

Pandas 라이브러리

inky4832@daum.net

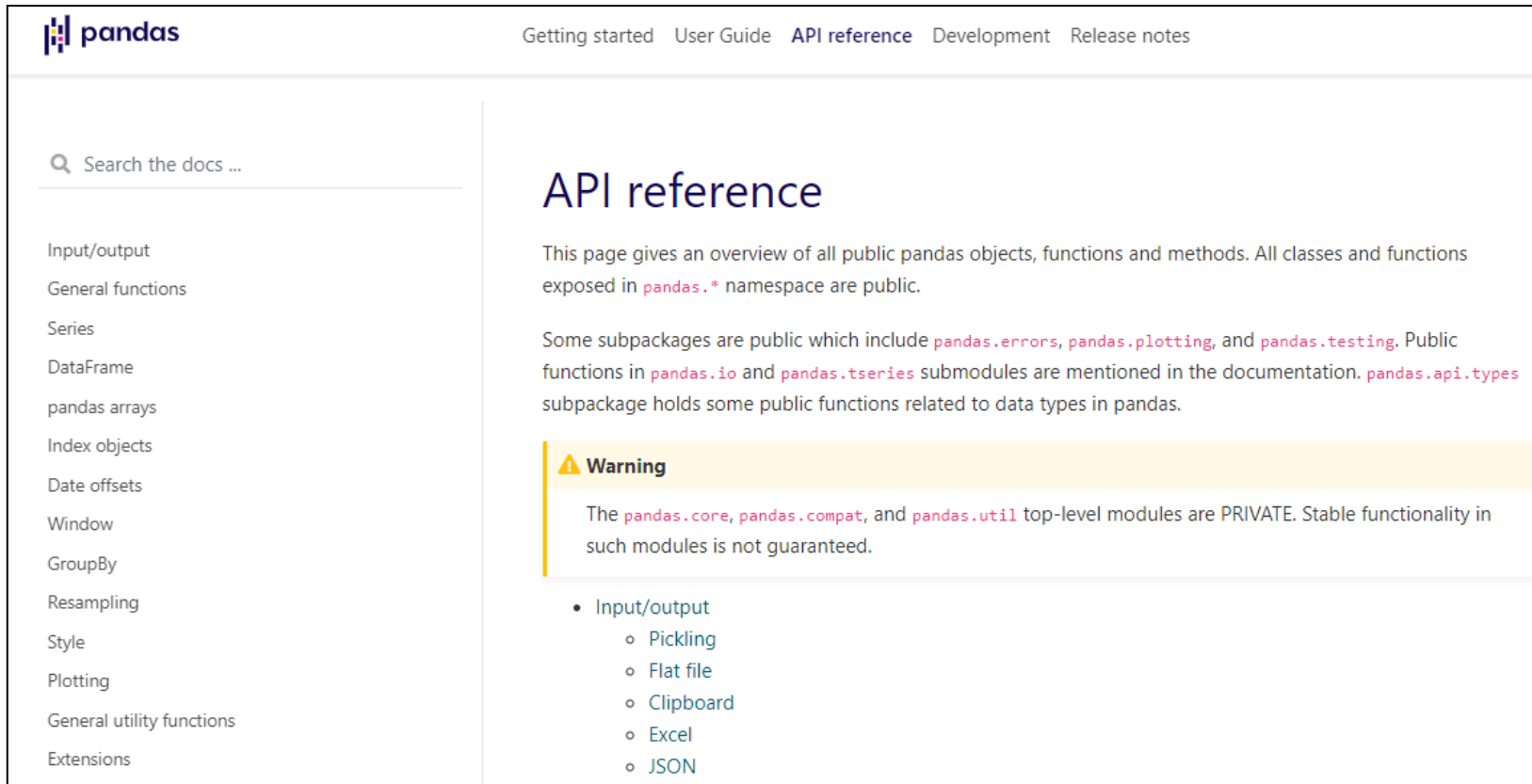
1장. Pandas 개요

Pandas 개요

DataFrame 및 Series

Pandas 다양한 기능 소개

<https://pandas.pydata.org/pandas-docs/stable/index.html>
https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf



The screenshot shows the Pandas API reference page. The header includes the Pandas logo and navigation links: Getting started, User Guide, API reference (selected), Development, and Release notes. A search bar is located on the left. The left sidebar contains a list of topics: Input/output, General functions, Series, DataFrame, pandas arrays, Index objects, Date offsets, Window, GroupBy, Resampling, Style, Plotting, General utility functions, and Extensions. The main content area is titled 'API reference' and contains an overview of public pandas objects, functions, and methods. It mentions that all classes and functions exposed in the `pandas.*` namespace are public. It also lists some subpackages that are public, including `pandas.errors`, `pandas.plotting`, and `pandas.testing`. Public functions in `pandas.io` and `pandas.tseries` submodules are mentioned. A warning box states that the `pandas.core`, `pandas.compat`, and `pandas.util` top-level modules are PRIVATE and their stable functionality is not guaranteed. A list of topics under 'Input/output' is shown: Pickling, Flat file, Clipboard, Excel, and JSON.

API reference

This page gives an overview of all public pandas objects, functions and methods. All classes and functions exposed in `pandas.*` namespace are public.

Some subpackages are public which include `pandas.errors`, `pandas.plotting`, and `pandas.testing`. Public functions in `pandas.io` and `pandas.tseries` submodules are mentioned in the documentation. `pandas.api.types` subpackage holds some public functions related to data types in pandas.

Warning

The `pandas.core`, `pandas.compat`, and `pandas.util` top-level modules are PRIVATE. Stable functionality in such modules is not guaranteed.

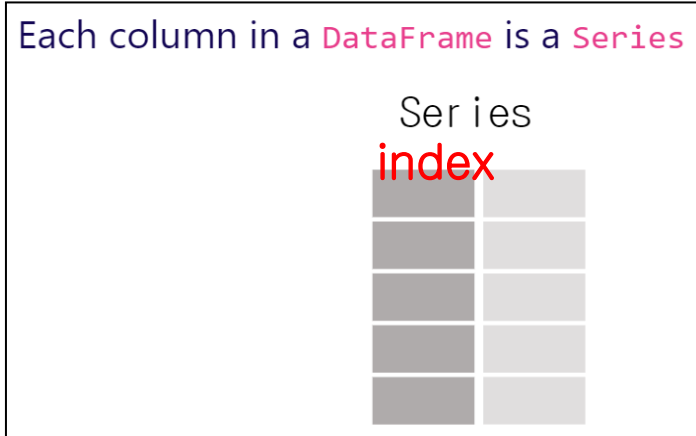
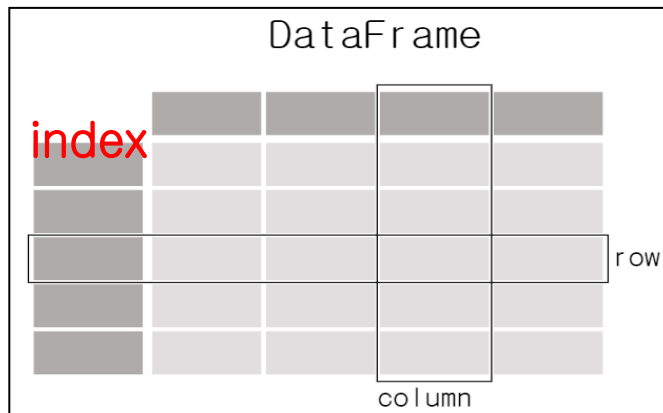
- Input/output
 - Pickling
 - Flat file
 - Clipboard
 - Excel
 - JSON

