo    Chijiaku   22
5    Adrienne    Anthoula   22
6      Axelia  Athanasios   22
7   Jon-Kabat        Zinn   22
8     Thabang       Mosoa   15
9    Kgaogelo       Mosoa   11,
```

```
   0  1  2  3
0  ×  1  2  3
1  1  1  2  3
2  2  2  4  6
3  3  3  6  9,
```

```
                         0                                                  1
\
0  Standard Representation                          Tabular Representation
1                    2 3 1  Risk levels of hazardous materials in this fac...
2              Health Risk                                       Flammability
3                  Level 3                                            Level 2
4              Health Risk                                       Flammability
5                  Level 3                                            Level 2

             2            3          4        5        6        7        8  \
0          NaN          NaN        NaN      NaN      NaN      NaN      NaN
1  Health Risk  Flammability  Reactivity  Special  Level 3  Level 2  Level 1
2   Reactivity      Special        NaN      NaN      NaN      NaN      NaN
3      Level 1          NaN        NaN      NaN      NaN      NaN      NaN
4   Reactivity      Special        NaN      NaN      NaN      NaN      NaN
5      Level 1          NaN        NaN      NaN      NaN      NaN      NaN

    9
0 NaN
1 NaN
2 NaN
3 NaN
4 NaN
5 NaN  ,
             0            1          2        3
0  Health Risk  Flammability  Reactivity  Special
1      Level 3      Level 2     Level 1      NaN,
```

```
    0                                                  1
0 NaN  Look up table in Wiktionary, the free dictionary.,
```

```
                                              0  \
0  vteVisualization of technical information
1                                        Fields
2                                   Image types
3                                        People
4                                Related topics

                                                    1
0                                                 NaN
1  Biological data visualization Chemical imaging...
2  Chart Diagram Engineering drawing Graph of a f...
3  Jacques Bertin Stuart Card Thomas A. DeFanti M...
4  Cartography Chartjunk Computer graphics in com...  ]
```

[16]:
```
df_list[1]
```

[16]:
```
            0            1    2
0  First name    Last name  Age
1        Tinu     Elejogun   14
2   Blaszczyk  Kostrzewski   25
3        Lily    McGarrett   18
4   Olatunkbo     Chijiaku   22
5    Adrienne     Anthoula   22
6      Axelia   Athanasios   22
7   Jon-Kabat         Zinn   22
8     Thabang        Mosoa   15
9    Kgaogelo        Mosoa   11
```

Further reference on supported file formats: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

4 Data indexing and selection

Let's look at the different way to access data from a dataframe

[18]:
```
csv_url = 'https://raw.githubusercontent.com/plotly/datasets/master/timeseries.
 ↪csv'
df = pd.read_csv(csv_url,
                 index_col=0, # set the index to the first column
                 parse_dates=True) # try to parse the date in the index

df
```

[18]:
```
                A       B       C      D      E      F      G
Date
2008-03-18  24.68  164.93  114.73  26.27  19.21  28.87  63.44
2008-03-19  24.18  164.89  114.75  26.22  19.07  27.76  59.98
```

```
2008-03-20  23.99  164.63  115.04  25.78  19.01  27.04  59.61
2008-03-25  24.14  163.92  114.85  27.41  19.61  27.84  59.41
2008-03-26  24.44  163.45  114.84  26.86  19.53  28.02  60.09
2008-03-27  24.38  163.46  115.40  27.09  19.72  28.25  59.62
2008-03-28  24.32  163.22  115.56  27.13  19.63  28.24  58.65
2008-03-31  24.19  164.02  115.54  26.74  19.55  28.43  59.20
2008-04-01  23.81  163.59  115.72  27.82  20.21  29.17  56.18
2008-04-02  24.03  163.32  115.11  28.22  20.42  29.38  56.64
2008-04-03  24.34  163.34  115.17  28.14  20.36  29.51  57.49
```

[21]:
```
# access one row:
# given a known specific index: loc
df.loc['2008-03-25']
# as if the df was a numpy array: iloc
df.iloc[3]  # acccess the 2d array structure
```

[21]:
```
A     24.14
B    163.92
C    114.85
D     27.41
E     19.61
F     27.84
G     59.41
Name: 2008-03-25 00:00:00, dtype: float64
```

[25]:
```
df['A']
df.A
```

[25]:
```
Date
2008-03-18    24.68
2008-03-19    24.18
2008-03-20    23.99
2008-03-25    24.14
2008-03-26    24.44
2008-03-27    24.38
2008-03-28    24.32
2008-03-31    24.19
2008-04-01    23.81
2008-04-02    24.03
2008-04-03    24.34
Name: A, dtype: float64
```

[26]:
```
df.values
```

[26]:
```
array([[ 24.68, 164.93, 114.73,  26.27,  19.21,  28.87,  63.44],
       [ 24.18, 164.89, 114.75,  26.22,  19.07,  27.76,  59.98],
       [ 23.99, 164.63, 115.04,  25.78,  19.01,  27.04,  59.61],
```

