



# Python Programming for Beginners

## Pandas Tutorial

2023학년도 2학기

Suk-Hwan Lee

**Artificial Intelligence**

*Creating the Future*

Dong-A University

Division of Computer Engineering &  
Artificial Intelligence

## References

### Pandas Tutorials

- <https://pandas.pydata.org/pandas-docs/stable/index.html>
- <https://wikidocs.net/2873>
- [http://taewan.kim/post/numpy\\_cheat\\_sheet/](http://taewan.kim/post/numpy_cheat_sheet/)
- “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

### Pandas 24 useful exercises with solutions

- <https://www.kaggle.com/icarofreire/pandas-24-useful-exercises-with-solutions>
- [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/index.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html)

### ➤ Pandas

- Python 데이터분석 라이브러리
- 행과 열로 이루어진 데이터 객체를 만들어 다룰 수 있게 되며 보다 안정적으로 대용량의 데이터들을 처리하는데 매우 편리한 도구



## pandas 소개

➤ Wikipedia : [https://en.wikipedia.org/wiki/Pandas\\_\(software\)](https://en.wikipedia.org/wiki/Pandas_(software))

### pandas (software)

From Wikipedia, the free encyclopedia

*Not to be confused with PANDAS, the Australian archival management system used for the Pandora Archive.*

**pandas** is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.<sup>[2]</sup> The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.<sup>[3]</sup> Its name is a play on the phrase "Python data analysis" itself.<sup>[4]</sup> Wes McKinney started building what would become pandas at AQR Capital while he was a researcher there from 2007 to 2010.<sup>[5]</sup>

#### Contents [hide]

- 1 Library features
- 2 Dataframes
- 3 History
- 4 See also
- 5 References
- 6 Further reading
- 7 External links



[출처] <https://pandas.pydata.org/>

### Library Highlights

- A fast and efficient **DataFrame** object for data manipulation with integrated indexing;
- Tools for **reading and writing data** between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
- Intelligent **data alignment** and integrated handling of **missing data**: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form;
- Flexible **reshaping** and pivoting of data sets;
- Intelligent label-based **slicing, fancy indexing**, and **subsetting** of large data sets;
- Columns can be inserted and deleted from data structures for **size mutability**;
- Aggregating or transforming data with a powerful **group by** engine allowing split-apply-combine operations on data sets;
- High performance **merging and joining** of data sets;
- **Hierarchical axis indexing** provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure;
- **Time series**-functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
- Highly **optimized for performance**, with critical code paths written in Cython or C.
- Python with *pandas* is in use in a wide variety of **academic and commercial** domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

## pandas 소개

 pandas

About us ▾ Getting started Documentation Community ▾ Contribute

# pandas

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

[Install pandas now!](#)

**Getting started**

- Install pandas
- Getting started

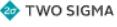
**Documentation**

- User guide
- API reference
- Contributing to pandas
- Release notes

**Community**

- About pandas
- Ask a question
- Ecosystem

With the support of:

The full list of companies supporting *pandas* is available in the sponsors page.

[출처] <https://pandas.pydata.org/>

## pandas documentation

Date: Aug 15, 2021 Version: 1.3.2

Download documentation: [PDF Version](#) | [Zipped HTML](#)

Useful links: [Binary Installers](#) | [Source Repository](#) | [Issues & Ideas](#) | [Q&A Support](#) | [Mailing List](#)

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.



### Getting started

New to pandas? Check out the getting started guides. They contain an introduction to pandas' main concepts and links to additional tutorials.

[To the getting started guides](#)



### User guide

The user guide provides in-depth information on the key concepts of pandas with useful background information and explanation.

[To the user guide](#)



### API reference

The reference guide contains a detailed description of the pandas API. The reference describes how the methods work and which parameters can be used. It assumes that you have an understanding of the key concepts.

[To the reference guide](#)



### Developer guide

Saw a typo in the documentation? Want to improve existing functionalities? The contributing guidelines will guide you through the process of improving pandas.

[To the development guide](#)

## pandas 소개

### ➤ Install

[https://pandas.pydata.org/docs/getting\\_started/install.html](https://pandas.pydata.org/docs/getting_started/install.html)

- Anaconda

The final step required is to install pandas. This can be done with the following command:

```
conda install pandas
```

To install a specific pandas version:

```
conda install pandas=0.20.3
```

- PyPI

pandas can be installed via pip from PyPI.

```
pip install pandas
```

- Installing using your Linux distribution's package manager

Ubuntu      stable      official Ubuntu repository      [sudo apt-get install python3-pandas](https://pandas.pydata.org/pandas-docs/stable/install/index.html#stable)

[출처] <https://pandas.pydata.org/>

- Start using pandas

```
In [1]: import pandas as pd
```

To load the pandas package and start working with it, import the package. The community agreed alias for pandas is `pd`, so loading pandas as `pd` is assumed standard practice for all of the pandas documentation.



### 잠깐 – 판다스의 특징

판다스는 다음과 같은 특징들을 갖는다.

1. 빠르고 효율적이며 다양한 표현력을 갖춘 자료구조.  
실세계 데이터 분석을 위해 만들어진 파이썬 패키지
2. 다양한 형태의 데이터에 적합  
**이종** 자료형의 열을 가진 테이블 데이터  
시계열 데이터  
레이블을 가진 다양한 행렬 데이터  
다양한 관측 통계 데이터
- 3 핵심 구조  
**시리즈Series** : 1차원 구조를 가진 하나의 열  
**데이터프레임DataFrame** : 복수의 열을 가진 2차원 데이터
4. 판다스가 잘 하는 일  
결측 데이터 처리  
데이터 추가 삭제 (새로운 열의 추가, 특정 열의 삭제 등)  
데이터 정렬과 다양한 데이터 조작



### 잠깐 – 판다스 or 판다

판다스라는 특이한 이름은 "panel data"라는 용어에서 유래되었다. 이 panel data라는 용어 역시 생소한 용어인데 이는 **계량경제학econometrics** 용어로 동일한 관찰자에 의하여 여러 회에 걸쳐 관측된 데이터 집합을 지칭하는 용어이다. 중국 쓰촨성 일대에 서식하는 동물인 판다와는 관계가 없으나 용어가 비슷하므로 많은 사람들이 판다스의 로고로 판다 그림을 사용하기도 한다.

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

### ➤ Pandas로 할 수 있는 일

#### 1) 데이터 불러오기 및 저장하기

- 파이썬 리스트, 딕셔너리, 넘파이 배열을 **DataFrame**으로 변환할 수 있다.
- Pandas로 CSV 파일이나, 엑셀 파일 등을 열 수 있다.
- URL을 통해 웹 사이트의 CSV 또는 JSON과 같은 원격 파일 또는 데이터베이스를 열 수 있다.

#### 2) 데이터 보기 및 검사

- `mean()`로 모든 열의 평균을 계산할 수 있다.
- `corr()`로 데이터프레임의 열 사이의 상관 관계를 계산할 수 있다.
- `count()`로 각 데이터프레임 열에서 null이 아닌 값의 개수를 계산할 수 있다.

#### 3) 필터, 정렬 및 그룹화

- `sort_values()`로 데이터를 정렬할 수 있다.
- 조건을 사용하여 열을 필터링할 수 있다.
- `groupby()`를 이용하여 기준에 따라 몇 개의 그룹으로 데이터를 분할할 수 있다.

#### 4) 데이터 정제

- 데이터의 누락 값을 확인할 수 있다.
- 특정한 값을 다른 값으로 대체할 수 있다.

## CSV (comma-separated values)

- CSV는 테이블 형식의 데이터를 저장하고 이동하는 데 사용되는 구조화된 텍스트 파일 형식이다. CSV는 쉼표로 구분한 변수 **comma separated variables**의 약자이다.
- CSV의 역사는 1972년으로 거슬러 올라가며 Microsoft Excel와 같은 **스프레드 시트spread sheet** 소프트웨어에 적합한 형식이다. 데이터 과학에서 사용되는 데이터 가운데 상당한 비율의 데이터들이 CSV 형식으로 공유되는 경우가 많다.



- Pandas는 데이터를 처리하고 분석하기 위한 모듈이므로 다양한 종류의 데이터 파일 형식을 지원한다
- 교재에서는 CSV로 저장된 데이터를 사용하는 것을 기본으로 삼을 것이다. 본격적으로 Pandas를 살펴보기 전에 CSV 데이터를 처리하는 것에 대해 먼저 살펴 보자.



[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

- CSV 파일은 필드를 나타내는 열과 레코드를 나타내는 행으로 구성
- 만약 데이터 중간에 구분자가 포함되어야 한다면 따옴표를 사용하여 필드를 묶어야 함
  - 예를 들어서 'Gildong, Hong'이라는 데이터가 있다고 하자. 데이터의 중간에 쉼표(,)가 포함되어 있다. 이러한 경우에는 구분자로 사용되는 쉼표와 구분하기 위하여 반드시 데이터를 따옴표로 감싸야 한다.
- CSV 파일의 첫 번째 레코드에는 열 제목이 포함되어 있을 수 있다.
- CSV 형식 자체의 요구사항이 아니라 단순히 일반적인 관행
- CSV 파일의 크기를 알 수 없고 잠재적으로 크기가 큰 경우 한 번에 모든 레코드를 읽지 않는 것이 좋다.**
- 이때는 현재 행을 읽고, 현재 행을 처리한 후에 삭제하고 다음 행을 가져오는 방식이 필요할 수도 있다. 아니면 특정한 크기만큼의 데이터를 읽어서 처리한 뒤에, 다음으로 또 그만큼의 크기를 가져오는 방식을 사용할 수도 있을 것이다.

### JavaScript Object Notation(JSON) 예시

다음은 한 사람에 관한 정보를 갖는 JSON 객체이다.

키-값 쌍(이름:값)의 패턴으로 표현된다.

```
1  {
2      "이름": "홍길동",
3      "나이": 25,
4      "성별": "여",
5      "주소": "서울특별시 양천구 목동",
6      "특기": ["농구", "도술"],
7      "가족관계": {"#": 2, "아버지": "홍판서", "어머니": "춘섬"},
8      "회사": "경기 수원시 팔달구 우만동"
9 }
```

## CSV (comma-separated values)

### ➤ CSV 데이터의 내용을 읽어 보자

- 파이썬 모듈 csv는 CSV reader와 CSV writer를 제공한다. 두 객체 모두 파일 핸들을 첫 번째 매개 변수로 사용한다. 필요한 경우 delimiter 매개 변수를 사용하여 구분자를 제공할 수 있다.
- 이 파일을 d: 드라이브의 data 폴더에 'weather.csv'로 저장했다고 가정하고, 그러면 이 파일의 경로 path는 'd:/data/weather.csv'가 된다.
- <https://github.com/dongupak/DataSciPy/tree/master/data/csv>

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

```
import csv
```

```
f = open('d:/data/weather.csv')  
data = csv.reader(f)  
for row in data:  
    print(row)  
f.close()
```

# CSV 파일을 열어서 f에 저장한다.  
# reader() 함수를 이용하여 읽는다.

```
['일시', '평균기온', '최대풍속', '평균풍속'] ←  
['2010-08-01', '28.7', '8.3', '3.4']  
['2010-08-02', '25.2', '8.7', '3.8']  
['2010-08-03', '22.1', '6.3', '2.9']  
...  
...
```

master DataSciPy / data / csv /

dknife Alternative filename

..

Life\_expectancy.csv Add files via upload

countries.csv

life\_expectancy.csv

weather.csv

2010년 8월~2020년 8월까지 울릉  
도의 기온과 풍속 데이터가 저장된 기  
상 데이터  
출처 : 기상자료개발포털사이트

weather.csv - Windows 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)  
일시,평균기온,최대풍속,평균풍속  
2010-08-01,28.7,8.3,3.4  
2010-08-02,25.2,8.7,3.8  
2010-08-03,22.1,6.3,2.9  
2010-08-04,25.3,6.6,4.2  
2010-08-05,27.2,9.1,5.6  
2010-08-06,26.8,9.8,8  
2010-08-07,27.5,9.1,5  
2010-08-08,26.6,5.9,4  
2010-08-09,26.9,5.1,3.1  
2010-08-10,25.6,10.2,5.5  
2010-08-11,24.6,9.4,4.8  
2010-08-12,23.7,8.7,2.6

## CSV (comma-separated values)

### ❖ CSV 헤더를 제거하는 방법

- `next()` 함수를 사용함

```
import csv

f = open('d:/data/weather.csv')
data = csv.reader(f)
header = next(data) ←
for row in data:
    print(row)
f.close()

['2010-08-01', '28.7', '8.3', '3.4']
['2010-08-02', '25.2', '8.7', '3.8']
['2010-08-03', '22.1', '6.3', '2.9']
['2010-08-04', '25.3', '6.6', '4.2']
...

```

# CSV 파일을 열어서 f에 저장한다.  
# csv의 `reader()` 함수를 이용하여 읽는다.  
# 헤더를 제거한다.  
# 반복 루프를 사용하여 데이터를 읽는다.  
# 파일을 닫는다.

### [Note] Iterator

- 반복 가능한 데이터: `iter()` 함수로 반복자를 구할 수 있는 데이터
- 반복자: `next()` 함수로 값을 하나씩 꺼낼 수 있는 데이터
- `iter()` 함수: 반복 가능한 데이터를 입력받아 반복자를 반환하는 함수
- `next()` 함수: 반복자를 입력받아 다음 출력값을 반환하는 함수

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

### ➤ CSV에서 원하는 데이터를 뽑아 보자

- 기상자료개방 포털 사이트에서 다운 받은 데이터 `weather.csv`를 사용
- 이제 이 데이터에서 평균 풍속 데이터만 추출하여 사용하고 싶다. CSV 파일에서 평균 풍속 데이터는 4번째 열에 저장되어 있다. 인덱스로는 3이 된다. 따라서 리스트에서 `row[3]`을 찾으면 된다

	col[0]	col[1]	col[2]	col[3]
row	A	B	C	D
1	일시	평균기온	최대풍속	평균풍속
2	2010-08-01	28.7	8.3	3.4
3	2010-08-02	25.2	8.7	3.8
4	2010-08-03	22.1	6.3	2.9

import csv

```
f = open('d:/data/weather.csv')
data = csv.reader(f)
header = next(data)
for row in data:
    print(row[3], end=',')
f.close()
```

# CSV 파일을 열어서 f에 저장한다.  
# `reader()` 함수를 이용하여 읽는다.  
# 헤더를 제거한다.  
# 반복 루프를 사용하여 데이터를 읽는다.  
# 평균풍속만 출력하고, 쉼표로 연결한다.  
# 파일을 닫는다.

3.4,3.8,2.9,4.2,5.6,8,5,4,3.1,5.5,4.8,2.6,4.6,4.4,10.3,3.2,1.6,2.1,1.9,3.2,4.2,2.5,6.6,2.3,4.9,6.2,4.2,2.6,5.3,1.7,3.2,3.3,4.3,7.6,6.6,2.5,7.2,3.8,1.8,3.9,1.6,2.2,1.2,2.2,7.9,5.8,4.1,6.1,1.8,2.8,5.6,2.1,2.2,3.3,3.2,5.9,5.5,1.3.1,3.4,3.7,2.7,2.6,3.1,2.5,5,3,2.9,2.1,3.9,6.3,3.9,2,3,6.1,7.1,4,3.5,5.8,6.6,7.2,5.6,3.5,3.2,2.9,3.2,3.3,2.5,7.5 ...

## CSV (comma-separated values)

### ➤ CSV에서 원하는 데이터를 뽑아 보자

- 반복문을 사용하여 import한 데이터의 네번째 열의 원소값 중에서 최대값을 구하자

```
# 위의 코드 import .. 부터 header =.. 까지가 생략되었음
max_wind = 0.0

for row in data:                                # 반복 루프를 사용하여 데이터를 읽는다.
    if row[3] == '' :                            # 평균 풍속 데이터가 없는 경우 0을 처리
        wind = 0
    else :
        wind = float(row[3])                     # 평균 풍속 데이터를 실수로 변환해 저장
    if max_wind < wind :                         # 최대 풍속을 갱신하는지 검사
        max_wind = wind                           # 현재까지의 최대 풍속보다 크면 새로 기록

print('지난 10년간 울릉도의 최대 풍속은 ', max_wind, 'm/s')
```

지난 10년간 울릉도의 최대 풍속은 14.9 m/s

예보-용어	바람강도(m/s)	비고
강한바람	9~14미만	<ul style="list-style-type: none"><li>초속 9~11미터 정도의 바람은 작은 나무 전체가 흔들리고 공원의 파라솔이 뒤집힐 정도의 바람이 불게 됨</li><li>한편 초속 11~14미터 정도면, 큰 나무가 흔들리고 우산을 들고 있기가 서 있기가 힘들게 됨</li></ul>
매우강한바람	14이상	<ul style="list-style-type: none"><li>육상에서는 강풍주의보의 수준이 됨</li></ul>

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

### ➤ LAB12-1 울릉도는 몇 월에 바람이 가장 강할까?

```
import csv
import matplotlib.pyplot as plt

f = open('d:/data/weather.csv')
data = csv.reader(f)
header = next(data)

monthly_wind = [ 0 for x in range(12) ]      # CSV 파일 열어 f에 저장
days_counted = [ 0 for x in range(12) ]        # reader() 함수로 읽기
                                                # 헤더를 제거

monthly_wind[0] = float(header[3])             # 매달 풍속을 담을 리스트, 초기화 0
days_counted[0] = 1                            # 각 달마다 측정된 일수, 초기화 0

for row in data:
    month = int(row[0][5:7])
    if row[3] != '' :
        wind = float(row[3])
        monthly_wind[month-1] += wind           # 0번 열에서 달 정보 추출
        days_counted[month-1] += 1              # 풍속 데이터 존재하는지 확인
                                                # 풍속을 얹어 온다.
                                                # 해당 달에 풍속 데이터 추가
                                                # 해당 달의 일수를 증가

for i in range(12) :
    monthly_wind[i] /= days_counted[i]          # 일수로 나누어 월평균 구하기

plt.plot(monthly_wind, 'blue')
plt.show()

f.close()                                       # 파일을 닫는다.
```

## Series & Dataframe

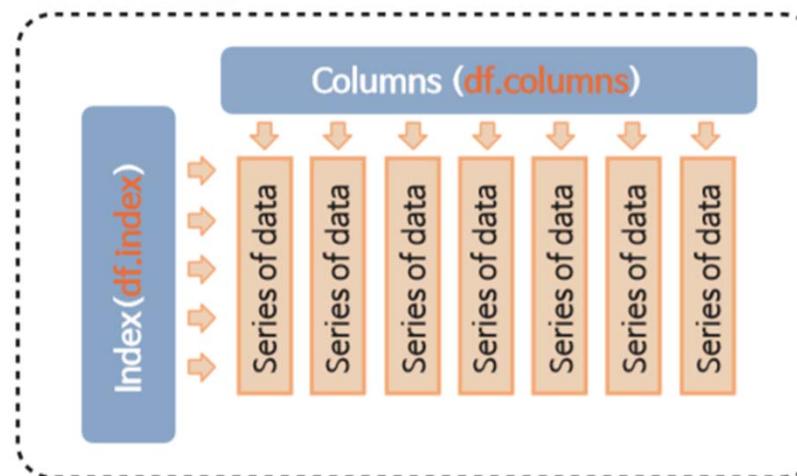
### ➤ Pandas의 데이터 구조 : Series와 DataFrame

- 앞서 다루어본 csv 모듈 이외에도 CSV 데이터를 처리할 수 있는 모듈이 있다. 이들 중 가장 강력한 외부 라이브러리인 Pandas를 알아보자.
- Pandas는 데이터 저장을 위하여 다음과 같은 2가지의 기본 데이터 구조를 제공하고 있다.
- 이들 데이터 구조는 모두 넘파이 배열을 이용하여 구현된다. 따라서 속도가 빠르다. 모든 데이터 구조는 값을 변경할 수 있으며, 시리즈를 제외하고는 크기도 변경할 수 있다. 각 행과 열은 이름이 부여되며, 행의 이름을 인덱스 index, 열의 이름을 컬럼스columns라 부른다.