2. Pandas 데이터 구조

https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/01_table_oriented.html

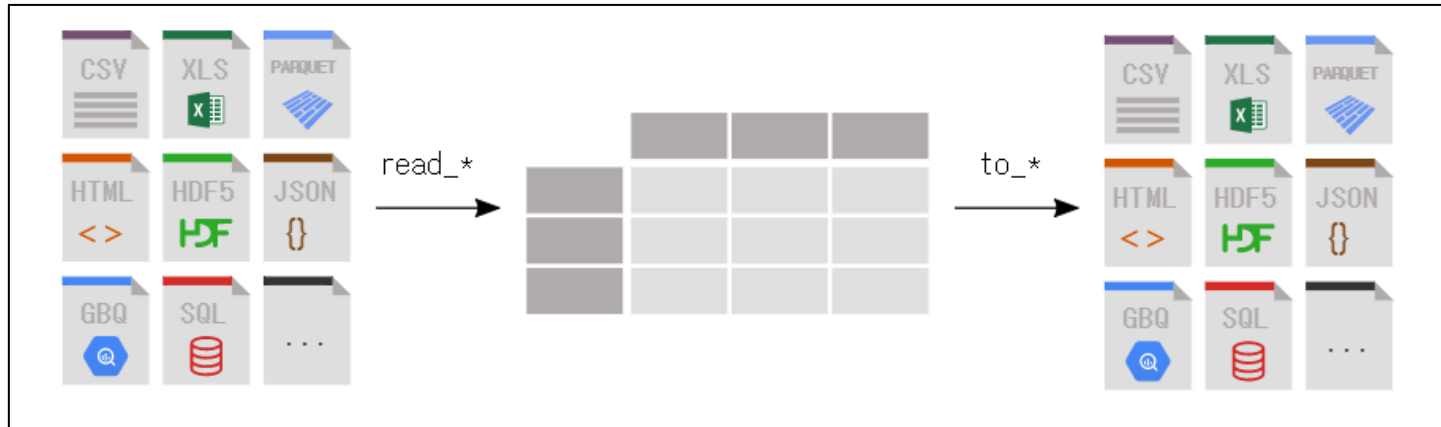
Data structures

Dimensions	Name	Description
1	Series	1D labeled homogeneously-typed array
2	DataFrame	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column



3. Pandas 기능

1) 파일 읽고 쓰기



2) DataFrame의 subset 만들기

https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/03_subset_data.html#how-do-i-select-a-subset-of-a-dataframe

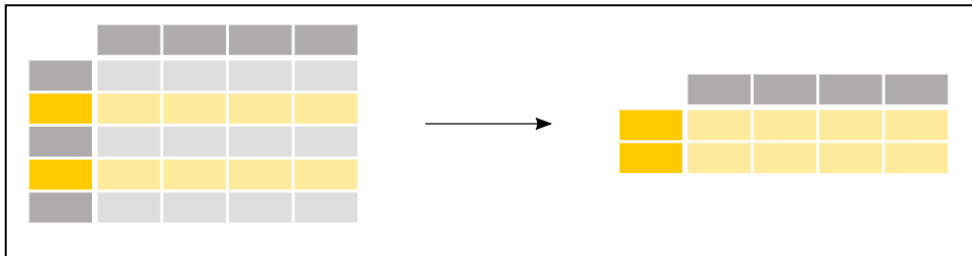
3. Pandas 기능

Pandas

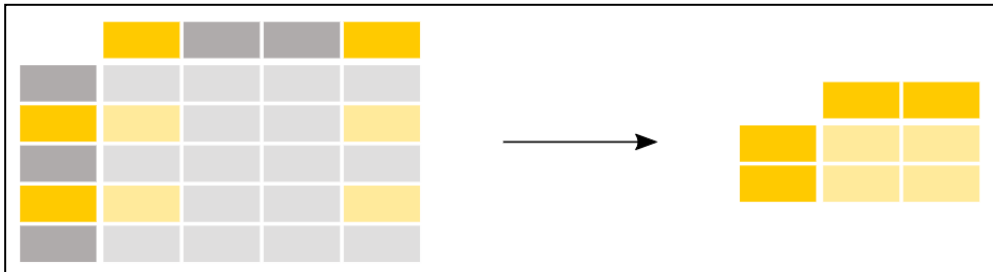
projection



selection

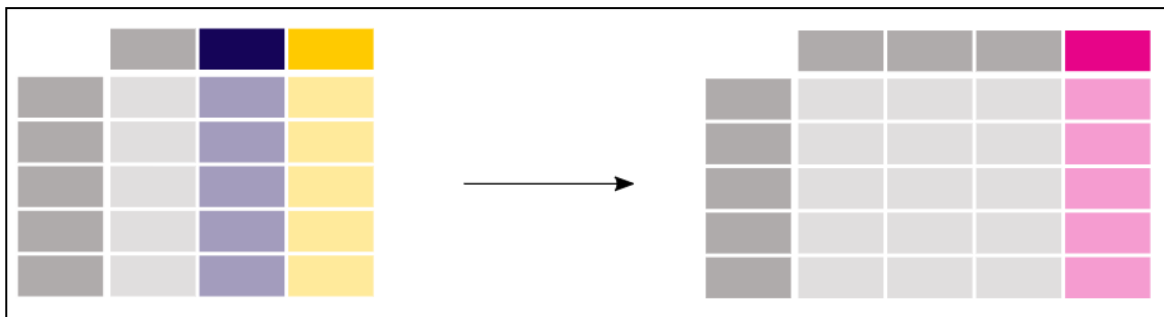
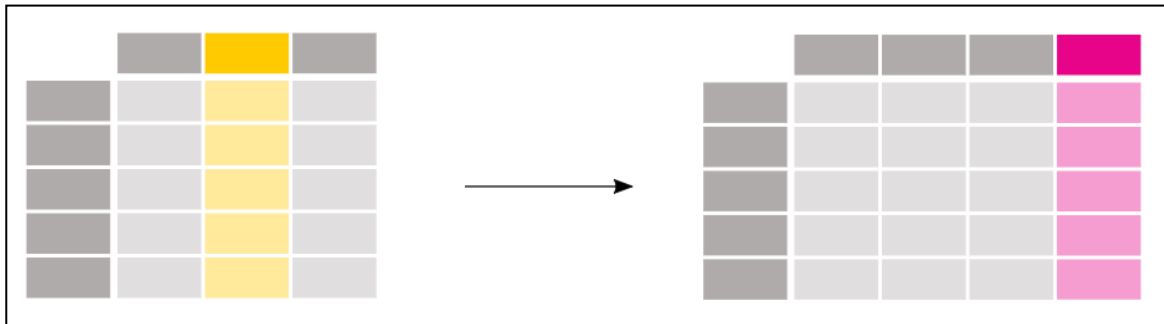


projection + selection



3) DataFrame에 새로운 컬럼 추가

https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/05_add_columns.html#how-to-create-new-columns-derived-from-existing-columns

