Pandas

안 화 수

pandas

- pandas

- 구조적 데이터 생성
  - Series를 활용한 데이터 생성 : Series()
  - 날짜 자동 생성 : date_range()
  - DataFrame을 활용한 데이터 생성 : DataFrame()

- 데이터 연산

- 데이터 원하는 부분 추출하기

- 데이터 통합하기

- 데이터 파일 읽고 쓰기

pandas

pandas

1. numpy / pandas 는 고급 데이터 분석과 수치 계산 등의 기능을 제공하는 확장 모듈 입니다.
2. numpy 는 같은 데이터 타입의 배열만 처리할 수 있는 데 반해서 pandas는 데이터 타입이 다양하게 섞여 있는 데이터도 처리할 수 있다.
3. pandas 는 데이터 분석 기능을 제공하는 라이브러리로서, CSV, 엑셀 파일 등의 데이터를 읽고 원하는 데이터 형식으로 변환해 줍니다.
4. numpy / pandas 두 라이브러리는 C언어로 작성돼 있으므로, 파이썬으로 만들어진 라이브러리 보다 처리 속도가 빠릅니다.
5. numpy / pandas 를 사용하려면, 표준 모듈이 아니므로 따로 설치해야 합니다. pip명령으로 설치하면 되는데, Anaconda 를 사용한다면 기본적으로 설치 되어 있습니다.

```
c:₩> pip install numpy
c:₩> pip install pandas
```

pandas

- **pandas**

pandas 모듈은 데이터 형태에 따라 1차원 데이터인 Series와 2차원 데이터인 DataFrame을 사용한다.

- 1차원 데이터 : Series
- 2차원 데이터 : DataFrame

```
import numpy as np
import panda as pd
```

Series

❖ **Series**

1차원 데이터는 Series()를 사용한다.

형식 : pd.Series (data , index = index)

❖ **pd_s.py**

```
import pandas as pd

# 1차원 데이터는 Series()를 사용한다.
s = pd.Series([1.0, 3.0, 5.0, 7.0, 9.0])
print(s)
```

Series

- **pd_series.py**

```
import numpy as np
import pandas as pd

# 1.스칼라(Scalar) 데이터
s1 = pd.Series(7, index=['a','b','c'])
print(s1)

# 2. 일차원 배열 데이터
s2 = pd.Series(np.random.randn(5))      # 난수 5개 발생
print(s2)

s3 = pd.Series(np.random.randn(5), index=['a','b','c','d','e'])
print(s3)

# 3. 리스트 데이터
s4 = pd.Series([1,2,3,4,5], index=['a','b','c','d','e'])
print(s4)

# 딕셔너리 데이터
s5 = pd.Series({'a':0, 'b':1, 'c':2})
print(s5)
```

date_range

- **날짜 자동 생성 : date_range()**

pandas에서 제공되는 date_range() 함수는 연속적인 날짜를 자동으로 생성해주는 역할을 한다.

형식 : pd.date_range (start , end, periods, freq = 'D')

start : 시작 날짜
end : 끝 날짜
periods : 날짜 데이터 생성 기간
freq : 입력하지 않으면 'D' 옵션이 설정돼 달력 날짜 기준으로 하루를 증가한다. (달력 날짜 기준 하루 기준)
start는 반드시 있어야하며 end나 periods는 둘 중 하나만 있어도 된다.

pd.data_range(start = '2019-01-01' , end = '2019-01-07')

date_range

- **날짜 자동 생성 : date_range()**

date_range.py

```
import pandas as pd

# 시작 날짜(start)와 끝 날짜(end)를 지정해서 날짜 데이터 생성
date1 = pd.date_range(start='2019-01-01', end='2019-01-07')
print(date1)

# 시작 날짜(start)와 날짜 생성 기간(periods)을 7로 지정해서 날짜 데이터 생성
date2 = pd.date_range(start='2019-01-01', periods=7)
print(date2)

# 2일씩 증가하는 날짜 생성
date3 = pd.date_range(start='2019-01-01', periods=4, freq='2D')
print(date3)
```

date_range

- **날짜 자동 생성 : date_range()**

```
# 1시간 주기로 10개의 시간을 생성
date4 = pd.date_range(start='2019-01-01 08:00', periods=10, freq='H')
print(date4)

# 30분 단위로 4개의 시간을 생성
date5 = pd.date_range(start='2019-01-01 10:00', periods=4, freq='30min')
print(date5)

# 10초 단위로 4개의 시간을 생성
date6 = pd.date_range(start='2019-01-01 10:00:00', periods=4, freq='10S')
print(date6)
```

DataFrame

- **DataFrame**

  1. pandas에서 사용하는 가장 기본 데이터는 데이터프레임(DataFrame)입니다.
  2. DataFrame을 정의할 때는 2차원 리스트를 매개변수로 사용합니다.