데이터 구조	차원	설명
시리즈	1	레이블이 붙어있는 1차원 벡터
데이터프레임	2	행과 열로 되어있는 2차원 테이블. 각 열은 시리즈로 되어 있다.

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

### DataFrame df



### Series

1      3      4      NaN      6      8

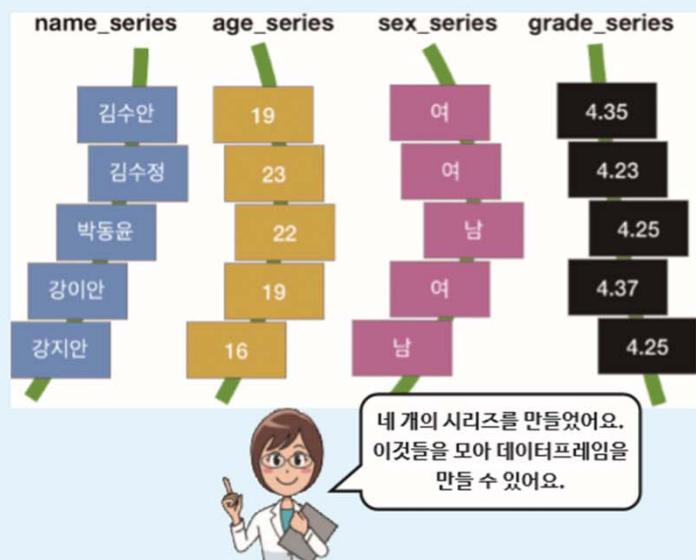
```
>>> import numpy as np
>>> import pandas as pd
>>> series = pd.Series([1, 3, 4, np.nan, 6, 8])
>>> series
0    1.0
1    3.0
2    4.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

np.nan은 값 없음(Not a Number)을 의미하는 것으로 수치 데이터가 없을 경우 이를 표기하는 방법이다. NaN과 동일한 표기임. nan이 있을 경우 실수형

➤ Pandas의 데이터 구조 : Series와 DataFrame

```
>>> name_series = pd.Series(['김수안', '김수정', '박동윤', '강이안', '강지안'])
>>> age_series = pd.Series([19, 23, 22, 19, 16])
>>> sex_series = pd.Series(['여', '여', '남', '여', '남'])
>>> grade_series = pd.Series([4.35, 4.23, 4.25, 4.37, 4.25])
>>> print(name_series, age_series, sex_series, grade_series)
0    김수안
1    김수정
2    박동윤
3    강이안
4    강지안
dtype: object
0    19
1    23
2    22
3    19
4    16
dtype: int64
0    여
1    여
2    남
3    여
4    남
dtype: object
0    4.35
1    4.23
2    4.25
3    4.37
4    4.25
dtype: float64
```

이름	나이	성별	평점
김수안	19	여	4.35
김수정	23	여	4.23
박동윤	22	남	4.45
강이안	19	여	4.37
강지안	16	남	4.25



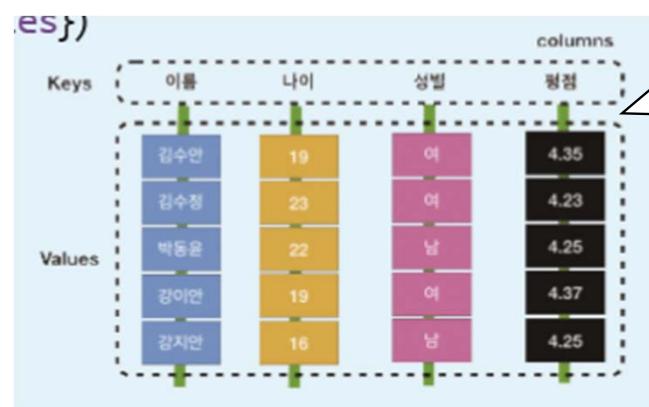
## Series & Dataframe

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

### ➤ Pandas의 데이터 구조 : Series와 DataFrame

```
>>> df = pd.DataFrame({'이름': name_series, '나이': age_series,
                      '성별': sex_series, '평점': grade_series})
>>> print(df)
   이름  나이 성별  평점
0  김수안    19   여   4.35
1  김수정    23   여   4.23
2  박동윤    22   남   4.25
3  강이안    19   여   4.37
4  강지안    16   남   4.25
```

Pandas의 DataFrame 클래스를  
사용해서 하나의 데이터프레임을 만  
들 수 있다



딕셔너리 형식의 데이터로 데이터프레임을 생성함

## Series & Dataframe

### ➤ Pandas로 데이터 읽기

- Pandas 모듈을 이용한 csv 파일을 읽어들여서 데이터프레임으로 바꾸는 작업을 간단히 할 수 있게 한다. 다음과 같이 `read_csv` 함수를 이용하면 된다. `countries.csv` 파일의 제 1행 제 1열은 비어 있음을 확인할 수 있다. 이것은 첫열은 데이터가 아니라 각 행의 인덱스로 사용되도록 하기 위해서이다.
- 이때 CSV 파일이 데이터프레임이 될 수 있도록 각 행이 같은 구조로 되어 있고, 각 열은 동일한 자료형을 가진 시리즈로 되어 있어야 한다. 예러가 없이 csv 파일을 읽어왔다면 df를 출력해보자.

countries.csv

		country	area	capital	population
0	KR	Korea	98480	Seoul	51780579
1	US	USA	9629091	Washington	331002825
2	JP	Japan	377835	Tokyo	125960000
3	CN	China	9596960	Beijing	1439323688
4	RU	Russia	17100000	Moscow	146748600

```
>>> import pandas as pd  
>>> df = pd.read_csv('d:/data/countries.csv')
```

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

각 열은 서로 다른 속성 레이  
블을 나타낸다.

```
>>> df  
      Unnamed: 0  country        area    capital  population  
0          KR   Korea     98480    Seoul    51780579  
1          US    USA    9629091  Washington  331002825  
2          JP   Japan    377835    Tokyo   125960000  
3          CN   China   9596960   Beijing  1439323688  
4          RU  Russia  17100000   Moscow  146748600
```

인덱스 번호는 Pandas가 추가  
한 열이다

countries.csv

	country	area	capital	population
KR	Korea	98480	Seoul	51780579
US	USA	9629091	Washington	331002825
JP	Japan	377835	Tokyo	125960000
CN	China	9596960	Beijing	1439323688
RU	Russia	17100000	Moscow	146748600

## Series & Dataframe

### ➤ 데이터를 설명하는 인덱스와 컬럼스 객체

- 데이터프레임에서는 다음과 같이 **인덱스index**와 **컬럼스columns** 객체를 정의하여 사용한다.
- 인덱스는 행들의 레이블이고 columns는 열들의 레이블이 저장된 객체이다.

비워 두었던 열 이름

CSV 파일의 첫 행으로 만들어진 **columns**

	Unnamed: 0	country	area	capital	population
0	KR	Korea	98480	Seoul	51780579
1	US	USA	9629091	Washington	331002825
2	JP	Japan	377835	Tokyo	125960000
3	CN	China	9596960	Beijing	1439323688
4	RU	Russia	17100000	Moscow	146748600

자동으로 생성된 **index**

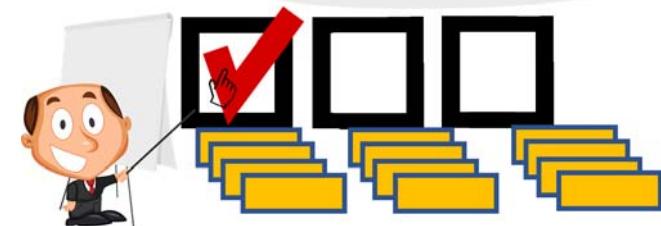
[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

첫 번째 열을 인덱스로 사용하겠다는 뜻.

```
import pandas as pd  
  
df = pd.read_csv('d:/data/countries.csv', index_col = 0)  
print(df)
```

	country	area	capital	population
KR	Korea	98480	Seoul	51780579
US	USA	9629091	Washington	331002825
JP	Japan	377835	Tokyo	125960000
CN	China	9596960	Beijing	1439323688
RU	Russia	17100000	Moscow	146748600

여러 열을 만드는 시리즈 가운데  
내가 원하는 시리즈를 선택해  
각 행의 인덱스로 사용할 수 있어요



### ➤ 열을 기준으로 데이터 선택하기

- 특정한 열만 선택하려면 아래와 같이 **대괄호 안에 열의 이름을 넣으면** 된다.
- 다음 코드는 countries.csv를 다시 읽고 있다. 그리고 처음에는 인덱스를 첫 열로 지정해서 df\_my\_index로 할당했고, 인덱스 지정 없이 만든 데이터프레임은 df\_no\_index로 할당했다. 두 데이터프레임에서 population 레이블을 가진 열을 추출하기 위해서는 **df['population']**이라고 하면 된다.

```
import pandas as pd

df_my_index = pd.read_csv('d:/data/countries.csv', index_col = 0)
df_no_index = pd.read_csv('d:/data/countries.csv')
print(df_my_index['population'])
print(df_no_index['population'])
```

```
KR      51780579
US      331002825
JP      125960000
CN      1439323688
RU      146748600
```

```
Name: population, dtype: int64
0      51780579
1      331002825
2      125960000
3      1439323688
4      146748600
Name: population, dtype: int64
```

인덱스 컬럼이 0이므로 KR, US, JP,..가 인덱스가 된다

인덱스 컬럼이 없을 경우 0, 1, 2, ..가 인덱스가 됨

```
import pandas as pd
```

```
df_my_index = pd.read_csv('d:/data/countries.csv', index_col = 0)
print(df_my_index[ ['area', 'population' ] ])
```

	area	population
KR	98480	51780579
US	9629091	331002825
JP	377835	125960000
CN	9596960	1439323688
RU	17100000	146748600

전체 데이터 중에서 두 개의 열을 선택하는 경우 : 선택을 원하는 열의 레이블을 리스트에 넣어서 전달

## Series & Dataframe

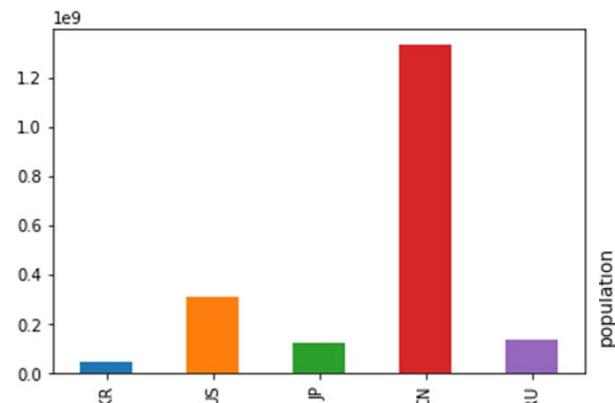
### ➤ 데이터 가시화하기

- 우리는 선택된 열을 그래프로 그릴 수 있다. 이를 위하여 [데이터프레임의 이름 다음에 plot\(\) 메소드만 추가하면 된다.](#) 각 국가의 인구만을 추출하여서 막대 그래프로 그려보면 다음과 같다.

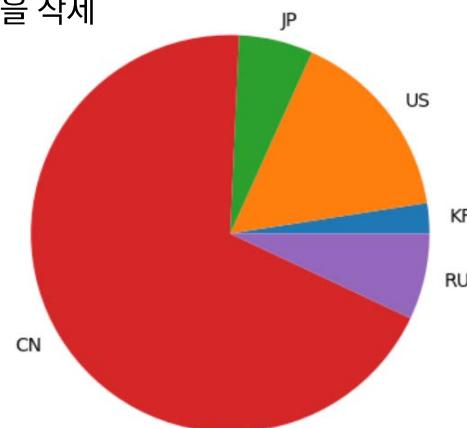
```
import pandas as pd
import matplotlib.pyplot as plt

countries_df = pd.read_csv('d:/data/countries.csv', index_col = 0)

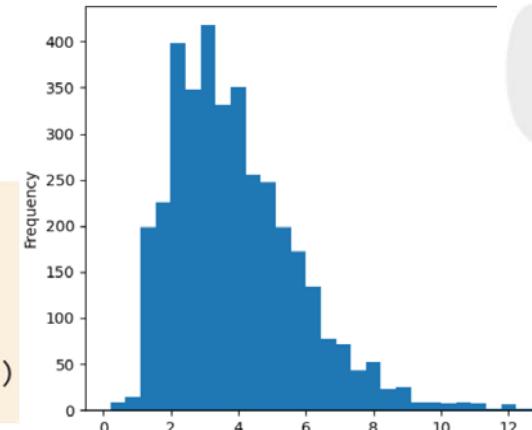
countries_df['population'].plot(kind='bar', color=('b', 'darkorange', 'g', 'r', 'm'))
plt.show()
```



kind='pie'로 지정하고, color 부분  
을 삭제



[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료



데이터를 차트로 표시하는 일을  
아주 간편하게 할 수 있도록  
지원하고 있어요.



```
import pandas as pd
import matplotlib.pyplot as plt
```

```
weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')
weather['평균풍속'].plot(kind='hist', bins=33)
plt.show()
```

한글 인코딩 문자를 읽어오기  
위해서 사용함

잠깐 – 판다스를 이용하여 csv를 여는 코드에서 눈여겨 볼 것

마이크로소프트의 윈도 시스템에서 생성되고 한글을 포함하고 있는 weather.csv는 읽을 때 한글을 어떤 [인코딩](#) 방식으로 처리할지 지정해야 한다. 위의 코드에 encoding='CP949'라고 표시된 것을 확인할 수 있을 것이다. 또 주의해서 볼 것은 csv 모듈로 읽을 때와 달리 판다스로 읽으면 풍속이 문자열이 아니라 실수 데이터로 바로 읽힌다는 것을 알 수 있다. float()를 이용하여 값을 실수로 바꿀 필요가 없는 것이다.

## Series & Dataframe

### ➤ Pandas에서도 슬라이싱으로 행 선택이 된다.

- 데이터프레임 중에서 몇 개의 행만을 가져오고자 할 때는 몇 가지의 방법이 있다. 우선 처음 5행만 얻으려면 `head()`를 사용할 수 있다. 마지막 5행만을 얻으려면 `tail()`을 사용한다.

```
>>> countries_df.head()  
country area capital population  
KR Korea 98480 Seoul 48422644  
US USA 9629091 Washington 310232863  
JP Japan 377835 Tokyo 127288000  
CN China 9596960 Beijing 1330044000  
RU Russia 17100000 Moscow 140702000
```

```
>>> countries_df[:3]  
country area capital population  
KR Korea 98480 Seoul 48422644  
US USA 9629091 Washington 310232863  
JP Japan 377835 Tokyo 127288000
```

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

```
>>> countries_df.loc['KR']  
country Korea  
area 98480  
capital Seoul  
population 48422644
```

행의 레이블이 'KR'인 행만을 선택하기

```
>>> countries_df['population'][3:]  
KR 48422644  
US 310232863  
JP 127288000
```

데이터프레임에서 특정한 요소 하나만을 선택하려면 loc함수에 행과 열의 레이블을 써주면 된다

```
>>> countries_df.loc['US', 'capital']  
'Washington'
```

```
>>> countries_df['capital'].loc['US']  
'Washington'
```

- ❖ DataFrame 인덱싱
- `df.loc[ "row" , "column" ]` : 행과 열의 이름
- `df.iloc[i, j]` : row, column의 인덱스값

## ➤ 새로운 열 생성

- Pandas를 이용하면 다른 열의 정보를 토대로 새로운 열을 생성할 수도 있다.
- 우리의 데이터프레임에 인구 밀도를 나타내는 열을 생성해보자. 앞 장에서 넘파이 배열에는 어떤 수를 곱하고 더하는 것이 가능하다고 하였다. Pandas는 넘파이를 기반으로 하기 때문에 Pandas 데이터프레임에도 동일하게 적용할 수 있다. 인구를 면적으로 나눠주면 된다.

```
import pandas as pd
import matplotlib.pyplot as plt

countries_df = pd.read_csv('d:/data/countries.csv', index_col = 0)
countries_df['density'] = countries_df['population'] / countries_df['area']
print(countries_df)
```

	country	area	capital	population	density
KR	Korea	98480	Seoul	51780579	525.797918
US	USA	9629091	Washington	331002825	34.375293
JP	Japan	377835	Tokyo	125960000	333.373033
CN	China	9596960	Beijing	1439323688	149.977044
RU	Russia	17100000	Moscow	146748600	8.581789

**countries\_df['density'] 열이 새롭게 추가되었음**



### 잠깐 – 데이터프레임의 열을 이용한 연산

인구밀도를 구하기 위해 새로운 열을 'density'라는 레이블로 생성해 보았다. 그리고 이 열의 데이터는 기존에 존재하던 데이터 중에서 인구수를 면적으로 나누어 얻을 수 있다. 그런데, 이를 위해서 각 행을 차례로 접근하여 해당 데이터 항목마다 연산을 수행하지 않는다. 데이터프레임의 어떤 열이 다른 열들의 값에 의해 결정될 때는 이 계산을 행별로 반복하여 일을 하는 것이 아니라 필요한 열을 통째로 접근하여 한번에 계산이 이루어지게 한다. 이것은 코드가 간결할 뿐만 아니라, 계산도 훨씬 빠르다. 데이터프레임이나 행렬 데이터를 다루면서 **for** 문을 사용한다면 '내가 잘못하고 있지 않는가? 이 **for** 문을 꼭 써야 하는가?'라는 생각을 항상 해야 한다.



## ➤ 데이터를 간편하게 분석할 수 있는 기능

- 이제 우리는 외부 파일을 읽어서 데이터프레임을 생성해서 필요한 행과 열을 선택할 수 있다.  
데이터프레임이 저장한 데이터를 간단히 분석하려면 **describe()** 함수를 호출해주면 된다.

```
import pandas as pd
weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')
print(weather.describe())
```

	평균기온	최대풍속	평균풍속
count	3653.000000	3649.000000	3647.000000
mean	12.942102	7.911099	3.936441
std	8.538507	3.029862	1.888473
min	-9.000000	2.000000	0.200000
25%	5.400000	5.700000	2.500000
50%	13.800000	7.600000	3.600000
75%	20.100000	9.700000	5.000000
max	31.300000	26.000000	14.900000

### ☞ 잠깐 – 판다스의 표준편차와 넘파이의 표준편차의 차이

평균기온 표준편차를 넘파이로 계산하면 다른 값이 나온다. 판다스 표준편차와 넘파이 표준편차를 각각 구하는 방법은 다음과 같다.