```
       [ 24.14, 163.92, 114.85,  27.41,  19.61,  27.84,  59.41],
       [ 24.44, 163.45, 114.84,  26.86,  19.53,  28.02,  60.09],
       [ 24.38, 163.46, 115.4 ,  27.09,  19.72,  28.25,  59.62],
       [ 24.32, 163.22, 115.56,  27.13,  19.63,  28.24,  58.65],
       [ 24.19, 164.02, 115.54,  26.74,  19.55,  28.43,  59.2 ],
       [ 23.81, 163.59, 115.72,  27.82,  20.21,  29.17,  56.18],
       [ 24.03, 163.32, 115.11,  28.22,  20.42,  29.38,  56.64],
       [ 24.34, 163.34, 115.17,  28.14,  20.36,  29.51,  57.49]])
```

[27]:
```
# combined access of rows and columns:
# access the first 5 rows and the
# last three columns
df.iloc[:5, -3:]
```

[27]:

Date	E	F	G
2008-03-18	19.21	28.87	63.44
2008-03-19	19.07	27.76	59.98
2008-03-20	19.01	27.04	59.61
2008-03-25	19.61	27.84	59.41
2008-03-26	19.53	28.02	60.09

[31]:
```
# Access rows from 2008-03-19 to 2008-03-30
# and only B, C, D columns
df.loc['2008-03-19':'2008-03-30', ['B', 'C', 'D']]
```

[31]:

Date	B	C	D
2008-03-19	164.89	114.75	26.22
2008-03-20	164.63	115.04	25.78
2008-03-25	163.92	114.85	27.41
2008-03-26	163.45	114.84	26.86
2008-03-27	163.46	115.40	27.09
2008-03-28	163.22	115.56	27.13

[32]:
```
(df.B > 163.3) & (df.index < '2008-03-31')
```

[32]:
```
Date
2008-03-18     True
2008-03-19     True
2008-03-20     True
2008-03-25     True
2008-03-26     True
2008-03-27     True
2008-03-28    False
2008-03-31    False
2008-04-01    False
```

```
2008-04-02    False
2008-04-03    False
Name: B, dtype: bool
```

[33]:
```
# query data for particular conditions (like in SQL)
# Method 1: subsetting
df[(df.B > 163.3) & (df.index < '2008-03-31')]

# Method 2: query() function
df.query('B > 163.3 & index < "2008-03-31"')
```

[33]:

Date	A	B	C	D	E	F	G
2008-03-18	24.68	164.93	114.73	26.27	19.21	28.87	63.44
2008-03-19	24.18	164.89	114.75	26.22	19.07	27.76	59.98
2008-03-20	23.99	164.63	115.04	25.78	19.01	27.04	59.61
2008-03-25	24.14	163.92	114.85	27.41	19.61	27.84	59.41
2008-03-26	24.44	163.45	114.84	26.86	19.53	28.02	60.09
2008-03-27	24.38	163.46	115.40	27.09	19.72	28.25	59.62

4.1 Dataframe data manipulation

Data can be also manipulated by adding and removing column. The operations usually returns a copy of the array so the result of the operation must be assigned to a new variable

[36]:
```
# add a column from the operation
# of two other columns
df['Z'] = df.A + df.B
df
```

[36]:

Date	A	B	C	D	E	F	G	Z	W
2008-03-18	24.68	164.93	114.73	26.27	19.21	28.87	63.44	189.61	13
2008-03-19	24.18	164.89	114.75	26.22	19.07	27.76	59.98	189.07	13
2008-03-20	23.99	164.63	115.04	25.78	19.01	27.04	59.61	188.62	13
2008-03-25	24.14	163.92	114.85	27.41	19.61	27.84	59.41	188.06	13
2008-03-26	24.44	163.45	114.84	26.86	19.53	28.02	60.09	187.89	13
2008-03-27	24.38	163.46	115.40	27.09	19.72	28.25	59.62	187.84	13
2008-03-28	24.32	163.22	115.56	27.13	19.63	28.24	58.65	187.54	13
2008-03-31	24.19	164.02	115.54	26.74	19.55	28.43	59.20	188.21	13
2008-04-01	23.81	163.59	115.72	27.82	20.21	29.17	56.18	187.40	13
2008-04-02	24.03	163.32	115.11	28.22	20.42	29.38	56.64	187.35	13
2008-04-03	24.34	163.34	115.17	28.14	20.36	29.51	57.49	187.68	13

[35]:
```
# add a single constant value
df['W'] = 13
```