  형식 : pd.DataFrame(data [, index = index_data , columns = columns_data])

- **pd_df.py**

```
import pandas as pd

# DataFrame을 정의할때 2차원 리스트를 매개변수로 사용한다.
a = pd.DataFrame([
        [10,20,30],
        [40,50,60],
        [70,80,90]
    ])

print(a)
```

DataFrame

❖ **pd_df01.py**

```
import numpy as np
import pandas as pd

#1. numpy로 2차원 배열 데이터로 DataFrame 생성
#  index와 columns 이  0,1,2 로 출력됨
data_list = np.array([[10,20,30],[40,50,60],[70,80,90]])
df1 = pd.DataFrame(data_list)
print(df1)

#2. index와 columns 를 지정된 날짜와 문자로 출력
data = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
index_date = pd.date_range('2019-09-01', periods=4)
columns_list = ['A','B','C']
df2 = pd.DataFrame(data, index=index_date, columns=columns_list)
print(df2)
```

DataFrame

```
#3. 어느 회사의 연도 및 지사별 고객수 데이터
# 이 데이터를 딕셔너리 타입의 데이터로 만들어 보자
table_data = {'연도' : [2015, 2016, 2016, 2017, 2017],
              '지사' : ['한국','한국','미국','한국','미국'],
              '고객수' : [200, 250, 450, 300, 500]}
print(table_data)

# 딕셔너리 데이터를 이용해서 DataFrame 으로 만들어 보자
df3 = pd.DataFrame(table_data)
print(df3)
print(df3.index)          # RangeIndex(start=0, stop=5, step=1)
print(df3.columns)        # Index(['연도', '지사', '고객수'], dtype='object')
print(df3.values)
```

DataFrame

❖ **원하는 데이터 추출하기**

: 1차원 리스트가 들어있는 딕셔너리 자료형의 데이터가 있을 때 키(key)로 원하는 열의 데이터를 추출할 수 있다.

❖ **pd_key.py**

```
import pandas as pd

# 키, 몸무게, 유형 데이터프레임 생성하기
tbl = pd.DataFrame({
        "weight": [ 80.0, 70.4, 65.5, 45.9, 51.2 ],
        "height": [ 170,  180,  155,  143,  154  ],
        "type":   [ "f", "n", "n", "t", "t"]
      })

# 몸무게 목록 추출하기
print("몸무게 목록")
print(tbl["weight"])

# 몸무게와 키 목록 추출하기
print("몸무게와 키 목록")
print(tbl[["weight","height"]])
```

DataFrame

❖ **원하는 데이터 추출하기**

: 1차원 리스트가 들어있는 딕셔너리 자료형의 데이터가 있을 때 키(key)로 원하는 열의 데이터를 추출할 수 있다.

❖ **pd_slice.py**

```
import pandas as pd

# 키, 몸무게, 유형 데이터프레임 생성하기
tbl = pd.DataFrame({
        "weight": [ 80.0, 70.4, 65.5, 45.9, 51.2, 72.5 ],
        "height": [ 170, 180, 155, 143, 154, 160 ],
        "type":    [ "f",   "n",  "n",   "t",   "t",   "f" ]
      })

# (0부터 세었을때) 2~3번째 데이터 출력
print("tbl[2:4]₩n", tbl[2:4])

# (0부터 세었을때) 3번째 이후의 데이터 출력
print("tbl[3:]₩n", tbl[3:])
```

DataFrame

❖ **원하는 데이터 추출하기**

: 1차원 리스트가 들어있는 딕셔너리 자료형의 데이터가 있을 때 키(key)로 원하는 열의 데이터를 추출할 수 있다.

