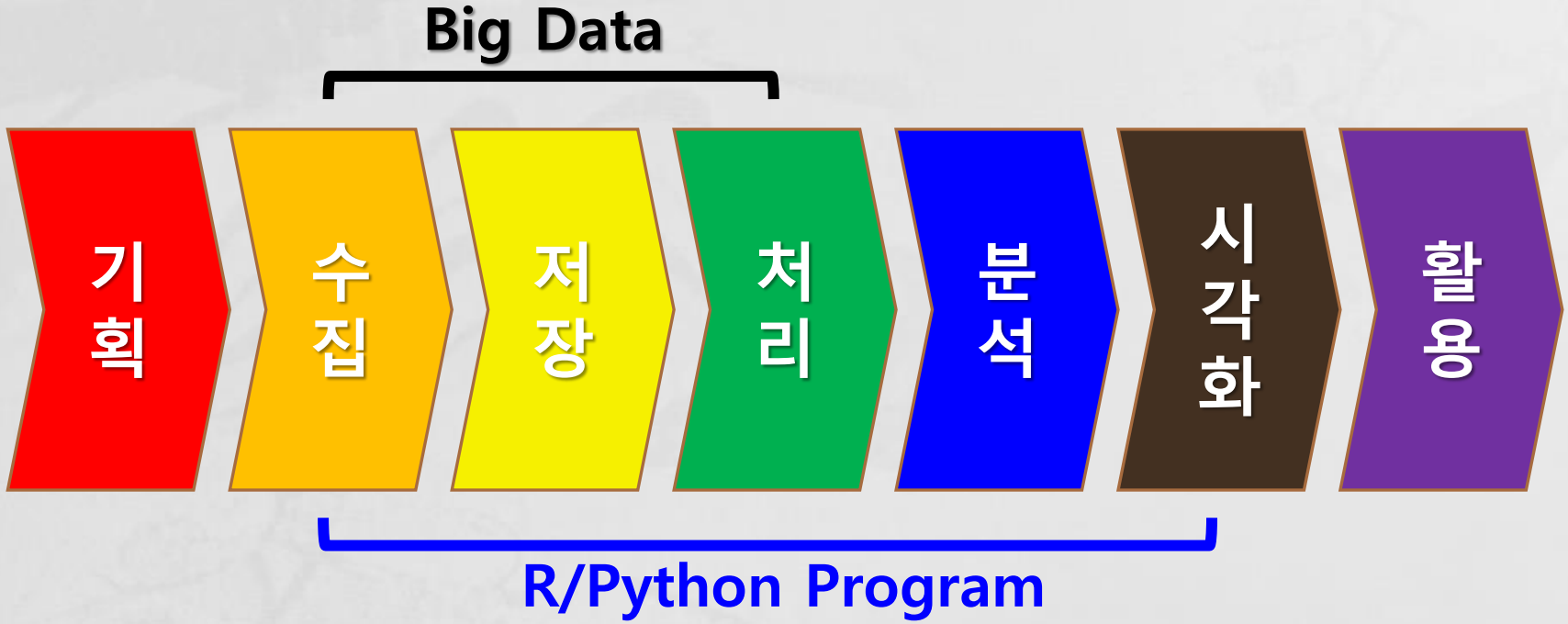
데이터 처리

Before we start

Big Data 단계



* 분석 목표 설정 -> 필요한 데이터 정의 -> 데이터 수집 및 전처리 -> 분석 -> 인사이트 도출

데이터 처리(Pandas)

판다스(pandas)

판다스 설명(pandas)

- series, DataFrame등의 자료구조를 활용한 데이터분석 기능을 제공해 주는 라이브러리
- 라이브러리 구성
- 여러종류의 클래스와 다양한 함수로 구성
- 시리즈와 데이터 프레임의 자료 구조 제공
- 시리즈(1차원 배열) 데이터프레임(2차원 행열구조)

판다스(pandas)

판다스의 목적

- 서로 다른 유형의 데이터를 공통된 포맷으로 정리하는 것
- 행과 열로 이루어진 2차원 데이터프레임을 처리 할 수 있는 함수제공 목적
- 실무 사용 형태 : 데이터 프레임

시리즈(Series)

Series

- pandas의 기본 객체 중 하나
- numpy의 ndarray를 기반으로 인덱싱을 기능을 추가하여 1차원 배열을 나타냄
- index를 지정하지 않을 시, 기본적으로 ndarray와 같이 0-based 인덱스 생성, 지정할 경우 명시적으로 지정된 index를 사용
- 같은 타입의 0개 이상의 데이터를 가질 수 있음

시리즈(Series)

Series

- 자료구조: 시리즈
  - 데이터가 순차적으로 나열된 1차원 배열 형태
  - 인덱스(index)와 데이터 값(value)이 일대일로 대을
  - 딕셔너리와 비슷한 구조 : {key(index):value}
- 시리즈의 인덱스
  - 데이터 값의 위치를 나타내는 이름표 역할
- 시리즈 생성 : 판다스 내장함수인 Series()이용
  - 리스트로 시리즈 만들기
  - 딕셔너리로 시리즈 만들기
  - 튜플로 시리즈 만들기

시리즈(Series)

판다스 모듈 import

- 대부분의 코드에서 pandas 모듈은 pd 라는 별칭을 사용함
- 데이터분석에서 pandas와 numpy 두 패키지는 기본 패키지로 본다
- numpy는 np라는 별칭을 사용
  - import pandas as pd
  - import numpy as np

시리즈(Series)

Series 생성하기

- data로만 생성하기
- index는 기본적으로 0부터 자동적으로 생성

```
# pd.Series(집합적 자료형)
# pd.Series(리스트)
s = pd.Series([1,2,3])
s
# 위 코드는 시리즈 생성 시 인덱스를 명시하지 않았음.  0 base 인덱스 생성
```

```
# pd.Series(튜플)
s = pd.Series((1.0,2.0,3.0))
s
```

시리즈(Series)

Series 생성하기

- object type

```
s2 = pd.Series(['a','a','c']) #dtype: object
s2
```

```
# 리스트내에 서로 다른 type의 data가 있으면 형변환 일어남- 문자열로 변환됨
s_1 = pd.Series(['a',1,3.0]) #dtype: object
s_1
```

시리즈(Series)

Series 생성하기

- 범위를 시리즈의 value 생성하는 데 사용하기

- range/np.arange 함수 사용

```
s = pd.Series(range(10,14)) # index 인수는 생략됨
s
```

```
np.arange(200)
```

```
s3 = pd.Series(np.arange(200))
s3
```

시리즈(Series)

결측값을 포함해서 시리즈 만들기

- 결측값 NaN - numpy 라는 모듈에서 생성할 수 있음
- 결측값 생성 위해서는 numpy 모듈 import

```
# NaN은 np.nan 속성을 이용해서 생성
s=pd.Series([1,2,3,np.nan,6,8])
s
# dtype: float64
# 판다스가 처리하는 자료구조인 시리즈와 데이터프레임에서 결측치가 있는 경우는 datatype이 float으로 변경(수치)
```

시리즈(Series)

인덱스 명시해서 시리즈 만들기

- 숫자 인덱스 지정
- s = pd.Series([값1,값2,값3],index=[1,2,3])

```
s=pd.Series([10,20,30],index=[1,2,3])
s
```

- 문자 인덱스 지정

```
s= pd.Series([95,100,88], index = ['홍길동','이몽룡','성춘향'])
s
```

시리즈(Series)

인덱스 활용

- 시리즈의 index
- 시리즈의 index는 index 속성으로 접근
- 시리즈.index.name 속성
- 시리즈의 인덱스에 이름을 붙일 수 있음

```
s.index.name = '광역시'
s
```

시리즈(Series)

시리즈의 값

- numpy 자료구조 - 1차원 배열
- values 속성으로 접근
- 시리즈.values
- 시리즈.name 속성
- 시리즈 데이터에 이름을 붙일 수 있음
- name 속성은 값의 의미 전달에 사용

```
s.values
# 시리즈의 값의  전체 형태는  array(numpy의 자료구조) 형태
```

인덱싱

인덱싱

- 데이터에서 특정한 데이터를 골라내는 것
- 시리즈의 인덱싱 종류
- 정수형 위치 인덱스(integer position)
- 인덱스 이름(index name) 또는 인덱스 라벨(index label)
  - 인덱스 별도 지정하지 않으면 0부터 시작하는 정수형 인덱스가 지정됨
- 원소접근
  - 정수형 인덱스 : 숫자 s[0]
  - 문자형 인덱스 : 문자 s['인천']

인덱싱

리스트 이용 인덱싱

- 자료의 순서를 바꾸거나 특정자료 여러개를 선택
- 인덱스값 여러개를 이용해 접근시 []안에 넣음

```
print(s)
# s[0,3,1] #KeyError: 'key of type tuple not found and not a MultiIndex'
s[0],s[3],s[1] #(9904312, 2466052, 3448737)
# 시리즈명[[인덱스리스트]] - 시리즈형태로 반환
s[[0,3,1]] #- 인덱스 리스트 내의 해당 인덱스의 item을 추출 후 시리즈 형태로 반환
```

인덱싱

시리즈 슬라이싱

- 정수형 위치 인덱스를 사용한 슬라이싱
- 시리즈[start:stop+1]

```
print(s)
s[[1,2]]
s[['부산','인천']]
s[1:3] # 시리즈 슬라이싱을 사용하면 시리즈로 반환
```

- 문자(라벨)인덱스 이용 슬라이싱
- 시리즈['시작라벨':'끝라벨'] : 표시된 라벨 범위 모두 추출

```
# 문자인덱스를 이용한 슬라이싱 가능
# 표시된 문자인덱스 범위 모두 추출
s["부산":"대구"]
```

인덱싱

문자 인덱스

- . 연산자를 이용하여 접근가능
- 인덱스 통한 데이터 업데이트
- 인덱스 재 사용 하기

```
s['서울'] = 10000000
s['서울']
```

```
print(s.index)
s1 = pd.Series(np.arange(4),s.index)
s1
```

시리즈 연산

벡터 연산

- numpy 배열처럼 pandas의 시리즈도 벡터화 연산 가능
- 벡터화 연산이란 집합적 자료형의 원소 각각을 독립적으로 계산을 진행하는 방법
- 단, 연산은 시리즈의 값에만 적용되며 인덱스 값은 변경 불가

```
# 시리즈 원소 각각에 대하여 + 4 연산을 진행 - 벡터화 연산
pd.Series([1,2,3]) + 4
```

```
# s 시리즈의 단위가 커서 단위를 변경하고 자 함 1/1000000
# 시리즈 자체를 1000000으로 나누면 됨, 벡터화연산을 진행 함
print(s)
s/1000000 # 대입하지 않았음
```

시리즈 연산

벡터 연산

- 백터화 인덱싱도 가능

```
# 벡터화 인덱싱도 가능
# 시리즈[조건]
# s시리즈 값 중 2500000 보다 크고 5000000보다 작은 원소를 추출
s[(s>250e4) & (s<500e4)]
# s 시리즈 각 원소값 각각에 대해서 조건식을 확인해서 결과가 True인 원소를 반환
```

시리즈 연산

Boolean selection

- boolean Series가 []와 함께 사용되면 True 값에 해당하는 값만 새로 반환되는 Series객체에 포함됨
- 다중조건의 경우, &(and), |(or)를 사용하여 연결 가능
  - s0 = pd.Series(np.arange(10), np.arange(10)+1)
  - S0
  - s0 > 5
  - s0[s0>5]
  - s0[s0%2 == 0]

시리즈 연산

Boolean selection

- 인덱스에도 관계연산이 가능
  - s0.index > 5
  - s0[s0.index>5]
  - # s0의 value가 5를 초과하고 8미만인 아이템(원소)만 추출하시오
  - s0[(s0>5) & (s0<8)]
  - (s0 >= 7).sum() # True의 개수 총 합
  - s0[s0 >= 7]).sum() # 조건의 결과가 True인 원소들의 합

시리즈 연산

두 시리즈간의 연산

- 두 시리즈간의 연산 가능
  - num_s1 + num_s2 # 시리즈간의 연산은 같은 인덱스를 찾아 연산을 진행

  # 동일한 인덱스는 연산을 진행하고 나머지 인덱스는 연산처리가 불가능 해서 NaN값으로 처리
  - num_s3 - num_s4
  - num_s3.values - num_s4.values

  # values 속성을 사용해 값만을 추출해 연산을 진행하게 되면 시리즈의 형태가 사라지므로

  # 동일 위치 원소들끼리 연산을 진행

  # 시리즈.values는 array 형태 반환

시리즈 연산

딕셔너리와 시리즈의 관계

- 시리즈 객체는 라벨(문자)에 의해 인덱싱이 가능
- 실질적으로는 라벨을 key로 가지는 딕셔너리 형과 같다고 볼 수 있음
- 딕셔너리에서 제공하는 대부분의 연산자 사용 가능
  - in 연산자 : T/F
  - for 루프를 통해 각 원소의 key와 value에 접근 할수 있음
- in 연산자/ for 반복문 사용

```
# 시리즈 각 원소 출력
for k, v in s.items() :
    print('%s=%d' % (k,v))
```