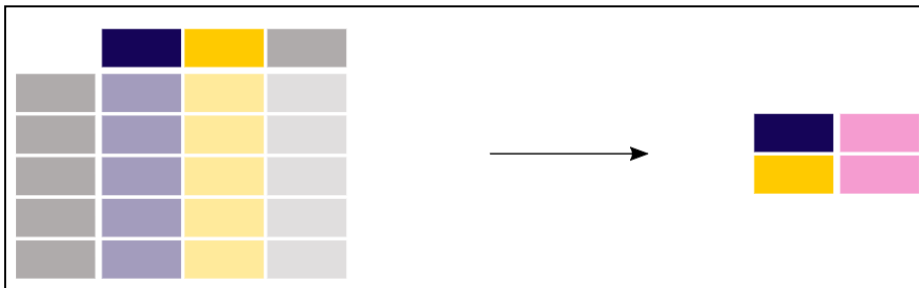


3. Pandas 기능

4) 다양한 통계 데이터 처리

https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/06_calculate_statistics.html#how-to-calculate-summary-statistics

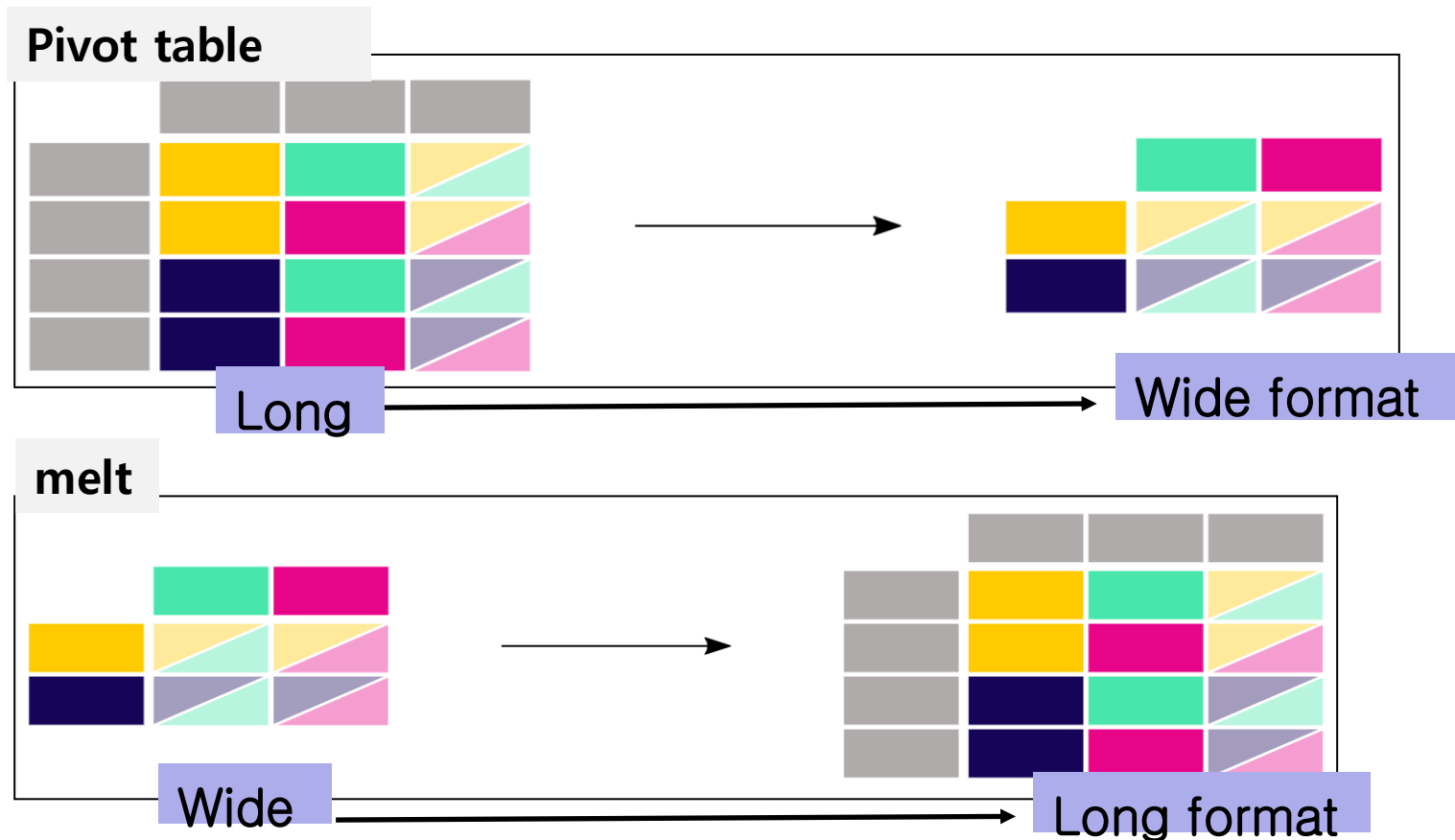
Aggregating statistics



3. Pandas 기능

5) DataFrame의 reshape 처리

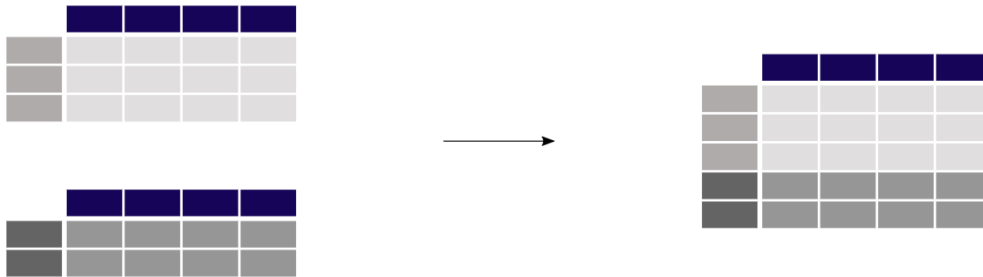
https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/07_reshape_table_layout.html#pivot-table



6) DataFrame의 조인

https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/08_combine_dataframes.html#how-to-combine-data-from-multiple-tables

Concatenating objects



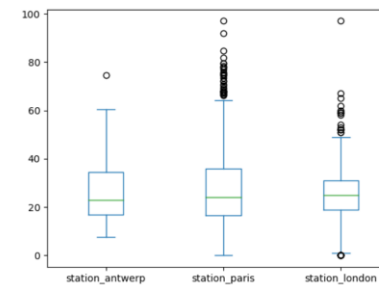
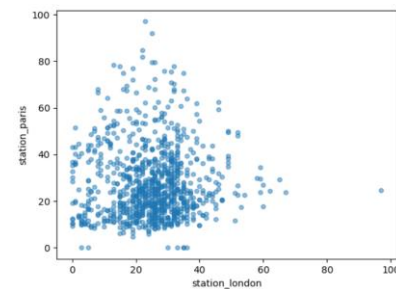
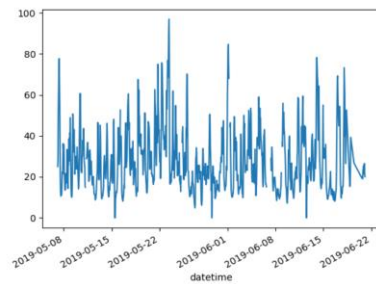
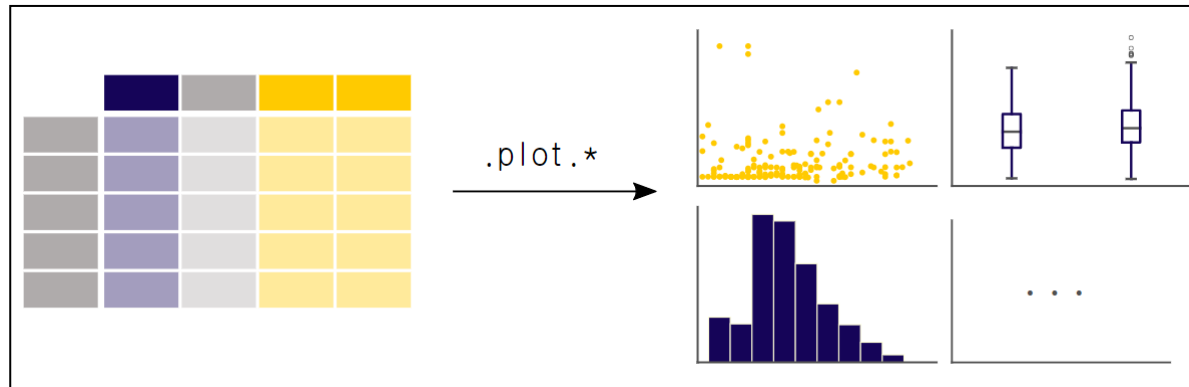
Join tables using a common identifier