❖ **pd_filter.py**

```
import pandas as pd

# 키, 몸무게, 유형 데이터프레임 생성하기
tbl = pd.DataFrame({
        "weight": [ 80.0, 70.4, 65.5, 45.9, 51.2, 72.5 ],
        "height": [ 170,  180,  155,  143,  154,  160  ],
        "gender": [ "f",  "m",  "m",  "f",  "f",  "m"  ]
       })

print("--- height가 160 이상인 것")
print(tbl[tbl.height >= 160])

print("--- gender가 m 인 것")
print(tbl[tbl.gender == "m"])
```

DataFrame

❖ **데이터 정렬(sort) : sort_values()**

: sort_values() 함수로 정렬 할 수 있다.

: 오름차순 정렬이 기본 정렬 방식이다.

❖ **pd_sort.py**

```
import pandas as pd

# 키, 몸무게, 유형 데이터프레임 생성하기
tbl = pd.DataFrame({
        "weight": [ 80.0, 70.4, 65.5, 45.9, 51.2, 72.5 ],
        "height": [ 170,  180,  155,  143,  154,  160  ],
        "gender": [ "f",  "m",  "m",  "f",  "f",  "m"  ]
       })

print("--- 키로 정렬 : 오름차순 정렬")
print(tbl.sort_values(by="height"))

print("--- 몸무게로 정렬 : 내림차순 정렬")
print(tbl.sort_values(by="weight", ascending=False))
```

DataFrame

- **데이터 반전**
  : 행과 열이 바뀜

- **pd_rot.py**

```
import pandas as pd

tbl = pd.DataFrame([
        ["A", "B", "C"],
        ["D", "E", "F"],
        ["G", "H", "I"]
      ])

print(tbl)

print("데이터 반전: 행과 열이 바뀜")
print(tbl.T)
```

정규화(normalization)

- **정규화(normalization)**

수식 : (요소값 - 최소값) / (최대값 - 최소값)
설명 : 전체 구간을 0~100으로 설정하여 데이터를 관찰하는 방법으로,
특정 데이터의 위치를 확인할 수 있게 해줌

- **키와 몸무게 정규화**

1. 키와 몸무게의 최대값을 각각 200과 100으로 결정한 뒤, 데이터를 나누면 데이터를 0과 1 사이의 값으로 만들 수 있습니다.
2. 다만 이 같은 방법은 키가 200이상, 몸무게가 100이상인 경우에는 제대로 정규화 하지 못 합니다.

3. 조금 더 정확하게 정규화 하는 방법

수식 : (요소값 - 최소값) / (최대값 - 최소값)
설명 : 전체 구간을 0~100으로 설정하여 데이터를 관찰하는 방법으로,
특정 데이터의 위치를 확인할 수 있게 해줌

정규화(normalization)

❖ **pd_norm.py**

```
import pandas as pd

# 키, 체중, 유형 데이터프레임 생성하기
tbl = pd.DataFrame({
        "weight": [ 80.0, 70.4, 65.5, 45.9, 51.2, 72.5 ],
        "height": [ 170,  180,  155,  143,  154,  160  ],
        "gender": [ "f",  "m",  "m",  "f",  "f",  "m"  ]
       })

# 최댓값과 최솟값 구하기
# 키와 몸무게 정규화하기
def norm(tbl, key):
    c = tbl[key]
    v_max = c.max()                          # 최대값
    v_min = c.min()                          # 최소값
    print(key, "=", v_min, "-", v_max)

    # 키와 몸무게 정규화
    tbl[key] = (c - v_min) / (v_max - v_min)

norm(tbl, "weight")
norm(tbl, "height")
print(tbl)
```

Pandas 데이터의 연산

- **Pandas 데이터의 연산**

1. pandas의 Series() 와 DataFrame()으로 생성한 데이터 끼리는 산술연산을 할 수 있다.

2. numpy 배열은 원소의 개수가 다르면 연산을 할 수 없지만, pandas는 원소의 갯수가 서로 달라도 산술연산을 할 수 있다.
   단, 연산할 수 없는 부분은 NaN 으로 표시된다.

3. 같은 Series() 데이터 끼리 산술연산

4. 같은 DataFrame() 데이터 끼리 산술연산

Pandas 데이터의 연산

- **Series 데이터 연산**
  pandas의 Series()로 생성한 데이터 끼리는 산술 연산을 할 수 있다.