```
df.head()
```

[35]:

Date	A	B	C	D	E	F	G	Z	W
2008-03-18	24.68	164.93	114.73	26.27	19.21	28.87	63.44	189.61	13
2008-03-19	24.18	164.89	114.75	26.22	19.07	27.76	59.98	189.07	13
2008-03-20	23.99	164.63	115.04	25.78	19.01	27.04	59.61	188.62	13
2008-03-25	24.14	163.92	114.85	27.41	19.61	27.84	59.41	188.06	13
2008-03-26	24.44	163.45	114.84	26.86	19.53	28.02	60.09	187.89	13

[37]:
```
# Remove column Z
# Method 1: inverse subsetting
df = df.loc[:, df.columns != 'Z']

# Method 2: calling appropriate function
df = df.drop(columns=['W'])
df.head()
```

[37]:

Date	A	B	C	D	E	F	G
2008-03-18	24.68	164.93	114.73	26.27	19.21	28.87	63.44
2008-03-19	24.18	164.89	114.75	26.22	19.07	27.76	59.98
2008-03-20	23.99	164.63	115.04	25.78	19.01	27.04	59.61
2008-03-25	24.14	163.92	114.85	27.41	19.61	27.84	59.41
2008-03-26	24.44	163.45	114.84	26.86	19.53	28.02	60.09

[38]:
```
# Add a row
row = {'A': 25, 'B': 133, 'C': 111, 'D': 26, 'E': 20, 'F': 29, 'G':60}
# appending like this will make us loose the time series index
df.append(row, ignore_index=True)
```

[38]:

	A	B	C	D	E	F	G
0	24.68	164.93	114.73	26.27	19.21	28.87	63.44
1	24.18	164.89	114.75	26.22	19.07	27.76	59.98
2	23.99	164.63	115.04	25.78	19.01	27.04	59.61
3	24.14	163.92	114.85	27.41	19.61	27.84	59.41
4	24.44	163.45	114.84	26.86	19.53	28.02	60.09
5	24.38	163.46	115.40	27.09	19.72	28.25	59.62
6	24.32	163.22	115.56	27.13	19.63	28.24	58.65
7	24.19	164.02	115.54	26.74	19.55	28.43	59.20
8	23.81	163.59	115.72	27.82	20.21	29.17	56.18
9	24.03	163.32	115.11	28.22	20.42	29.38	56.64
10	24.34	163.34	115.17	28.14	20.36	29.51	57.49
11	25.00	133.00	111.00	26.00	20.00	29.00	60.00

[40]:
```
# Add a row
df.loc[pd.to_datetime('2008-11-10')] = [25, 133, 111, 26, 20, 29, 60]
```