```
pandas_std = weather['평균기온'].std()
numpy_std = np.std( weather['평균기온'] )
print(pandas_std, numpy_std)
```

8.538507014753446 8.537338236838895



```
...
print('평균 분석 -----')
print(weather.mean())
print('표준편차 분석 -----')
print(weather.std())
```

평균 분석 -----
평균기온 12.942102
최대풍속 7.911099
평균풍속 3.936441
dtype: float64
표준편차 분석 -----
평균기온 8.538507
최대풍속 3.029862
평균풍속 1.888473
dtype: float64

판다스의 표준편차는 디폴트로 **베셀 보정Bessel's correction**을 적용하는데, 이것은 표준편차를 구할 때, 표본 크기 n 대신에 n-1을 적용하는 것으로 모분산 추정에서 편향을 보정하는 역할을 한다.

## ➤ 데이터 집계 분석도 손쉽게

- 데이터의 전체적인 특징이 어떠한지를 분석하는 것은 매우 중요한 일이다. 앞서 살펴본 `describe()` 함수는 데이터프레임의 데이터 특성을 전체적으로 요약해 주는데, 이 분석 내용 각각은 하나씩 떼어서 적용할 수도 있다. 예를 들어 다음 코드를 보자.

```
>>> weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')
>>> weather.count()
평균기온      3653
최대풍속      3649
평균풍속      3647
dtype: int64

>>> weather['최대풍속'].count()
3649
```

weather.csv 파일이 담고 있는 데이터가  
3 개의 열을 가지고 있고, 각각의 열에 담긴  
데이터가 3653, 3649, 3647개라는 것을 알 수 있다

```
>>> weather[['최대풍속','평균풍속']].count()
최대풍속      3649
평균풍속      3647
dtype: int64
```

여러 개의 열을 분석하고 싶을 때는 원하는 열의 레이블  
들을 리스트로 제공

```
>>> weather[['최대풍속','평균풍속']].mean()
최대풍속      7.911099
평균풎속      3.936441
dtype: float64
```

min(), max(), mean(), sum()  
등도 적용 가능

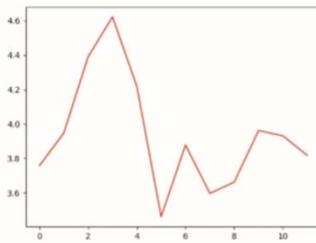
```
>>> weather.mean()[['최대풍속', '평균풎속']]
최대풍속      7.911099
평균풎속      3.936441
dtype: float64
```

## 예제)

### ▶ LAB12-2 판다스로 울릉도의 바람 세기 분석하기

앞서 사용했던 울릉도의 기상 데이터에 기록된 매일의 평균 풍속 데이터를 바탕으로 몇 월의 바람이 가장 강한지 분석해 보았다. 이번에는 이 작업을 판다스를 이용하여 해 보려고 한다. 결과는 앞의 것도 동일하게 나올 것이다.

#### 원하는 결과



```
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt

weather = pd.read_csv('d:/data/weather.csv', encoding='CP949')
monthly = [ None for x in range(12) ] # 달별로 구분된 12개의 데이터프레임
monthly_wind = [ 0 for x in range(12) ] # 각 달의 평균 풍속을 담을 리스트
# 마지막에 해당 행의 데이터가 측정된 달을 기록한 열을 추가
weather['month'] = pd.DatetimeIndex(weather['일시']).month

for i in range(12) :
    monthly[i] = weather[ weather['month'] == i + 1 ] # 달별로 분리
    monthly_wind[i] = monthly[i].mean()['평균풍속'] # 개별 데이터 분석

plt.plot(monthly_wind, 'red')
plt.show()
```

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

일시	평균기온	최대풍속	평균풍속
2010-08-01	28.7	8.3	3.4
2010-08-02	25.2	8.7	3.8
2010-08-03	22.1	6.3	2.9
2010-08-04	25.3	6.6	4.2
2010-08-05	27.2	9.1	5.6
2010-08-06	26.8	9.8	8
2010-08-07	27.5	9.1	5
2010-08-08	26.6	5.9	4

```
pd.DatetimeIndex(weather['일시'])
```

```
DatetimeIndex(['2010-08-01', '2010-08-02', '2010-08-03', '2010-08-04',
                 '2010-08-05', '2010-08-06', '2010-08-07', '2010-08-08',
                 '2010-08-09', '2010-08-10',
                 ...
                 '2020-07-22', '2020-07-23', '2020-07-24', '2020-07-25',
                 '2020-07-26', '2020-07-27', '2020-07-28', '2020-07-29',
                 '2020-07-30', '2020-07-31'],
                dtype='datetime64[ns]', name='일시', length=3653, freq=None)
```

```
pd.DatetimeIndex(weather['일시']).month
```

```
Int64Index([8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
             ...
             7, 7, 7, 7, 7, 7, 7, 7, 7, 7],
            dtype='int64', name='일시', length=3653)
```

```
weather[ weather['month'] == 8 ]
```

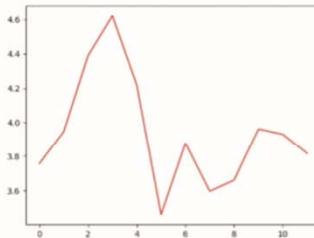
	일시	평균기온	최대풍속	평균풍속	month
0	2010-08-01	28.7	8.3	3.4	8
1	2010-08-02	25.2	8.7	3.8	8
2	2010-08-03	22.1	6.3	2.9	8
3	2010-08-04	25.3	6.6	4.2	8
4	2010-08-05	27.2	9.1	5.6	8

## 예제)

### ▶ LAB12-2 판다스로 울릉도의 바람 세기 분석하기

앞서 사용했던 울릉도의 기상 데이터에 기록된 매일의 평균 풍속 데이터를 바탕으로 몇 월의 바람이 가장 강한지 분석해 보았다. 이번에는 이 작업을 판다스를 이용하여 해 보려고 한다. 결과는 앞의 것도 동일하게 나올 것이다.

#### 원하는 결과



```
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt

weather = pd.read_csv('d:/data/weather.csv', encoding='CP949')
monthly = [ None for x in range(12) ] # 달별로 구분된 12개의 데이터프레임
monthly_wind = [ 0 for x in range(12) ] # 각 달의 평균 풍속을 담을 리스트
# 마지막에 해당 행의 데이터가 측정된 달을 기록한 열을 추가
weather['month'] = pd.DatetimeIndex(weather['일시']).month

for i in range(12) :
    monthly[i] = weather[ weather['month'] == i + 1 ] # 달별로 분리
    monthly_wind[i] = monthly[i].mean()['평균풍속'] # 개별 데이터 분석

plt.plot(monthly_wind, 'red')
plt.show()
```

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

### ▶ 비교 : csv 모듈을 사용하는 것 보다 간결하고 강력함

```
import csv
import matplotlib.pyplot as plt

f = open('d:/data/weather.csv')
data = csv.reader(f)
header = next(data)

monthly_wind = [ 0 for x in range(12) ]
days_counted = [ 0 for x in range(12) ]

for row in data:
    month = int(row[0][5:7])
    if row[3] != '' :
        wind = float(row[3])
        monthly_wind[month-1] += wind
        days_counted[month-1] += 1

for i in range(12) :
    monthly_wind[i] /= days_counted[i]

plt.plot(monthly_wind, 'blue')
plt.show()

f.close()
```

# CSV 파일 열어 f에 저장  
# reader() 함수로 읽기  
# 헤더를 제거

# 매달 풍속을 담을 리스트, 초기화 0  
# 각 달마다 측정된 일수, 초기화 0

# 0번 열에서 달 정보 추출  
# 풍속 데이터 존재하는지 확인  
# 풍속을 얻어 온다.  
# 해당 달에 풍속 데이터 추가  
# 해당 달의 일수를 증가

# 일수로 나누어 월평균 구하기

# 파일을 닫는다.

## ➤ 데이터를 특정한 값에 기반하여 묶는 기능 : 그룹핑

- 조금 더 효율적인 방법이 있는데 그것은 **groupby()**라는 함수이다. **groupby()** 함수에 넘길 인자로는 우리가 그룹을 묶을 때에 사용될 열의 레이블이다. 해당 열에 있는 데이터가 동일하면 하나의 그룹으로 묶이는 것이다. 그리고 여기에 **mean()**을 적용하면 해당 그룹의 데이터들이 가진 값의 평균을 구할 수 있다.

해당 데이터를 그룹별로 모두  
더하여 값을 확인

```
...
weather = pd.read_csv('d:/data/weather.csv', encoding='CP949')
weather['month'] = pd.DatetimeIndex(weather['일시']).month
means = weather.groupby('month').mean()

print(means)
```

month	평균기온	최대풍속	평균풍속
1	1.598387	8.158065	3.757419
2	2.136396	8.225357	3.946786
3	6.250323	8.871935	4.390291
4	11.064667	9.305017	4.622483
5	16.564194	8.548710	4.219355
6	19.616667	6.945667	3.461000
7	23.328387	7.322581	3.877419
8	24.748710	6.853226	3.596129
9	20.323667	6.896333	3.661667
10	15.383871	7.766774	3.961613
11	9.889667	8.013333	3.930667
12	3.753548	8.045484	3.817097

```
>>> weather.tail()
      일시  평균기온  최대풍속  평균풍속  month
3648  2020-07-27    22.1     4.2     1.7    7
3649  2020-07-28    21.9     4.5     1.6    7
3650  2020-07-29    21.6     3.2     1.0    7
3651  2020-07-30    22.9     9.7     2.4    7
3652  2020-07-31    25.7     4.8     2.5    7
```

DatetimeIndex() 함수를 통해서  
weather['month'] 열이 새롭게 추가  
되었음..

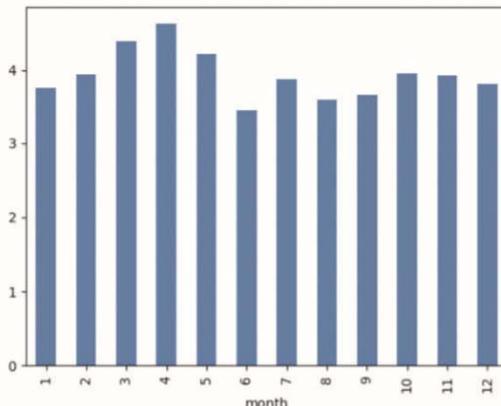
```
sum_data = weather.groupby('month').sum()
print(sum_data)
```

month	평균기온	최대풍속	평균풍속
1	495.5	2529.0	1164.8
2	604.6	2303.1	1105.1
3	1937.6	2750.3	1356.6
4	3319.4	2782.2	1377.5
5	5134.9	2650.1	1308.0
6	5885.0	2083.7	1038.3
7	7231.8	2270.0	1202.0
8	7672.1	2124.5	1114.8
9	6097.1	2068.9	1098.5
10	4769.0	2407.7	1228.1
11	2966.9	2404.0	1179.2
12	1163.6	2494.1	1183.3

### ➤ LAB12-3 울릉도는 몇 월에 바람이 가장 강할까? groupby() 활용

울릉도의 기상 데이터에 기록된 매일의 평균 풍속 데이터를 바탕으로 몇 월의 바람이 가장 강한지 분석해 보았다. 이번에는 판다스가 제공하는 groupby() 함수로 더욱 효율적으로 만들어 보라.

원하는 결과



```
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt

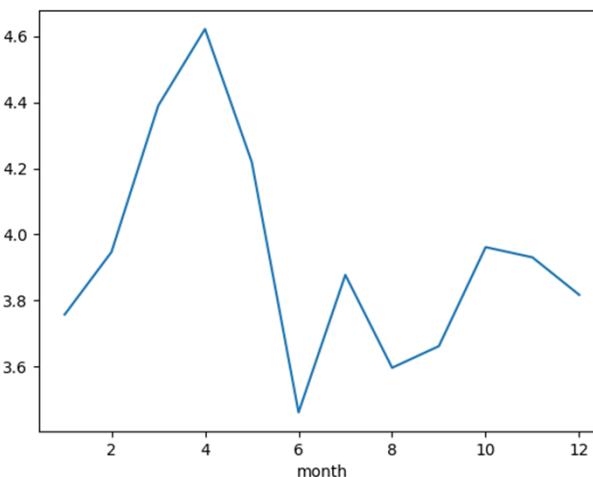
weather = pd.read_csv('d:/data/weather.csv', encoding='CP949')

weather['month'] = pd.DatetimeIndex(weather['일시']).month
means = weather.groupby('month').mean()
means['평균풍속'].plot(kind = 'bar')

plt.show()
```

LAB 12-1, 12-2보다 간단한  
코딩: 강력한 성능을 보임

해당 부분을 생략하면 LAB  
12-1, 12-2의 결과와 동일함



## Series & Dataframe

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

### ➤ 조건에 맞게 골라내자 : 필터링

- weather 데이터프레임에서 '최대풍속' 레이블로 되어 있는 열의 값이 10.0을 넘는지 확인하여 참과 거짓을 얻는 방법은 다음과 같다.

```
>>> weather['최대풍속'] >= 10.0
```

일시

2010-08-01 False  
2010-08-02 False  
2010-08-03 False  
...  
2020-07-29 False  
2020-07-30 False  
2020-07-31 False

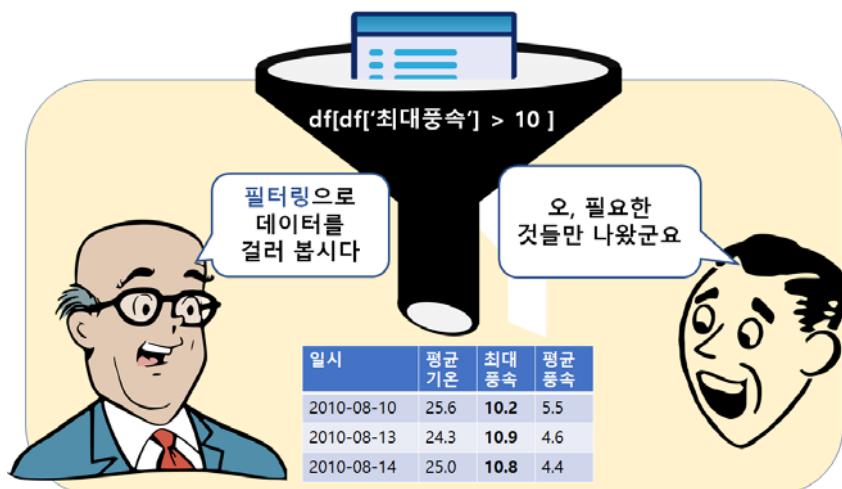
넘파이의 논리 인덱싱과 동일  
한문법

```
weather = pd.read_csv('./data/csv/weather.csv', index_col=0, encoding='CP949')  
print(weather['최대풍속']>=10)
```

'최대풍속' 레이블로 되어 있는 열의 값이  
10.0을 넘는지 확인하여 참값을 얻는 방법

```
>>> weather[ weather['최대풍속'] >= 10.0 ]
```

일시	평균기온	최대풍속	평균풍속
2010-08-10	25.6	10.2	5.5
2010-08-13	24.3	10.9	4.6
2010-08-14	25.0	10.8	4.4
...	...	...	...
2020-07-13	17.8	10.3	4.6
2020-07-14	17.8	12.7	9.4
2020-07-20	23.0	11.2	7.3



일시	평균기온	최대풍속	평균풍속
2010-08-01	28.7	8.3	3.4
2010-08-02	25.2	8.7	3.8
2010-08-03	22.1	6.3	2.9
2010-08-04	25.3	6.6	4.2
2010-08-05	27.2	9.1	5.6
2010-08-06	26.8	9.8	8
2010-08-07	27.5	9.1	5
2010-08-08	26.6	5.9	4

### ▶ 빠진 값을 찾기 : isna

- 데이터 과학자가 사용하는 실제 데이터는 완벽하지 않고 상당한 수의 결손값을 가지고 있거나 의심스러운 값을 가지고 있다. 결손값은 왜 생길까? 데이터가 아예 수집되지 않았거나, 측정 장치의 고장, 사건 사고 등으로 데이터를 확보할 수 없을 수도 있다. 따라서 데이터를 처리하기 전에 반드시 거쳐야 하는 절차가 데이터 정제이다. 판다스에서는 결손값을 **NAN**으로 나타낸다(혹은 NA로 표기함). 판다스는 결손값**missing data**을 탐지하고 수정하는 함수를 제공한다.
- weather.csv 데이터 역시 이러한 결손값이 존재한다. 데이터에 결손값이 있는지를 확인하는 함수는 **isna()**이다. 평균풍속이 측정되지 않았는지를 다음과 같이 확인해 보자.

weather['평균풍속'].**isna()**



[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

- weather[ weather['평균풍속'].**isna()** ]라고 하면, 이 조건을 이용하여 데이터프레임의 일부를 가져올 것이다. 즉 평균 풍속이 측정되지 않은 날들만 추출해 보고 싶다면 아래 코드로 가능하다. 지난 10년의 데이터 가운데 평균풍속이 기록되지 않은 날은 6일임을 알 수 있다.

```
import pandas as pd
```

```
weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')
missing_data = weather [ weather['평균풍속'].isna() ]
print(missing_data)
```

일시	평균기온	최대풍속	평균풍속
2012-02-11	-0.7	NaN	NaN
2012-02-12	0.4	NaN	NaN
2012-02-13	4.0	NaN	NaN
2015-03-22	10.1	11.6	NaN
2015-04-01	7.3	12.1	NaN
2019-04-18	15.7	11.7	NaN

## ➤ 빠진 값을 찾고 삭제하기 : dropna

축이 0이면 결손데이터를 포함한 행을 삭제하고  
축이 1이면 결손데이터를 포함한 열을 삭제한다.

inplace가 True이면 원본 데이터에서 결손데이터를 삭제하고  
False인 경우는 원본은 그대로 두고 고쳐진 데이터프레임 반환

**pandas.DataFrame.dropna(axis=0, how='any', inplace=False)**

how의 값이 'any'이면 결손 데이터가 하나라도 포함되면 제거 대상이 되고,  
'all'이면 axis 인자에 따라서 행 혹은 열 전체가 결손 데이터이어야 제거한다.



```
>>> weather.dropna(axis=0, how="any", inplace=True)
>>> weather.loc['2012-02-11']
... raise KeyError(key) from err
KeyError: '2012-02-11'
```

일시	평균기온	최대풍속	평균풍속
2012-02-11	-0.7	NaN	NaN
2012-02-12	0.4	NaN	NaN
2012-02-13	4.0	NaN	NaN
2015-03-22	10.1	11.6	NaN
2015-04-01	7.3	12.1	NaN
2019-04-18	15.7	11.7	NaN

### ➤ 빠진 데이터를 깨끗하게 메워 보자 : fillna

- 우리가 사용하고 있는 weather.csv의 결손값을 새로운 값으로 채워보자. 아래의 코드는 fillna() 함수를 이용하여 결손값을 0으로 채우고 있다. 그리고 이러한 작업이 원본에 반영되도록 inplace=True로 설정했음을 유의해서 보자. 평균풍속의 결손값이 존재했던 2012년 2월 11일의 데이터를 출력해 보자. 결손값 NaN이 아니라 0이 채워진 것을 확인할 수 있다.