시리즈 연산

딕셔너리로 시리즈 만들기

- Series({key:value,key1:value1....})
- 인덱스 -> key
- 값 -> value
- 딕셔너리의 원소는 순서를 갖지 않음
- 딕셔너리로 생성된 시리즈의 원소도 순서가 보장되지 않음
- 만약 순서를 보장하고 싶으면 인덱스를 리스트로 지정해야 함

시리즈 연산

시리즈 데이터의 갱신,추가, 삭제

- 인덱싱을 이용하면 딕셔너리 처럼 갱신, 추가 가능
  - # s 시리즈의 부산의 인구 값을 1630000으로 변경
  - s['부산'] = 1630000
  - # 시리즈내의 원소 삭제 - del 명령을 사용
  - del s['서울']
  - # 시리즈에 새로운 원소 추가
  - s['대구'] = 1875000

시리즈 연산

Series 함수

- Series size, shape, unique, count, value_counts 함수
- size(속성) : 개수 반환
- shape(속성) : 튜플형태로 shape반환
- unique: 유일한 값만 ndarray로 반환
- count : NaN을 제외한 개수를 반환
- mean: NaN을 제외한 평균
- value_counts: NaN을 제외하고 각 값들의 빈도를 반환

시리즈 연산

날짜 자동 생성 : date_range

- 판다스 패키지의 date_range 함수 (날짜생성)
- pd.date_range(start=None, end=None, periods=None, freq='D')
- start : 시작날짜/ end= 끝날짜 / periods = 날짜 생성기간/ fref = 날짜 생성 주기
- start는 필수 옵션/end나 periods는 둘 중 하나가 있어야 함/ freq는 기본 Day로 설정

시리즈 연산

날짜 자동 생성 : date_range

- 판다스 패키지의 date_range 함수 (날짜생성)

B	비즈니스 데이
C	커스텀 비즈니스 데이
D	일별
W	주별
M	월별 말일
BM	비즈니스 월별
MS	월별 시작일
BMS	비즈니스 월별 시작일
Q	분기별 말일
BQ	비즈니스 분기별
QS	쿼터 시작일
BQS	비지니스 분기 시작일
A	연도별 말일
BA	비즈니스 연도별 말일
AS	연도별 시작일
BAS	비즈니스 연도별 시작일

H	시간별
T	분별
S	초별
L	밀리초(Milliseconds)
U	마이크로초(Microseconds)

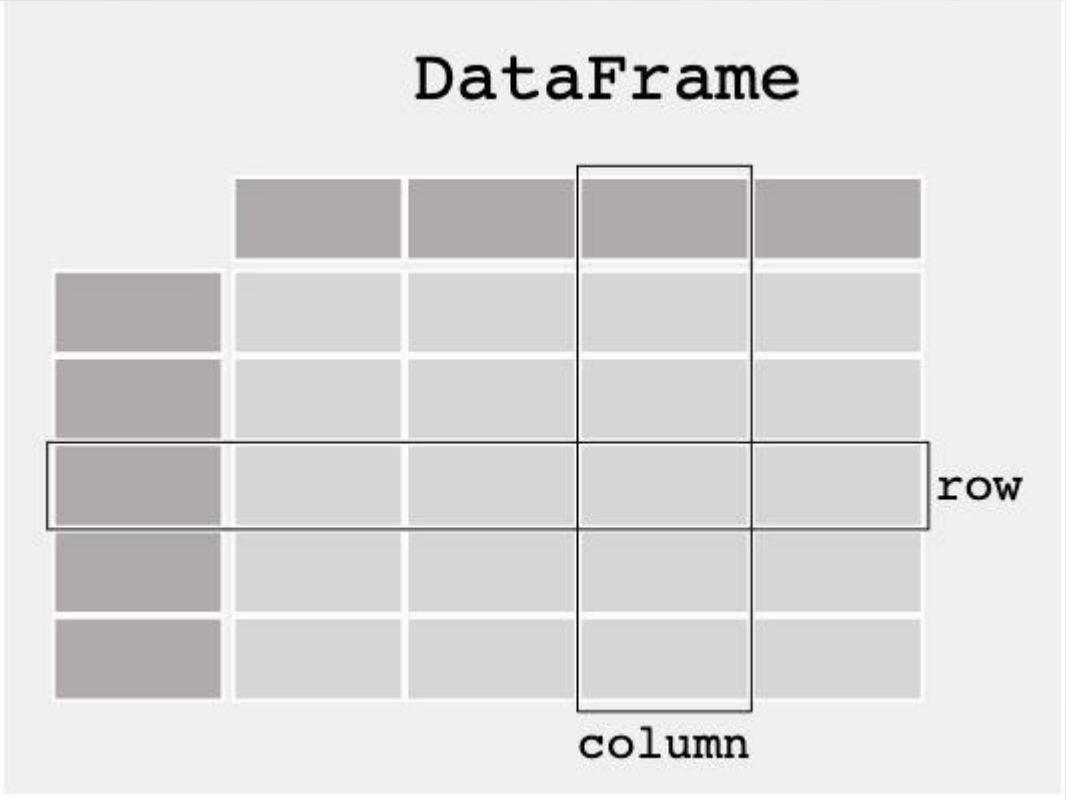
데이터 프레임

데이터 프레임

- Pandas 라이브러리에서 기본적으로 데이터를 다루는 단위는

  DataFrame : spreadsheet와 같은 개념

- Structured Data
- 또는 Panel Data
- 또는 Tabular Data라고 부름



데이터 프레임

데이터 프레임

- 2차원 행렬 데이터에 인덱스를 붙인 것
- 행과 열로 만들어지는 2차원 배열 구조
- R의 데이터 프레임 에서 유래
- 데이프레임의 각 열은 시리즈로 구성되어 있음
- DataFrame()함수를 사용해서 생성

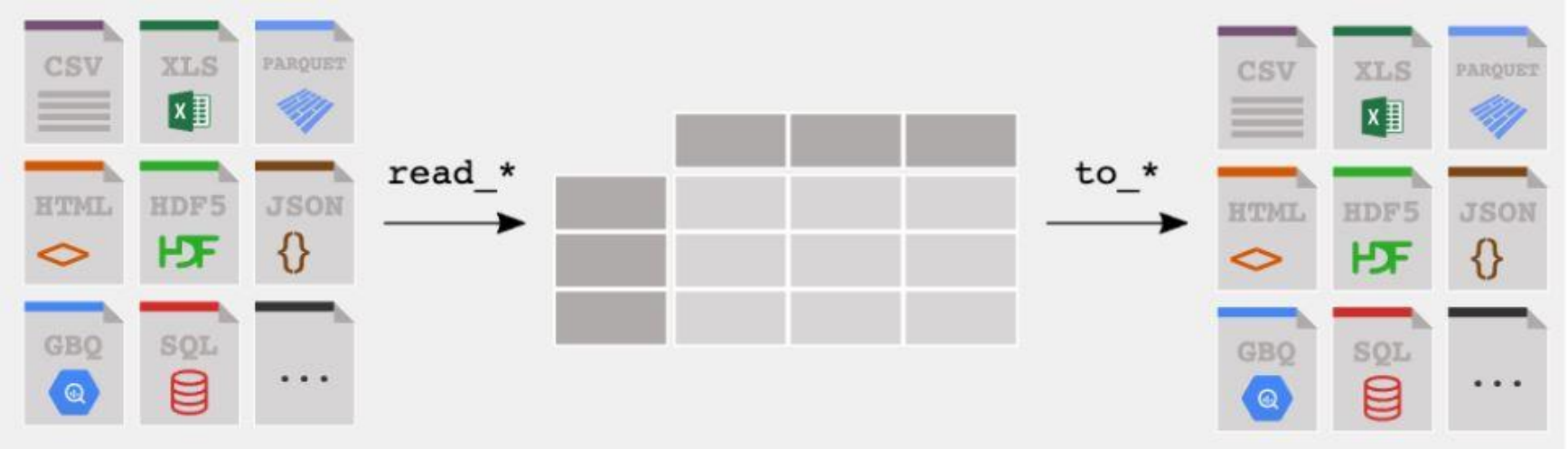
데이터 프레임

외부파일 <-> 데이터 프레임으로 변환

- read_* 함수 사용 ex. csv를 데이터프레임으로 변환해서 가져오기

  pd.read_csv('파일명')

- to_* 함수사용 ex. 데이터 프레임을 csv 파일로 내보내기 데이터프레임.to_csv('파일명')



데이터 프레임 생성

리스트로 데이터 프레임 만들기

- DataFrame([[list1],[list2]]) - 리스트 안에 리스트 형태로 인수를 전달(2차원 리스트 형태로 전달)
- 각 list는 한 행으로 구성됨
- 행의 원소 개수가 다르면 None 값으로 저장
- index 인수값이 없으면 기본 인덱스 (위치인덱스)가 생성됨

```
import pandas as pd
import numpy as np
```

```
df = pd.DataFrame([['a','b','c'],['a','a','g'],['a','a']])
df
```

데이터 프레임 생성

딕셔너리로 데이터프레임 생성

- dict의 key -> column name

```
# 열 데이터를 dict로 작성하는 것이 일반적 임
# 열방향 인덱스(dict의 key), 행방향 인덱스(자동생성 숫자)
df1 = pd.DataFrame(
{
    'A':[90,80,70],
    'B':[85,98,75],
    'C':[88,99,77],
    'D':[87,89,86]
},index = [1,2,3] # 행 인덱스
)
df1
# 딕셔너리의 key는 컬럼네임, index= 인수는 로우네임
```

데이터 프레임 생성

딕셔너리로 데이터프레임 생성

- index 파라미터와 columns 파라미터를 사용해서 df의 의미 전달

```
data = {
    "2015": [9904312, 3448737, 2890451, 2466052],
    "2010": [9631482, 3393191, 2632035, 2000002],
    "2005": [9762546, 3512547, 2517680, 2456016],
    "2000": [9853972, 3655437, 2466338, 2473990],
    "지역": ["수도권", "경상권", "수도권", "경상권"],
    "2010-2015 증가율":[0.0283, 0.0163, 0.0982,0.0141]
}

df3 = pd.DataFrame(data)
df3
```

```
columns =['지역','2000','2005','2010','2015', '2010-2015 증가율']
index = ['서울','부산','인천','대구']
df3 = pd.DataFrame(data, index=index, columns=columns)
df3
```

데이터 프레임 생성

시리즈로 데이터 프레임 생성

- 각 Series의 인덱스 -> columname

```
a = pd.Series([100, 200, 300], ['a', 'b', 'd'])
b = pd.Series([101, 201, 301], ['a', 'b', 'k'])
c = pd.Series([110, 210, 310], ['a', 'b', 'c'])
```

```
pd.DataFrame([a,b,c],index=[100,101,102])
```

데이터 프레임 생성

csv 데이터로 부터 Dataframe 생성

- 데이터 분석을 위해, dataframe을 생성하는 가장 일반적인 방법
- 데이터 소스로부터 추출된 csv(comma separated values) 파일로 부터 생성
- pandas.read_csv 함수 사용

```
# data 출처: https://www.kaggle.com/hesh97/titanicdataset-traincsv/data
train_data = pd.read_csv('./data/train.csv') # 파일경로와 파일명을 정확히 전달해야 함
train_data.head() # df의 위 5행만 출력
```

데이터 프레임 생성

read_csv 함수 파라미터

- sep - 각 데이터 값을 구별하기 위한 구분자(separator) 설정
- header - header를 무시할 경우, None 설정
- index_col - index로 사용할 column 설정
- usecols - 실제로 dataframe에 로딩할 columns만 설정정

```
train_data = pd.read_csv('data/train.csv',
                         index_col = 'PassengerId',
                         usecols= ['PassengerId','Survived','Name','Sex','Age'])
```

```
# df.columns  속성 : df의 컬럼명을 저장하고 있는 속성
train_data.columns
```

인덱스와 컬럼의 이해

인덱스와 컬럼의 이해

- 인덱스(index)
  - index 속성
  - 각 아이템을 특정할 수 있는 고유의 값을 저장
  - 복잡한 데이터의 경우, 멀티 인덱스로 표현 가능

```
# df의 컬럼명(열인덱스)을 확인 - df.columns 속성
print(df3.columns)
type(df3.columns)
```

- 컬럼(column)
  - columns 속성
  - 각각의 특성(feature)을 나타냄
  - 복잡한 데이터의 경우, 멀티 컬럼으로 표현 가능

```
#df의 행 인덱스를 확인 - df.index 속성
print(type(df3.index))
df3.index
```