```
df
```

[40]:

Date	A	B	C	D	E	F	G
2008-03-18	24.68	164.93	114.73	26.27	19.21	28.87	63.44
2008-03-19	24.18	164.89	114.75	26.22	19.07	27.76	59.98
2008-03-20	23.99	164.63	115.04	25.78	19.01	27.04	59.61
2008-03-25	24.14	163.92	114.85	27.41	19.61	27.84	59.41
2008-03-26	24.44	163.45	114.84	26.86	19.53	28.02	60.09
2008-03-27	24.38	163.46	115.40	27.09	19.72	28.25	59.62
2008-03-28	24.32	163.22	115.56	27.13	19.63	28.24	58.65
2008-03-31	24.19	164.02	115.54	26.74	19.55	28.43	59.20
2008-04-01	23.81	163.59	115.72	27.82	20.21	29.17	56.18
2008-04-02	24.03	163.32	115.11	28.22	20.42	29.38	56.64
2008-04-03	24.34	163.34	115.17	28.14	20.36	29.51	57.49
2008-11-10	25.00	133.00	111.00	26.00	20.00	29.00	60.00

[41]:
```
# remove a row based on the date
df.drop(index=pd.to_datetime('2008-04-03'))
```

[41]:

Date	A	B	C	D	E	F	G
2008-03-18	24.68	164.93	114.73	26.27	19.21	28.87	63.44
2008-03-19	24.18	164.89	114.75	26.22	19.07	27.76	59.98
2008-03-20	23.99	164.63	115.04	25.78	19.01	27.04	59.61
2008-03-25	24.14	163.92	114.85	27.41	19.61	27.84	59.41
2008-03-26	24.44	163.45	114.84	26.86	19.53	28.02	60.09
2008-03-27	24.38	163.46	115.40	27.09	19.72	28.25	59.62
2008-03-28	24.32	163.22	115.56	27.13	19.63	28.24	58.65
2008-03-31	24.19	164.02	115.54	26.74	19.55	28.43	59.20
2008-04-01	23.81	163.59	115.72	27.82	20.21	29.17	56.18
2008-04-02	24.03	163.32	115.11	28.22	20.42	29.38	56.64
2008-11-10	25.00	133.00	111.00	26.00	20.00	29.00	60.00

[45]:
```
# Remove rows where total row sum is less
# than 437
mask = df.sum(axis=1) < 437
df[mask]
```

[45]:

Date	A	B	C	D	E	F	G
2008-03-19	24.18	164.89	114.75	26.22	19.07	27.76	59.98
2008-03-20	23.99	164.63	115.04	25.78	19.01	27.04	59.61
2008-03-28	24.32	163.22	115.56	27.13	19.63	28.24	58.65
2008-04-01	23.81	163.59	115.72	27.82	20.21	29.17	56.18
2008-11-10	25.00	133.00	111.00	26.00	20.00	29.00	60.00

5 Groupby operations

pandas dataframes have a very powerful SQL-like groupby methods to help achieve some data manipulation that would require lot of work otherwise. Grouping can be useful for computing aggregated data statistics or for visualization.

[49]:
```
url = 'https://raw.githubusercontent.com/Pierian-Data/AutoArima-Time-Series-Blog/
 ↪master/Electric_Production.csv'
df = pd.read_csv(url, index_col=0, parse_dates=True)
df.head()
```

[49]:
```
            IPG2211A2N
DATE
1985-01-01     72.5052
1985-02-01     70.6720
1985-03-01     62.4502
1985-04-01     57.4714
1985-05-01     55.3151
```

[50]:
```
# I want to group by month
group = df.groupby(by=df.index.year)
group
```

[50]:
```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x7fb7c8166860>
```

As you see nothing is printed out. This is because the groupby object is expecting that some aggregation function is executed in order to print out the results as a new dataframe. However we can inspect the internal groups in the groupby object

[51]:
```
group.groups
```

[51]:
```
{1985: DatetimeIndex(['1985-01-01', '1985-02-01', '1985-03-01', '1985-04-01',
                      '1985-05-01', '1985-06-01', '1985-07-01', '1985-08-01',
                      '1985-09-01', '1985-10-01', '1985-11-01', '1985-12-01'],
                     dtype='datetime64[ns]', name='DATE', freq=None),
 1986: DatetimeIndex(['1986-01-01', '1986-02-01', '1986-03-01', '1986-04-01',
                      '1986-05-01', '1986-06-01', '1986-07-01', '1986-08-01',
                      '1986-09-01', '1986-10-01', '1986-11-01', '1986-12-01'],
                     dtype='datetime64[ns]', name='DATE', freq=None),
 1987: DatetimeIndex(['1987-01-01', '1987-02-01', '1987-03-01', '1987-04-01',
                      '1987-05-01', '1987-06-01', '1987-07-01', '1987-08-01',
                      '1987-09-01', '1987-10-01', '1987-11-01', '1987-12-01'],
                     dtype='datetime64[ns]', name='DATE', freq=None),
 1988: DatetimeIndex(['1988-01-01', '1988-02-01', '1988-03-01', '1988-04-01',
                      '1988-05-01', '1988-06-01', '1988-07-01', '1988-08-01',
                      '1988-09-01', '1988-10-01', '1988-11-01', '1988-12-01'],
                     dtype='datetime64[ns]', name='DATE', freq=None),
 1989: DatetimeIndex(['1989-01-01', '1989-02-01', '1989-03-01', '1989-04-01',
```