- **pd_oper01.py**

```
import pandas as pd

#1. 원소의 갯수가 같은 경우
s1 = pd.Series([1,2,3,4,5])
s2 = pd.Series([10,20,30,40,50])

print(s1+s2)
print(s1-s2)
print(s1*s2)
print(s1/s2)

#2. 원소의 갯수가 다른 경우
# 1)numpy배열은 원소의 갯수가 서로 다르면 산술연산을 할수 없지만,
#    pandas는 원소의 갯수가 달라도 산술연산할 수 있다.
# 2)연산을 할 수 없는 경우에는 NaN 으로 출력됨
s3 = pd.Series([1,2,3,4])
s4 = pd.Series([10,20,30,40,50])
print(s3+s4)
print(s3-s4)
print(s3*s4)
print(s3/s4)
```

Pandas 데이터의 연산

- **DataFrame 데이터 연산**
  pandas의 DataFrame()으로 생성한 데이터 끼리는 산술 연산을 할 수 있다.

- **pd_oper02.py**

```
import pandas as pd

table_data1 = {'A' : [1, 2, 3, 4, 5],
               'B' : [10, 20, 30, 40, 50],
               'C' : [100, 200, 300, 400, 500]}

table_data2 = {'A' : [6, 7, 8],
               'B' : [60, 70, 80],
               'C' : [600, 700, 800]}

df1 = pd.DataFrame(table_data1)
print(df1)

df2 = pd.DataFrame(table_data2)
print(df2)

# 데이터프레임 데이터 df1과 df2의 길이지 같지 않아도 산술연산을 할 수 있다.
print(df1 + df2)
print(df1 - df2)
print(df1 * df2)
print(df1 / df2)
```

Pandas의 통계분석 함수

- pandas의 통계분석 함수

- pandas 에서는 통계에서 자주 사용하는 함수들을 지원한다

sum() : 원소의 합
mean() : 평균
var() : 분산
std() : 표준편차
max() : 최대값
min() : 최소값
cumsum() : 각 원소의 누적 합
cumprod() : 각 원소의 누적 곱
describe() : 평균, 표준편차, 최소값, 최대값 등을 한꺼번에 구해줌

Pandas의 통계분석 함수

- **pandas의 통계분석 함수**

- **pd_oper03.py**

```
import pandas as pd

# 2012년 부터 2016년까지 우리나라 계절별 강수량 데이터 ( 단위 : mm )
table_data = {'봄' : [256.5, 264.3, 215.9, 223.2, 312.8],
              '여름' : [770.6, 567.5, 599.8, 387.1, 446.2],
              '가을' : [363.5, 231.2, 293.1, 247.7, 381.6],
              '겨울' : [139.3, 59.9, 76.9, 109.1, 108.1]}
columns_list = ['봄', '여름', '가을', '겨울']
index_list = ['2012', '2013', '2014', '2015', '2016']

# index : 연도,  columns : 계절
df = pd.DataFrame(table_data, index=index_list, columns=columns_list)
print(df)
```

Pandas의 통계분석 함수

- **pandas의 통계분석 함수**

```
# axis 인자를 설정하지 않으면 기본값이 0이 설정됨
# axis = 0 이면 열(column)별로 합을 구함 : 각 계절별 강수량 합을 구함
# axis = 1 이면 행(row)별로 합을 구함 :  각 연도별 강수량 합을 구함

# 2012년 ~ 2016년 계절별 강수량의 합
print('계절별 강수량 합')
sum0 = df.sum(axis=0)
print(sum0)

# 2012년 ~ 2016년 연도별 강수량의 합
print('연도별 강수량 합')
sum1 = df.sum(axis=1)
print(sum1)
```

Pandas의 통계분석 함수

- **pandas의 통계분석 함수**

```
# 2012년 ~ 2016년 계절별 강수량 평균
print('계절별 강수량 평균')
mean0 = df.mean(axis=0)
print(mean0)

# 2012년 ~ 2016년 연도별 강수량 평균
print('연도별 강수량 평균')
mean1 = df.mean(axis=1)
print(mean1)

# describe() 함수를 이용하면, 평균, 표준편차, 최소값, 최대값 등을 한번에
# 구할수 있다.
describe = df.describe()
print(describe)
```