인덱스와 컬럼의 이해

행/열 인덱스 이름 설정

- index.name
- columns.name

```
# 데이터 프레임내의 데이터만 접근하려면 values 속성을 사용
print(type(df3.values))
df3.values # df의 value는 2차원 ndarray
```

```
type(df3.values[0])
df3.values[0]
```

인덱스와 컬럼의 이해

행/열 인덱스 이름 설정

- index.name
- columns.name

```
# 데이터 프레임내의 데이터만 접근하려면 values 속성을 사용
print(type(df3.values))
df3.values # df의 value는 2차원 ndarray
```

```
type(df3.values[0])
df3.values[0]
```

인덱스와 컬럼의 이해

dataframe 데이터 파악하기

- shape 속성 (row, column)
- describe 함수 - 숫자형 데이터의 통계치 계산
- info 함수 - 데이터 타입, 각 아이템의 개수 등 출력
  - len(train_data) # df의행수
  - print(train_data.size) # df의 값의 개수
  - train_data.shape # df의 행과 열 수
  - train_data.info() # 데이터의 요약(개요)정보 반환
  - train_data.describe() # 수치형 데이터에 대해서만 기본 통계량을 반환

인덱스와 컬럼의 이해

dataframe 데이터 파악하기

- 데이터 프레임 전치
- 판다스 데이터 프레임은 전치를 포함해서 Numpy 2차원 배열의 대부분 속성이나 메서드를 지원함.
- 전치 : 행과 열을 바꾸는 기능
- df.T

```
# 확인 후 설명
df3.T['서울'].values
df3.T['서울']['2000']
```

데이터 프레임 내용 변경

열추가, 열삭제, 내용 갱신

- 해당열이 있으면 내용 갱신, 열이 없으면 추가
- 열추가 : df[열이름(key)]=values
- 열 내용 갱신 : df[열이름(key)]=values
- DF의 행추가
  - pd의 인덱서 사용
  - concate() 사용 - 추가하고자 하는 data를 df로 새로 생성후 결합

데이터 프레임 인덱싱

데이터 프레임 인덱싱

- 열인덱싱
- 인덱서를 사용하지않는 행 인덱싱
- []기호를 이용해서 인덱싱할때 주의점
- []기호는 열 위주 인덱싱이 원칙

데이터 프레임 인덱싱

열인덱싱

- 열 라벨(컬럼명)을 키값으로 생각하고 인덱싱
- 인덱스로 라벨값을 하나 넣으면 시리즈 객체가 반환
- 라벨의 배열이나 리스트를 넣으면 부분적 df 가 반환

```
# 인덱스로 키워드 사용 -  시리즈 형태로 반환
print(df3['지역'])
type(df3['지역'])
```

```
# 열 1개 추출시 . 연산자 사용 가능(시리즈로 반환)
df3.지역
```

```
# 인덱스 값으로 컬럼명의 리스트를 사용하면 반환되는 데이터는 DF
df3[['지역']]
```

데이터 프레임 인덱싱

열인덱싱

- 판다스 데이터 프레임에 열이름(컬럼명)이 문자열일 경우에는
- 수치 인덱스를 사용할 수 없음
- 위치 인덱싱 기능을 사용할 수 없음 : keyerror 발생

```
# 위치적으로 맨 처음 열을 반환받기 위해 위치 인덱스 사용해 봄
try :
    df3[0] # <class 'KeyError'>
except Exception as e :
    print(type(e))

# df3[0]은 컬럼명이 0인 컬럼을 찾으라는 의미인데
# df3에는 컬럼명이 0인 컬럼은 없음 - 위치 인덱스 사용 불가
```

데이터 프레임 인덱싱

행 단위 인덱싱

- 행단위 인덱싱을 하고자 하면 인덱서라는 특수 기능을 사용하지 않는 경우 슬라이싱을 해야 함(인덱서는 바로 뒤에 배움)
- 인덱스 값이 문자(라벨)면 문자슬라이싱도 가능하다.
- 위치값 슬라이싱도 가능

```
# 첫번째 행 추출
df3['서울']
```

```
df3[1:3] # [시작값 : 끝값+1]
```

```
# 키워드 인덱스를 이용
df3['부산':'인천']
```

데이터 프레임 인덱싱

개별요소 접근

- 개별요소 접근 [열][행]

```
df3['2015'] # 시리즈 반환
```

```
df3['2015']['부산']
# 2010 열을 시리즈로 우선 반환받고 시리즈에서 부산 인덱스에 해당하는 값을 반환
```

```
df3[['2015']] # df로 반환
```

```
# df3[['2015']]['부산'] # key에러 발생
# df3[['2015']] 데이터 프레임을 반환하기 때문
```

```
# 부산데이터 추출
df3[['2005','2010']]['부산':'부산']
```

```
### 열 인덱싱에 슬라이싱 사용 - 사용불가
# df3[['2000':'2010']]
```

데이터 프레임 데이터 삭제

행 삭제

- drop() 함수 사용
- drop(index=[삭제할 행 인덱스])
- 원본반영하지 않음
- 삭제와 동시에 원본 반영하려면
- drop(index=[삭제할 행 인덱스], inplace=True)

데이터 프레임 데이터 삭제

drop() 이용 열 삭제

- drop(columns=[삭제할 열])

```
df3.drop(columns=['2010-2015 증가율','2010'])
```

- drop()함수 columns/index 미 표기시
- drop([삭제할행 또는 열],axis=0/1)
- axis : 0(행),1(열)

```
df3.drop(['대구'],axis=0)
```

```
df3.drop(['2010-2015 증가율'],axis=1)
```

데이터 프레임 인덱서

데이터 프레임에서 인덱서 사용

- 데이터 프레임 인덱서 : loc, iloc
- Pandas는 numpy행렬과 같이 쉼표를 사용한 (행 인덱스, 열 인덱스) 형식의 2차원 인덱싱을 지원
- 특별한 인덱서(indexer) 속성을 제공
- loc : 라벨값 기반의 2차원 인덱싱
- iloc : 순서를 나타내는 정수 기반의 2차원 인덱싱

데이터 프레임 인덱서

데이터 프레임에서 인덱서 사용

- 행과 열을 동시에 인덱싱 하는 구조는 기본 자료구조 인덱스와 차이가 있음
- df['열']
- df[:'행'] 슬라이싱이 반드시 필요
- df['열'][:'행']

데이터 프레임 인덱서

데이터 프레임에서 인덱서 사용

- loc, iloc 속성을 사용하는 인덱싱
- pandas 패키지는 [행번호,열번호] 인덱싱 불가
  - iloc 속성 사용하면 가능
  - iloc[행번호,열번호] - 가능
  - loc[행제목,열제목] -가능

데이터 프레임 인덱서

loc 인덱서 : 행 우선 인덱서

- df[열이름값] # 기본 인덱싱, 열우선
- df.loc[행인덱싱 값] #행우선 인덱서
- df.loc[행인덱싱 값,열인덱싱 값]
- 인덱싱 값
  - 인덱스 데이터(index name, column name)
  - 인덱스 데이터 슬라이스
  - 같은 행 인덱스를 갖는 불리언 시리즈(행 인덱싱인 경우)
  - 조건으로 추출 가능
  - 위 값을 반환하는 함수

데이터 프레임 인덱서

loc 인덱서 사용 요소 값 접근

- 인덱싱으로 행과 열을 모두 받는 경우
- 문법 : df.loc[행인덱스,열인덱스] - 라벨(문자열)인덱스 사용

```
# loc인덱서를 사용한 원소값을 변경
# df.loc[행,열] = 값
df.loc['a','A'] = 50
df
```

```
df.loc[['a','b']]['A'] # 시리즈
df.loc[['a','b'],'A'] # 시리즈
df.loc[['a','b'],['A']] # df 반환
```

데이터 프레임 인덱서

loc 인덱서 사용 요소 값 접근

- loc를 이용한 indexing 정리

```
# a행의 모든열 추출
df.loc['a'] # a행 모든열 추출, 시리즈로 반환
df.loc[['a']] # a행의 모든 열 추출, df 반환
df.loc['a',:] # a행의 모든 열 추출, 시리즈
df.loc[['a'],:] # a행의 모든 열 추출, df 반환
```

```
# B행부터 모든행의 A열을 추출
df.loc['b':] #b행부터 모든행
df.loc['b':,'A'] # 시리즈 반환
df.loc['b':]['A'] #시리즈 반환
df.loc['b':][['A']] # df반환
df.loc['b':,['A']] # df반환
df.loc['b':,'A':'A']
```

```
#a행의 B,C열을 추출하시오
df.loc['a'] # a행의 모든열 시리즈로 반환
df.loc['a', 'B':'C'] # 시리즈로 반환
df.loc[['a']] # df로 반환
df.loc[['a'],"B":"C"] # df로 반환

df.loc['a', ['B','C']] # 시리즈로 반환
```

```
# a,b 행의 B,D열을 데이터 프레임으로 반환
df.loc['a':'b']
df.loc[['a','b']]
df.loc[['a','b']][['B','D']]
df.loc[['a','b'],['B','D']]
```

데이터 프레임 인덱서

iloc 인덱서(위치 인덱스)

- 라벨(name)이 아닌 위치를 나타내는 정수 인덱스만 받음
- 위치 정수값은 0부터 시작
- 데이터프레임.iloc[행,열]

```
# 0행 데이터에서 끝에서 두번째 열부터 끝까지 반환
df
df.iloc[0:1,-2:] # df
df.iloc[0,-2:] # 시리즈
```

```
df # 컬럼명과 로우명이 문자형으로 설정됨
df.iloc[0,1]
```

```
# iloc에 슬라이싱 적용 # 행과 열 모두 슬라이싱 적용 - df
df.iloc[0:2,1:2]
```

```
# df 형태 추출 : df.iloc[행슬라이싱, 열슬라이싱]
df.iloc[2:3,1:2]
```

데이터 프레임 인덱서

value_counts()

- 원소들을 분류하여 갯수를 세는 함수 : value_counts()

```
df = pd.DataFrame({'num_legs': [2, 4, 4, 6],
                   'num_wings': [2, 0, 0, 0]},
                  index=['falcon', 'dog', 'cat', 'ant'])
df
```

```
df.num_legs
df.num_legs.value_counts()
```

```
df.num_legs
df.num_legs.value_counts()
```

```
# 동일한 값을 갖는 행의 수를 반환
df.value_counts()
```

pandas 데이터처리 및 변환관련 함수

데이터 개수 세기

- 가장 간단한 분석은 개수를 세기 : count()함수 이용
- NaN값은 세지 않음
- 각 열마다 데이터 개수를 세기때문에 누락된 부분을 찾을 때 유용
- 난수 발생시켜 dataframe 생성
  - 난수 seed(값)라는 함수를 사용할 수 있음
  - seed의 의미 : 난수 알고리즘에서 사용하는 기본 값으로
  - 시드값이 같으면 동일한 난수가 발생함
  - 난수 함수 사용시 매번 고정시켜야 함
  - 계속 변경되는 난수를 받고 싶으면 함수등을 이용해서 시드값이 매번 변하게 작업해야 함. Time.tiem()

pandas 데이터처리 및 변환관련 함수

카테고리 값 세기

- 시리즈의 값이 정수,문자열 등 카테고리 값인 경우에
- 시리즈.value_counts()메서드를 사용해 각각의 값이 나온 횟수를 셀 수 있음
- 파라미터 normalize=True 를 사용하면 각 값 및 범주형 데이터의 비율을 계산
- 시리즈.value_counts(normalize=True)

pandas 데이터처리 및 변환관련 함수

범주형 데이터에 value_counts() 함수 적용

- 범주형 데이터 : 관측 결과가 몇개의 범주 또는 항목의 형태로 나타나는 자료
- ex. 성별(남,여), 선호도(좋다, 보통, 싫다), 혈액형(A,B,O,AB) 등
- 행을 하나의 value로 설정하고 동일한 행이 몇번 나타났는지 반환
- 행의 경우가 인덱스로 개수된 값이 value로 표시되는 Series 반환

pandas 데이터처리 및 변환관련 함수

정렬함수 -데이터 정렬 시 사용

- sort_index(ascending=True/False) : 인덱스를 기준으로 정렬
- ascending 생략하면 오름차순 정렬
- sort_value(ascending=True/False) : 데이터 값을 기준으로 정렬

pandas 데이터처리 및 변환관련 함수

데이터 프레임 정렬

- df.sort_values() : 특정열 값 기준 정렬
- 데이터프레임은 2차원 배열과 동일하기 때문에
- 정렬시 기준열을 줘야 함 (by 인수 사용 : 생략 불가)
- by = 기준열, by=[기준열1,기준열2]
- 오름차순/내림차순 : ascending = True/False (생략하면 오름차순)
- df.sort_index() : DF의 INDEX 기준 정렬
- 오름차순/내림차순 : ascending = True/False (생략하면 오름차순)

pandas 데이터처리 및 변환관련 함수

행/열 합계

- df.sum() 함수 사용
- 행과 열의 합계를 구할때는 sum(axis=0/1) - axis는 0이 기본
- 각 열의 합계를 구할때는 sum(axis=0)
- 각 행의 합계를 구할때는 sum(axis=1)
- # 예제 DF 생성

pandas 데이터처리 및 변환관련 함수

df의 기본 함수

- mean(axis=0/1)
- min(axis=0/1)
- max(axis=0/1)

```
# 각 열의 최소값
df2.min(axis=0)
# 각 행의 최소값
df2.min(axis=1)
```

```
# 각 열의 최대값
df2.max(axis=0)
# 각 행의 최대값
df2.max(axis=1)
```

pandas 데이터처리 및 변환관련 함수

행/열 삭제 - drop() 사용 예제

- df.drop('행이름',0) : 행삭제
- 행삭제 후 df로 결과를 반환
- df.drop('행이름',1) : 열 삭제
- 행삭제 후 df로 결과를 반환
- 원본에 반영되지 않으므로 원본수정하려면 저장 해야 함
- inplace=True

pandas 데이터처리 및 변환관련 함수

NaN 값 처리 함수

- df.dropna(axis=0/1)
- NaN값이 있는 열 또는 행을 삭제
- 원본 반영되지 않음
- df.fillna(0)
- NaN값을 정해진 숫자로 채움
- 원본 반영 되지 않음

pandas 데이터처리 및 변환관련 함수

데이터프레임의 형변환

- df.astype(자료형)