```
               '1989-05-01', '1989-06-01', '1989-07-01', '1989-08-01',
               '1989-09-01', '1989-10-01', '1989-11-01', '1989-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
1990: DatetimeIndex(['1990-01-01', '1990-02-01', '1990-03-01', '1990-04-01',
               '1990-05-01', '1990-06-01', '1990-07-01', '1990-08-01',
               '1990-09-01', '1990-10-01', '1990-11-01', '1990-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
1991: DatetimeIndex(['1991-01-01', '1991-02-01', '1991-03-01', '1991-04-01',
               '1991-05-01', '1991-06-01', '1991-07-01', '1991-08-01',
               '1991-09-01', '1991-10-01', '1991-11-01', '1991-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
1992: DatetimeIndex(['1992-01-01', '1992-02-01', '1992-03-01', '1992-04-01',
               '1992-05-01', '1992-06-01', '1992-07-01', '1992-08-01',
               '1992-09-01', '1992-10-01', '1992-11-01', '1992-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
1993: DatetimeIndex(['1993-01-01', '1993-02-01', '1993-03-01', '1993-04-01',
               '1993-05-01', '1993-06-01', '1993-07-01', '1993-08-01',
               '1993-09-01', '1993-10-01', '1993-11-01', '1993-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
1994: DatetimeIndex(['1994-01-01', '1994-02-01', '1994-03-01', '1994-04-01',
               '1994-05-01', '1994-06-01', '1994-07-01', '1994-08-01',
               '1994-09-01', '1994-10-01', '1994-11-01', '1994-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
1995: DatetimeIndex(['1995-01-01', '1995-02-01', '1995-03-01', '1995-04-01',
               '1995-05-01', '1995-06-01', '1995-07-01', '1995-08-01',
               '1995-09-01', '1995-10-01', '1995-11-01', '1995-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
1996: DatetimeIndex(['1996-01-01', '1996-02-01', '1996-03-01', '1996-04-01',
               '1996-05-01', '1996-06-01', '1996-07-01', '1996-08-01',
               '1996-09-01', '1996-10-01', '1996-11-01', '1996-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
1997: DatetimeIndex(['1997-01-01', '1997-02-01', '1997-03-01', '1997-04-01',
               '1997-05-01', '1997-06-01', '1997-07-01', '1997-08-01',
               '1997-09-01', '1997-10-01', '1997-11-01', '1997-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
1998: DatetimeIndex(['1998-01-01', '1998-02-01', '1998-03-01', '1998-04-01',
               '1998-05-01', '1998-06-01', '1998-07-01', '1998-08-01',
               '1998-09-01', '1998-10-01', '1998-11-01', '1998-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
1999: DatetimeIndex(['1999-01-01', '1999-02-01', '1999-03-01', '1999-04-01',
               '1999-05-01', '1999-06-01', '1999-07-01', '1999-08-01',
               '1999-09-01', '1999-10-01', '1999-11-01', '1999-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
2000: DatetimeIndex(['2000-01-01', '2000-02-01', '2000-03-01', '2000-04-01',
               '2000-05-01', '2000-06-01', '2000-07-01', '2000-08-01',
               '2000-09-01', '2000-10-01', '2000-11-01', '2000-12-01'],
              dtype='datetime64[ns]', name='DATE', freq=None),
```