DataFrame 데이터 추출

- **DataFrame 데이터 추출**

- **DataFrame_data.head()** : 첫 5개 행데이터 반환
- **DataFrame_data.head(n)** : 첫 n개 행데이터 반환
- **DataFrame_data.tail()** : 마지막 5개 행데이터 반환
- **DataFrame_data.tail(n)** : 마지막 n개 행데이터 반환
- **DataFrame_data[행 시작위치 : 행 끝위치]**
  : 행 시작위치 ~ 행 끝위치 – 1 까지 행데이터 반환
  행 위치는 0부터 시작
- **DataFrame_data.loc[index_name]**
  : index가 index_name인 행 데이터 반환
- **DataFrame_data.loc[start_index_name : end_index_name]**
  : index가 start_index_name에서 end_index_name까지 구간의
  행 데이터를 반환

DataFrame 데이터 추출

- **DataFrame 데이터 추출**

- **DataFrame_data[column_name]**
  : 하나의 열(column_name)로 지정한 데이터반환

- **DataFrame_data[column_name] [start_index_name : end_index_name]**
  : 하나의 열(column)을 선택한 후 start_index_name ~ end_index_name

- **DataFrame_data[column_name] [start_index_pos : end_index_pos]**
  : 하나의 열(column)을 선택한 후 start_index_pos ~ end_index_pos-1

- **DataFrame 데이터 중에서 하나의 원소 선택하는 방법들**
  1. DataFrame_data.loc[index_name] [column_name]
  2. DataFrame_data.loc[index_name , column_name]
  3. DataFrame_data[column_name][index_name]
  4. DataFrame_data[column_name][index_pos]
  5. DataFrame_data[column_name].loc[index_name]

- **DataFrame_data.T** : DataFrmae의 행과 열을 바꾸는 전치행렬을 만들어준다.

DataFrame 데이터 추출

- **DataFrame 데이터 추출하기 (1 / 4)**
  2011년 부터 2017년까지 노선별 KTX이용자 수(단위: 천명) 데이터
- **pd_extract.py**

```
import numpy as np
import pandas as pd

KTX_data = {'경부선 KTX': [39060, 39896, 42005, 43621, 41702, 41266, 32427],
            '호남선 KTX': [7313, 6967, 6873, 6626, 8675, 10622, 9228],
            '경전선 KTX': [3627, 4168, 4088, 4424, 4606, 4984, 5570],
            '전라선 KTX': [309, 1771, 1954, 2244, 3146, 3945, 5766],
            '동해선 KTX': [np.nan,np.nan, np.nan, np.nan, 2395, 3786, 6667]}
col_list = ['경부선 KTX','호남선 KTX','경전선 KTX','전라선 KTX','동해선 KTX']
index_list = ['2011', '2012', '2013', '2014', '2015', '2016', '2017']

df_KTX = pd.DataFrame(KTX_data, columns = col_list, index = index_list)
print(df_KTX)                                   # DataFrame 데이터 출력
print(df_KTX.index)                             # index 정보 출력
print(df_KTX.columns)                           # columns 정보 출력
print(df_KTX.values)                            # values 정보 출력
```

DataFrame 데이터 추출

- **DataFrame 데이터 추출하기 (2 / 4)**

```
print(df_KTX.head())                    # DataFrame 데이터의 첫 5개 행 데이터 출력
print(df_KTX.head(3))                   # DataFrame 데이터의 첫 3개 행 데이터 출력

print(df_KTX.tail())                    # DataFrame 데이터의 마지막 5개 행 데이터 출력
print(df_KTX.tail(3))                   # DataFrame 데이터의 마지막 3개 행 데이터 출력

# 행의 index 번호로 데이터 추출
# DataFrame_data[행 시작위치 : 행 끝위치] : 행 시작위치 ~ 행 끝위치-1  까지의 행데이터 반환
# 행의 위치는 0부터 시작
print(df_KTX[1:2])                      # 1의 행 데이터 출력
print(df_KTX[2:5])                      # 2에서 4의 행 데이터 출력

# 행의 index 항목 이름으로 데이터 추출
# DataFrame_data.loc[index_name]
print(df_KTX.loc['2011'])               # 2011년 행 데이터 추출

# 행의 index 항목 이름으로 구간을 지정해서 연속된 구간의 행 데이터 추출
# DataFrame_data.loc[start_index_name : end_index_name]
print(df_KTX.loc['2013' : '2016'])      # 2013년부터 2016년까지 행 데이터 추출
```