3. Pandas 기능

7) 시각화

https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/04_plotting.html#how-to-create-plots-in-pandas



https://pandas.pydata.org/docs/getting_started/comparison/comparison_with_sql.html#comparison-with-sql

SELECT

In SQL, selection is done using a comma-separated list of columns you'd like to select (or a `*` to select all columns):

```
SELECT total_bill, tip, smoker, time
FROM tips;
```

With pandas, column selection is done by passing a list of column names to your DataFrame:

```
In [6]: tips[["total_bill", "tip", "smoker", "time"]]
```

```
Out[6]:
```

	total_bill	tip	smoker	time
0	16.99	1.01	No	Dinner
1	10.34	1.66	No	Dinner
2	21.01	3.50	No	Dinner
3	23.68	3.31	No	Dinner
4	24.59	3.61	No	Dinner

2장. DataFrame 생성

DataFrame 생성

Dict 이용

중첩 리스트 이용

랜덤값 이용

Series 이용

https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html#dataframe

DataFrame

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series
- Another DataFrame

Along with the data, you can optionally pass **index** (row labels) and **columns** (column labels) arguments. If you pass an index and / or columns, you are guaranteeing the index and / or columns of the resulting DataFrame. Thus, a dict of Series plus a specific index will discard all data not matching up to the passed index.

2. DataFrame 생성

1) dict 이용

문법 :

변수 = `pd.DataFrame(dict타입)`

```
# 1. dict 이용한 DataFrame 생성
df = pd.DataFrame({"a": [4, 5, 6],
                   "b": [7, 8, 9],
                   "c": [10, 11, 12]},
                  index = [1, 2, 3])

print("1. DataFrame: \n", df)
print("2. 컬럼 정보: \n", df.columns)
print("3. 인덱스 정보: \n", df.index, df.keys())
print("4. 값 정보: \n", df.values)
```

```
1. DataFrame:
   a  b  c
1  4  7 10
2  5  8 11
3  6  9 12
2. 컬럼 정보:
Index(['a', 'b', 'c'], dtype='object')
3. 인덱스 정보:
Int64Index([1, 2, 3], dtype='int64') Index(['a', 'b', 'c'], dtype='object')
4. 값 정보:
[[ 4  7 10]
 [ 5  8 11]
 [ 6  9 12]]
```

2. DataFrame 생성

2) 중첩리스트 이용

문법 :

변수 = pd.DataFrame(중첩리스트, index=리스트, columns=리스트)

```
# 2. 중첩 리스트 이용한 DataFrame 생성 => ndarray도 가능
df = pd.DataFrame([[4, 7, 10],[5, 8, 11],[6, 9, 12]],
                  index=[1, 2, 3],
                  columns=['a', 'b', 'c'])
print("1. DataFrame: \n", df)
print("2. 컬럼 정보: \n", df.columns)
print("3. 인덱스 정보: \n", df.index, df.keys())
print("4. 값 정보: \n", df.values)
```

```
1. DataFrame:
   a  b  c
1  4  7 10
2  5  8 11
3  6  9 12
2. 컬럼 정보:
Index(['a', 'b', 'c'], dtype='object')
3. 인덱스 정보:
Int64Index([1, 2, 3], dtype='int64') Index(['a', 'b', 'c'], dtype='object')
4. 값 정보:
[[ 4  7 10]
 [ 5  8 11]
 [ 6  9 12]]
```


2. DataFrame 생성

3) 랜덤값 이용

문법 :

변수 = pd.DataFrame(랜덤값)

```
data = np.random.random((5,2)) # 5행 2열
df = pd.DataFrame(data, columns=['A','B'])
print("1. np.random.random((행,열)) 활용한 df \n", df)

data = np.random.randint(5, size=(2, 4)) # 0 ~ 4 범위에서 2x4 행렬 반환
df = pd.DataFrame(data, columns=['A','B','C','D'])
print("2. np.random.randint(5, size=(2, 4)) 활용한 df \n", df)

data = np.random.choice(['foo','bar','baz'], size=(3,4))
df = pd.DataFrame(data, columns=['A','B','C','D'])
print("3. np.random.choice(['foo','bar','baz'], size=(3,4)) 활용한 df:\n", df)
```

	A	B
0	0.332535	0.097839
1	0.321543	0.615278
2	0.537106	0.709441
3	0.663948	0.562781
4	0.081860	0.504816

	A	B	C	D
0	2	0	3	4
1	3	1	2	0

	A	B	C	D
0	baz	foo	baz	bar
1	foo	foo	bar	foo
2	baz	foo	baz	baz

2. DataFrame 생성

4) Series 이용

문법 :

변수 = pd.DataFrame([Series,...])

```
# 3. DataFrame 생성 - Series 사용
print("3. DataFrame 생성")
name = pd.Series(["유관순", "안중근"])
age = pd.Series([18, 31])
birthday = pd.Series(['1920/09/28', '1910/03/26'])

hero = pd.DataFrame([name, age, birthday])
hero.columns = ["hero1", "hero2"]
hero.index = ["이름", "나이", "생일"]
print(hero)
```

3. DataFrame 생성		
	hero1	hero2
이름	유관순	안중근
나이	18	31
생일	1920/09/28	1910/03/26

3장. 인덱스 생성

인덱스 생성 방법

set_index, reset_index, reindex

ignore_index

문법 1:

```
pd.DataFrame( ..., index=함수표현식 )
```

1) RangeIndex

```
# 1. RangeIndex 생성
print("1. RangeIndex 생성")
temps = pd.DataFrame({"City": ["Seoul", "Pusan"],
                      "Temperature": [32, 34]}, index=range(2))
print(temps, temps.index)
```

```
1. RangeIndex 생성
   City  Temperature
0  Seoul           32
1  Pusan           34 RangeIndex(start=0, stop=2, step=1)
```

2) Int64Index

```
# 2. Int64Index 생성
print("2. Int64Index 생성")
df_i64 = pd.DataFrame(np.arange(10, 15), index=np.arange(1, 6))
df_i64.columns=["num"]
print(df_i64, df_i64.index)
```

```
2. Int64Index RangIndex 생성
   num
1    10
2    11
3    12
4    13
5    14 Int64Index([1, 2, 3, 4, 5], dtype='int64')
```