```
2001: DatetimeIndex(['2001-01-01', '2001-02-01', '2001-03-01', '2001-04-01',
                     '2001-05-01', '2001-06-01', '2001-07-01', '2001-08-01',
                     '2001-09-01', '2001-10-01', '2001-11-01', '2001-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2002: DatetimeIndex(['2002-01-01', '2002-02-01', '2002-03-01', '2002-04-01',
                     '2002-05-01', '2002-06-01', '2002-07-01', '2002-08-01',
                     '2002-09-01', '2002-10-01', '2002-11-01', '2002-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2003: DatetimeIndex(['2003-01-01', '2003-02-01', '2003-03-01', '2003-04-01',
                     '2003-05-01', '2003-06-01', '2003-07-01', '2003-08-01',
                     '2003-09-01', '2003-10-01', '2003-11-01', '2003-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2004: DatetimeIndex(['2004-01-01', '2004-02-01', '2004-03-01', '2004-04-01',
                     '2004-05-01', '2004-06-01', '2004-07-01', '2004-08-01',
                     '2004-09-01', '2004-10-01', '2004-11-01', '2004-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2005: DatetimeIndex(['2005-01-01', '2005-02-01', '2005-03-01', '2005-04-01',
                     '2005-05-01', '2005-06-01', '2005-07-01', '2005-08-01',
                     '2005-09-01', '2005-10-01', '2005-11-01', '2005-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2006: DatetimeIndex(['2006-01-01', '2006-02-01', '2006-03-01', '2006-04-01',
                     '2006-05-01', '2006-06-01', '2006-07-01', '2006-08-01',
                     '2006-09-01', '2006-10-01', '2006-11-01', '2006-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2007: DatetimeIndex(['2007-01-01', '2007-02-01', '2007-03-01', '2007-04-01',
                     '2007-05-01', '2007-06-01', '2007-07-01', '2007-08-01',
                     '2007-09-01', '2007-10-01', '2007-11-01', '2007-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2008: DatetimeIndex(['2008-01-01', '2008-02-01', '2008-03-01', '2008-04-01',
                     '2008-05-01', '2008-06-01', '2008-07-01', '2008-08-01',
                     '2008-09-01', '2008-10-01', '2008-11-01', '2008-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2009: DatetimeIndex(['2009-01-01', '2009-02-01', '2009-03-01', '2009-04-01',
                     '2009-05-01', '2009-06-01', '2009-07-01', '2009-08-01',
                     '2009-09-01', '2009-10-01', '2009-11-01', '2009-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2010: DatetimeIndex(['2010-01-01', '2010-02-01', '2010-03-01', '2010-04-01',
                     '2010-05-01', '2010-06-01', '2010-07-01', '2010-08-01',
                     '2010-09-01', '2010-10-01', '2010-11-01', '2010-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2011: DatetimeIndex(['2011-01-01', '2011-02-01', '2011-03-01', '2011-04-01',
                     '2011-05-01', '2011-06-01', '2011-07-01', '2011-08-01',
                     '2011-09-01', '2011-10-01', '2011-11-01', '2011-12-01'],
                    dtype='datetime64[ns]', name='DATE', freq=None),
2012: DatetimeIndex(['2012-01-01', '2012-02-01', '2012-03-01', '2012-04-01',
                     '2012-05-01', '2012-06-01', '2012-07-01', '2012-08-01',
                     '2012-09-01', '2012-10-01', '2012-11-01', '2012-12-01'],
```

```
                dtype='datetime64[ns]', name='DATE', freq=None),
 2013: DatetimeIndex(['2013-01-01', '2013-02-01', '2013-03-01', '2013-04-01',
                      '2013-05-01', '2013-06-01', '2013-07-01', '2013-08-01',
                      '2013-09-01', '2013-10-01', '2013-11-01', '2013-12-01'],
                dtype='datetime64[ns]', name='DATE', freq=None),
 2014: DatetimeIndex(['2014-01-01', '2014-02-01', '2014-03-01', '2014-04-01',
                      '2014-05-01', '2014-06-01', '2014-07-01', '2014-08-01',
                      '2014-09-01', '2014-10-01', '2014-11-01', '2014-12-01'],
                dtype='datetime64[ns]', name='DATE', freq=None),
 2015: DatetimeIndex(['2015-01-01', '2015-02-01', '2015-03-01', '2015-04-01',
                      '2015-05-01', '2015-06-01', '2015-07-01', '2015-08-01',
                      '2015-09-01', '2015-10-01', '2015-11-01', '2015-12-01'],
                dtype='datetime64[ns]', name='DATE', freq=None),
 2016: DatetimeIndex(['2016-01-01', '2016-02-01', '2016-03-01', '2016-04-01',
                      '2016-05-01', '2016-06-01', '2016-07-01', '2016-08-01',
                      '2016-09-01', '2016-10-01', '2016-11-01', '2016-12-01'],
                dtype='datetime64[ns]', name='DATE', freq=None),
 2017: DatetimeIndex(['2017-01-01', '2017-02-01', '2017-03-01', '2017-04-01',
                      '2017-05-01', '2017-06-01', '2017-07-01', '2017-08-01',
                      '2017-09-01', '2017-10-01', '2017-11-01', '2017-12-01'],
                dtype='datetime64[ns]', name='DATE', freq=None),
 2018: DatetimeIndex(['2018-01-01'], dtype='datetime64[ns]', name='DATE',
freq=None)}
```

Sometimes it might be helpful to visualize mentally (and on the screen) what is happening to the groupby. The groupby in this case will group by the month in the index. A visual results is equivalent to add a new index like this:

[54]:
```
df.set_index([df.index.year, df.index])
```

[54]:

		IPG2211A2N
DATE	DATE	
1985	1985-01-01	72.5052
	1985-02-01	70.6720
	1985-03-01	62.4502
	1985-04-01	57.4714
	1985-05-01	55.3151
	1985-06-01	58.0904
	1985-07-01	62.6202
	1985-08-01	63.2485
	1985-09-01	60.5846
	1985-10-01	56.3154
	1985-11-01	58.0005
	1985-12-01	68.7145
1986	1986-01-01	73.3057
	1986-02-01	67.9869
	1986-03-01	62.2221