- 전체 데이터에 대해서 형변환을 진행

```
# 결측치를 0으로 채우면서 df의 데이터를 정수형으로 형변환
df2.fillna(0).astype(int)
df2.fillna(0).astype(float)
```


```
# 결측치를 0으로 채우면서 df의 데이터를 정수형으로 형변환
df2.fillna(0).astype(int)
df2.fillna(0).astype(float)
```


```
test_df = df2.fillna(0).astype(float)
test_df
```

```
test_df[1]=test_df[1].astype(int)
test_df
```

pandas 데이터처리 및 변환관련 함수

열 또는 행에 동일한 연산 반복 적용할 때 : apply() 함수

- apply() 함수는 DataFrame의 행이나 열에 복잡한 연산을 vectorizing할 수 있게 해주는 함수로 매우 많이 활용되는 함수임
- 동일한 연산(함수화 되어있어야함)을 모든열에 혹은 모든 행에 반복 적용하고자 할때 사용
- apply(반복적용할 함수, axis=0/1)
  - 0 : 열마다 반복
  - 1 : 행마다 반복
  - 생략시 기본값 : 0

pandas 데이터처리 및 변환관련 함수

데이터프레임의 기본 집계함수

- 데이터프레임의 기본 집계함수(sum, min, max, mean 등)들은 행/열 단위 벡터화 연산을 수행함
- apply() 함수를 사용할 필요가 없음
- 일반적으로 apply() 함수 사용은 복잡한 연산을 해결하기 위한 lambda 함수나 사용자 정의 함수를 각 열 또는 행에 일괄 적용시키기 위해 사용

pandas 데이터처리 및 변환관련 함수

1회성함수 lambda 함수를 apply()에 사용

- 1회성함수 lambda 함수를 apply()에 사용 예제

```
(lambda x : x.max()-x.min())(df3['a'])
```

```
3
```

```
df3.apply(lambda x : x.max()-x.min(),0)
```

pandas 데이터처리 및 변환관련 함수

데이터값을 카테고리 값으로 변환

- 값의 크기를 기준으로하여 카테고리 값으로 변환하고 싶을때

- cut(data,bins,labels)
  - data : 구간 나눌 실제 값,
  - bins : 구간 경계값
  - labels: 카테고리값

- qcut(data,구간수,labels)

```
#구간을 나눌 실제 값 : 관측 데이터
ages=[0,0.5,4,6,4,5,2,10,21,23,37,15,38,31,61,20,41,31,100]
```

```
# data : 구간 나눌 실제 값, bins : 구간 경계값, label: 카테고리값
data = ages

# 구간 경계값
# 구간 최소값 < 구간 <= 구간 최대값
bins = [0,4,18,25,35,60,100]
# 0~4 구간 : 영유아 0살 < 영유아 <=4

# 각 구간의 이름 : labels - 카테고리명
# 순서는 구간(bins)의 순서와 동일해야 함
labels = ['영유아','미성년자','청년','중년','장년','노년']
```

pandas 데이터처리 및 변환관련 함수

Categorical 클래스 객체

- 카테고리명 속성 : Categorical.categories
- 카테고리명 저장
- 코드 속성 : Categorical.codes
- 인코딩한 카테고리 값을 정수로 갖음

pandas 데이터처리 및 변환관련 함수

qcut()

- 구간 경계선을 지정하지 않고 데이터 개수가 같도록 지정한 수의 구간으로 분할하기 : qcut()
- 형식 : pd.qcut(data,구간수,labels=[d1,d2....])
  - 예)1000개의 데이터를 4구간으로 나누려고 한다면
  - qcut 명령어를 사용 한 구간마다 250개씩 나누게 됨
  - 예외)같은 숫자인 경우에는 같은 구간으로 처리함

pandas 데이터처리 및 변환관련 함수

set_index(), reset_index()

- 데이터 프레임 인덱스 설정을 위해 set_index(), reset_index()
- set_index() : 기존 행 인덱스를 제거하고 데이터 열 중 하나를 인덱스로 설정해주는 함수
- reset_index() : 기존 행인덱스를 제거하고 기본인덱스로 변경
- 기본인덱스 : 0부터 1씩 증가하는 정수 인덱스
  - 따로 설정하지 않으면 기존 인덱스는 데이터열로 추가 됨

데이터 프레임 병합

데이터 프레임 병합

- pandas는 두개 이상의 데이터 프레임을 하나로 합치는
- 병합(merge)이나 연결(concate)을 지원
- merge 명령을 사용한 데이터 프레임 병합
- merge :
  - 두개의 데이터 프레임의 공통 열 이나 인덱스를 기준으로
  - 두개의 데이터프레임을 합친다.
  - 이때 기준이되는 열 데이터를 key라고 부른다

데이터 프레임 병합

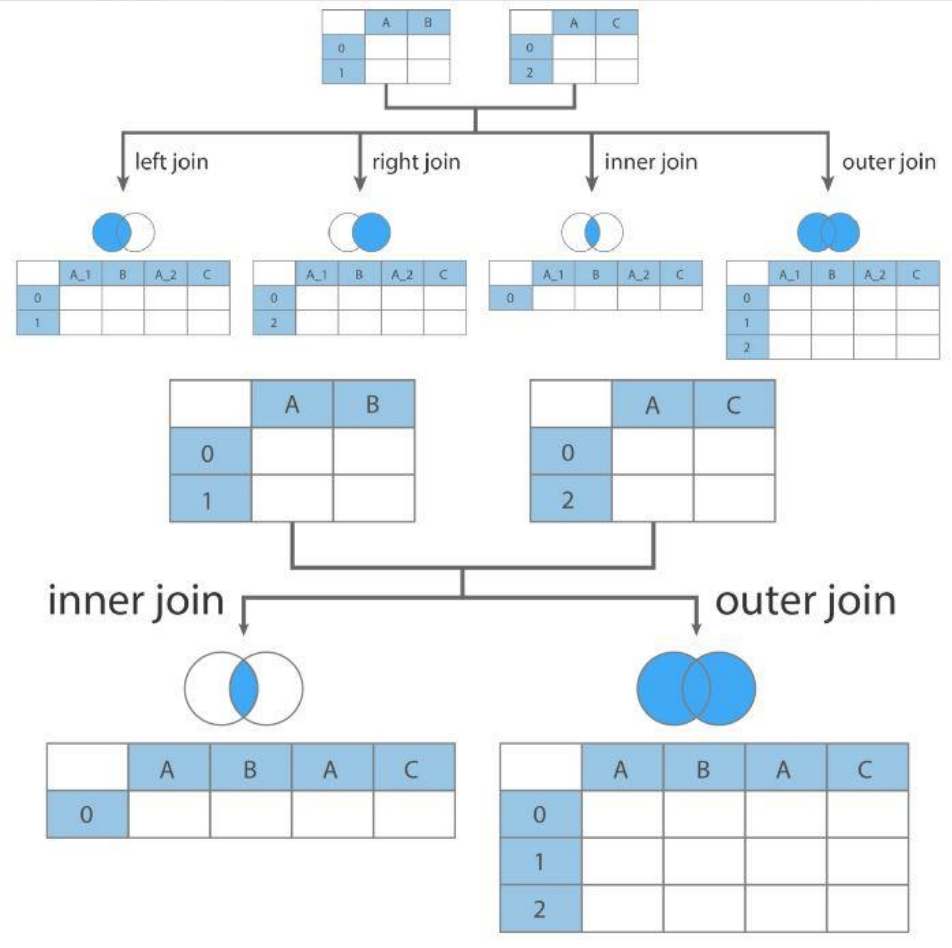
merge 명령을 사용한 데이터 프레임 병합

- df.merge(df1) : 두 df를 병합
- 기본은 inner join : 양쪽에 동일하게 존재하는 키만 표시
- key : 기준열을 의미
  - 실제 데이터 필드거나 행 인덱스
- 병합방식
  - inner join : 양쪽 df에서 모두 키가 존재하는 data만 표시
  - outer join : 한쪽에만 키가 존재하면 data를 표시
  - 병합방식을 설정 : how=inner(생략가능), how=outer

데이터 프레임 병합

merge 명령을 사용한 데이터 프레임 병합

- pandas는 두개 이상의 데이터 프레임을 하나로 병합



데이터 프레임 병합

merge 명령을 사용한 데이터 프레임 병합

- merge 명령으로 두 df를 병합하는 문법
  - 모든 인수 생략(병합 df를 제외한) 공통 이름을 갖고 있는 열
  - '고객번호'가 키가 됨
  - 양쪽에 모두 존재하는 키의 data만 보여주는 inner join 방식을 사용
- outer join 방식은 키 값이 한쪽에만 있어도 데이터를 보여 줌
  - pd.merge(df1,df2, how = 'outer')
  - 어느 한 df에 데이터가 존재하지 않으면 NaN으로 표시됨

데이터 프레임 병합

merge 명령을 사용한 데이터 프레임 병합

- how = inner/outer/left/right
- how=left : 왼쪽 df에 있는 모든 키의 데이터는 표시
- how=right : 오른쪽 df 에 있는 모든 키의 데이터는 표시
- 동일한 키 값이 있는 경우
  - 키값이 같은 데이터가 여러개 있는 경우에는 있을 수 있는 모든 경우의 수를 따져서 조합을 만들어 냄

데이터 프레임 병합

merge 명령을 사용한 데이터 프레임 병합

- key
  - 두 데이터 프레임에서 이름이 같은 열은 모두 키가 될 수 있음
  - 이름이 같아도 키가되면 안되는 열이 있으면 on 인수로 기준열을 명시
- 기준열을 직접 지정 : on=기준열 이름
  - 반환 결과에 동일 필드명이 있을경우에는 필드명_x, 필드명_y로 필드명을 변경해서 표현한다

데이터 프레임 병합

merge 명령을 사용한 데이터 프레임 병합

- 일반 데이터 열이 아닌 인덱스를 기준으로 merge 할 수 도 있음
  - 인덱스를 기준열로 사용하려면
  - left_index = True 또는
  - right_index = True 설정을 하게 됨
- 양쪽 데이터프레임에 key가 모두 인덱스 인 경우
- merge 명령어 대신 join 메서드를 사용가능
  - 사용 방법은 동일

데이터 프레임 병합

concat 명령을 사용한 데이터 연결

- pd.concat(objs, # Series, DataFrame, Panel object

  axis=0, # 0: 위+아래로 합치기, 1: 왼쪽+오른쪽으로 합치기

  join='outer', # 'outer': 합집합(union), 'inner': 교집합(intersection)