3) Float64Index

```
# 3. Float64Index 생성
print("3. Float64Index 생성")
df_f64 = pd.DataFrame(np.arange(0, 10, 2),
                      index=np.arange(0.0, 10.0, 2))
df_f64.columns=["num"]
print(df_f64, df_f64.index)
```

```
3. Float64Index 생성
      num
0.0      0
2.0      2
4.0      4
6.0      6
8.0      8 Float64Index([0.0, 2.0, 4.0, 6.0, 8.0], dtype='float64')
```

4) DatetimeIndex

```
# 4. DatetimeIndex 생성
print("4. DatetimeIndex 생성")
rng = pd.date_range('2020-10-02', periods=5)
df = pd.DataFrame({"num":range(5)}, index=rng)
print(df, df.index)
```

```
      num
2020-10-02  0
2020-10-03  1
2020-10-04  2
2020-10-05  3
2020-10-06  4 DatetimeIndex(['2020-10-02', '2020-10-03', '2020-10-04', '2020-10-05',
                              '2020-10-06'],
                              dtype='datetime64[ns]', freq='D')
```

2. 컬럼을 인덱스 변경

문법:

```
df.set_index(컬럼명, inplace=True)
```

```
df.reset_index(inplace=True, drop=True)
```

```
# 1. DataFrame 생성
df = pd.DataFrame({
    "date": ['2021', '2022'],
    "City": ["Seoul", "Seoul"],
    "Temperature": [32, 34]
})
print("1. 원본 DataFrame")
print(df)
```

1. 원본 DataFrame			
	date	City	Temperature
0	2021	Seoul	32
1	2022	Seoul	34

```
# 2. 기존 컬럼을 인덱스로 변경
print("2. 기존 컬럼을 인덱스로 변경")
df.set_index('date', inplace=True)
print(df)
```

2. 기존 컬럼을 인덱스로 변경		
	City	Temperature
date		
2021	Seoul	32
2022	Seoul	34

```
# 3. 새로운 인덱스로 변경
print("3. 새로운 인덱스로 변경")
# df.reset_index(inplace=True)
df.reset_index(inplace=True, drop=True)
print(df)
```

3. 인덱스 재배치

문법:

```
new_df = df.reindex(index=리스트)  
ignore_index=True
```

1. DataFrame 생성

```
df = pd.DataFrame(np.arange(10, 15), index=list("BADEC"))  
print("1. DataFrame 생성")  
print(df)
```

1. DataFrame 생성

	0
B	10
A	11
D	12
E	13
C	14

2. index 재배치

```
new_df = df.reindex(index=list("ABCDE"))  
print("2. index 재배치")  
print(new_df)
```

2. index 재배치

	0
A	11
B	10
C	14
D	12
E	13

3. 인덱스 재배치

```
df1 = pd.DataFrame({'a':[12,2]},  
                    index=[1,2])  
df2 = pd.DataFrame({'a':[120,20]},  
                    index=[1,2])  
  
new_df = pd.concat([df1, df2])  
print("3. df 병합")  
print(new_df)  
print("4. df 병합, ignore_index=True")  
new_df = pd.concat([df1, df2], ignore_index=True)  
print(new_df)
```

```
3. df 병합  
   a  
1  12  
2   2  
1 120  
2  20  
4. df 병합, ignore_index=True  
   a  
0  12  
1   2  
2 120  
3  20
```


4장. DataFrame의 subset 처리

다양한 색인 처리 방법

`df.loc[], df.iloc[]`

색인 이용한 값 변경

`filter` 함수

1. Subset 처리- column

문법 :

1. 단일컬럼 조회 (Series 반환)

`df. 컬럼명`

`df['컬럼명']`

2. 다중컬럼 조회 (DataFrame 반환)

`df[['컬럼명','컬럼명',...]]`

```
# 1. 싱글 및 멀티 컬럼 조회
df = pd.DataFrame({"col1": [4, 5, 6, 6],
                   "col2": [7, 8, 9, 9],
                   "col3": [10, 11, 12, 12]},
                  index = list("ABCD"))

print("1. col1 컬럼만 조회")
print(df.col1)      # Series 반환

print("2. col1 컬럼만 조회")
print(df['col1'])    # Series 반환

print("3. col1와 col2 컬럼 조회")
print(df[['col2', 'col1']]) # fancy 색인 비슷 , # DataFrame 반환
```

```
1. col1 컬럼만 조회
A    4
B    5
C    6
D    6
Name: col1, dtype: int64
2. col1 컬럼만 조회
A    4
B    5
C    6
D    6
Name: col1, dtype: int64
3. col1와 col2 컬럼 조회
   col2  col1
A      7     4
B      8     5
C      9     6
D      9     6
```

2. Subset 처리 - row

문법 :

`df.loc[XXX]` , XXX는 인덱스 라벨, fancy, 슬라이싱, boolean

`df.iloc[YYY]`, YYY는 인덱스 위치, fancy, 슬라이싱, boolean

		col1	col2	col3
0	A	4	7	10
1	B	5	8	11
2	C	6	9	12
3	D	6	9	12
4	E	1	2	10

df.loc[라벨]

```
print("1. A 행 출력(인덱싱 label)")
print(df.loc["A"]) # Series 반환
```

```
print("2. A 와 B행 출력(fancy label)")
print(df.loc[["A","B"]]) # DataFrame 반환
```

```
print("3. B행부터 D행까지 출력(slicing label)")
print(df.loc["B":"D"]) # DataFrame 반환
```

```
print("4. A,C,E행 출력(boolean label)")
print(df.loc[[True,False,True,False,True]])
```

df.iloc[위치]

```
print("1. A 행 출력(인덱싱 위치)")
print(df.iloc[0]) # Series 반환
```

```
print("2. A 와 B행 출력(fancy 위치)")
print(df.iloc[[0,1]]) # DataFrame 반환
```

```
print("3. B행부터 D행까지 출력(slicing label)")
print(df.iloc[1:-1]) # DataFrame 반환
```

```
print("4. A,C,E행 출력(boolean label)")
print(df.iloc[[True,False,True,False,True]])
```

3. Subset 처리 – row와 column

문법 :

`df.loc[row, column]`

`df.iloc[row, column]`

		0	1	2
		col1	col2	col3
0	A	4	7	10
1	B	5	8	11
2	C	6	9	12
3	D	6	9	12
4	E	1	2	10

df.loc[라벨, 라벨]

```
print("1. A행 col1 출력")  
print(df.loc["A", "col1"])
```

```
print("2. A,B행 col1 출력")  
print(df.loc[["A","B"], "col1"])
```

```
print("3. A,B행 col1,col3 출력")  
print(df.loc[["A","B"], ["col1","col3"]])
```

```
print("4. B~D행, col2~ col3까지 출력")  
print(df.loc["B":"D", "col2":"col3"])
```

df.iloc[위치, 위치]

```
print("1. A행 col1 출력")  
print(df.iloc[0, 0])
```

```
print("2. A,B행 col1 출력")  
print(df.iloc[[0,1], 0])
```

```
print("3. A,B행 col1,col3 출력")  
print(df.iloc[[0,1], [0,1]])
```

```
print("4. B~D행, col2~ col3까지 출력")  
print(df.iloc[1:4, 1:])
```

4. loc와 iloc 활용한 값 변경

문법 :