```
import pandas as pd

weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')
weather.fillna(0, inplace = True) # 결손값을 0으로 채움, inplace를 True로 설정해 원본 데이터를 수정
print(weather.loc['2012-02-11'])

평균기온    -0.7
최대풍속     0.0
평균풍속     0.0
Name: 2012-02-11, dtype: float64
```

☞

```
weather.fillna( weather['평균풍속'].mean(), inplace = True)
print(weather.loc['2012-02-11'])
```

☞

```
평균기온    -0.700000
최대풍속     3.936441
평균풍속     3.936441
Name: 2012-02-11, dtype: float64
```

측정이 누락된 2012년 2월 11월의 풍속을 전체 데이터 평균으로 채울 수 있다



#### 잠깐 – inplace 매개변수

파andas 모듈은 데이터프레임에 대한 조작을 하는 다양한 함수를 제공한다. 그리고 많은 함수들이 inplace 매개 변수를 가지고 있다. 이것은 조작이 원본 데이터에 이루어지는 것인지, 아니면 사본을 만들어 사본을 변경하는지를 결정하게 된다.

파andas를 다룰 때 흔히 범하는 실수는 inplace의 디폴트 인자가 False임을 잘 모르고, 원본 데이터프레임이 변경되었다고 생각하는 것이다.

아래 코드를 실행하면 여전히 NaN이 출력될 것이다.

```
weather.fillna(0)
print(weather.loc['2012-02-11'][2])
```

➤ 빠진 데이터를 깨끗하게 메워 보자

```
>>> weather['최대풍속'].fillna( weather['최대풍속'].mean(), inplace = True)
```

```
>>> print(weather.loc['2012-02-11'])
```

평균기온 -0.700000

최대풍속 7.911099

평균풍속 NaN

측정이 누락된 2012년 2월 11월의 풍속을 최대 풍속 데이터 평균으로 채울 수 있다

```
Name: 2012-02-11, dtype: float64
```

```
>>> weather['평균풍속'].fillna( weather['평균풍속'].mean(), inplace = True)
```

```
>>> print(weather.loc['2012-02-11'])
```

평균기온 -0.700000

최대풍속 7.911099

평균풍속 3.936441

측정이 누락된 2012년 2월 11월의 풍속을 평균 풍속 데이터 평균으로 채울 수 있다

```
Name: 2012-02-11, dtype: float64
```

```
weather.fillna( weather['평균풍속'].mean(), inplace = True)  
print(weather.loc['2012-02-11'])
```

평균기온 -0.700000

최대풍속 3.936441

평균풍속 3.936441

```
Name: 2012-02-11, dtype: float64
```

## ➤ 데이터 구조를 변경 : pivot

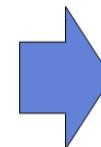
- 데이터프레임을 CSV를 읽어서 생성할 수도 있지만, 딕셔너리 데이터를 이용하여 생성할 수도 있다. 이때 키(key)는 열의 레이블이 되고, 딕셔너리의 키에 해당되는 값(value)은 열을 채우는 데이터를 가진 리스트가 된다. 딕셔너리의 한 항목이 시리즈 데이터가 되는 것이다. 다음과 같은 방식으로 데이터프레임을 만들어 보자.

```
import pandas as pd

df_1 = pd.DataFrame({'item' : ['ring0', 'ring0', 'ring1', 'ring1'],
                     'type' : ['Gold', 'Silver', 'Gold', 'Bronze'],
                     'price': [20000, 10000, 50000, 30000]})
```

	item	type	price
0	ring0	Gold	20000
1	ring0	Silver	10000
2	ring1	Gold	50000
3	ring1	Bronze	30000

index	item	type	price
0	ring0	Gold	20000
1	ring0	Silver	10000
2	ring1	Gold	50000
3	ring1	Bronze	30000



item	Bronze	Gold	Silver
ring0	NaN	20000	10000
ring1	30000	50000	NaN

```
df_2 = df_1.pivot(index='item', columns='type', values='price')
print(df_2)
```

item	type	Bronze	Gold	Silver
ring0		NaN	20000.0	10000.0
ring1		30000.0	50000.0	NaN



### ➤ concat() 함수로 데이터프레임을 합쳐보자

- 일반적으로 데이터들은 하나의 큰 테이블로 저장되지 않고, 작은 테이블로 나누어져 있는 경우가 많다. 이것은 저장과 관리의 편의성 때문이기도 하고, 데이터 수집의 시기, 주제 등이 달라 별도로 생성된 경우가 많기 때문이다. 이 절에서는 이러한 데이터를 하나로 합치는 방법 가운데 하나인 concat() 함수를 살펴보자.
- 우선 다음과 같이 데이터프레임을 두 개 준비해 보자. 딕셔너리 데이터를 이용하여 만들 수 있고, index를 원하는 값으로 설정할 수 있다.

```
df_1 = pd.DataFrame( {'A' : ['a10', 'a11', 'a12'],
                      'B' : ['b10', 'b11', 'b12'],
                      'C' : ['c10', 'c11', 'c12']} , index = ['가', '나', '다'] )

df_2 = pd.DataFrame( {'B' : ['b23', 'b24', 'b25'],
                      'C' : ['c23', 'c24', 'c25'],
                      'D' : ['d23', 'd24', 'd25']} , index = ['다', '라', '마'] )
```

	df_1			df_2		
	A	B	C	B	C	D
가	a10	b10	c10			
나	a11	b11	c11			
다	a12	b12	c12	b23	c23	d23
				b24	c24	d24
				b25	c25	d25

인덱스

합칠 데이터프레임의 리스트

축이 0이면 테이블의 행을 늘려서 붙여 나감  
축이 1이면 테이블의 열을 늘려며 붙여 나감

**pandas.concat( df\_list, axis=0, join='outer' )**

join 매개변수는 테이블들을 붙일 때 레이블들을 어떻게 사용할지 결정한다.  
이것의 인자가 'outer'이면 레이블들의 합집합으로 생성하고 'inner'이면  
레이블들의 교집합으로 생성된다.

```
df_3 = pd.concat( [df_1, df_2] ) # df_1, df_2 두 데이터프레임을 합쳐서 df_3를 생성
print(df_3)
```

	A	B	C	D
가	a10	b10	c10	NaN
나	a11	b11	c11	NaN
다	a12	b12	c12	NaN
	NaN	b23	c23	d23
	NaN	b24	c24	d24
	NaN	b25	c25	d25

## ➤ LAB12-4 다양한 방법으로 concat 적용해 보기

앞서 생성한 df\_1과 df\_2 데이터프레임을 합치는 데에 concat의 axis와 join 매개변수에 인자  
를 다양하게 적용하여 결과를 확인해보라.

### 원하는 결과

pandas.concat( [df\_1, df2], axis = 0, join = 'outer' )

	A	B	C	D
가	a10	b10	c10	NaN
나	a11	b11	c11	NaN
다	a12	b12	c12	NaN
다	NaN	b23	c23	d23
라	NaN	b24	c24	d24
마	NaN	b25	c25	d25

pandas.concat( [df\_1, df2], axis = 1, join = 'outer' )

	A	B	C	B	C	D
가	a10	b10	c10	NaN	NaN	NaN
나	a11	b11	c11	NaN	NaN	NaN
다	a12	b12	c12	b23	c23	d23
라	NaN	NaN	NaN	b24	c24	d24
마	NaN	NaN	NaN	b25	c25	d25

pandas.concat( [df\_1, df2], axis = 0, join = 'inner' )

	B	C
가	b10	c10
나	b11	c11
다	b12	c12
다	b23	c23
라	b24	c24
마	b25	c25

import pandas as pd

```
df_1 = pd.DataFrame( {'A' : ['a10', 'a11', 'a12'],
                      'B' : ['b10', 'b11', 'b12'],
                      'C' : ['c10', 'c11', 'c12']} , index = ['가', '나', '다'] )

df_2 = pd.DataFrame( {'B' : ['b23', 'b24', 'b25'],
                      'C' : ['c23', 'c24', 'c25'],
                      'D' : ['d23', 'd24', 'd25']} , index = ['다', '라', '마'] )

print( pd.concat( [df_1, df_2] , axis = 0, join = 'outer' ) )
print( pd.concat( [df_1, df_2] , axis = 0, join = 'inner' ) )
print( pd.concat( [df_1, df_2] , axis = 1, join = 'outer' ) )
print( pd.concat( [df_1, df_2] , axis = 1, join = 'inner' ) )
```

pandas.concat( [df\_1, df2], axis = 1, join = 'inner' )

	A	B	C	B	C	D
다	a12	b12	c12	b23	c23	d23

## Series & Dataframe

#### ▶ 데이터베이스 join 방식의 데이터 병합 – merge

- 데이터베이스는 **join**이라는 연산을 지원한다. 이 조인 연산과 같은 방식의 데이터 병합을 지원하는 판다스 함수가 **merge()** 함수이다. **merge()** 함수는 데이터프레임 df\_1을 df\_2와 병합하려고 할 때, 다음과 같은 방식을 사용한다.

`df_1.merge(df_2, 다양한 선택 사항을 결정하는 키워드 인자들... )`

#### 현재의 데이터프레임과 결합할 데이터프레임

결합의 방  
'left'  
'right'



이것들이 중요한  
매개변수들이에요

`DataFrame.merge(right, how='inner', on=None)`

조인 연산을 수행하기 위해 사용할 레이블  
(왼쪽과 오른쪽 데이터프레임 모두에 존재해야 함)

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

	A	B	C
가	a10	b10	c10
나	a11	b11	c11
다	a12	b12	c12
인덱스			

인덱스

		df_2		
인덱스	값	B	C	D
다	b23	c23	d23	
라	b24	c24	d24	
마	b25	c25	d25	

10

```
df_3 = df_1.merge(df_2, how='outer', on='B')  
print(df_3)
```

— 1 —

	A	B	C_x	C_y	D
0	a10	b10	c10	NaN	NaN
1	a11	b11	c11	NaN	NaN
2	a12	b12	c12	NaN	NaN
3	NaN	b23	NaN	c23	d23
4	NaN	b24	NaN	c24	d24
5	NaN	b25	NaN	c25	d25

이 키를 기준으로 조인이 이루어짐

드 데이터프레임이 동시에 가져

출동하는 레이블은 점미사를 붙여 구분

▼ 글과는密切한 접두어를 둘러보자

	A	B	C_x	C_y	D
0	a10	b10	c10	NaN	NaN
1	a11	b11	c11	NaN	NaN
2	a12	b12	c12	NaN	NaN
3	NaN	b23	NaN	c23	d23
4	NaN	b24	NaN	c24	d24
5	NaN	b25	NaN	c25	d25

### ➤ 인덱스를 키로 활용하여 merge 적용해 보기

- 앞에서 살펴본 merge() 함수의 동작 결과를 살펴보면, 두 데이터를 결합할 때 사용할 키가 될 레이블을 지정하면, 해당 레이블에 있는 값들을 이용하여 테이블을 생성하고 있다. 그런데, 원래 테이블에 있던 인덱스는 사라진 것을 확인할 수 있다.
- 많은 경우 인덱스가 키의 역할을 수행하는 경우도 있다. 이런 경우에 **인덱스를 키로 사용**하라고 할 수도 있다. 이러한 방식으로 merge를 수행하려면 다음과 같이 코딩을 하면 된다.

```
df_1.merge(df_2, how = 'outer', left_index = True, right_index = True )
```

- 이 방식으로 merge를 적용해보자. 앞서 사용한 데이터프레임을 그대로 사용하면, 그림과 같은 모양의 테이블 데이터가 존재한다.

	A	B	C
인덱스	a10	b10	c10
인덱스	a11	b11	c11
인덱스	a12	b12	c12

	B	C	D
인덱스	b23	c23	d23
인덱스	b24	c24	d24
인덱스	b25	c25	d25

```
df_3 = df_1.merge(df_2, how='outer', left_index = True, right_index = True )
print(df_3)
```

	A	B_x	C_x	B_y	C_y	D
가	a10	b10	c10	NaN	NaN	NaN
나	a11	b11	c11	NaN	NaN	NaN
다	a12	b12	c12	b23	c23	d23
라	NaN	NaN	NaN	b24	c24	d24
마	NaN	NaN	NaN	b25	c25	d25



## ➤ LAB12-5 다양한 방법으로 merge 적용해 보기

앞서 생성한 df\_1과 df\_2 데이터프레임을 합치는 데에 merge()의 how 매개변수에는 네 종류의 인자를 넘길 수 있다. on='B'를 유지한채로 how를 변경하여 다양한 결과를 확인해 보라.

### 원하는 결과

full outer					
	A	B	C_x	C_y	D
0	a10	b10	c10	NaN	NaN
1	a11	b11	c11	NaN	NaN
2	a12	b12	c12	NaN	NaN
right outer					
	A	B	C_x	C_y	D
0	NaN	b23	NaN	c23	d23
1	NaN	b24	NaN	c24	d24
2	NaN	b25	NaN	c25	d25
inner					
	Empty DataFrame				
	Columns: [A, B, C_x, C_y, D]				
	Index: []				

```
import pandas as pd

df_1 = pd.DataFrame( {'A' : ['a10', 'a11', 'a12'],
                      'B' : ['b10', 'b11', 'b12'],
                      'C' : ['c10', 'c11', 'c12']} , index = ['가', '나', '다'] )

df_2 = pd.DataFrame( {'B' : ['b23', 'b24', 'b25'],
                      'C' : ['c23', 'c24', 'c25'],
                      'D' : ['d23', 'd24', 'd25']} , index = ['다', '라', '마'] )

print('left outer \n' , df_1.merge(df_2, how='left', on='B' ) )
print('right outer \n' , df_1.merge(df_2, how='right', on='B' ) )
print('full outer \n' , df_1.merge(df_2, how='outer', on='B' ) )
print('inner \n' , df_1.merge(df_2, how='inner', on='B' ) )
```

	A	B	C
가	a10	b10	c10
나	a11	b11	c11
다	a12	b12	c12

인덱스

	B	C	D
다	b23	c23	d23
라	b24	c24	d24
마	b25	c25	d25

인덱스

## Series & Dataframe

[출처] “따라하며 배우는 파이썬과 데이터 과학”, 생능출판사 강의자료

### ➤ 데이터를 크기에 따라 나열하자 : sort\_values

```
import pandas as pd
import matplotlib.pyplot as plt

countries_df = pd.read_csv('d:/data/countries.csv', index_col = 0)
sorted = countries_df.sort_values('population')
print(sorted)
```

	country	area	capital	population
KR	Korea	98480	Seoul	51780579
JP	Japan	377835	Tokyo	125960000
RU	Russia	17100000	Moscow	146748600
US	USA	9629091	Washington	331002825
CN	China	9596960	Beijing	1439323688

```
countries_df = pd.read_csv('d:/data/countries.csv', index_col = 0)
countries_df.sort_values('population', inplace = True)
print(countries_df)
```

```
countries = pd.read_csv('d:/data/countries.csv', index_col = 0, encoding='CP949')
countries.sort_values(['population', 'area'], ascending = False, inplace = True)
print(countries)
```

	country	area	capital	population
CN	China	9596960	Beijing	1439323688
US	USA	9629091	Washington	331002825
RU	Russia	17100000	Moscow	146748600
JP	Japan	377835	Tokyo	125960000
KR	Korea	98480	Seoul	51780579

[https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort\\_values.html](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_values.html)

```
>>> df = pd.DataFrame({
...     'col1': ['A', 'A', 'B', np.nan, 'D', 'C'],
...     'col2': [2, 1, 9, 8, 7, 4],
...     'col3': [0, 1, 9, 4, 2, 3],
...     'col4': ['a', 'B', 'c', 'D', 'e', 'F']
... })
>>> df
   col1  col2  col3  col4
0     A     2     0     a
1     A     1     1     B
2     B     9     9     c
3    NaN     8     4     D
4     D     7     2     e
5     C     4     3     F
```

```
>>> df.sort_values(by=['col1', 'col2'])
   col1  col2  col3  col4
1     A     1     1     B
0     A     2     0     a
2     B     9     9     c
5     C     4     3     F
4     D     7     2     e
3    NaN     8     4     D
```

## Pandas Tutorial

[https://pandas.pydata.org/docs/getting\\_started/intro\\_tutorials/index.html](https://pandas.pydata.org/docs/getting_started/intro_tutorials/index.html)

<https://github.com/pandas-dev/pandas>

## Getting started tutorials

- What kind of data does pandas handle?
- How do I read and write tabular data?
- How do I select a subset of a `DataFrame`?
- How to create plots in pandas?
- How to create new columns derived from existing columns?
- How to calculate summary statistics?
- How to reshape the layout of tables?
- How to combine data from multiple tables?
- How to handle time series data with ease?
- How to manipulate textual data?

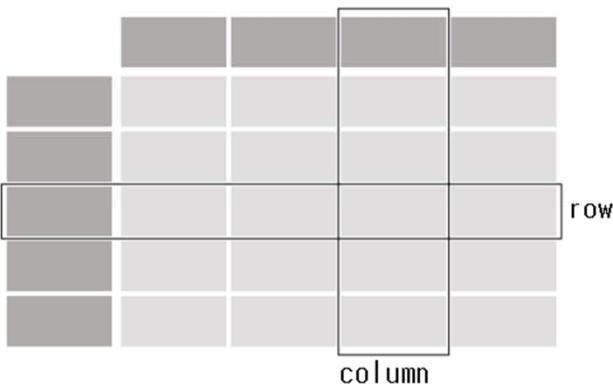
[출처]

[https://pandas.pydata.org/docs/getting\\_started/intro\\_tutorials/02\\_read\\_write.html](https://pandas.pydata.org/docs/getting_started/intro_tutorials/02_read_write.html)

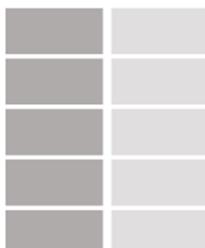
<https://github.com/pandas-dev/pandas>

### ➤ pandas data table representation

DataFrame



Series



- Each column in a **DataFrame** is a **Series**

You can create a **Series** from scratch as well:

```
In [5]: ages = pd.Series([22, 35, 58], name="Age")
In [6]: ages
Out[6]:
0    22
1    35
2    58
Name: Age, dtype: int64
```

```
In [2]: df = pd.DataFrame(
...:     {
...:         "Name": [
...:             "Braund, Mr. Owen Harris",
...:             "Allen, Mr. William Henry",
...:             "Bonell, Miss. Elizabeth",
...:         ],
...:         "Age": [22, 35, 58],
...:         "Sex": ["male", "male", "female"],
...:     }
...: )
In [3]: df
Out[3]:
   Name  Age  Sex
0 Braund, Mr. Owen Harris    22  male
1 Allen, Mr. William Henry    35  male
2 Bonnell, Miss. Elizabeth    58 female
```

A screenshot of LibreOffice Calc showing a spreadsheet titled "Untitled 1 - LibreOffice Calc". The spreadsheet has columns labeled A through F and rows numbered 1 through 8. The data is as follows:

	Name	Age	Sex
2	Braund, Mr. Owen Harris	22	male
3	Allen, Mr. William Henry	35	male
4	Bonnell, Miss. Elizabeth	58	female
5			
6			
7			
8			

- Basic statistics of the numerical data of my data table

```
In [9]: df.describe()
Out[9]:
              Age
count    3.000000
mean    38.333333
std     18.230012
min    22.000000
25%    28.500000
50%    35.000000
75%    46.500000
max    58.000000
```

## Pandas Tutorial

### ➤ How do I read and write tabular data?