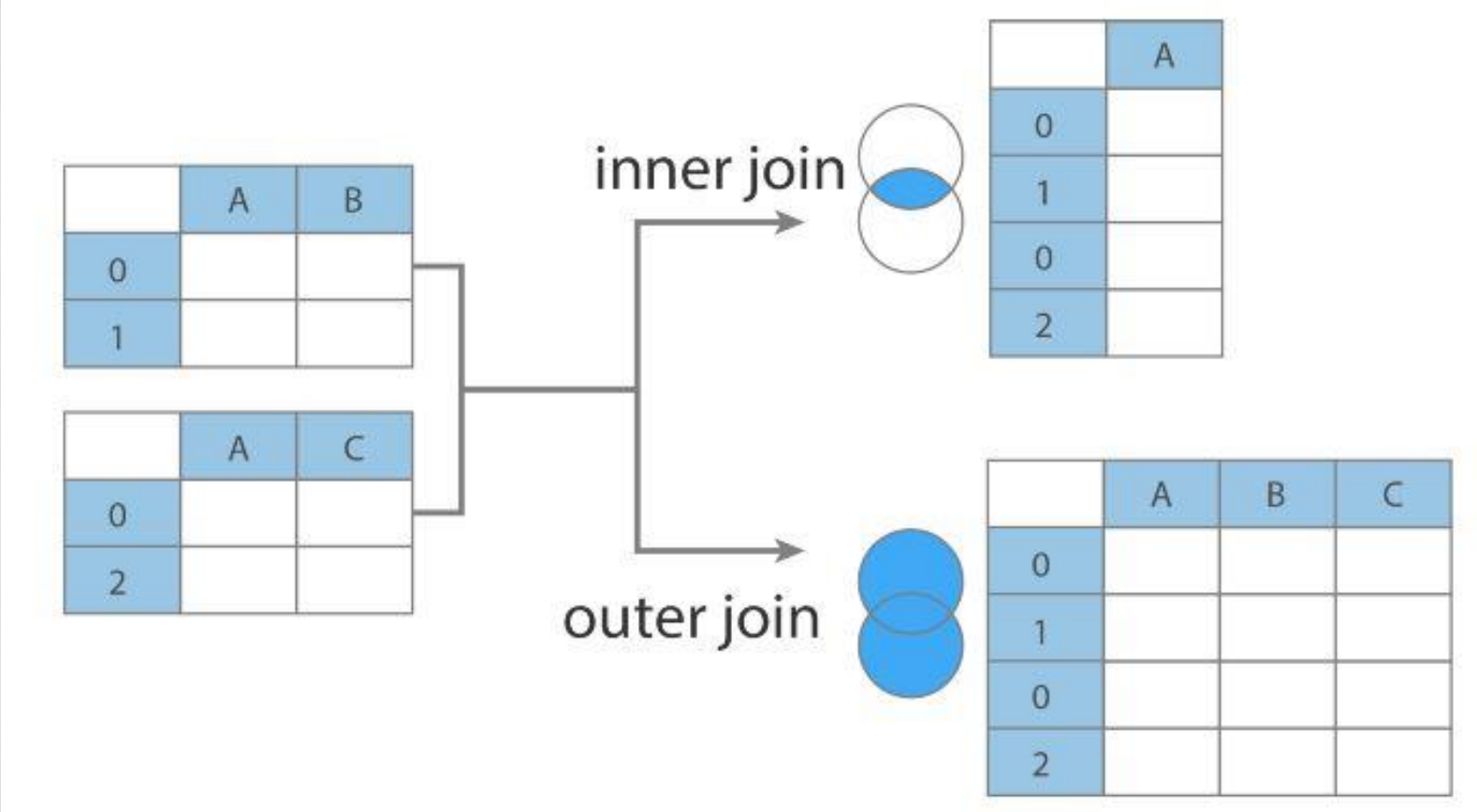
  ignore_index=False, # False: 기존 index 유지, True: 기존 index 무시

  keys=None, # 계층적 index 사용하려면 keys 튜플 입력)
- concat 명령을 사용하면 기준열 없이 데이터를 연결
- 기본은 위 아래로 데이터 행 결합(row bind) axis 속성을 1로 설정하면 열 결합(column bind)을 수행
- 단순히 두 시리즈나 데이터프레임을 연결하기 때문에 인덱스 값이 중복

데이터 프레임 병합

concat 명령을 사용한 데이터 연결

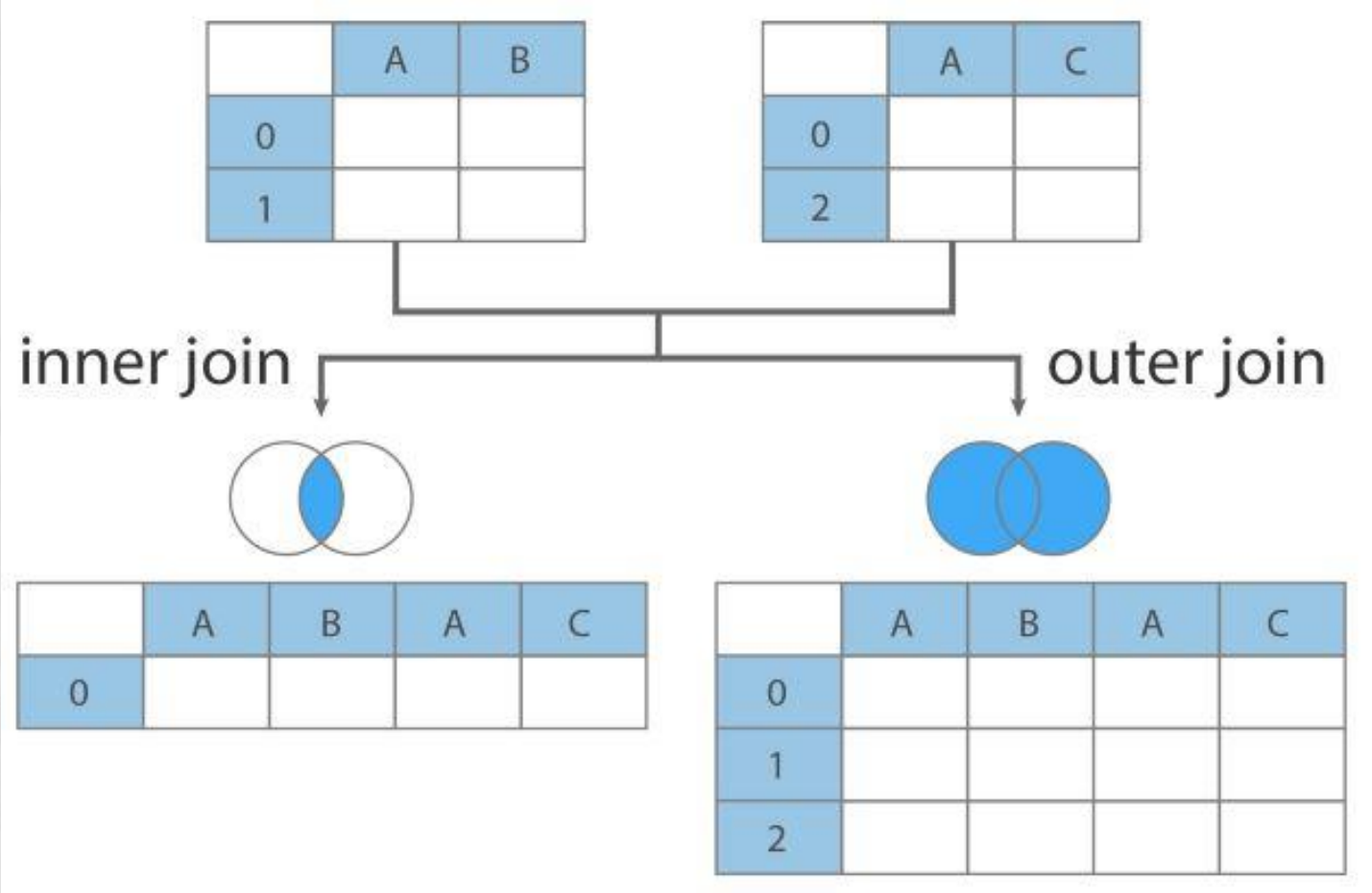
- pd.concat([df1,df2],axis=0)



데이터 프레임 병합

concat 명령을 사용한 데이터 연결

- pd.concat([df1,df2],axis=1)



데이터 프레임 병합

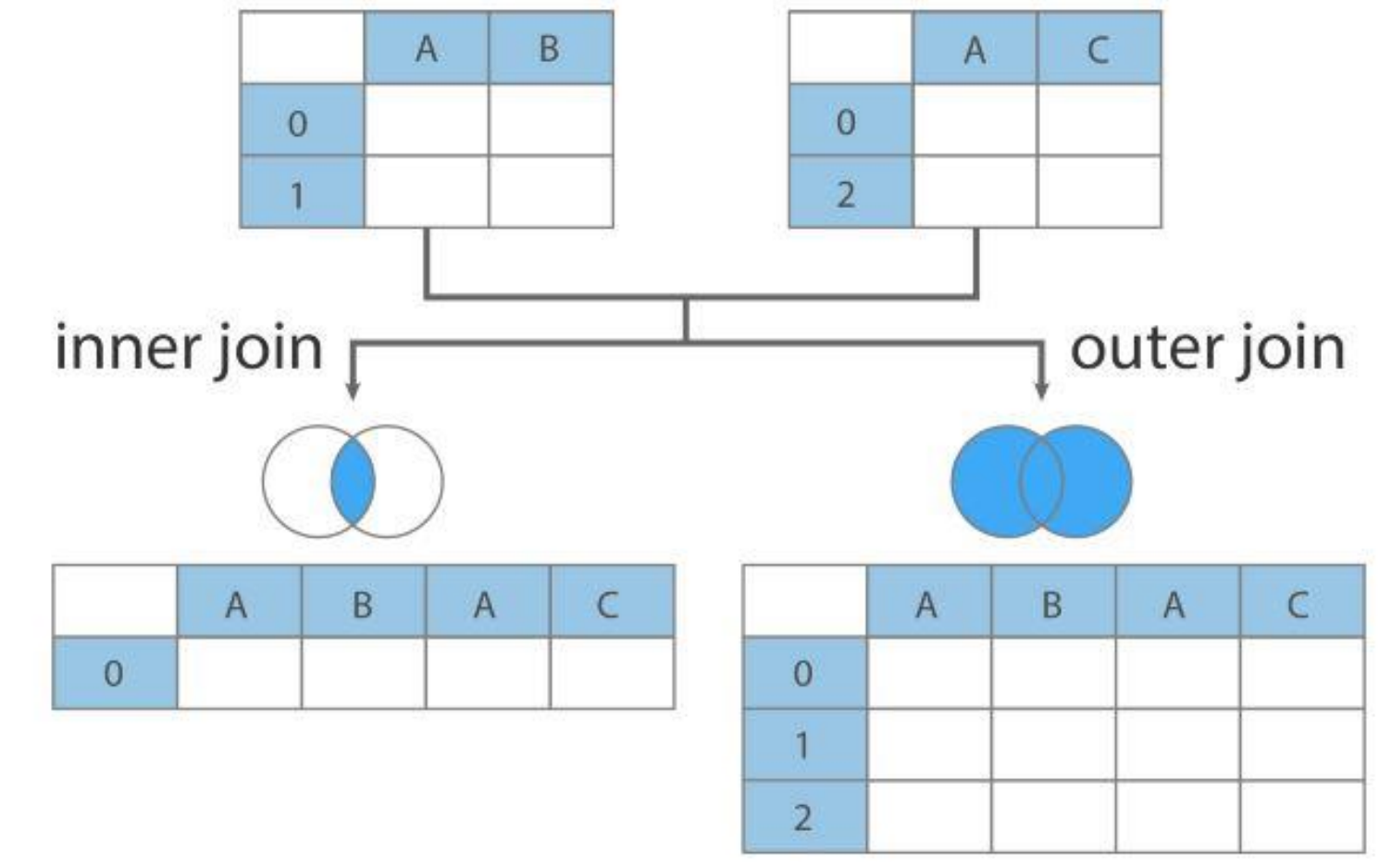
concate를 이용한 열 병합

- axis=1 설정
- pd.concat([df1,df2],axis=1,join='inner/outer')
- 데이터프레임들의 열을 결합
- 모든행을 표시하고 해당 행의 데이터가 없는 열의 원소는 NaN으로 표시된다 : 기본설정(join='outer')
- 병합하는 데이터프레임에 중복되는 인덱스의 행만 표시 : join='inner'

데이터 프레임 병합

concate를 이용한 열 병합

- concate를 이용한 병합



피봇테이블

피봇테이블

- 가지고 있는 데이터원본을 원하는 형태의 가공된 정보를 보여주는 것
- 자료의 형태를 변경하기 위해 많이 사용하는 방법
- 방법 : 두개의 키를 사용해서 데이터를 선택
- pivot_table(data,values=None,index=None,columns=None,aggfunc='mean',margins=False,margins_name='All')
  - data : 분석할 데이터 프레임. 메서드 형식일때는 필요하지 않음 ex)df1.pivot_table()
  - values : 분석할 데이터 프레임에서 분석할 열
  - index : 행 인덱스로 들어갈 키열 또는 키열의 리스트
  - columns : 열 인덱스로 들어갈 키열 또는 키열의 리스트
  - fill_value : NaN이 표출될 때 대체값 지정
  - margins : 모든 데이터를 분석한 결과를 행으로 표출할 지 여부
  - margins_name : margins가 표출될 때 그 열(행)의 이름

피봇테이블

피봇테이블

- 피봇테이블을 작성할 때 반드시 설정해야 되는 인수
  - data : 사용 데이터 프레임
  - index : 행 인덱스로 사용할 필드(기준 필드로 작용됨)
  - 인덱스 명을 제외한 나머지 값(data)은 수치 data 만 사용함
  - 기본 함수가 평균(mean)함수 이기 때문에 각 데이터의 평균값이 반환