```
      1986-04-01     57.0329
      1986-05-01     55.8137
      1986-06-01     59.9005
      1986-07-01     65.7655
      1986-08-01     64.4816
      1986-09-01     61.0005
      1986-10-01     57.5322
      1986-11-01     59.3417
      1986-12-01     68.1354
1987  1987-01-01     73.8152
      1987-02-01     70.0620
      1987-03-01     65.6100
      1987-04-01     60.1586
      1987-05-01     58.8734
      1987-06-01     63.8918
...                      ...
2015  2015-08-01    110.5925
      2015-09-01    101.9204
      2015-10-01     91.5959
      2015-11-01     93.0628
      2015-12-01    103.2203
2016  2016-01-01    117.0837
      2016-02-01    106.6688
      2016-03-01     95.3548
      2016-04-01     89.3254
      2016-05-01     90.7369
      2016-06-01    104.0375
      2016-07-01    114.5397
      2016-08-01    115.5159
      2016-09-01    102.7637
      2016-10-01     91.4867
      2016-11-01     92.8900
      2016-12-01    112.7694
2017  2017-01-01    114.8505
      2017-02-01     99.4901
      2017-03-01    101.0396
      2017-04-01     88.3530
      2017-05-01     92.0805
      2017-06-01    102.1532
      2017-07-01    112.1538
      2017-08-01    108.9312
      2017-09-01     98.6154
      2017-10-01     93.6137
      2017-11-01     97.3359
      2017-12-01    114.7212
2018  2018-01-01    129.4048
```

```
[397 rows x 1 columns]
```

We can now apply some functions. For example, we want to compute the mean value for each column for the three months.

[57]:
```
group.mean()
```

[57]:
```
      IPG2211A2N
DATE
1985   62.165667
1986   62.709892
1987   65.740275
1988   69.716358
1989   71.895167
1990   73.313433
1991   75.111850
1992   75.120908
1993   77.678992
1994   79.255058
1995   82.060867
1996   84.382417
1997   84.236192
1998   86.544075
1999   89.190892
2000   91.790125
2001   91.460792
2002   94.107075
2003   95.700108
2004   97.119225
2005   99.208242
2006   98.873225
2007  101.984167
2008  101.663992
2009   98.829783
2010  102.596883
2011  102.290475
2012   99.999992
2013  102.337350
2014  103.732783
2015  103.052908
2016  102.764375
2017  101.944842
2018  129.404800
```

[58]:
```
# more sofisticated: we want to apply a
# custom function. We can use a
# a lambda function
```

```
def custom_function(array):
    return np.std(array)

group.apply(lambda x: x.min())
group.apply(custom_function)
```

[58]:

DATE	IPG2211A2N
1985	5.501963
1986	5.091016
1987	4.651648
1988	5.735234
1989	5.796188
1990	5.278903
1991	5.331017
1992	5.633756
1993	6.302929
1994	7.061054
1995	6.664940
1996	6.980029
1997	6.925562
1998	6.022114
1999	7.200824
2000	6.885528
2001	7.898586
2002	6.343009
2003	8.369536
2004	8.164091
2005	8.620553
2006	7.618600
2007	8.048787
2008	8.240594
2009	9.173033
2010	10.785192
2011	9.909236
2012	8.338682
2013	8.259866
2014	9.377916
2015	9.828350
2016	10.116512
2017	8.531759
2018	0.000000

[59]:
```
# We can also groupby by multiple labels:
# for example, we want to groupby year
# and month
group2 = df.groupby(by=[df.index.year, df.index.month])
```