`df.loc[row, column]= 값`

`df.iloc[row, column]= 값`

		0	1	2
		col1	col2	col3
0	A	4	7	10
1	B	5	8	11
2	C	6	9	12
3	D	6	9	12
4	E	1	2	10

`df.loc[라벨, 라벨]=값`

```
print("1. A행값을 모두 100으로 변경")  
df.loc["A"]=100
```

```
print("2. B,C 행값을 모두 200으로 변경")  
df.loc[["B","C"]]=200
```

```
print("3. B,E 행의 col1,col2값 300으로 변경")  
df.loc[["B","E"],["col1","col2"]]=300
```

`df.iloc[위치, 위치]=값`

```
print("4. 마지막행의 마지막 열 값 1100으로 변경")  
df.iloc[-1,-1]=1000
```

```
print("5. 1행~3행 2열~끝 값 -1으로 변경")  
df.iloc[0:3, 1:]=-1
```

5. Subset 처리- filter함수

문법 :

```
df.filter(items=리스트, axis=0|1)
```

```
df.filter(like=str, axis=0|1)
```

	one	two	three
mouse	1	2	3
rabbit	4	5	6
habbit	7	8	9

items=리스트 속성

```
print("2. 열 라벨로 검색, axis=1")
df2 = df.filter(items=['one', 'two'], axis=1)
print(df2)
```

```
print("3. 행 라벨로 검색, axis=0")
df2 = df.filter(items=['mouse'], axis=0)
print(df)
```

like=str 속성

```
print("4. 열 라벨로 like 검색, axis=1")
df2 = df.filter(like='e', axis=1)
print(df2)
```

```
print("5. 행 라벨로 like 검색, axis=0")
df2 = df.filter(like='bit', axis=0)
print(df2)
```

5장. 컬럼 및 행 추가/삭제

컬럼 추가/삽입/삭제
행 추가/ 삭제

1. 컬럼 추가

문법 :

1. new_df = df.assign(컬럼명=리스트|함수)
2. df['컬럼명']=값
3. new_df = pd.concat([df, df2], axis=1)

	이름	국어	수학
1	홍길동	30	20
2	이순신	26	12
3	유관순	11	20
4	강감찬	10	12

df.assign()

```
print("2. 영어 컬럼 추가")
df = df.assign(영어=[30,42,53,21])
```

```
print("3. 총합 컬럼 추가")
df = df.assign(총합=
    lambda df: df["국어"]+df["수학"]+df["영어"])
```

df['컬럼']

```
print("4. 평균 컬럼 추가")
df['평균']=np.round(df['총합']/4,2)
```

```
df['합격여부'] = ["합격" if avg > 25
                  else "불합격" for avg in df['평균']]
```


2. 컬럼 삽입

문법 :

`df.insert(컬럼인덱스, 컬럼명, 값)`

	이름	국어	수학
1	홍길동	30	20
2	이순신	26	12
3	유관순	11	20
4	강감찬	10	12

```
print("2. 영어점수 중간 삽입")  
df.insert(1, '영어', [100, 100, 100, 100])
```

	이름	영어	국어	수학
1	홍길동	100	30	20
2	이순신	100	26	12
3	유관순	100	11	20
4	강감찬	100	10	12

3. 컬럼 삭제

문법 :

1. 단일 컬럼 삭제

```
df.pop('컬럼명')  
del df['컬럼명']
```

2. 다중 컬럼 삭제

```
new_df = df.drop(columns=리스트)  
new_df = df.drop(리스트, axis=1)
```

단일컬럼 삭제

```
print("2. 영어 컬럼 삭제")  
del df['영어']  
  
print("3. 과학 컬럼 삭제")  
df.pop('과학')
```

	이름	국어	수학	영어	과학	체육	사회	컴퓨터
1	홍길동	30	20	20	20	20	20	20
2	이순신	26	12	12	12	12	12	12
3	유관순	11	20	20	20	20	20	20
4	강감찬	10	12	12	12	12	12	12

다중컬럼 삭제

```
print("4. 사회,체육 컬럼 삭제")  
df = df.drop(columns=['사회','체육'])  
  
print("5. 국어,수학, 컬럼 삭제")  
df = df.drop(['국어','수학'], axis=1)
```

4. 행 추가

문법 :

```
new_df = df.append(df2, ignore_index=True)
```

```
new_df = pd.concat([df, df2, df3,..], axis=0 , ignore_index=True)
```

df

	Name	age	birthday
0	유관순	18	1920/09/28
1	안중근	31	1910/03/26

df2

	Name	age	birthday
0	홍길동	22	1990/09/28
1	강감찬	43	1980/03/26

```
print("1. df.append 행 추가")
new_df = df.append(df2, ignore_index=True)
print(new_df)
```

```
print("2. df.concat 행 추가")
new_df = pd.concat([df2,df], axis=0, ignore_index=True)
print(new_df)
```

5. 행 삭제

문법 :

```
new_df = df.drop(index=리스트)  
new_df = df.drop(리스트, axis=0)
```

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12
4	6	1	32
5	7	2	23
6	8	4	12

```
# 1. df.drop(index=[인덱스명, 인덱스명])  
df = df.drop(index=[1,4])  
print(df)
```

```
# 2. df.drop([인덱스명, 인덱스명], axis=0)  
df = df.drop([3,2], axis=0)  
print(df)
```

6장. null 처리/정렬

nan, null 처리
정렬

문법 :

1. Pandas 함수 이용

```
bool = pd.isna(스칼라|Series|df)
bool = pd.isnull(스칼라|Series|df)
bool = pd.notnull(스칼라|Series|df)
```

2. DataFrame 함수 이용

```
bool = df.isnull() 또는 df.isna()
bool = df[컬럼명].isnull()
bool = df[[컬럼, 컬럼2]].isnull()
```

	col1	col2	col3	col4
1	1.0	2.0	NaN	NaN
2	1.0	2.0	3.0	NaN
3	1.0	2.0	3.0	NaN
4	NaN	2.0	3.0	NaN
5	1.0	NaN	3.0	NaN

Pandas 이용

```
print("2. null 찾기")
print(df.isna())
print(df["col1"].isna())
print(df[["col1", "col2"]].isna())

print(df.isnull())
print(df["col1"].isnull())
print(df[["col1", "col2"]].isnull())
```

DataFrame 이용

```
print("2. null 찾기")
print(pd.isna(df))
print(pd.isna(df["col1"]))
print(pd.isna(df[["col1", "col2"]]))

print(pd.isnull(df))
print(pd.isnull(df["col1"]))
print(pd.isnull(df[["col1", "col2"]]))
```

2. Null 삭제

문법 :

```
df.dropna(axis=0|1, how='any|all', inplace=False)
```

	col1	col2	col3	col4
1	1	2.0	3.0	2.0
2	1	2.0	NaN	NaN
3	1	2.0	NaN	NaN
4	1	2.0	NaN	NaN
5	1	NaN	NaN	NaN

```
print("2. nan이 하나라도 있는 행 삭제:")
new_df = df.dropna(axis=0)
print(new_df)

print("3. nan이 하나라도 있는 컬럼 삭제:")
new_df = df.dropna(axis=1)
print(new_df)
```

```
print("4. 모든 행이 nan 행 삭제:")
new_df = df.dropna(axis=0, how='all')
print(new_df)

print("5. 모든 컬럼이 nan 컬럼 삭제:")
new_df = df.dropna(axis=1, how='all')
print(new_df)
```