그룹 분석

그룹 분석

- 만약 키가 지정하는 조건에 맞는 데이터가 하나 이상이라서 데이터 그룹을 이루는 경우에는 그룹의 특성을 보여주는 그룹분석(group analysis)을 해야 함
- 그룹분석은 피봇테이블과 달리 키에 의해서 결정되는 데이터가 여러개가 있을 경우 미리 지정한 연산을 통해 그 그룹 데이터의 대표값을 계산 하는 것
- 판다스에서는 groupby 메서드를 사용하여 아래 내용 처럼 그룹분석을 진행
- 분석하고자 하는 시리즈나 데이터프레임에 groupby 메서드를 호출하여 그룹화 수행
- 그룹 객체에 대해 그룹연산을 수행

그룹 분석

groupby 메서드

- groupby 메서드는 데이터를 그룹 별로 분류하는 역할을 함
- groupby 메서드의 인수
  - 열 또는 열의 리스트
  - 행 인덱스
- 연산 결과로 그룹 데이터를 나타내는 GroupBy 클래스 객체를 반환
  - 이 객체에는 그룹별로 연산을 할 수 있는 그룹연산 메서드가 있음

그룹 분석

GroupBy 클래스 객체의 그룹연산 메서드

- size, count: 그룹 데이터의 갯수
- mean, median, min, max: 그룹 데이터의 평균, 중앙값, 최소, 최대
- sum, prod, std, var, quantile : 그룹 데이터의 합계, 곱, 표준편차, 분산, 사분위수
- first, last: 그룹 데이터 중 가장 첫번째 데이터와 가장 나중 데이터

그룹 분석

이 외에도 많이 사용되는 그룹 연산

- agg, aggregate
  - 만약 원하는 그룹연산이 없는 경우 함수를 만들고 이 함수를 agg에 전달
  - 또는 여러가지 그룹연산을 동시에 하고 싶은 경우 함수 이름 문자열의 리스트를 전달
- describe
  - 하나의 그룹 대표값이 아니라 여러개의 값을 데이터프레임으로 구함
- apply
  - describe 처럼 하나의 대표값이 아닌 데이터프레임을 출력하지만 원하는 그룹연산이 없는 경우에 사용
- transform
  - 그룹에 대한 대표값을 만드는 것이 아니라 그룹별 계산을 통해 데이터 자체를 변형

그룹 분석

그룹 함수 예제

- apply()/agg()
  - DF 등에 벡터라이징 연산을 적용하는 함수(axis = 0/1 이용하여 행/열 적용가능)
  - agg 함수는 숫자 타입의 스칼라만 리턴하는 함수를 적용
    - apply의 특수한 경우
  - 스칼라 : 하나의 수치(數値)만으로 완전히 표시되는 양. 방향의 구별이 없는 물리적 수량임. 질량·에너지·밀도(密度)·전기량(電氣量) 따위.

그룹 분석

그룹함수 및 피봇 테이블 이용 간단한 분석 예제

- 식당에서 식사 후 내는 팁(tip)과 관련된 데이터이용
- seaborn 패키지 내 tips 데이터셋 사용
  - total_bill: 식사대금
  - tip: 팁
  - sex: 성별
  - smoker: 흡연/금연 여부
  - day: 요일
  - time: 시간
  - size: 인원

데이터 시각화(Matplotlib)

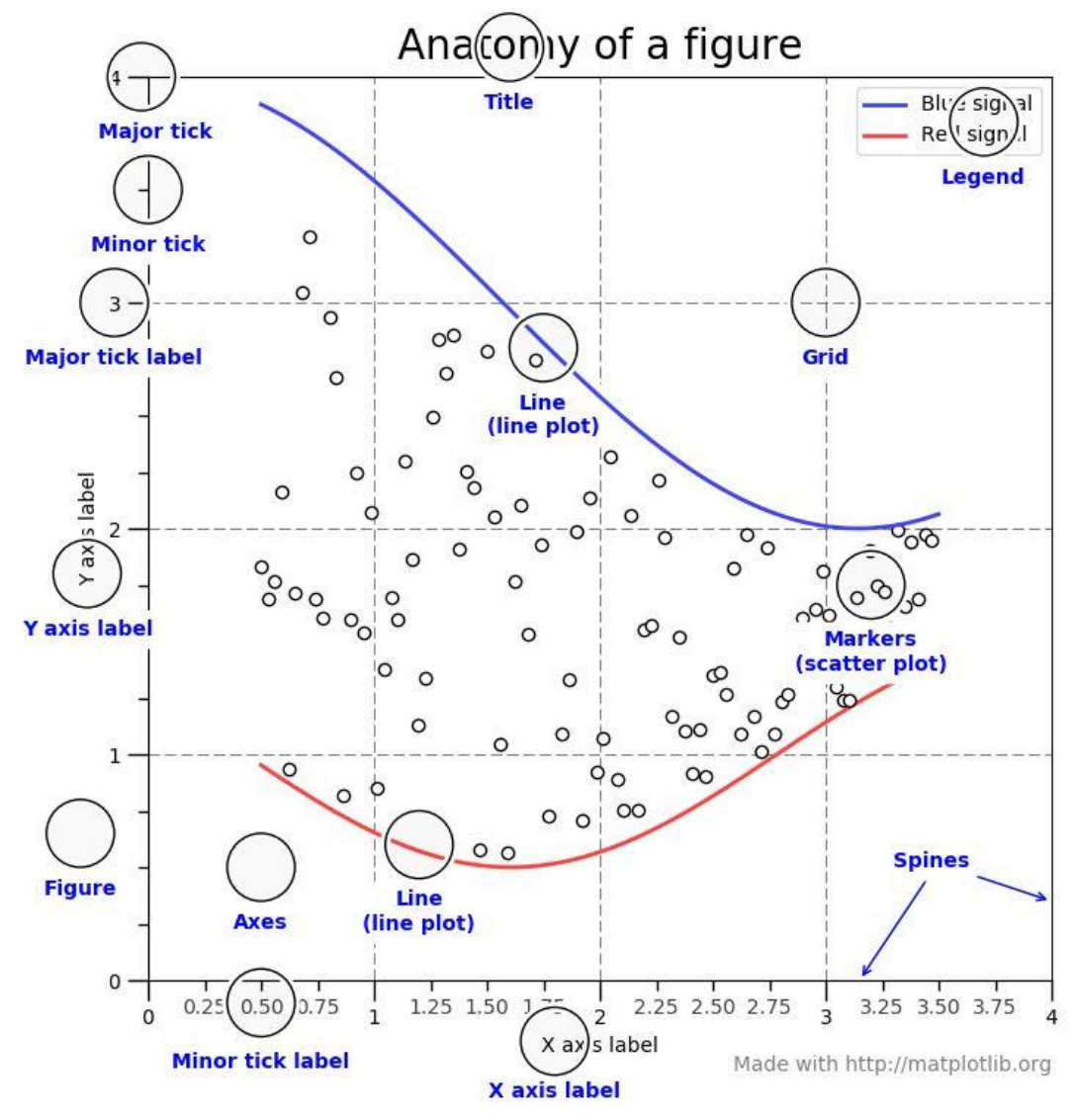
Matplotlib

Matplotlib

- 시각화 패키지
- 파이썬 표준 시각화 도구로 불림
- 2D 평면 그래프에 관한 다양한 포맷과 기능 지원
- 데이터 분석 결과를 시각화 하는데 필요한 다양한 기능을 제공
- .matplotlib 주 패키지 사용시
  - import matplotlib as mpl
- pylab 서브 패키지 사용시 : 주로 사용
  - import matplotlib.pyplot as plt
- 매직 명령어 %matplotlib inline
  - 주피터 노트북 사용시 노트북 내부에 그림을 표시하도록 지정하는 명령어

Matplotlib

그래프 용어 정리



https://matplotlib.org/

라인 플롯 : plot()

plot()

- 기본으로 선을 그리는 함수
- 데이터가 시간, 순서 등에 따라 변화를 보여주기 위해 사용
- show()
  - 각화명령(그래프 그리는 함수) 후 실제로 차트로 렌더링 하고 마우스 이벤트등의 지시를 기다리는 함수
  - 주피터 노트북 에서는 셀 단위로 플롯 명령을 자동으로 렌더링 주므로 show 명령이 필요 없지만
  - 일반 파이썬 인터프리터(파이참)로 가동되는 경우를 대비해서 항상 마지막에 실행하도록 함

https://matplotlib.org/

라인 플롯 : plot()

plot()

- 관련 함수 및 속성
  - figure(x,y) : 그래프 크기 설정 : 단위 인치
  - title() : 제목 출력
  - xlim(범위값): x 축 범위
  - ylim(범위값) : y 축 범위
  - xticks():yticks() : 축과 값을 잇는 실선
  - legend() : 범례
  - xlabel() : x축라벨(값)
  - ylabel() : y축라벨(값)
  - grid() : 그래프 배경으로 grid 사용 결정 함수

https://matplotlib.org/

라인 플롯 : plot()

plot()

- line plot 에서 자주 사용되는 스타일 속성(약자로도 표기 가능)
  - color:c(선색깔)
  - linewidth : lw(선 굵기)
  - linestyle: ls(선스타일)
  - marker:마커 종류
  - markersize : ms(마커크기)
  - markeredgecolor:mec(마커선색깔)
  - markeredgewidth:mew(마커선굵기)
  - markerfacecolor:mfc(마커내부색깔)

https://matplotlib.org/

라인 플롯 : plot()

plot()

- 라인스타일 기호 지정

지정자	선 스타일
'-'	실선(디폴트 값)
'--'	파선
':'	점선
'-.'	일 점 쇄선

https://matplotlib.org/

라인 플롯 : plot()

plot()

- color(c) : 선색깔
- linewidth(lw) : 선굵기
- linestyle(ls) : 선스타일
- marker : 마커의 종류
- markersize(ms) : 마커의 크기
- markeredgecolor(mec) : 마커 선 색깔
- markeredgewidth(mew) : 마커 선 굵기
- markerfacecolor(mfc) : 마커 내부 색깔

https://matplotlib.org/

라인 플롯 : plot()

그래프 제목 및 축 레이블 설정

- plot.title(data,loc=, pad=, fontsize=)
- loc= 'right'|'left'| 'center'| 'right'로 설정할 수 있으며 디폴트는 'center'
- pad=point 은 타이틀과 그래프와의 간격 (오프셋)을 포인트(숫자) 단위로 설정
- fontsize=제목폰트크기
- plot.xlabel()
- plot.ylabel()

라인 플롯 : plot()

subplot()

- subplot() : 하나의 윈도우(figure)안에여러개의 플롯을 배열 형태로 표시
  - 그리드 형태의 Axes 객체들 생성
- 형식 : subplot(인수1,인수2,인수3)
- 인수1 과 인수2는 전체 그리드 행렬 모양 지시
- 인수3 : 그래프 위치 번호
  - subplot(2,2,1) 가 원칙이나 줄여서 221로 쓸 수 있음
  - subplot(221) 2행 2열의 그리드에서 첫번째 위치
  - subplot(224) 2행 2열의 그리드에서 마지막 위치
- tight_layout(pad=) : 플롯간 간격을 설정
  - pad = 간격값(실수)

라인 플롯 : plot()

subplot()

- plt.subplots(행,열)
  - 여러개의 Axes 객체를 동시에 생성해주는 함수
  - 행렬 형태의 객체로 반환
- 두개의 반환값이 있음 :
  - 첫번 째 반환은 그래프 객체 전체 이름 - 거의 사용하지 않음
  - 두번 째 반환값에 Axes 객체를 반환 함
  - 두번 째 반환값이 필요하므로 반환 값 모두를 반환받아 두번 째 값을 사용해야 함
  - ex. fig, axes = plt.subplots(2,2)

라인 플롯 : plot()

범례(legend)표시

- Location String

Location String	Location Code	설명
'best'	0	그래프의 최적의 위치에 표시합니다. (디폴트)
'upper right'	1	그래프의 오른쪽 위에 표시합니다.
'upper left'	2	그래프의 왼쪽 위에 표시합니다.
'lower left'	3	그래프의 왼쪽 아래에 표시합니다.
'lower right'	4	그래프의 오른쪽 아래에 표시합니다.
'right'	5	그래프의 오른쪽에 표시합니다.
'center left'	6	그래프의 왼쪽 가운데에 표시합니다.
'center right'	7	그래프의 오른쪽 가운데에 표시합니다.
'lower center'	8	그래프의 가운데 아래에 표시합니다.
'upper center'	9	그래프의 가운데 위에 표시합니다.
'center'	10	그래프의 가운데에 표시합니다.

https://matplotlib.org/

세로 막대 그래프 그리기: bar()

bar()

- bar(x,y,color=[],alpha=)
- color = [] : 색상값 설정
- alpha = 투명도 설정
- barh(x,y,color=[], alpha=) #가로 막대 그래프 그리기
- 데이터 프레임으로 바 그래프 그리기