DataFrame 데이터 추출

- **DataFrame 데이터 추출하기 (3 / 4)**

```
# 열(column) 항목 이름으로 데이터 추출
# DataFrame_data[column_name]
# columns 항목중 '경부선 KTX'의 열 데이터 추출
print(df_KTX['경부선 KTX'])

# 하나의 열을 선택 후 index 범위의 데이터 추출
# DataFrame_data[column_name][index_name]
# DataFrame_data[column_name][index_pos]
# '경부선 KTX' 열을 선택한 후 2012년에서 2014년까지 데이터 추출
print(df_KTX['경부선 KTX']['2012':'2014'])
# '경부선 KTX' 열을 선택한 후 2행에서 4행 데이터 추출
print(df_KTX['경부선 KTX'][2:5])

# DataFrame 의 행과 열을 바꾸는 전치 행렬을 만들어 준다.
print(df_KTX.T)
```

DataFrame 데이터 추출

- **DataFrame 데이터 추출하기 (4 / 4)**

```
# DataFrame 데이터 중에서 하나의 원소 선택하는 방법들

# 1. DataFrame_data.loc[index_name] [column_name]
print(df_KTX.loc['2016']['호남선 KTX'])                # 10622.0

# 2. DataFrame_data.loc[index_name , column_name]
print(df_KTX.loc['2016','호남선 KTX'])                 # 10622.0

# 3. DataFrame_data[column_name][index_name]
print(df_KTX['호남선 KTX']['2016'])                    # 10622

# 4. DataFrame_data[column_name][index_pos]
print(df_KTX['호남선 KTX'][5])                         # 10622

# 5. DataFrame_data[column_name].loc[index_name]
print(df_KTX['호남선 KTX'].loc['2016'])                # 10622
```

DataFrame 데이터 통합하기

- **DataFrame 데이터 통합하기**
  2개의 DataFrame데이터를 하나로 통합하는 방법을 살펴보자.

- **DataFrame 데이터 통합방법**

  1. 세로로 증가하는 방향으로 통합하기

  ```
  DataFrame_data1.append(DataFrame_data2 [,ignore_index=True])
  ```

  2. 가로로 증가하는 방향으로 통합하기

  ```
  DataFrame_data1.join(DataFrame_data2)
  ```

  3. 특정 열을 기준으로 통합하기

  ```
  DataFrame_left_data.merge(DataFrame_right_data,
                            how=merge_method,
                            on=key_label)
  ```

DataFrame 데이터 통합하기

- **merge() 함수의 how 선택 인자에 따른 통합 방법**

```
DataFrame_left_data.merge( DataFrame_right_data ,
                           how = merge_method ,
                           on = key_label )
```

how 선택 인자	설 명
left	왼쪽 데이터는 모두 선택하고 지정된 열(key)에 값이 있는 오른쪽 데이터를 선택
right	오른쪽 데이터는 모두 선택하고 지정된 열(key)에 값이 있는 왼쪽 데이터를 선택
outer	지정된 열(key)을 기준으로 왼쪽과 오른쪽 데이터를 모두 선택
inner	지정된 열(key)을 기준으로 왼쪽과 오른쪽 데이터 중 공통 항목만 선택(기본값)

on 인자는 통합하려는 기준이 되는 특정 열의 라벨 이름을 입력한다.
해당 항목이 없으면 NaN 으로 출력됨

DataFrame 데이터 통합하기

❖ **DataFrame 데이터 통합하기 (1 / 6)**

❖ **pd_join.py**

```
import numpy as np
import pandas as pd

# 두 학급의 시험 점수가 담긴 DataFrame 생성
df1 = pd.DataFrame({'Class1': [95, 92, 98, 100],
                    'Class2': [91, 93, 97, 99]})
print(df1)

# 두 학급에 전학온 학생들의 점수 DataFrame 생성
df2 = pd.DataFrame({'Class1': [87, 89],
                    'Class2': [85, 90]})
print(df2)
```

DataFrame 데이터 통합하기

DataFrame 데이터 통합하기 (2 / 6)

```
# 1. 세로 방향으로 통합하기 : append() 함수 이용
# ignore_index=True 로 설정하지 않으면 기존 index 번호가 그대로 유지된다.
print(df1.append(df2))                                    # 기존 index번호가 유지됨

# ignore_index=True 로 설정하면 새로운 index 번호가 할당된다.
print(df1.append(df2, ignore_index=True))       # 새로운 index번호가 할당됨

# 만약 columns 가 같지 않은 DataFrame 데이터를 append() 함수로 추가하면
# 데이터가 없는 부분은 NaN 으로 채워진다.
# 이를 확인하기 위해서 열(column)이 하나만 있는 DataFrame df3를 생성
df3 = pd.DataFrame({'Class1': [96, 83]})
print(df3)

# 열이 두 개인 DataFrame df2에 열이 하나인 DataFrame df3를 추가
print(df2.append(df3, ignore_index=True))       # 데이터가 없는 부분은 NaN
```