```
group2.mean()
```

[59]:

		IPG2211A2N
DATE	DATE	
1985	1	72.5052
	2	70.6720
	3	62.4502
	4	57.4714
	5	55.3151
	6	58.0904
	7	62.6202
	8	63.2485
	9	60.5846
	10	56.3154
	11	58.0005
	12	68.7145
1986	1	73.3057
	2	67.9869
	3	62.2221
	4	57.0329
	5	55.8137
	6	59.9005
	7	65.7655
	8	64.4816
	9	61.0005
	10	57.5322
	11	59.3417
	12	68.1354
1987	1	73.8152
	2	70.0620
	3	65.6100
	4	60.1586
	5	58.8734
	6	63.8918
...		...
2015	8	110.5925
	9	101.9204
	10	91.5959
	11	93.0628
	12	103.2203
2016	1	117.0837
	2	106.6688
	3	95.3548
	4	89.3254
	5	90.7369
	6	104.0375
	7	114.5397

```
      8       115.5159
      9       102.7637
      10       91.4867
      11       92.8900
      12      112.7694
2017  1       114.8505
      2        99.4901
      3       101.0396
      4        88.3530
      5        92.0805
      6       102.1532
      7       112.1538
      8       108.9312
      9        98.6154
      10       93.6137
      11       97.3359
      12      114.7212
2018  1       129.4048

[397 rows x 1 columns]
```

[61]:
```
# if we want save this dataframe in a more usable way
# for plotting or for using it with a different application
# we can flatten the index with reset_index().
# Judt be sure that if there is a multi index
# (it usually happens when grouping by multiple columns)
# the indexes have different names
agg_mean = group2.mean()
agg_mean.index = agg_mean.index.set_names(['year', 'month'])
agg_mean.head()
```

[61]:
```
            IPG2211A2N
year month
1985 1         72.5052
     2         70.6720
     3         62.4502
     4         57.4714
     5         55.3151
```

[62]:
```
# flatten multi index
agg_mean.reset_index()
```

[62]:
```
     year  month  IPG2211A2N
0    1985      1     72.5052
1    1985      2     70.6720
2    1985      3     62.4502
3    1985      4     57.4714
```

4	1985	5	55.3151
5	1985	6	58.0904
6	1985	7	62.6202
7	1985	8	63.2485
8	1985	9	60.5846
9	1985	10	56.3154
10	1985	11	58.0005
11	1985	12	68.7145
12	1986	1	73.3057
13	1986	2	67.9869
14	1986	3	62.2221
15	1986	4	57.0329
16	1986	5	55.8137
17	1986	6	59.9005
18	1986	7	65.7655
19	1986	8	64.4816
20	1986	9	61.0005
21	1986	10	57.5322
22	1986	11	59.3417
23	1986	12	68.1354
24	1987	1	73.8152
25	1987	2	70.0620
26	1987	3	65.6100
27	1987	4	60.1586
28	1987	5	58.8734
29	1987	6	63.8918
..	...	...	...
367	2015	8	110.5925
368	2015	9	101.9204
369	2015	10	91.5959
370	2015	11	93.0628
371	2015	12	103.2203
372	2016	1	117.0837
373	2016	2	106.6688
374	2016	3	95.3548
375	2016	4	89.3254
376	2016	5	90.7369
377	2016	6	104.0375
378	2016	7	114.5397
379	2016	8	115.5159
380	2016	9	102.7637
381	2016	10	91.4867
382	2016	11	92.8900
383	2016	12	112.7694
384	2017	1	114.8505
385	2017	2	99.4901
386	2017	3	101.0396

```
387  2017      4     88.3530
388  2017      5     92.0805
389  2017      6    102.1532
390  2017      7    112.1538
391  2017      8    108.9312
392  2017      9     98.6154
393  2017     10     93.6137
394  2017     11     97.3359
395  2017     12    114.7212
396  2018      1    129.4048

[397 rows x 3 columns]
```

This covers a brief outlook of the pandas features. There are so many other features that is difficult to cover them all, so we will look at some pratical examples