스캐터 플롯(scatter plot) : scatter()

scatter()

- 버블차트 : 점 하나의 크기 또는 색상을 이용해서 서로 다른 데이터 값을 표시하는 그래프
  - s 인수 : size
  - c 인수 : color

```
#color map 을 이용해서 그래프 그리기
colormap = t

plt.scatter(t,y,s=50,c=colormap,marker='>')

plt.colorbar() # 색상값의 가중치를 bar로 출력
plt.show()
```

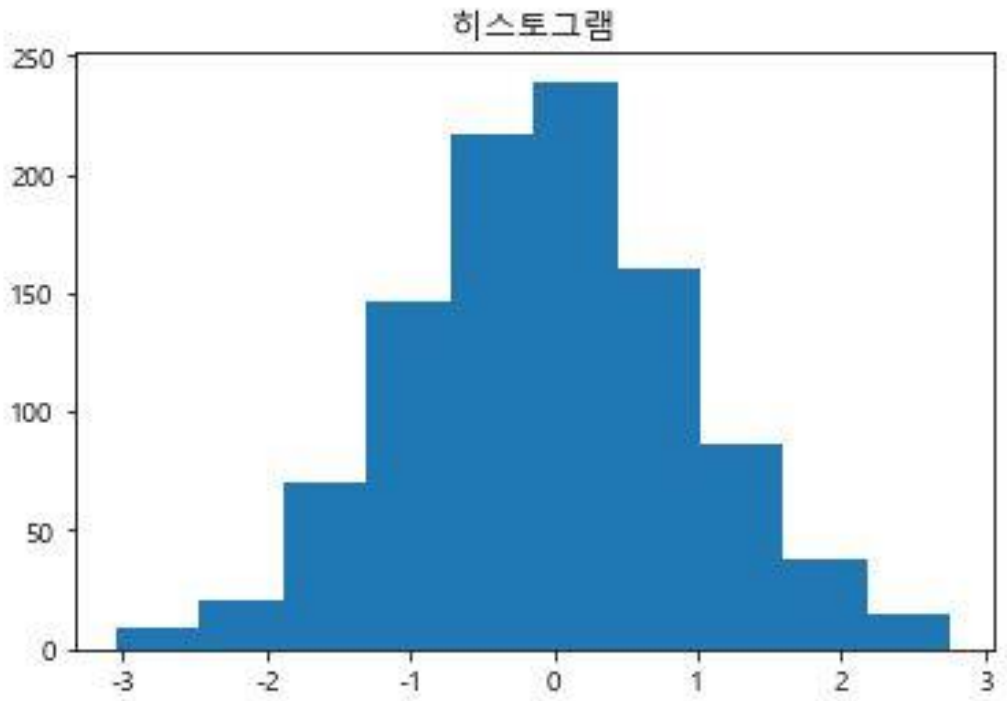

https://matplotlib.org/

히스토그램 : hist()

hist()

- hist()

```
np.random.seed(0)
x=np.random.randn(1000) #난수 1000개 발생
plt.title('히스토그램')
plt.hist(x)
plt.show()
```



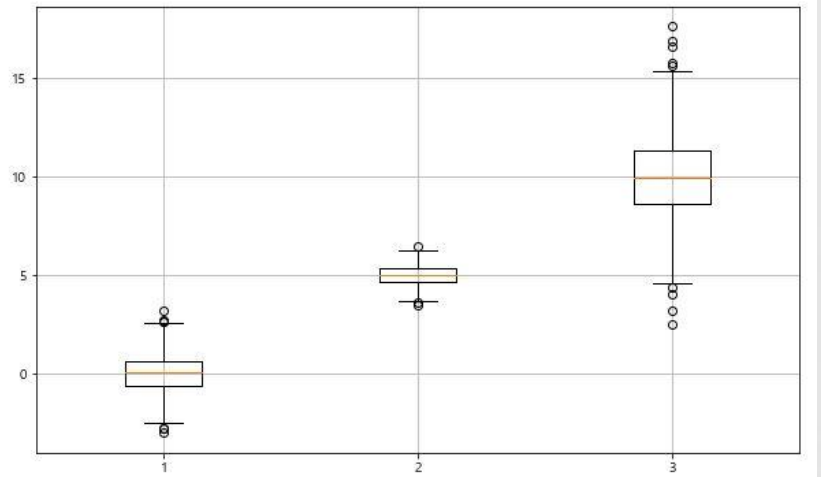
https://matplotlib.org/

박스플롯 : boxplot()

boxplot()

- 다차원 array 형태로 무작위 샘플을 생성
- np.random.normal(정규분포평균,표준편차,(행열) or 개수)
- 정규분포 확률 밀도에서 표본 추출해주는 함수

```
#박스 그래프
plt.figure(figsize=(10,6))
plt.boxplot([s1,s2,s3])
plt.grid()
plt.show()
```

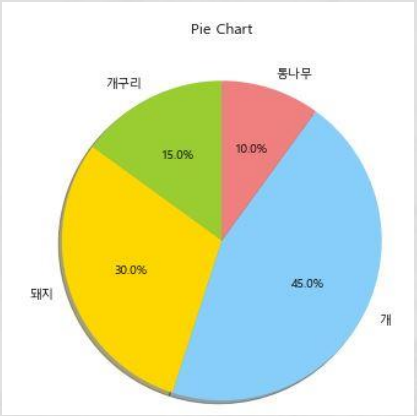


https://matplotlib.org/

파이차트:pie()

pie()

- 카테고리 별 값의 상대적인 비교를 할때 주로 사용하는 차트
- 원의 형태를 유지할 수 있도록 다음 명령을 실행해야 함.
- 콘솔에서는 별 다른 변화 없음. plot 창에서는 필요함
- plt.axis('equal')




```
#예제 데이터 생성
labels=['개구리','돼지','개','통나무']
size=[15,30,45,10]
colors=['yellowgreen','gold','lightskyblue','lightcoral']
explode=(0,0.4,0,0.5)


plt.figure(figsize=(10,6))
plt.title('Pie Chart')
plt.pie(size,  labels=labels, colors=colors,
        autopct= '%1.1f%%', shadow=True,startangle=90)
```


https://matplotlib.org/

seaborn 패키지

seaborn 패키지: 좀 더 편리한 시각화 도구

- matplotlib의 기능에 스타일을 확장한 파이썬 시각화 도구
- 패키지는 설치 해야 함(pip install seaborn) 단, 아나콘다 내부에는 이미 설치되어 있음
- seaborn을 사용하기 위해서는 matplotlib가 import 되어 있어야 함.
- 스타일 색상을 지원하기위해서 파렛트를 지원
- 그래프는 대부분 matplotlib에 있는 그래프를 사용하고, seaborn이 몇개의 그래프를 더 가지고 있음

seaborn 패키지

seaborn 스타일링

- set_style: background color, grid, spine, tick을 정의하여 그림의 전반적인 모양을 스타일링
- set_context: 프리젠테이션이나 보고서와 같은 다양한 매체에 활용할 수 있도록 스타일링
- set_style: 그림의 전반적인 모양 스타일링

seaborn 패키지

seaborn 스타일링

- set_style: 그림의 전반적인 모양 스타일링
  - Built-in Themes (내장 테마) 활용하기
  - Seaborn에는 5가지 기본 제공 테마가 있음
  - darkgrid, whitegrid, dark, white, ticks.
  - 기본값은 darkgrid이지만, 원하는대로 변경이 가능

seaborn 패키지

seaborn 스타일링

- sns.despine() 함수
  - 축선 표시 여부 결정하는 함수(right=T/F,left=T/F,top=T/F,bottom=T/F)
  - 각 속성값이
  - True 면 : 표시하지 않는다.
  - False 면: 표시함
- sns.set_context() 함수
  - 전체 스케일 조정
  - paper, notebook, talk, poster. 여기서 기본값은 notebook이다.
  - 스케일로도 전반적인 사이즈를 조정할 수 있지만 실제 폰트 크기를 조정하는 파라미터는 별도로 있음
  - sns.set_context() 안에 font_scale 사용

sns 의 기본 데이터 셋

Distribution Plot

- Distribution Plot은 데이터의 분포를 시각화하는데 도움이 됨
- 이 그래프를 사용하여 데이터의 평균(mean), 중위수(median), 범위(range), 분산(variance), 편차(deviation) 등을 이해 할 수 있음

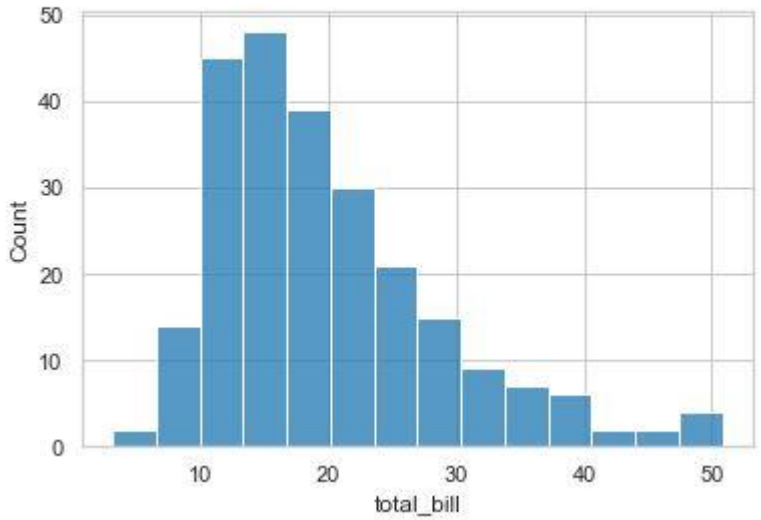
sns 의 기본 데이터 셋

Hist Plot

- 변수에 대한 히스토그램을 표시
- 하나 혹은 두 개의 변수 분포를 나타내는 전형적인 시각화 도구로 범위에 포함되는 관측수를 세어 표시

```
sns.histplot(x=tips['total_bill'])
```

```
<AxesSubplot:xlabel='total_bill', ylabel='Count'>
```



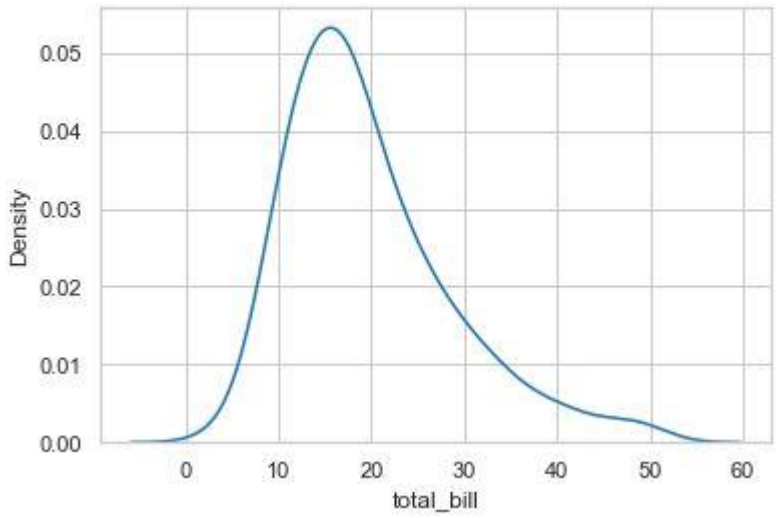
sns 의 기본 데이터 셋

KDE Plot

- 하나 혹은 두 개의 변수에 대한 분포를 그림
- histplot은 절대량이라면 kdeplot은 밀도 추정치를 시각화
- 결과물로는 연속된 곡선의 그래프를 얻을 수 있음

```
sns.kdeplot(x=tips['total_bill'])
```

```
<AxesSubplot:xlabel='total_bill', ylabel='Density'>
```



sns 의 기본 데이터 셋

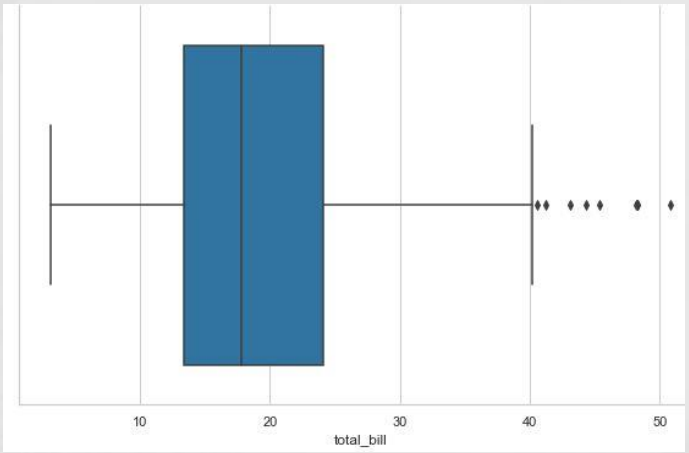
Box Plot

- 최대(maximum), 최소(minimum), mean(평균), 1 사분위수(first quartile), 3 사분위수(third quartile)를 보기 위한 그래프
- 특이치(outliar)를 발견하기에도 좋음
- 단일 연속형 변수에 대해 수치를 표시하거나, 연속형 변수를 기반으로 서로 다른 범주형 변수를 분석할 수 있음

```
#tips 데이터를 이용한 box plot 그리기
#seaborn 패키지의 boxplot() 이용해서 그리기

sns.set_style('whitegrid')

plt.figure(figsize=(10,6))
sns.boxplot(x=tips['total_bill'])
sns.despine()
plt.show()
```