```
In [1]: import pandas as pd
```

Data used for this tutorial:

Titanic data

This tutorial uses the Titanic data set, stored as CSV. The data consists of the following data columns:

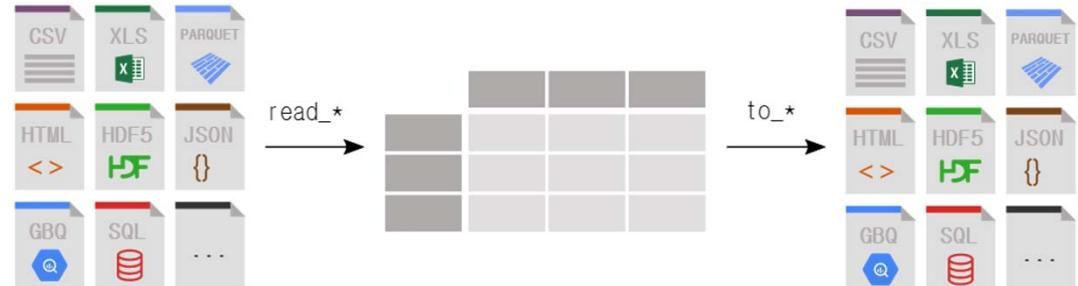
- PassengerId: Id of every passenger.
- Survived: This feature have value 0 and 1. 0 for not survived and 1 for survived.
- Pclass: There are 3 classes: Class 1, Class 2 and Class 3.
- Name: Name of passenger.
- Sex: Gender of passenger.
- Age: Age of passenger.
- SibSp: Indication that passenger have siblings and spouse
- Parch: Whether a passenger is alone or have family.
- Ticket: Ticket number of passenger.
- Fare: Indicating the fare.
- Cabin: The cabin of passenger.
- Embarked: The embarked category.

[출처]

[https://pandas.pydata.org/docs/getting\\_started/intro\\_tutorials/02\\_read\\_write.html](https://pandas.pydata.org/docs/getting_started/intro_tutorials/02_read_write.html)

<https://github.com/pandas-dev/pandas>

CSV, XLS, PARQUET, HTML, HDF5, JSON, GBQ, SQL



`read_csv, read_excel, read_parquet,  
read_html, read_hdf5, read_json,  
read_gbq, read_sql`

`to_csv, to_excel, to_parquet,  
to_html, to_hdf5, to_json, to_gbq,  
to_sql`

<https://github.com/pandas-dev/pandas/blob/master/doc/data/titanic.csv>

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925		S
4	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	6	0	3	Moran, Mr. James	male		0	0	330877	8.4583		Q
7	7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S

## ➤ How do I read and write tabular data?

- Analyze the Titanic passenger data, available as a CSV file.

```
import pandas as pd
titanic = pd.read_csv("data/titanic.csv")
```

titanic

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0 26.0	1 0	0	PC 17599 STON/O2. 3101282	71.2833 7.9250	C85 NaN	C S
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
3	4	1	1	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
4	5	0	3									
...	...	...	...		...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W.C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

- See the first 8 rows of a pandas DataFrame.

```
titanic.head(8)
```

When displaying a DataFrame, the first and last 5 rows will be shown by default:

[출처]

[https://pandas.pydata.org/docs/getting\\_started/intro\\_tutorials/02\\_read\\_write.html](https://pandas.pydata.org/docs/getting_started/intro_tutorials/02_read_write.html)

<https://github.com/pandas-dev/pandas>

### ➤ How do I read and write tabular data?

- A check on how pandas interpreted each of the column data types can be done by requesting the pandas **dtypes** attribute:

```
titanic.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name             object
Sex              object
Age              float64
SibSp            int64
Parch            int64
Ticket           object
Fare             float64
Cabin            object
Embarked         object
dtype: object
```

The data types in this DataFrame are integers (int64), floats (float64) and strings (object).

- Exporting data out of pandas is provided by different to\_\* methods.
- Convert the Titanic data as a spreadsheet.

```
titanic.to_excel("data/titanic.xlsx", sheet_name="passengers", index=False)
titanic = pd.read_excel("data/titanic.xlsx", sheet_name="passengers")
titanic.head()
```

Whereas `read_*` functions are used to read data to pandas, the `to_*` methods are used to store data. The `to_excel()` method stores the data as an excel file. In the example here, the `sheet_name` is named *passengers* instead of the default *Sheet1*. By setting `index=False` the row index labels are not saved in the spreadsheet.

- ❖ No module named ‘openpyxl’ : Terminal 창에서 conda install openpyxl 설치

```
ModuleNotFoundError: No module named 'openpyxl'
```

```
[ ]:
```

```
Terminal 2
```



Note

When asking for the `dtypes`, no brackets are used! `dtypes` is an attribute of a `DataFrame` and `Series`. Attributes of `DataFrame` or `Series` do not need brackets. Attributes represent a characteristic of a `DataFrame/Series`, whereas a method (which requires brackets) *do* something with the `DataFrame/Series` as introduced in the first tutorial.

```
새로운 크로스 플랫폼 PowerShell 사용 https://aka.ms/pscore6
PS C:\Users\sky> conda install openpyxl
Collecting package metadata (current_repotdata.json): done
Solving environment: done
```

### ➤ How do I read and write tabular data?

- A technical summary of a **DataFrame**

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId 891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object  
 4   Sex          891 non-null    object  
 5   Age          714 non-null    float64 
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object  
 9   Fare          891 non-null    float64 
 10  Cabin        204 non-null    object  
 11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

#### REMEMBER

- Getting data in to pandas from many different file formats or data sources is supported by `read_*` functions.
- Exporting data out of pandas is provided by different `to_*` methods.
- The `head/tail/info` methods and the `dtypes` attribute are convenient for a first check.

### ➤ How do I select specific a subset from a DataFrame?

#### 1) How do I select specific columns from a DataFrame?



- The age of the Titanic passengers.

```
ages = titanic["Age"]
ages.head()
```

0 22.0  
1 38.0  
2 26.0  
3 35.0  
4 35.0  
Name: Age, dtype: float64

```
type(titanic["Age"])
pandas.core.series.Series
```

titanic["Age"].shape  
(891,)

- ✓ To select a single column, **use square brackets [] with the column name** of the column of interest.

[출처]

[https://pandas.pydata.org/docs/getting\\_started/intro\\_tutorials/02\\_read\\_write.html](https://pandas.pydata.org/docs/getting_started/intro_tutorials/02_read_write.html)

<https://medium.com/epfl-extension-school/selecting-data-from-a-pandas-dataframe-53917dc39953>

- The age and sex of the Titanic passengers.

```
In [8]: age_sex = titanic[["Age", "Sex"]]
```

```
In [9]: age_sex.head()
```

```
Out[9]:
      Age      Sex
0   22.0    male
1   38.0  female
2   26.0  female
3   35.0  female
4   35.0    male
```

```
In [10]: type(titanic[["Age", "Sex"]])
```

```
Out[10]: pandas.core.frame.DataFrame
```

```
In [11]: titanic[["Age", "Sex"]].shape
```

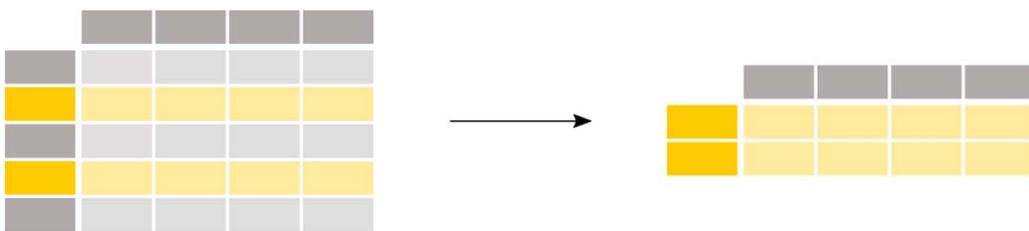
```
Out[11]: (891, 2)
```

- ✓ The selection returned a DataFrame with 891 rows and 2 columns. Remember, a DataFrame is 2-dimensional with both a row and column dimension.

## Pandas Tutorial

### ➤ How do I select specific a subset from a DataFrame?

#### 2) How do I select specific rows from a DataFrame?¶



- The passengers older than 35 years.

```
above_35 = titanic[titanic["Age"] > 35]
above_35.head()
```

above\_35.shape

(217, 12)

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
6	7	0	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
11	12	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
13	14	0	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	Nan	S
15	16	1	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000	Nan	S

[출처]

[https://pandas.pydata.org/docs/getting\\_started/intro\\_tutorials/02\\_read\\_write.html](https://pandas.pydata.org/docs/getting_started/intro_tutorials/02_read_write.html)

<https://medium.com/epfl-extension-school/selecting-data-from-a-pandas-dataframe-53917dc39953>

- Titanic passengers from cabin class 2 and 3.

```
class_23 = titanic[titanic["Pclass"].isin([2, 3])]
class_23.head()
```

class\_23.shape

(675, 12)

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Nan	S
2	3	1	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	Nan	S
4	5	0	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	Nan	S
5	6	0	Moran, Mr. James	male	Nan	0	0	330877	8.4583	Nan	Q
7	8	0	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	Nan	S

Similar to the conditional expression, the `isin()` conditional function returns a `True` for each row the values are in the provided list. To filter the rows based on such a function, use the conditional function inside the selection brackets `[]`. In this case, the condition inside the selection brackets `titanic["Pclass"].isin([2, 3])` checks for which rows the `Pclass` column is either 2 or 3.

The above is `equivalent` to filtering by rows for which the class is either 2 or 3 and combining the two statements with an `|` (or) operator:

```
class_23 = titanic[(titanic["Pclass"] == 2) | (titanic["Pclass"] == 3)]
class_23.head()
```

[출처]

[https://pandas.pydata.org/docs/getting\\_started/intro\\_tutorials/02\\_read\\_write.html](https://pandas.pydata.org/docs/getting_started/intro_tutorials/02_read_write.html)

<https://medium.com/epfl-extension-school/selecting-data-from-a-pandas-dataframe-53917dc39953>

### ➤ How do I select specific a subset from a DataFrame?

#### 2) How do I select specific rows from a DataFrame?

- Passenger data for which the age is known.

```
age_no_na = titanic[titanic["Age"].notna()]
age_no_na.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3		female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
age_no_na.shape
```

(714, 12)

The `notna()` conditional function returns a `True` for each row the values are not an `Null` value.

As such, this can be combined with the selection brackets `[]` to filter the data table.

### ➤ How do I select specific a subset from a DataFrame?

#### 3) How do I select specific rows and columns from a DataFrame?



- The names of the passengers older than 35 years.

```
adult_names = titanic.loc[titanic["Age"] > 35, "Name"]
adult_names.head()
```

```
1    Cumings, Mrs. John Bradley (Florence Briggs Th...
6                               McCarthy, Mr. Timothy J
11                             Bonnell, Miss. Elizabeth
13                            Andersson, Mr. Anders Johan
15                  Hewlett, Mrs. (Mary D Kingcome)
Name: Name, dtype: object
```

In this case, a subset of both rows and columns is made in one go and just using selection brackets `[]` is not sufficient anymore. The `loc/iloc` operators are required in front of the selection brackets `[]`. When using `loc/iloc`, the part before the comma is the rows you want, and the part after the comma is the columns you want to select.

[출처]

[https://pandas.pydata.org/docs/getting\\_started/intro\\_tutorials/02\\_read\\_write.html](https://pandas.pydata.org/docs/getting_started/intro_tutorials/02_read_write.html)

<https://medium.com/epfl-extension-school/selecting-data-from-a-pandas-dataframe-53917dc39953>

- rows 10 till 25 and columns 3 to 5.

```
titanic.iloc[9:15, 2:5]
```

	Pclass	Name	Sex
9	2	Nasser, Mrs. Nicholas (Adele Achem)	female
10	3	Sandstrom, Miss. Marguerite Rut	female
11	1	Bonnell, Miss. Elizabeth	female
12	3	Saundercock, Mr. William Henry	male
13	3	Andersson, Mr. Anders Johan	male
14	3	Vestrom, Miss. Hulda Amanda Adolfina	female

[Note]

- The `iloc` operator to **select by integer positions** : The `iloc` operator allows us to slice both rows and columns **using their position**.
- The `loc` operator to **select by labels and names** : The `loc` operator is similar to the `iloc` operator except that instead of **referencing rows and columns using their position in the DataFrame** we use the index labels and column names respectively.

```
df.iloc[::2, 2:]
```

```
df.loc[:, 'Height (m)':'First ascent']
```

## ➤ How to create plots in pandas?

```
In [1]: import pandas as pd  
In [2]: import matplotlib.pyplot as plt
```

Data used for this tutorial:

### Air quality data

For this tutorial, air quality data about  $NO_2$  is used, made available by `openaq` and using the `py-openaq` package. The `air_quality_no2.csv` data set provides  $NO_2$  values for the measurement stations *FR04014*, *BETR801* and *London Westminster* in respectively Paris, Antwerp and London.

[https://github.com/pandas-dev/pandas/blob/master/doc/data/air\\_quality\\_no2.csv](https://github.com/pandas-dev/pandas/blob/master/doc/data/air_quality_no2.csv)

```
air_quality = pd.read_csv("data/air_quality_no2.csv", index_col=0, parse_dates=True)  
air_quality.head()
```

	station_antwerp	station_paris	station_london
datetime			
2019-05-07 02:00:00	NaN	NaN	23.0
2019-05-07 03:00:00	50.5	25.0	19.0
2019-05-07 04:00:00	45.0	27.7	19.0
2019-05-07 05:00:00	NaN	50.4	16.0
2019-05-07 06:00:00	NaN	61.9	NaN

```
air_quality.shape
```

(1035, 3)

### Note

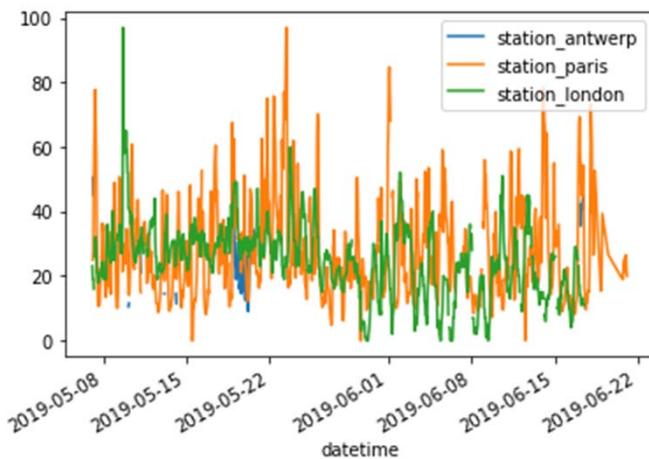
The usage of the `index_col` and `parse_dates` parameters of the `read_csv` function to define the first (0th) column as index of the resulting `DataFrame` and convert the dates in the column to `Timestamp` objects, respectively.

## ➤ How to create plots in pandas?

- Quick visual check of the data.

```
air_quality.plot()
```

```
<AxesSubplot:xlabel='datetime'>
```

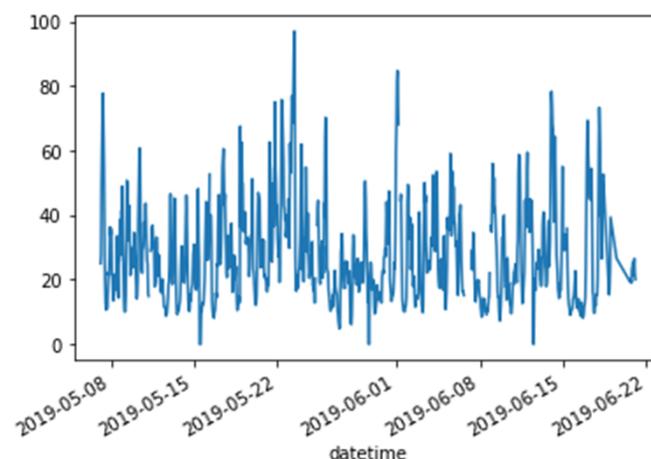


With a DataFrame, pandas creates by default one line plot for each of the columns with numeric data.

- Plot only the columns of the data table with the data from Paris.

```
air_quality["station_paris"].plot()
```

```
<AxesSubplot:xlabel='datetime'>
```

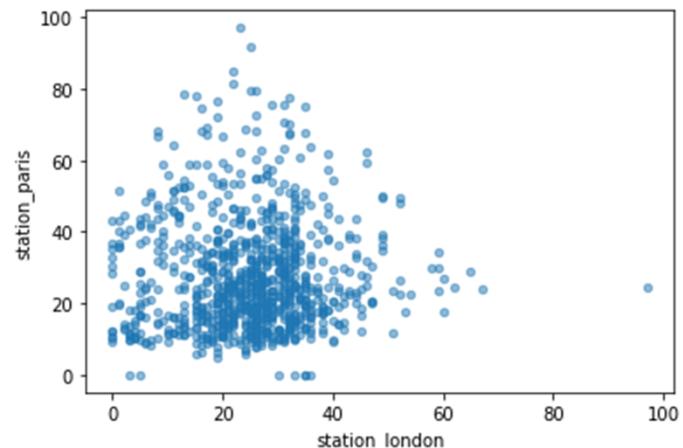


To plot a specific column, use the selection method of the subset data tutorial in combination with the plot() method. Hence, the plot() method works on both Series and DataFrame.

- Visually compare the N02 values measured in London versus Paris.

```
air_quality.plot.scatter(x="station_london",
y="station_paris", alpha=0.5)
```

```
<AxesSubplot:xlabel='station_london', ylabel='station_paris'>
```



## ➤ How to create plots in pandas?

- Apart from the [default line plot](#) when using the plot function, a number of alternatives are available to plot data.
- Let's use some standard Python to get an overview of the available plot methods:

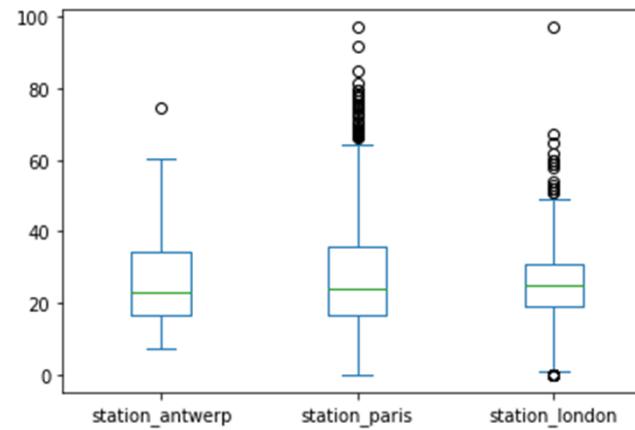
```
[method_name
for method_name in dir(air_quality.plot)
if not method_name.startswith("_")]
['area',
'bar',
'barh',
'box',
'density',
'hexbin',
'hist',
'kde',
'line',
'pie',
'scatter']
```

```
dir(air_quality.plot)
['__annotations__', '__subclasshook__',
 '__call__', '__weakref__',
 '__class__', '_accessors',
 '__delattr__', '_all_kinds',
 '__dict__', '_common_kinds',
 '__dir__', '_constructor',
 '__doc__', '_dataframe_kinds',
 '__eq__', '_dir_additions',
 '__format__', '_dir_deletions',
 '__ge__', '_get_call_args',
 '__getattribute__', '_hiddenAttrs',
 '__gt__', '_kind_aliases',
 '__hash__', '_parent',
 '__init__', '_reset_cache',
 '__init_subclass__', '_series_kinds',
 '__le__', 'area',
 '__lt__', 'bar',
 '__module__', 'barh',
 '__ne__', 'box',
 '__new__', 'density',
 '__reduce__', 'hexbin',
 '__reduce_ex__', 'hist',
 '__repr__', 'kde',
 '__setattr__', 'line',
 '__sizeof__', 'pie',
 '__str__', 'scatter']
```

- One of the options is DataFrame.plot.box(), which refers to a boxplot.