3. Null 변경

문법 :

```
df.fillna(값)  
df.fillna({컬럼명:값, 컬럼명:값})  
df.fillna(method='bfill|ffill|None')
```

	col1	col2	col3	col4
1	1.0	2.0	3.0	NaN
2	1.0	2.0	3.0	NaN
3	1.0	2.0	3.0	NaN
4	1.0	2.0	3.0	NaN
5	NaN	NaN	NaN	NaN

```
print("2. 모든 nan 값을 N/A로 변경")  
new_df = df.fillna('N/A')  
print(new_df)  
  
print("3. nan 값을 지정된 컬럼만 특정 값으로 변경")  
new_df = df.fillna({'col1':'N/A', 'col3':999})  
print(new_df)
```

	col1	col2	col3
1	1.0	1.0	1.0
2	NaN	NaN	2.0
3	3.0	3.0	NaN
4	4.0	4.0	4.0
5	NaN	NaN	NaN

```
print("4. 모든 NaN값을 Nan의 앞(forward)의 값으로 변경")  
new_df = df.fillna(method="ffill")  
print(new_df)  
  
print("5. 모든 NaN값을 Nan의 뒤(backward)의 값으로 변경")  
df.fillna(method="bfill", inplace=True)  
print(df)
```

문법 :

1. 컬럼 정렬

```
df.sort_values( by=컬럼값, ascending=True, inplace=False,  
                ignore_index=False, kind="quicksort",  
                na_position="last")
```

2. 위치(position) 정렬

```
df.sort_index( axis=0|1, ascending=True, inplace=False,  
              ignore_index=False, kind="quicksort",  
              na_position="last")
```

	mpg	cylinders	displacement	...	model_year	origin	name
H	18.0	8.0	307.0	...	70.0	usa	chevrolet chevelle malibu
D	15.0	8.0	350.0	...	70.0	usa	buick skylark 320
A	18.0	8.0	318.0	...	70.0	usa	plymouth satellite
F	16.0	8.0	304.0	...	70.0	usa	amc rebel sst
C	NaN	NaN	NaN	...	NaN	NaN	NaN
B	15.0	8.0	429.0	...	70.0	usa	ford galaxie 500
E	14.0	8.0	454.0	...	70.0	usa	chevrolet impala
G	14.0	8.0	440.0	...	70.0	usa	plymouth fury iii
I	14.0	8.0	455.0	...	70.0	usa	pontiac catalina
J	15.0	8.0	390.0	...	70.0	usa	amc ambassador dpl

```
print("2. mpg 컬럼 오름차순 정렬")
```

```
new_df = df.sort_values(by='mpg', ascending=True)
```

```
print("3. ['mpg','displacement'] 다중정렬")
```

```
new_df = df.sort_values(by=['mpg','displacement'], ascending=False)
```

```
print("4. na_position='first' 정렬")
```

```
new_df = df.sort_values(by='mpg', ascending=True, na_position='first')
```

	mpg	cylinders	displacement	...	model_year	origin	name
H	18.0	8.0	307.0	...	70.0	usa	chevrolet chevelle malibu
D	15.0	8.0	350.0	...	70.0	usa	buick skylark 320
A	18.0	8.0	318.0	...	70.0	usa	plymouth satellite
F	16.0	8.0	304.0	...	70.0	usa	amc rebel sst
C	NaN	NaN	NaN	...	NaN	NaN	NaN
B	15.0	8.0	429.0	...	70.0	usa	ford galaxie 500
E	14.0	8.0	454.0	...	70.0	usa	chevrolet impala
G	14.0	8.0	440.0	...	70.0	usa	plymouth fury iii
I	14.0	8.0	455.0	...	70.0	usa	pontiac catalina
J	15.0	8.0	390.0	...	70.0	usa	amc ambassador dpl

```
print("2. 행 단위 index 정렬:axis= 0")
```

```
new_df = df.sort_index()
```

```
print(new_df)
```

```
print("3. 열 단위 index 정렬:axis= 1")
```

```
new_df = df.sort_index(axis=1, ascending=False)
```

```
print(new_df)
```

7장. 유틸리티 함수

유틸리티 함수

1. 유틸리티 함수 - 일반

함수명	설명
df.max() , df.min()	최대(소)값 반환
df.cummax(), df.cummin()	누적 최대(소)값 반환
df.idxmax() , df.idxmin()	최대(소)값 label 반환
df.sum() , df.cumsum()	(누적)합계 반환
df.mean() , df.median()	평균, 중앙값 반환
df.var() , df.std()	분산, 표준편차 반환
df.count() , df.describe()	행 개수, 통합통계정보 반환
df.replace()	값 변경
df.rename(columns index)	컬럼명 및 인덱스명 변경
df.all(), df.any()	모든(특정) 컬럼(행)값의 참/거짓 여부
df.duplicated(), df.drop_duplicates()	중복조회, 중복제거
df.apply()	임의의 함수 적용
df.eval()	문자열 이용한 연산
df.isin()	값 존재 여부
df.nunique()	unique 개수 반환
df.query()	조건식 이용한 조회

1. 유틸리티 함수 - 일반

함수명	설명
df['컬럼'].between(start, end)	범위 포함 여부
df['컬럼'].unique()	unique한 값 반환
df['컬럼'].value_counts()	값 빈도수 반환 (nan 제외)
df['컬럼'].str.함수	문자열 함수 적용

df[컬럼].str.함수

```
# 2. replace
df = hero['name'].str.replace("Hello", "hello")
print(df)
print()

# 3. slice ==> str[0], str[:3], str[-1], str[::-1]
df = hero['name'].str[::-1]
print(df)
print()

# 4. contains
animals = pd.Series(['cat', 'dog', 'bear', 'cow', 'bird', 'owl', 'rabbit', 'snake'])
print(animals)
print(animals.str.contains('a')) # animals.str.contains('a|b') a또는 b
print(animals[animals.str.contains('a')])
```

2. 유틸리티 함수 - 날짜데이터

문법 :

```
pd.to_datetime('날짜', format=포맷)
pd.date_range('날짜', '날짜', periods=n, freq='D')
df['날짜컬럼'].dt.year
```

```
print("문자열을 날짜타입으로 변환1: ", pd.to_datetime('2019/1/1'))
print("문자열을 날짜타입으로 변환2: ", pd.to_datetime('2019 1 1'))

print("문자열을 날짜타입으로 변환5: ", pd.to_datetime('2019년12월14일', format='%Y년%m월%d일'))
print("문자열을 날짜타입으로 변환6: ", pd.to_datetime('2019년12월14일 12:23:52', format='%Y년%m월%d일 %H:%M:%S'))
```

```
date= pd.date_range("2019-11-1", "2019-11-30")
print("date_range: ", date) # DatetimeIndex(['

print("3. 년, 월, 일, 주(week, 해당년도의 몇번째 주):", df['xxx'].dt.year,
                                             df['xxx'].dt.month,
                                             df['xxx'].dt.day,
                                             df['xxx'].dt.isocalendar().week)

print("4. 날짜데이터를 문자데이터로 변환:", df['xxx'], df['xxx'].astype(str)+" 1")
```


8장. 병합(merge) 및 groupby

병합 개요

Inner 및 outer 병합

Index 병합

groupby

agg 함수

1. 병합 (merge)