sns 의 기본 데이터 셋

Box Plot

- tips 데이터를 이용한 box plot 그리기
- seaborn 패키지의 boxplot() 이용해서 그리기

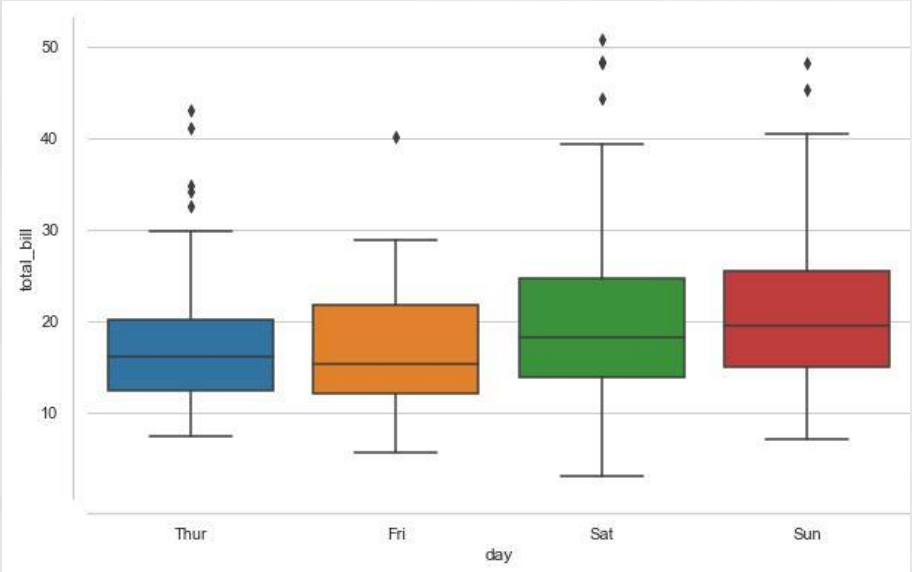
```
#tips 데이터를 이용한 box plot 그리기
#seaborn 패키지의 boxplot() 이용해서 그리기
#요일별 영수증 금액의 분포 확인

sns.set_style('whitegrid')

plt.figure(figsize=(10,6))

sns.boxplot(x=tips['day'], y=tips['total_bill'])
sns.despine(offset=10) # x,y축간의 간격을 설정하는 인수

plt.show()
```

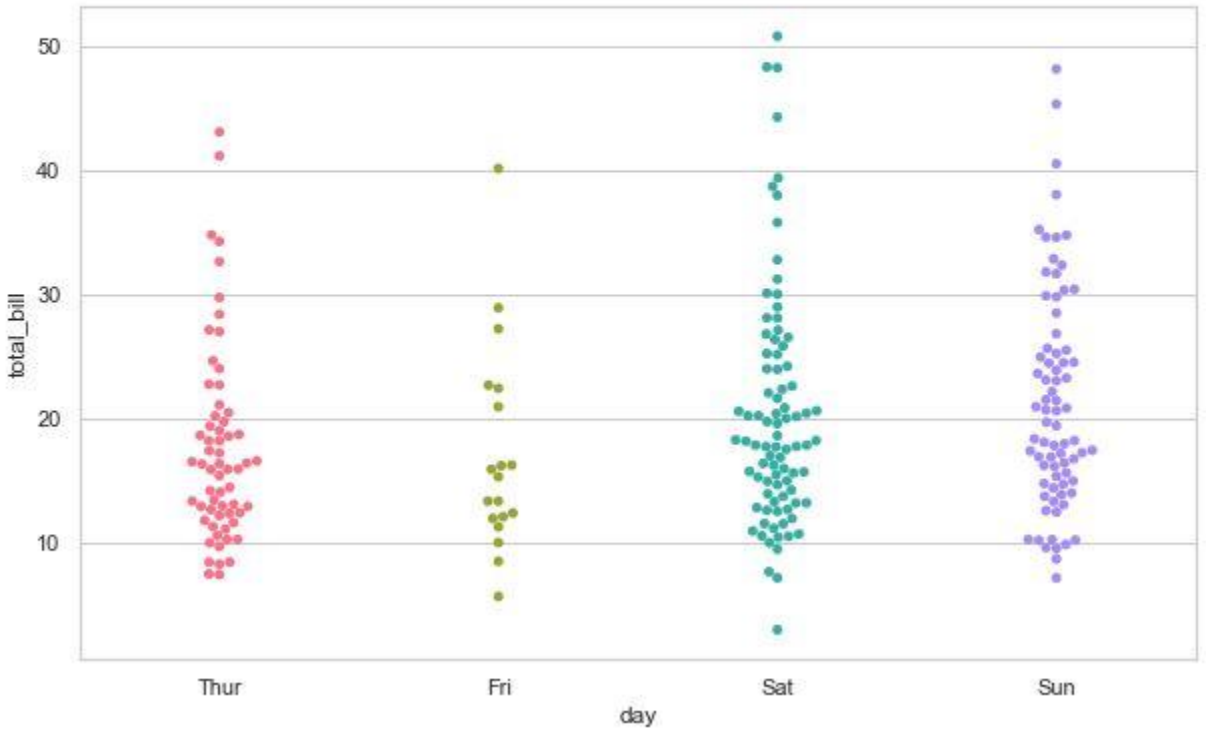


sns 의 기본 데이터 셋

Swarm Plot

- 데이터 포인트 수와 함께 각 데이터의 분포도 제공

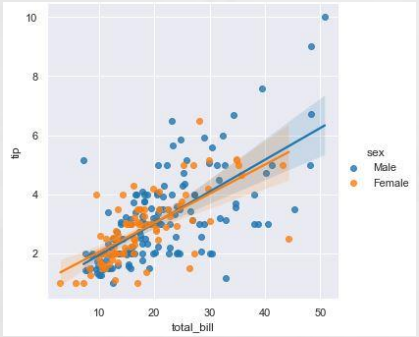
```
#데이터가 모여 있는 정도를 보여주는 plot : seaborn.swarmplot()
plt.figure(figsize=(10,6))
sns.swarmplot(x='day', y='total_bill', data=tips, palette='husl')
plt.show()
```



sns 의 기본 데이터 셋

seaborn.lmplot()

- 스캐터그래프를 그릴때 회귀선을 기본으로 출력해주는 그래프 함수
- size 인수 : 그래프 크기
  - height 인수로 변경해서 사용하라는 메시지가 출력될 수 있음

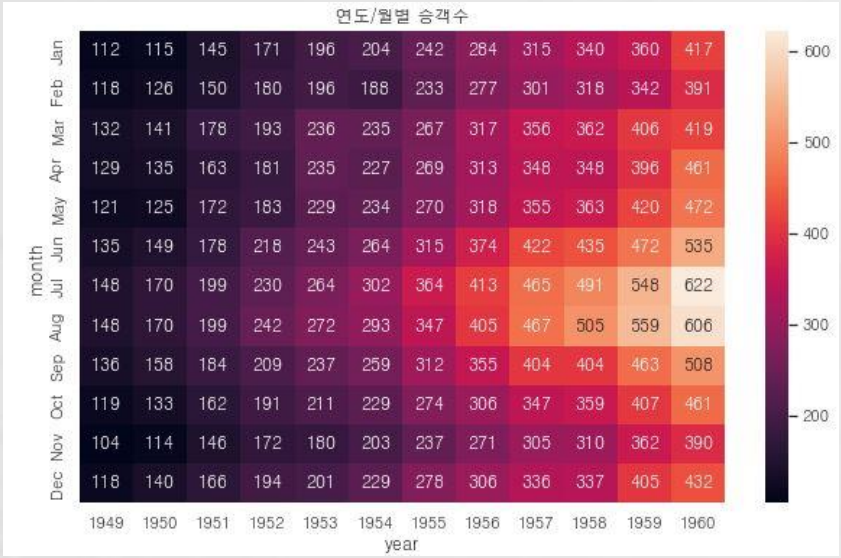


- hue 인수 : 카테고리 데이터를 분류하여 그룹화하되 하나의 그래프에 두개의 값을 모두 표현
- col 인수 : 카테고리 데이터를 분류하여 그룹화하되 그룹의 개수만큼 그래프를 생성해서 표현

sns 의 기본 데이터 셋

heatmap 그래프

- 히트맵 (heat map)
  - 열분포도
  - 2차원 수치 데이터를 색으로 표시



- 두개의 카테고리 값에 대한 값 변화를 한 눈에 알기 쉬움
- 대용량 데이터 시각화에도 사용성이 높음

sns 의 기본 데이터 셋

heatmap 그래프

- sns.heatmap

(heat, # 전처리한 데이터프레임

annot = True, # 숫자 표시 여부

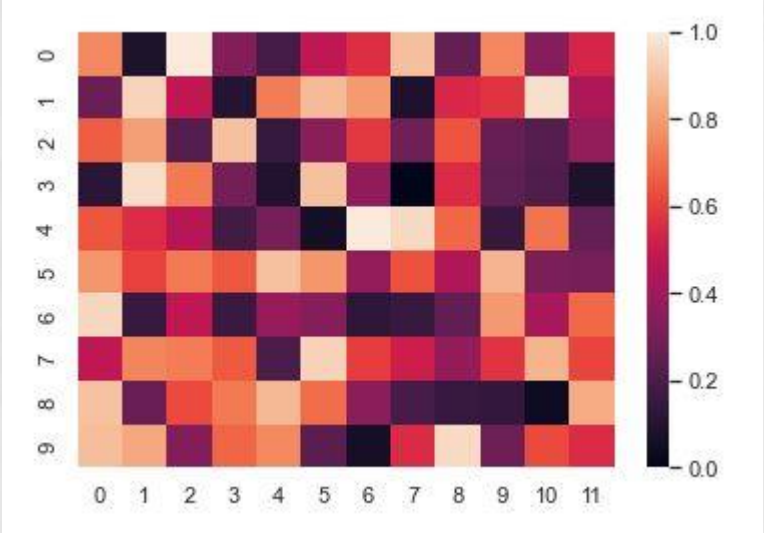
ax = ax, # 히트맵을 그릴 격자

linewidths = 0.4, # 선의 굵기

linecolor = 'white', # 선의 색깔

fmt = '.1f', # 소수점 포맷팅 형태

cmap = 'YlOrRd') # colormap 형태



데이터 시각화(FOLIUM)

FOLIUM 패키지

FOLIUM 패키지

- 지도 이용해 data 시각화 하는 도구
- 아나콘다 프롬프트에서 설치
- pip install folium : 주로 사용
- pip install Folium
- OPEN STREET MAP과 같은 지도 데이터에 leaflet.js 를 이용해서 위치 정보를 시각화 하기 위한 라이브러리
- 마커형태로 위치정보를 지도상에 표현할 수 있다

FOLIUM 패키지

지도 생성 방법

- Map()메소드에 중심 좌표값을 지정해서 간단하게 생성
- 위도와 경도를 data로 지도를 그려줌
- 크롬에서는 출력이 가능하고
- 익스플로러에서는 저장만 가능(출력되지 않는다.)

```
# 초기 map 생성
#중심좌표 인수 location=[lon,lat]
#확대비율 정의 인수 zoon_start=13

map_osm = folium.Map(location=[45.5236,-122.6750], zoom_start=13)
map_osm
```

FOLIUM 패키지

마커설정(popup 설정도 가능)

- 마커(특정 위치를 표시하는 표식)
  - popup(마커 클릭시에 나타나는 정보) : 환경에 따라 지원 안되는 경우도 있음
- 마커 생성
  - folium.Marker():인자값 위,경도 값 리스트, popup 문자
- 마커 생성 후 부착 하는 코드
  - folium.Marker().add_to(지도객체변수)

FOLIUM 패키지

단계 구분도

- 데이터를 지도 그림에 반영시켜서 전달하는 그래프
- 보통 folium 에 layer로 얹게 됨
- map 객체의 choropleth() 이용해서 작업
- 사용 인수
  - geo_data : 지도 파일 경로와 파일명
  - data : 지도에 표현되어야 할 값 변수
  - columns = [key로 사용할 data, 실제 data의 필드명]
  - key_on : 지도 경계파일인 json에서 사용할 키 값이며 지도 Data는 표준 형식
    - key_on 지칭 문법 : feature(키워드).json에서 나타나키 필드명
- folium.LayerControl().add_to(map) : 단계구분도를 map에 표시하는 함수