```
air_quality.plot.box()
```

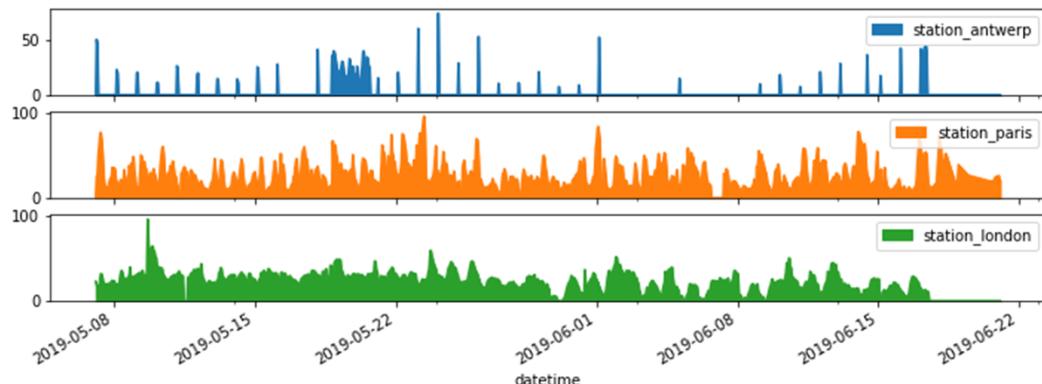
<AxesSubplot:>



## ➤ How to create plots in pandas?

- Each of the columns in a separate subplot.

```
axs = air_quality.plot.area(figsize=(12, 4), subplots=True)
```



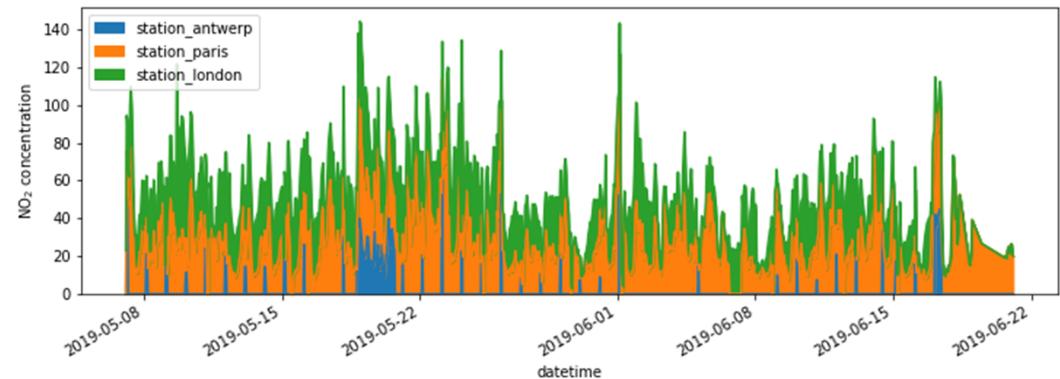
- further customize, extend or save the resulting plot.

```
# Create an empty matplotlib Figure and Axes
fig, axs = plt.subplots(figsize=(12, 4))
```

```
# Use pandas to put the area plot on the prepared Figure/Axes
air_quality.plot.area(ax=axs)
```

```
# Do any matplotlib customization you like
axs.set_ylabel("NO$_2$ concentration")
```

```
# Save the Figure/Axes using the existing matplotlib method.
fig.savefig("no2_concentrations.png")
```



➤ How to create new columns derived from existing columns?



- air\_quliaty\_no2.csv

```
air_quality = pd.read_csv("data/air_quality_no2.csv",
                           index_col=0, parse_dates=True)
air_quality.head()
```

	station_antwerp	station_paris	station_london
datetime			
2019-05-07 02:00:00	NaN	NaN	23.0
2019-05-07 03:00:00	50.5	25.0	19.0
2019-05-07 04:00:00	45.0	27.7	19.0
2019-05-07 05:00:00	NaN	50.4	16.0
2019-05-07 06:00:00	NaN	61.9	NaN

- Express the NO2 concentration of the station in London in mg/m<sup>3</sup>  
(If we assume temperature of 25 degrees Celsius and pressure of 1013 hPa, the conversion factor is 1.882)

```
air_quality["london_mg_per_cubic"] = air_quality["station_london"] * 1.882
air_quality.head()
```

	station_antwerp	station_paris	station_london	london_mg_per_cubic
datetime				
2019-05-07 02:00:00			23.0	43.286
2019-05-07 03:00:00	50.5	25.0	19.0	35.758
2019-05-07 04:00:00	45.0	27.7	19.0	35.758
2019-05-07 05:00:00		50.4	16.0	30.112
2019-05-07 06:00:00		61.9	NaN	NaN

To create a new column, use the [] brackets with the new column name at the left side of the assignment.

**Note**

The calculation of the values is done **element wise**. This means all values in the given column are multiplied by the value 1.882 at once. You do not need to use a loop to iterate each of the rows!

➤ How to create new columns derived from existing columns?



- Check the ratio of the values in Paris versus Antwerp and save the result in a new column

```
air_quality["ratio_paris_antwerp"] = (
    air_quality["station_paris"] / air_quality["station_antwerp"]
)
air_quality.head()
```

The calculation is again element-wise, so the / is applied for the values in each row.

	station_antwerp	station_paris	station_london	london_mg_per_cubic	ratio_paris_antwerp
datetime					
2019-05-07 02:00:00	NaN	NaN	23.0	43.286	NaN
2019-05-07 03:00:00	50.5	25.0	19.0	35.758	0.495050
2019-05-07 04:00:00	45.0	27.7	19.0	35.758	0.615556
2019-05-07 05:00:00	NaN	50.4	16.0	30.112	NaN
2019-05-07 06:00:00	NaN	61.9	NaN	NaN	NaN

## ➤ How to create new columns derived from existing columns?

- Rename the data columns to the corresponding station identifiers used by openAQ

```
air_quality_renamed = air_quality.rename(
    columns={
        "station_antwerp": "BETR801",
        "station_paris": "FR04014",
        "station_london": "London Westminster",
    }
)
air_quality_renamed.head()
```

	BETR801	FR04014	London Westminster	london_mg_per_cubic	ratio_paris_antwerp
datetime					
2019-05-07 02:00:00	NaN	NaN	23.0	43.286	NaN
2019-05-07 03:00:00	50.5	25.0	19.0	35.758	0.495050
2019-05-07 04:00:00	45.0	27.7	19.0	35.758	0.615556
2019-05-07 05:00:00	NaN	50.4	16.0	30.112	NaN
2019-05-07 06:00:00	NaN	61.9	NaN	NaN	NaN

- The mapping should not be restricted to fixed names only, but can be a mapping function as well. For example, converting the column names to lowercase letters can be done using a function as well:

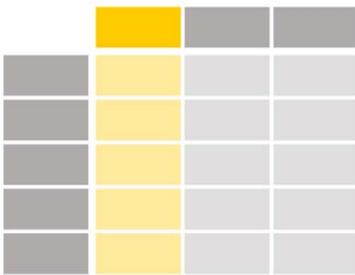
```
air_quality_renamed = air_quality_renamed.rename(columns=str.lower)
air_quality_renamed.head()
```

	betr801	fr04014	london westminster	london_mg_per_cubic	ratio_paris_antwerp
datetime					
2019-05-07 02:00:00	NaN	NaN	23.0	43.286	NaN
2019-05-07 03:00:00	50.5	25.0	19.0	35.758	0.495050
2019-05-07 04:00:00	45.0	27.7	19.0	35.758	0.615556
2019-05-07 05:00:00	NaN	50.4	16.0	30.112	NaN
2019-05-07 06:00:00	NaN	61.9	NaN	NaN	NaN

- The `rename()` function can be used for both row labels and column labels. Provide a dictionary with the keys the current names and the values the new names to update the corresponding names.

## ➤ How to calculate summary statistics?

### 1) Aggregating statistics



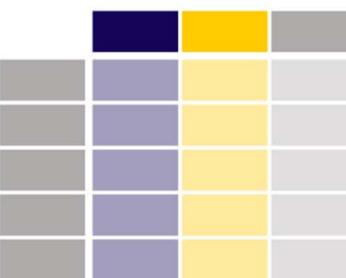
- titanic.csv

```
titanic = pd.read_csv("data/titanic.csv")
```

- Average age of the Titanic passengers

```
titanic["Age"].mean()
```

```
29.69911764705882
```



- Median age and ticket fare price of the Titanic passengers?

```
titanic[["Age", "Fare"]].median()
```

```
Age      28.0000
Fare    14.4542
dtype: float64
```

```
titanic[["Age", "Fare"]].describe()
```

	Age	Fare
count	714.000000	891.000000
mean	29.699118	32.204208
std	14.526497	49.693429
min	0.420000	0.000000
25%	20.125000	7.910400
50%	28.000000	14.454200
75%	38.000000	31.000000
max	80.000000	512.329200

## ➤ How to calculate summary statistics?

### 1) Aggregating statistics

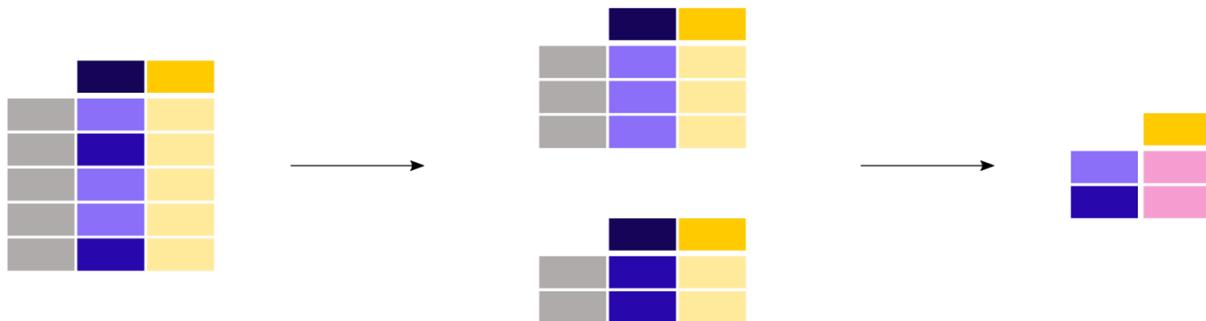
- Instead of the predefined statistics, specific combinations of aggregating statistics for given columns can be defined using the `DataFrame.agg()` method:

```
titanic.agg(
    {
        "Age": ["min", "max", "median", "skew"],
        "Fare": ["min", "max", "median", "mean"],
    }
)
```

	Age	Fare
min	0.420000	0.000000
max	80.000000	512.329200
median	28.000000	14.454200
skew	0.389108	NaN
mean	NaN	32.204208

- Details about descriptive statistics are provided in the user guide section on [descriptive statistics](#).

### 2) Aggregating statistics grouped by category



- Average age for non-survived versus survived Titanic passengers

```
titanic[["Survived", "Age"]].groupby("Survived").mean()
```

	Age
<b>Survived</b>	
0	30.626179
1	28.343690

As our interest is the average age for each gender, a subselection on these two columns is made first: `titanic[["Survived", "Age"]]`. Next, the `groupby()` method is applied on the `Survived` column to make a group per category. The average age for each gender is calculated and returned.

## ➤ How to calculate summary statistics?

### 2) Aggregating statistics grouped by category

Calculating a given statistic (e.g. `mean` age) for each category in a column (e.g. male/female in the `Sex` column) is a common pattern. The `groupby` method is used to support this type of operations. More general, this fits in the more general `split-apply-combine` pattern:

- **Split** the data into groups
- **Apply** a function to each group independently
- **Combine** the results into a data structure

The apply and combine steps are typically done together in pandas.

- In the previous example, we explicitly selected the 2 columns first. If not, the `mean` method is applied to each column containing numerical columns:

```
titanic.groupby("Survived").mean()
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare
Survived						
0	447.016393	2.531876	30.626179	0.553734	0.329690	22.117887
1	444.368421	1.950292	28.343690	0.473684	0.464912	48.395408

```
titanic.groupby("Survived")["Age"].mean()
```

```
Survived
0    30.626179
1    28.343690
Name: Age, dtype: float64
```

Count number of records by category

```
titanic      .groupby("Survived")      [ "Age" ]      .mean()
```



#### Note

The `Pclass` column contains numerical data but actually represents 3 categories (or factors) with respectively the labels '1', '2' and '3'. Calculating statistics on these does not make much sense. Therefore, pandas provides a `Categorical` data type to handle this type of data. More information is provided in the user guide `Categorical data` section.

## ➤ How to calculate summary statistics?

### 2) Aggregating statistics grouped by category

- Mean ticket fare price for each of the sex and cabin class combinations?

```
titanic.groupby(["Sex", "Pclass"])["Fare"].mean()
```

Sex	Pclass	Fare
female	1	106.125798
	2	21.970121
	3	16.118810
male	1	67.226127
	2	19.741782
	3	12.661633

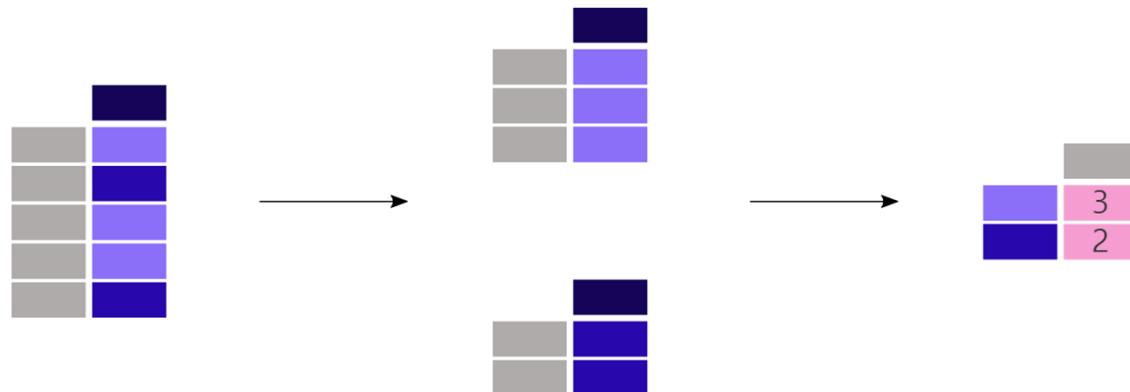
Name: Fare, dtype: float64

- Grouping can be done by multiple columns at the same time. Provide the column names as a list to the `groupby()` method.

#### Note

Both `size` and `count` can be used in combination with `groupby`. Whereas `size` includes `NaN` values and just provides the number of rows (size of the table), `count` excludes the missing values. In the `value_counts` method, use the `dropna` argument to include or exclude the `NaN` values.

### 3) Count number of records by category



```
titanic["Pclass"].value_counts()
```

Pclass	Count
3	491
1	216
2	184

Name: Pclass, dtype: int64

```
titanic.groupby("Pclass")["Pclass"].count()
```

Pclass	Count
1	216
2	184
3	491

Name: Pclass, dtype: int64

*The number of passengers in each of the cabin classes*

*The function is a shortcut, as it is actually a `groupby` operation in combination with counting of the number of records within each group:*

## ➤ How to reshape the layout of tables?

```
titanic = pd.read_csv("data/titanic.csv")
air_quality = pd.read_csv(
    "data/air_quality_long.csv", index_col="date.utc", parse_dates=True
)
```

### 1) Sort table rows

- Sort the Titanic data according to the age of the passengers.

```
titanic.sort_values(by="Age").head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
803	804	1	3	Thomas, Master. Assad Alexander	male	0.42	0	1	2625	8.5167	NaN	C
755	756	1	2	Hamalainen, Master. Viljo	male	0.67	1	1	250649	14.5000	NaN	S
644	645	1	3	Baclini, Miss. Eugenie	female	0.75	2	1	2666	19.2583	NaN	C
469	470	1	3	Baclini, Miss. Helene Barbara	female	0.75	2	1	2666	19.2583	NaN	C
78	79	1	2	Caldwell, Master. Alden Gates	male	0.83	0	2	248738	29.0000	NaN	S

- Sort the Titanic data according to the cabin class and age in descending order

```
titanic.sort_values(by=['Pclass', 'Age'], ascending=False).head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
851	852	0	3	Svensson, Mr. Johan	male	74.0	0	0	347060	7.7750	NaN	S
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	370369	7.7500	NaN	Q
280	281	0	3	Duane, Mr. Frank	male	65.0	0	0	336439	7.7500	NaN	Q
483	484	1	3	Turkula, Mrs. (Hedwig)	female	63.0	0	0	4134	9.5875	NaN	S
326	327	0	3	Nysveen, Mr. Johan Hansen	male	61.0	0	0	345364	6.2375	NaN	S

## ➤ How to reshape the layout of tables?

### 2) Long to wide table format

- Let's use a small subset of the air quality data set. We focus on NO2 data and only use the first two measurements of each location (i.e. the head of each group). The subset of data will be called no2\_subset

```
# filter for no2 data only
no2 = air_quality[air_quality["parameter"] == "no2"]

# use 2 measurements (head) for each location (groupby)
no2_subset = no2.sort_index().groupby(["location"]).head(2)
no2_subset
```

	city	country	location	parameter	value	unit
date.utc						
2019-04-09 01:00:00+00:00	Antwerpen	BE	BETR801	no2	22.5	µg/m³
2019-04-09 01:00:00+00:00	Paris	FR	FR04014	no2	24.4	µg/m³
2019-04-09 02:00:00+00:00	London	GB	London Westminster	no2	67.0	µg/m³
2019-04-09 02:00:00+00:00	Antwerpen	BE	BETR801	no2	53.5	µg/m³
2019-04-09 02:00:00+00:00	Paris	FR	FR04014	no2	27.4	µg/m³
2019-04-09 03:00:00+00:00	London	GB	London Westminster	no2	67.0	µg/m³

```
# filter for no2 data only
no2 = air_quality[air_quality["parameter"] == "no2"]
no2_subset0 = no2.sort_index().groupby(["location"]).head(1)
no2_subset0
```

	city	country	location	parameter	value	unit
	date.utc					
2019-04-09 01:00:00+00:00	Antwerpen	BE	BETR801	no2	22.5	µg/m³
2019-04-09 01:00:00+00:00	Paris	FR	FR04014	no2	24.4	µg/m³
2019-04-09 02:00:00+00:00	London	GB	London Westminster	no2	67.0	µg/m³

### ➤ How to reshape the layout of tables?

#### 2) Long to wide table format



- the values for the three stations as separate columns next to each other

```
no2_subset.pivot(columns="location", values="value")
```

location	BETR801	FR04014	London Westminster
date.utc			
2019-04-09 01:00:00+00:00	22.5	24.4	Nan
2019-04-09 02:00:00+00:00	53.5	27.4	67.0
2019-04-09 03:00:00+00:00	Nan	Nan	67.0

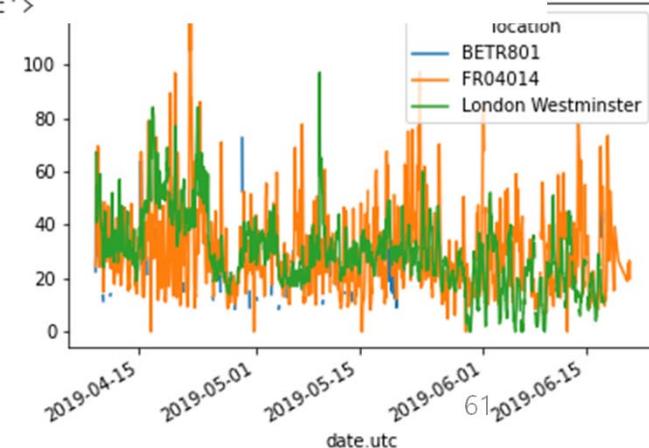
The `pivot()` function is purely reshaping of the data: a single value for each index/column combination is required.

```
no2.head()
```

city	country	location	parameter	value	unit
date.utc					
2019-06-21 00:00:00+00:00	Paris	FR	FR04014	no2	20.0 µg/m³
2019-06-20 23:00:00+00:00	Paris	FR	FR04014	no2	21.8 µg/m³
2019-06-20 22:00:00+00:00	Paris	FR	FR04014	no2	26.5 µg/m³
2019-06-20 21:00:00+00:00	Paris	FR	FR04014	no2	24.9 µg/m³
2019-06-20 20:00:00+00:00	Paris	FR	FR04014	no2	21.4 µg/m³

```
no2.pivot(columns="location", values="value").plot()
```

<AxesSubplot:xlabel='date.utc'>



## Reshaping by pivoting DataFrame objects¶

### Pivot

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

```
df.pivot(index='foo',
         columns='bar',
         values='baz')
```



bar	A	B	C
foo			
one	1	2	3
two	4	5	6

## Reshaping by stacking and unstacking

### stack

		A	B
first	second		
bar	one	1	2
	two	3	4
baz	one	5	6
	two	7	8

```
stacked = df2.stack()
```



first	second		
bar	one	A	1
	two	B	2
baz	one	A	3
	two	B	4
bar	one	A	5
	two	B	6
baz	one	A	7
	two	B	8

MultIndex

Closely related to the `pivot()` method are the related `stack()` and `unstack()` methods available on `Series` and `DataFrame`. These methods are designed to work together with `MultiIndex` objects (see the section on hierarchical indexing). Here are essentially what these methods do:

- `stack`: “pivot” a level of the (possibly hierarchical) column labels, returning a `DataFrame` with an index with a new inner-most level of row labels.
- `unstack`: (inverse operation of `stack`) “pivot” a level of the (possibly hierarchical) row index to the column axis, producing a reshaped `DataFrame` with a new inner-most level of column labels.