컬럼 이용

문법 :

1. inner 병합

가. 공통컬럼 이용

```
pd.merge(df, df2, how="inner", on="컬럼명")
```

나. 비공통컬럼 이용

```
pd.merge(df, df2, how="inner", left_on="컬럼명",  
right_on="컬럼명" )
```

2. outer 병합

가. 공통컬럼 이용

```
pd.merge(df, df2, how="left|right|outer", on="컬럼명")
```

나. 비공통컬럼 이용

```
pd.merge(df, df2, how="left|right|outer", left_on="컬럼명",  
right_on="컬럼명" )
```

1. 병합 (merge)

컬럼과 index 이용

문법 :

```
pd.merge(df, df2, left_on="컬럼명", right_on="인덱스" )
```

```
pd.merge(df, df2, left_on="컬럼명", right_index=True )
```

1. 병합 (merge)

index 이용

문법 :

```
pd.merge(df, df2, left_on="인덱스", right_on="인덱스" )
```

```
pd.merge(df, df2, left_on="인덱스", right_index=True )
```

```
pd.merge(df, df2, right_on="인덱스", left_index=True )
```

```
pd.merge(df, df2, left_index=True, right_index=True )
```

문법 :

`df.groupby(by=컬럼명).그룹함수`
`df.groupby(by=리스트)[컬럼명].그룹함수`

```
df = sns.load_dataset('mpg')
print("1. 원본 DataFrame: \n", df.head())
gb = df.groupby(by="origin").size() # count() 함수 및 df['origin'].value_counts() 와 동일
print("1. df.groupby(by='origin').size(): \n", gb)
print()
gb2 = df.groupby(by="origin")['cylinders'].sum() # mean(), median(), min(), max(), count()
print("2. df.groupby(by='origin')['cylinders'].sum(): \n", gb2)
print("DataFrame 으로 변경\n", pd.DataFrame(gb2))
print()
gb3 = df.groupby(by=["model_year", "origin"])['cylinders'].sum() # mean(), median(), min(), max(), count()
print("3. 다중그룹: \n", gb3)
df = pd.DataFrame(gb3)
print("3. 다중그룹(DataFrame으로 변경): \n", df)
```

3. groupby + agg 함수

문법 :

```
df.groupby(by=컬럼명).agg(함수)  )
```

```
employee = {"empno":['A1', 'A2', 'A3', 'A4', 'A5'],
            "ename":['홍길동', '유관순', '안중근', '강감찬', '이순신'],
            "sal": [1000, 1500, 2300, 3400, 4500],
            "hireday": ['2019/01/02', '2018/01/02', '2017/01/02', '2016/01/02', '2015/01/02'],
            "deptno": [10, 20, 10, 30, 10]}
employee_df = pd.DataFrame(employee)
```

```
# 사용자 정의 함수
def my_mean(values):
    #print("**: ", values)
    n = len(values)
    sum=0
    for v in values:
        sum+=v
    return sum/n
```

```
sal_average2 = employee_df.groupby(by='deptno')['sal'].agg(my_mean)
print("2. 빌트인 함수 사용- mean(): \n", pd.DataFrame(sal_average2))
sal_average3 = employee_df.groupby(by='deptno')['sal'].agg(평균=my_mean)
print("3. 빌트인 함수 사용- mean(): \n", pd.DataFrame(sal_average3))

sal_average3 = employee_df.groupby(by='deptno')['sal'].agg('mean')
print("4. 빌트인 함수 사용- mean(): \n", pd.DataFrame(sal_average3))
```

3. groupby + agg 함수

```
# 2. 한꺼번에 여러 함수를 리스트에 설정하고 실행
info = employee_df.groupby(by='deptno')['sal'].agg([np.mean, np.sum, Series.count, len, np.min, np.max])
print(pd.DataFrame(info))
```

	mean	sum	count	len	amin	amax
deptno						
10	2600	7800	3	3	1000	4500
20	1500	1500	1	1	1500	1500
30	3400	3400	1	1	3400	3400

```
info = employee_df.groupby(by='deptno')['sal'].agg(['mean', 'sum', 'count', 'min', 'max'])
print(pd.DataFrame(info))
```

	mean	sum	count	min	max
deptno					
10	2600	7800	3	1000	4500
20	1500	1500	1	1500	1500
30	3400	3400	1	3400	3400

```
info = employee_df.groupby(by='deptno').agg({
    'sal': ['sum', 'mean', 'max'],
    'deptno': 'count',
})
```

9장. 파일 I/O

CSV 파일 입출력

1. 파일 I/O – csv 파일

문법 :

```
df = pd.read_csv("파일명", index_cols=0)
```

```
df = pd.read_csv("파일명", nrows=n, header=0, name=리스트컬럼명)
```

```
# read in msft.csv into a DataFrame
msft = pd.read_csv("./data/msft.csv")
print("1. 기본 : \n", msft.head())
#           Date  Open  High  Low  Close  Volume
# 0  7/21/2014  83.46  83.53  81.81  81.93  2359300
# 1  7/18/2014  83.30  83.40  82.52  83.35  4020800

msft = pd.read_csv("./data/msft.csv", index_col=0)
print("2. 특정 컬럼을 인덱스 설정 : \n", msft.head())
#           Open  High  Low  Close  Volume
# Date
# 7/21/2014  83.46  83.53  81.81  81.93  2359300
# 7/18/2014  83.30  83.40  82.52  83.35  4020800

msft = pd.read_csv("./data/msft.csv", header=0,
                    names = ['date', 'open', 'high', 'low',
                             'close', 'volume'])
print("3. 컬럼명 변경 : \n", msft.head())
```

2. 파일 I/O - 엑셀파일

문법 : *pip install xlrd*

```
df = pd.read_excel("파일명", sheet_name=이름, nrows=n, header=0)
```

1. 기본 엑셀파일 읽기 (기본적으로 첫 sheet를 읽는다)

```
df = pd.read_excel("./data/stocks.xlsx") # # only reads first sheet (msft in this case)
print("1. 기본 엑셀 읽기: \n", df.head())
```

2. 특정 sheet 엑셀파일 읽기

```
df = pd.read_excel("./data/stocks.xlsx", sheet_name='aapl' )
print("2. 특정 sheet 엑셀파일 읽기: \n", df.head())
```

3. 특정 sheet 엑셀파일 읽기

```
aapl = pd.read_excel("./data/stocks.xlsx", sheet_name='aapl', skiprows=100, nrows=6, header=0,
                    names = ['date', 'open', 'high', 'low',
                              'close', 'volume'] )
print("2. 특정 sheet 엑셀파일 읽기: \n", aapl)
```

3. 파일 I/O – json/html

문법 : *pip install lxml*

```
df = pd.read_html("파일명")
```

```
json = pd.read_json(파일명)
```

```
df.to_json(파일명)
```

```
df = pd.read_csv("./data/msft.csv").head()
print("1. 원본: \n", df)
print()

print("2. json 파일 쓰기: \n")
df.to_json("msft.json")

json = pd.read_json("msft.json")
print("3. json 파일 읽기: \n", json)
print()

# pip install lxml
url = "http://www.fdic.gov/bank/individual/failed/banklist.html"
# read it
banks = pd.read_html(url)
print(banks)
df = pd.read_excel("./data/stocks.xlsx")
# write the first tw rows to HTML
df.head(2).to_html("stocks.html")
```



감사합니다.