### unstack

first	second		
bar	one	A	1
	two	B	2
baz	one	A	3
	two	B	4
bar	one	A	5
	two	B	6
baz	one	A	7
	two	B	8

MultIndex

```
stacked.unstack()
```

		A	B
first	second		
bar	one	1	2
	two	3	4
baz	one	5	6
	two	7	8

MultIndex

## Reshaping by stacking and unstacking

### unstack(1)

stacked			
first	second		
bar	one	A	1
		B	2
baz	two	A	3
		B	4
baz	one	A	5
		B	6
baz	two	A	7
		B	8

MultIndex

stacked.unstack(1)  
or  
stacked.unstack('second')

	second	one	two
first			
bar	A	1	3
	B	2	4
baz	A	5	7
	B	6	8

MultIndex

### unstack(0)

stacked			
first	second		
bar	one	A	1
		B	2
baz	two	A	3
		B	4
baz	one	A	5
		B	6
baz	two	A	7
		B	8

MultIndex

stacked.unstack(0)  
or  
stacked.unstack('first')

	first	bar	baz
second			
one	A	1	5
	B	2	6
two	A	3	7
	B	4	8

MultIndex

## Reshaping by melt

df3

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

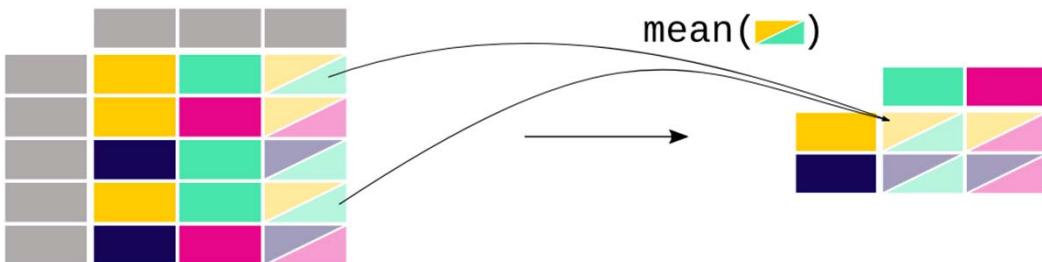
df3.melt(id\_vars=['first', 'last'])

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

The top-level `melt()` function and the corresponding `DataFrame.melt()` are useful to massage a `DataFrame` into a format where one or more columns are *identifier variables*, while all other columns, considered *measured variables*, are “unpivoted” to the row axis, leaving just two non-identifier columns, “variable” and “value”. The names of those columns can be customized by supplying the `var_name` and `value_name` parameters.

## ➤ How to reshape the layout of tables?

### 3) Pivot table



- Mean concentrations for NO<sub>2</sub> and PM<sub>2.5</sub> in each of the stations in table form

```
air_quality.pivot_table(
    values="value", index="location", columns="parameter", aggfunc="mean"
)
```

parameter	no2	pm25
location		
BETR801	26.950920	23.169492
FR04014	29.374284	NaN
London Westminster	29.740050	13.443568

- In the case of `pivot()`, the data is only rearranged.
- When multiple values need to be aggregated (in this specific case, the values on different time steps) `pivot_table()` can be used, providing an aggregation function (e.g. mean) on how to combine these values.

- Pivot table is a well known concept in spreadsheet software. When interested in summary columns for each variable separately as well, put the margin parameter to True:

	parameter	no2	pm25	All
	location			
BETR801	26.950920	23.169492	24.982353	
FR04014	29.374284		NaN	29.374284
London Westminster	29.740050	13.443568	21.491708	
All	29.430316	14.386849	24.222743	

#### Note

In case you are wondering, `pivot_table()` is indeed directly linked to `groupby()`. The same result can be derived by grouping on both `parameter` and `location`:

```
air_quality.groupby(["parameter", "location"]).mean()
```

## ➤ How to reshape the layout of tables?

### 4) Wide to long format

```
no2_pivoted = no2.pivot(columns="location", values="value").reset_index()
no2_pivoted.head()
```

location	date.utc	BETR801	FR04014	London Westminster
0	2019-04-09 01:00:00+00:00	22.5	24.4	NaN
1	2019-04-09 02:00:00+00:00	53.5	27.4	67.0
2	2019-04-09 03:00:00+00:00	54.5	34.2	67.0
3	2019-04-09 04:00:00+00:00	34.5	48.5	41.0
4	2019-04-09 05:00:00+00:00	46.5	59.5	41.0



- Collect all air quality NO<sub>2</sub> measurements in a single column (long format)

```
no_2 = no2_pivoted.melt(id_vars="date.utc")
no_2.head()
```

	date.utc	location	value
0	2019-04-09 01:00:00+00:00	BETR801	22.5
1	2019-04-09 02:00:00+00:00	BETR801	53.5
2	2019-04-09 03:00:00+00:00	BETR801	54.5
3	2019-04-09 04:00:00+00:00	BETR801	34.5
4	2019-04-09 05:00:00+00:00	BETR801	46.5

The `pandas.melt()` method on a `DataFrame` converts the data table from wide format to long format. The column headers become the variable names in a newly created column.

## ➤ How to reshape the layout of tables?

### 4) Wide to long format

- The `pandas.melt()` method can be defined in more detail:

```
no_2 = no2_pivoted.melt(  
    id_vars="date.utc",  
    value_vars=["BETR801", "FR04014", "London Westminster"],  
    value_name="NO_2",  
    var_name="id_location",  
)
```

	date.utc	id_location	NO_2
0	2019-04-09 01:00:00+00:00	BETR801	22.5
1	2019-04-09 02:00:00+00:00	BETR801	53.5
2	2019-04-09 03:00:00+00:00	BETR801	54.5
3	2019-04-09 04:00:00+00:00	BETR801	34.5
4	2019-04-09 05:00:00+00:00	BETR801	46.5
...	...	...	...
5110	2019-06-20 20:00:00+00:00	London Westminster	NaN
5111	2019-06-20 21:00:00+00:00	London Westminster	NaN
5112	2019-06-20 22:00:00+00:00	London Westminster	NaN
5113	2019-06-20 23:00:00+00:00	London Westminster	NaN
5114	2019-06-21 00:00:00+00:00	London Westminster	NaN

5115 rows × 3 columns

The result is the same, but in more detail defined:

- `value_vars` defines explicitly which columns to *melt* together
- `value_name` provides a custom column name for the values column instead of the default column name `value`
- `var_name` provides a custom column name for the column collecting the column header names. Otherwise it takes the index name or a default `variable`

Hence, the arguments `value_name` and `var_name` are just user-defined names for the two generated columns. The columns to melt are defined by `id_vars` and `value_vars`.

## ➤ How to combine data from multiple tables?

Used data

### Air quality Nitrate data

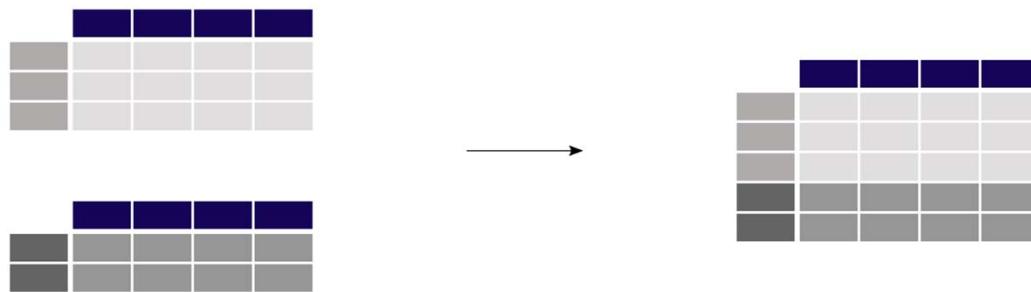
```
In [2]: air_quality_no2 = pd.read_csv("data/air_quality_no2_long.csv",
...                                     parse_dates=True)
...
...
In [3]: air_quality_no2 = air_quality_no2[["date.utc", "location",
...                                         "parameter", "value"]]
...
In [4]: air_quality_no2.head()
Out[4]:
      date.utc location parameter  value
0  2019-06-21 00:00:00+00:00  FR04014       no2    20.0
1  2019-06-20 23:00:00+00:00  FR04014       no2    21.8
2  2019-06-20 22:00:00+00:00  FR04014       no2    26.5
3  2019-06-20 21:00:00+00:00  FR04014       no2    24.9
4  2019-06-20 20:00:00+00:00  FR04014       no2    21.4
```

### Air quality Particulate matter data

```
In [5]: air_quality_pm25 = pd.read_csv("data/air_quality_pm25_long.csv",
...                                       parse_dates=True)
...
...
In [6]: air_quality_pm25 = air_quality_pm25[["date.utc", "location",
...                                         "parameter", "value"]]
...
In [7]: air_quality_pm25.head()
Out[7]:
      date.utc location parameter  value
0  2019-06-18 06:00:00+00:00  BETR801       pm25   18.0
1  2019-06-17 08:00:00+00:00  BETR801       pm25    6.5
2  2019-06-17 07:00:00+00:00  BETR801       pm25   18.5
3  2019-06-17 06:00:00+00:00  BETR801       pm25   16.0
4  2019-06-17 05:00:00+00:00  BETR801       pm25    7.5
```

## ➤ How to combine data from multiple tables?

### 1) Concatenating objects



- Combine the measurements of NO<sub>2</sub> and PM<sub>25</sub>, two tables with a similar structure, in a single table

```
In [8]: air_quality = pd.concat([air_quality_pm25, air_quality_no2], axis=0)
```

```
In [9]: air_quality.head()
```

```
Out[9]:
      date.utc location parameter  value
0  2019-06-18 06:00:00+00:00  BETR801    pm25  18.0
1  2019-06-17 08:00:00+00:00  BETR801    pm25   6.5
2  2019-06-17 07:00:00+00:00  BETR801    pm25  18.5
3  2019-06-17 06:00:00+00:00  BETR801    pm25  16.0
4  2019-06-17 05:00:00+00:00  BETR801    pm25   7.5
```

- The `concat()` function performs concatenation operations of multiple tables along one of the axis (row-wise or column-wise).

- By default concatenation is along axis 0, so the resulting table combines the rows of the input tables. Let's check the shape of the original and the concatenated tables to verify the operation:

```
In [10]: print('Shape of the ``air_quality_pm25`` table: ', air_quality_pm25.shape)
Shape of the ``air_quality_pm25`` table: (1110, 4)
```

```
In [11]: print('Shape of the ``air_quality_no2`` table: ', air_quality_no2.shape)
Shape of the ``air_quality_no2`` table: (2068, 4)
```

```
In [12]: print('Shape of the resulting ``air_quality`` table: ', air_quality.shape)
Shape of the resulting ``air_quality`` table: (3178, 4)
```

- Hence, the resulting table has  $3178 = 1110 + 2068$  rows.

air_quality				
	date.utc	location	parameter	value
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5
...	...	...	...	...
2063	2019-05-07 06:00:00+00:00	London Westminster	no2	26.0
2064	2019-05-07 04:00:00+00:00	London Westminster	no2	16.0
2065	2019-05-07 03:00:00+00:00	London Westminster	no2	19.0
2066	2019-05-07 02:00:00+00:00	London Westminster	no2	19.0
2067	2019-05-07 01:00:00+00:00	London Westminster	no2	23.0

3178 rows × 4 columns

## ➤ How to combine data from multiple tables?

### 1) Concatenating objects

- Sorting the table on the datetime information illustrates also the combination of both tables, with the `parameter` column defining the origin of the table (either `no2` from table `air_quality_no2` or `pm25` from table `air_quality_pm25`):

```
In [13]: air_quality = air_quality.sort_values("date.utc")
```

```
In [14]: air_quality.head()
```

```
Out[14]:
```

		date.utc	location	parameter	value
2067	2019-05-07 01:00:00+00:00	London Westminster		no2	23.0
1003	2019-05-07 01:00:00+00:00		FR04014	no2	25.0
100	2019-05-07 01:00:00+00:00		BETR801	pm25	12.5
1098	2019-05-07 01:00:00+00:00		BETR801	no2	50.5
1109	2019-05-07 01:00:00+00:00	London Westminster		pm25	8.0

#### Note

The existence of multiple row/column indices at the same time has not been mentioned within these tutorials. *Hierarchical indexing* or *MultIndex* is an advanced and powerful pandas feature to analyze higher dimensional data.

Multi-indexing is out of scope for this pandas introduction. For the moment, remember that the function `reset_index` can be used to convert any level of an index to a column, e.g. `air_quality.reset_index(level=0)`

- In this specific example, the `parameter` column provided by the data ensures that each of the original tables can be identified. This is not always the case. the `concat` function provides a convenient solution with the `keys` argument, adding an additional (hierarchical) row index. For example:

```
air_quality_ = pd.concat([air_quality_pm25, air_quality_no2], keys=["PM25", "NO2"])
air_quality_
```

			date.utc	location	parameter	value
PM25	0	2019-06-18 06:00:00+00:00		BETR801	pm25	18.0
	1	2019-06-17 08:00:00+00:00		BETR801	pm25	6.5
	2	2019-06-17 07:00:00+00:00		BETR801	pm25	18.5
	3	2019-06-17 06:00:00+00:00		BETR801	pm25	16.0
	4	2019-06-17 05:00:00+00:00		BETR801	pm25	7.5
...						
NO2	2063	2019-05-07 06:00:00+00:00	London Westminster		no2	26.0
	2064	2019-05-07 04:00:00+00:00	London Westminster		no2	16.0
	2065	2019-05-07 03:00:00+00:00	London Westminster		no2	19.0
	2066	2019-05-07 02:00:00+00:00	London Westminster		no2	19.0
	2067	2019-05-07 01:00:00+00:00	London Westminster		no2	23.0

3178 rows × 4 columns

## ➤ How to combine data from multiple tables?

### 2) Join tables using a common identifier



- Add the station coordinates, provided by the stations metadata table, to the corresponding rows in the measurements table.

#### ⚠ Warning

The air quality measurement station coordinates are stored in a data file `air_quality_stations.csv`, downloaded using the `py-openaq` package.

```
In [17]: stations_coord = pd.read_csv("data/air_quality_stations.csv")
```

```
In [18]: stations_coord.head()
```

```
Out[18]:
```

	location	coordinates.latitude	coordinates.longitude
0	BELAL01	51.23619	4.38522
1	BELHB23	51.17030	4.34100
2	BELLD01	51.10998	5.00486
3	BELLD02	51.12038	5.02155
4	BELR833	51.32766	4.36226

#### i Note

The stations used in this example (FR04014, BETR801 and London Westminster) are just three entries enlisted in the metadata table. We only want to add the coordinates of these three to the measurements table, each on the corresponding rows of the `air_quality` table.

```
In [19]: air_quality.head()
```

```
Out[19]:
```

	date.utc	location	parameter	value
2067	2019-05-07 01:00:00+00:00	London	Westminster	no2 23.0
1003	2019-05-07 01:00:00+00:00		FR04014	no2 25.0
100	2019-05-07 01:00:00+00:00		BETR801	pm25 12.5
1098	2019-05-07 01:00:00+00:00		BETR801	no2 50.5
1109	2019-05-07 01:00:00+00:00	London	Westminster	pm25 8.0

## ➤ How to combine data from multiple tables?

### 2) Join tables using a common identifier

```
In [20]: air_quality = pd.merge(air_quality, stations_coord, how="left", on="location")
```

```
In [21]: air_quality.head()
```

```
Out[21]:
```

	date.utc	location	parameter	value	coordinates.latitude	coordinates.longitude
0	2019-05-07 01:00:00+00:00	London Westminster	no2	23.0	51.49467	
1	2019-05-07 01:00:00+00:00		FR04014	no2	48.83724	
2	2019-05-07 01:00:00+00:00		FR04014	no2	48.83722	
3	2019-05-07 01:00:00+00:00		BETR801	pm25	51.20966	
4	2019-05-07 01:00:00+00:00		BETR801	no2	51.20966	

Using the `merge()` function, for each of the rows in the `air_quality` table, the corresponding coordinates are added from the `air_quality_stations_coord` table.

Both tables have the column `location` in common which is used as a key to combine the information. By choosing the `left` join, only the locations available in the `air_quality` (left) table, i.e. FR04014, BETR801 and London Westminster, end up in the resulting table. The `merge` function supports multiple join options similar to database-style operations.

```
In [18]: stations_coord.head()
```

```
Out[18]:
```

	location	coordinates.latitude	coordinates.longitude
0	BELAL01	51.23619	4.38522
1	BELHB23	51.17030	4.34100
2	BELLD01	51.10998	5.00486
3	BELLD02	51.12038	5.02155
4	BELR833	51.32766	4.36226

```
In [19]: air_quality.head()
```

```
Out[19]:
```

	date.utc	location	parameter	value	
2067	2019-05-07 01:00:00+00:00	London Westminster	no2	23.0	
1003	2019-05-07 01:00:00+00:00		FR04014	no2	25.0
100	2019-05-07 01:00:00+00:00		BETR801	pm25	12.5
1098	2019-05-07 01:00:00+00:00		BETR801	no2	50.5
1109	2019-05-07 01:00:00+00:00	London Westminster	pm25	8.0	

## ➤ How to combine data from multiple tables?

### 2) Join tables using a common identifier

- Add the parameter `full description and name`, provided by the parameters metadata table, to the measurements table

```
In [22]: air_quality_parameters = pd.read_csv("data/air_quality_parameters.csv")
```

```
In [23]: air_quality_parameters.head()
```

```
Out[23]:
      id           description   name
0    bc        Black Carbon     BC
1    co  Carbon Monoxide     CO
2   no2  Nitrogen Dioxide    NO2
3    o3        Ozone            O3
4  pm10 Particulate matter less than 10 micrometers in...  PM10
```

```
air_quality = pd.merge(air_quality, air_quality_parameters,
                      how='left', left_on='parameter', right_on='id')
air_quality.head()
```

#### ⚠ Warning

The air quality parameters metadata are stored in a data file `air_quality_parameters.csv`, downloaded using the `py-openaq` package.

*Compared to the previous example, there is no common column name. However, the `parameter` column in the `air_quality` table and the `id` column in the `air_quality_parameters_name` both provide the measured variable in a common format. The `left_on` and `right_on` arguments are used here (instead of just `on`) to make the link between the two tables.*

	date.utc	location	parameter	value	coordinates.latitude	coordinates.longitude	id	description	name
0	2019-05-07 01:00:00+00:00	London Westminster	no2	23.0	51.49467	-0.13193	no2	Nitrogen Dioxide	NO2
1	2019-05-07 01:00:00+00:00	FR04014	no2	25.0	48.83724	2.39390	no2	Nitrogen Dioxide	NO2
2	2019-05-07 01:00:00+00:00	FR04014	no2	25.0	48.83722	2.39390	no2	Nitrogen Dioxide	NO2
3	2019-05-07 01:00:00+00:00	BETR801	pm25	12.5	51.20966	4.43182	pm25	Particulate matter less than 2.5 micrometers i...	PM2.5
4	2019-05-07 01:00:00+00:00	BETR801	no2	50.5	51.20966	4.43182	no2	Nitrogen Dioxide	NO2

## ➤ How to handle time series data with ease?

### Air quality data

For this tutorial, air quality data about  $NO_2$  and Particulate matter less than 2.5 micrometers is used, made available by `openaq` and downloaded using the `py-openaq` package. The `air_quality_no2_long.csv` data set provides  $NO_2$  values for the measurement stations *FR04014*, *BETR801* and *London Westminster* in respectively Paris, Antwerp and London.

```
In [3]: air_quality = pd.read_csv("data/air_quality_no2_long.csv")
In [4]: air_quality = air_quality.rename(columns={"date.utc": "datetime"})
In [5]: air_quality.head()
Out[5]:
   city country        datetime location parameter  value    unit
0  Paris     FR 2019-06-21 00:00:00+00:00  FR04014    no2  20.0  µg/m³
1  Paris     FR 2019-06-20 23:00:00+00:00  FR04014    no2  21.8  µg/m³
2  Paris     FR 2019-06-20 22:00:00+00:00  FR04014    no2  26.5  µg/m³
3  Paris     FR 2019-06-20 21:00:00+00:00  FR04014    no2  24.9  µg/m³
4  Paris     FR 2019-06-20 20:00:00+00:00  FR04014    no2  21.4  µg/m³
```

```
In [6]: air_quality.city.unique()
Out[6]: array(['Paris', 'Antwerpen', 'London'], dtype=object)
```

### 1) Using pandas datetime properties

- Work with the dates in the column `datetime` as `datetime` objects instead of plain text

```
In [7]: air_quality["datetime"] = pd.to_datetime(air_quality["datetime"])
```

```
In [8]: air_quality["datetime"]
```

```
Out[8]:
```

0	2019-06-21 00:00:00+00:00
1	2019-06-20 23:00:00+00:00
2	2019-06-20 22:00:00+00:00
3	2019-06-20 21:00:00+00:00
4	2019-06-20 20:00:00+00:00
	...
2063	2019-05-07 06:00:00+00:00
2064	2019-05-07 04:00:00+00:00
2065	2019-05-07 03:00:00+00:00
2066	2019-05-07 02:00:00+00:00
2067	2019-05-07 01:00:00+00:00

```
Name: datetime, Length: 2068, dtype: datetime64[ns, UTC]
```

Initially, the values in `datetime` are character strings and do not provide any `datetime` operations (e.g. extract the year, day of the week,...). By applying the `to_datetime` function, pandas interprets the strings and convert these to `datetime` (i.e. `datetime64[ns, UTC]`) objects. In pandas we call these `datetime` objects similar to `datetime.datetime` from the standard library as `pandas.Timestamp`.

## ➤ How to handle time series data with ease?

### 1) Using pandas datetime properties

- Add a new column to the DataFrame containing only the month of the measurement

```
air_quality["month"] = air_quality["datetime"].dt.month
air_quality.head()
```

	city	country	datetime	location	parameter	value	unit	month
0	Paris	FR	2019-06-21 00:00:00+00:00	FR04014	no2	20.0	µg/m³	6
1	Paris	FR	2019-06-20 23:00:00+00:00	FR04014	no2	21.8	µg/m³	6
2	Paris	FR	2019-06-20 22:00:00+00:00	FR04014	no2	26.5	µg/m³	6
3	Paris	FR	2019-06-20 21:00:00+00:00	FR04014	no2	24.9	µg/m³	6
4	Paris	FR	2019-06-20 20:00:00+00:00	FR04014	no2	21.4	µg/m³	6

By using `Timestamp` objects for dates, a lot of time-related properties are provided by pandas. For example the `month`, but also `year`, `weekofyear`, `quarter`, ... All of these properties are accessible by the `dt` accessor.

- Average NO<sub>2</sub> concentration for each day of the week for each of the measurement locations?

```
air_quality.groupby(
    [air_quality["datetime"].dt.weekday, "location"])["value"].mean()
```

	datetime	location	value
0	BETR801		27.875000
	FR04014		24.856250
	London Westminster		23.969697
1	BETR801		22.214286
	FR04014		30.999359
	London Westminster		24.885714
2	BETR801		21.125000
	FR04014		29.165753
	London Westminster		23.460432
3	BETR801		27.500000
	FR04014		28.600690
	London Westminster		24.780142
4	BETR801		28.400000
	FR04014		31.617986
	London Westminster		26.446809
5	BETR801		33.500000
	FR04014		25.266154
	London Westminster		24.977612
6	BETR801		21.896552
	FR04014		23.274306
	London Westminster		24.859155
		Name: value, dtype: float64	

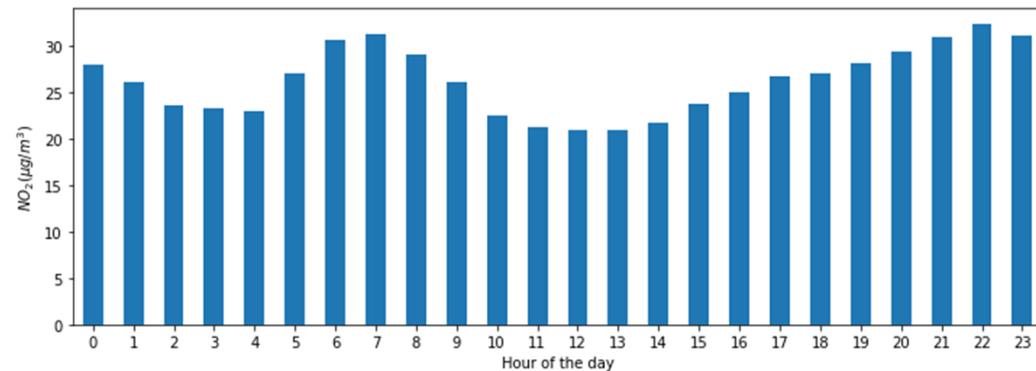
Here, we want to calculate a given statistic (e.g. mean NO<sub>2</sub>) for each weekday and for each measurement location. To group on weekdays, we use the `datetime` property `weekday` (with Monday=0 and Sunday=6) of pandas `Timestamp`, which is also accessible by the `dt` accessor. The grouping on both locations and weekdays can be done to split the calculation of the mean on each of these combinations.

## ➤ How to handle time series data with ease?

### 1) Using pandas datetime properties

- Plot the typical NO<sub>2</sub> pattern during the day of our time series of all stations together. In other words, what is the average value for each hour of the day?

```
fig, axs = plt.subplots(figsize=(12, 4))
air_quality.groupby(air_quality["datetime"].dt.hour)[“value”].mean().plot(
    kind='bar', rot=0, ax=axs)
plt.xlabel("Hour of the day"); # custom x label using matplotlib
plt.ylabel("$NO_2 (\mu g/m^3)$");
```



Similar to the previous case, we want to calculate a given statistic (e.g. mean NO<sub>2</sub>) for each hour of the day and we can use the split-apply-combine approach again. For this case, we use the datetime property `hour` of pandas `Timestamp`, which is also accessible by the `dt` accessor.

### 2) Datetime as index

- `pivot()` was introduced to reshape the data table with each of the measurements locations as a separate column:

```
no_2 = air_quality.pivot(index="datetime", columns="location", values="value")
no_2.head()
```

	location	BETR801	FR04014	London Westminster
datetime				
2019-05-07 01:00:00+00:00		50.5	25.0	23.0
2019-05-07 02:00:00+00:00		45.0	27.7	19.0
2019-05-07 03:00:00+00:00		NaN	50.4	19.0
2019-05-07 04:00:00+00:00		NaN	61.9	16.0
2019-05-07 05:00:00+00:00		NaN	72.4	NaN

#### Note

By pivoting the data, the datetime information became the index of the table. In general, setting a column as an index can be achieved by the `set_index` function.

## ➤ How to handle time series data with ease?

### 2) Datetime as index

- Working with a datetime index (i.e. `DatetimeIndex`) provides powerful functionalities. For example, we do not need the `dt` accessor to get the time series properties, but have these properties available on the index directly:

```
no_2.index.year, no_2.index.weekday
```

```
(Int64Index([2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019,
             ...
             2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019],
            dtype='int64', name='datetime', length=1033),
 Int64Index([1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
             ...
             3, 3, 3, 3, 3, 3, 3, 3, 3, 4],
            dtype='int64', name='datetime', length=1033))
```

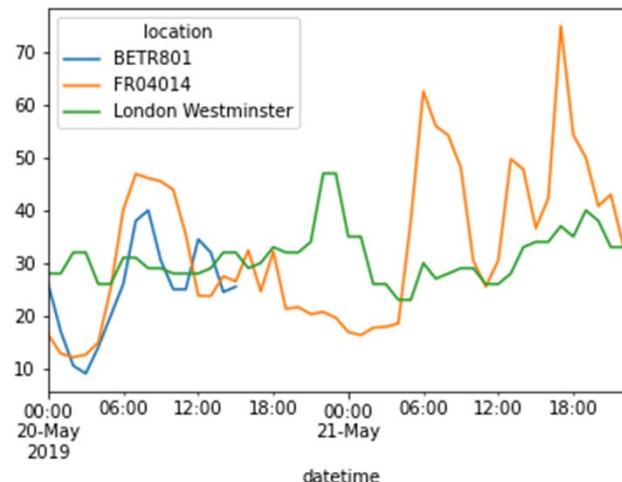
```
no_2 = air_quality.pivot(index="datetime", columns="location", values="value")
no_2.head()
```

location	BETR801	FR04014	London Westminster
datetime			
2019-05-07 01:00:00+00:00	50.5	25.0	23.0
2019-05-07 02:00:00+00:00	45.0	27.7	19.0
2019-05-07 03:00:00+00:00	NaN	50.4	19.0
2019-05-07 04:00:00+00:00	NaN	61.9	16.0
2019-05-07 05:00:00+00:00	NaN	72.4	NaN

- Create a plot of the NO<sub>2</sub> values in the different stations from the 20th of May till the end of 21st of May

```
no_2["2019-05-20":"2019-05-21"].plot()
```

```
<AxesSubplot:xlabel='datetime'>
```



- By providing a string that parses to a datetime, a specific subset of the data can be selected on a `DatetimeIndex`.

## ➤ How to handle time series data with ease?

### 3) Resample a time series to another frequency

- Aggregate the current hourly time series values to the monthly maximum value in each of the stations.

```
monthly_max = no_2.resample("M").max()
monthly_max
```

location	BETR801	FR04014	London Westminster
datetime			
2019-05-31 00:00:00+00:00	74.5	97.0	97.0
2019-06-30 00:00:00+00:00	52.5	84.7	52.0

- A very powerful method on time series data with a datetime index, is the ability to **resample()** time series to another frequency (e.g., converting secondly data into 5-minutely data).

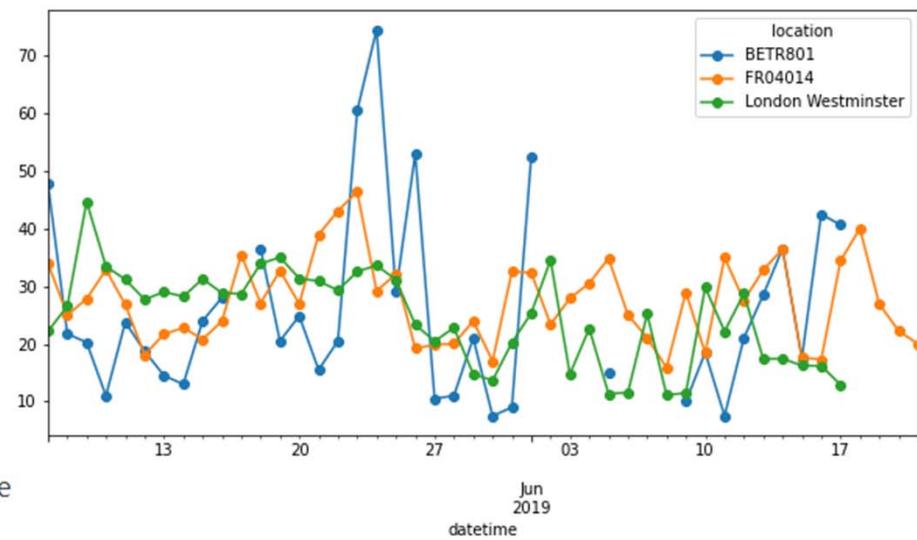
The **resample()** method is similar to a groupby operation:

- it provides a time-based grouping, by using a string (e.g. `M`, `5H`,...) that defines the target frequency
- it requires an aggregation function such as `mean`, `max`,...

- Make a plot of the daily mean NO<sub>2</sub> value in each of the stations.

```
no_2.resample("D").mean().plot(style="-o", figsize=(10, 5))
```

```
<AxesSubplot:xlabel='datetime'>
```



## ➤ How to manipulate textual data?

- Make all name characters lowercase.

```
titanic = pd.read_csv("data/titanic.csv")
```

```
titanic["Name"].str.lower()
```

```
0           braund, mr. owen harris
1    cumings, mrs. john bradley (florence briggs th...
2           heikkinen, miss. laina
3    futrelle, mrs. jacques heath (lily may peel)
4           allen, mr. william henry
...
886          montvila, rev. juozas
887           graham, miss. margaret edith
888      johnston, miss. catherine helen "carrie"
889           behr, mr. karl howell
890          dooley, mr. patrick
Name: Name, Length: 891, dtype: object
```

- Create a new column Surname that contains the surname of the passengers by extracting the part before the comma.

```
titanic["Name"].str.split(",")
```

```
0           [Braund, Mr. Owen Harris]
1    [Cumings, Mrs. John Bradley (Florence Briggs ...
2           [Heikkinen, Miss. Laina]
3    [Futrelle, Mrs. Jacques Heath (Lily May Peel)]
4           [Allen, Mr. William Henry]
...
886          [Montvila, Rev. Juozas]
887           [Graham, Miss. Margaret Edith]
888      [Johnston, Miss. Catherine Helen "Carrie"]
889           [Behr, Mr. Karl Howell]
890          [Dooley, Mr. Patrick]
Name: Name, Length: 891, dtype: object
```

Similar to datetime objects in the time series tutorial having a `dt` accessor, a number of specialized string methods are available when using the `str` accessor. These methods have in general matching names with the equivalent built-in string methods for single elements, but are applied element-wise (remember element-wise calculations?) on each of the values of the columns.

## ➤ How to manipulate textual data?

- Using the `Series.str.split()` method, each of the values is returned as a list of 2 elements. The first element is the part before the comma and the second element is the part after the comma.

```
titanic["Surname"] = titanic["Name"].str.split(",").str.get(0)
titanic["Surname"]
```

```
0      Braund
1      Cumings
2     Heikkinen
3      Futrelle
4       Allen
...
886    Montvila
887    Graham
888   Johnston
889     Behr
890    Dooley
Name: Surname, Length: 891, dtype: object
```

As we are only interested in the first part representing the surname (element 0), we can again use the `str` accessor and apply `Series.str.get()` to extract the relevant part. Indeed, these string functions can be concatenated to combine multiple functions at once!

- Extract the passenger data about the countesses on board of the Titanic.

```
titanic["Name"].str.contains("Countess")
```

```
0      False
1      False
2      False
3      False
4      False
...
886    False
887    False
888    False
889    False
890    False
Name: Name, Length: 891, dtype: bool
```

The string method `Series.str.contains()` checks for each of the values in the column `Name` if the string contains the word `Countess` and returns for each of the values `True` (`Countess` is part of the name) or `False` (`Countess` is not part of the name). This output can be used to subselect the data using conditional (boolean) indexing introduced in the subsetting of data tutorial. As there was only one countess on the Titanic, we get one row as a result.

```
titanic[titanic["Name"].str.contains("Countess")]
```

PassengerId	Survived	Pclass	Name	Sex			
759	760	1	1 Rothes, the Countess. of (Lucy Noel Martha Dye...	female			
Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Surname
33.0	0	0	110152	86.5	B77	S	Rothes

## ➤ How to manipulate textual data?

- Which passenger of the Titanic has the longest name?

```
titanic["Name"].str.len()
```

```
0      23
1      51
2      22
3      44
4      24
...
886     21
887     28
888     40
889     21
890     19
Name: Name, Length: 891, dtype: int64
```

```
titanic["Name"].str.len().idxmax()
```

307

```
titanic.loc[titanic["Name"].str.len().idxmax(), "Name"]
```

'Penasco y Castellana, Mrs. Victor de Satode (Maria Josefa Perez de Soto y Vallejo)'

### Note

More powerful extractions on strings are supported, as the `Series.str.contains()` and `Series.str.extract()` methods accept regular expressions, but out of scope of this tutorial.

To get the longest name we first have to get the lengths of each of the names in the `Name` column. By using pandas string methods, the `Series.str.len()` function is applied to each of the names individually (element-wise).

Next, we need to get the corresponding location, preferably the index label, in the table for which the name length is the largest. The `idxmax()` method does exactly that. It is not a string method and is applied to integers, so no str is used.

Based on the index name of the row (307) and the column (`Name`), we can do a selection using the `loc` operator, introduced in the tutorial on subsetting.