DataFrame 데이터 통합하기

DataFrame 데이터 통합하기 (3 / 6)

```
# 2. 가로 방향으로 통합하기 : join() 함수 이용
# df1과 index방향으로 크기가 같은 DataFrame 생성
df4 = pd.DataFrame({'Class3': [93, 91, 95, 98]})
print(df4)

# df1에 join()함수를 이용해서 df4를 가로방향으로 추가
print(df1.join(df4))

# index라벨을 지정한 DataFrame 데이터의 경우에도 index 가 같으면 join() 함수를
# 이용해서 가로방향으로 데이터를 추가할 수 있다.
index_label = ['a','b','c','d']
df1a = pd.DataFrame({'Class1' : [95, 92, 98, 100],
                     'Class2' : [91, 93, 97, 99]}, index=index_label)
df4a = pd.DataFrame({'Class3' : [93, 91, 95, 98]}, index=index_label)
print(df1a.join(df4a))
```

DataFrame 데이터 통합하기

- **DataFrame 데이터 통합하기 (4 / 6)**

```
# index의 크기가 다른 DataFrame 데이터를 join() 함수를 이용해서 추가하면,
# 데이터가 없는 부분은 NaN으로 채워진다.
df5 = pd.DataFrame({'Class4': [82, 92]})
print(df5)

# index의 크기가 2인 DataFrame df5 를 join()함수를 이용해 index 크기가 4인
# DataFrame df1에 추가
print(df1.join(df5))
```

DataFrame 데이터 통합하기

DataFrame 데이터 통합하기 (5 / 6)

```
# 3. 특정 열을 기준으로 통합하기
# 1월부터 4월까지 제품 A와 B의 판매량 데이터를 변수 df_A_B 에 할당하고, 같은 기간 동안
# 제품 C와 D의 판매량 데이터를 변수 df_C_D 에 할당
df_A_B = pd.DataFrame({'판매월': ['1월', '2월', '3월', '4월'],
                       '제품A': [100, 150, 200, 130],
                       '제품B': [90, 110, 140, 170]})
print(df_A_B)

df_C_D = pd.DataFrame({'판매월': ['1월', '2월', '3월', '4월'],
                       '제품C': [112, 141, 203, 134],
                       '제품D': [90, 110, 140, 170]})
print(df_C_D)

# 두 DataFrame 의 공통 컬럼인 '판매월'을 중심으로 DataFrame 데이터를 통합
print(df_A_B.merge(df_C_D))
```

DataFrame 데이터 통합하기

❖ DataFrame 데이터 통합하기 (6 / 6)

```
# 특정 열을 기준으로 일부만 공통 데이터를 가진 DataFrame 데이터를 통합한 예
df_left = pd.DataFrame({'key':['A','B','C'], 'left': [1, 2, 3]})
print(df_left)
df_right = pd.DataFrame({'key':['A','B','D'], 'right': [4, 5, 6]})
print(df_right)

print(df_left.merge(df_right, how='left', on = 'key'))        # 왼쪽을 먼저 선택
print(df_left.merge(df_right, how='right', on = 'key'))       # 오른쪽을 먼저 선택
print(df_left.merge(df_right, how='outer', on = 'key'))       # 양쪽을 모두 선택
print(df_left.merge(df_right, how='inner', on = 'key'))       # 공통 항목만 선택
```

Pandas로 CSV 파일 읽기

- **Pandas로 CSV 파일 읽기**

  pandas 는 표형식의 데이터 파일을 DataFrame 형식의 데이터로 읽어오는 방법과, DataFrame 형식의 데이터를 표형식으로 파일로 저장할 수 있는 방법을 제공한다.

- Pandas로 CSV(Comma Separated Value)파일 읽기

  DataFrame_date = pd.read_csv(filename, encoding='utf-8')

  파이썬에서 텍스트 파일을 생성하면 기본 문자 인코딩 형식이 utf-8 이다.
  하지만 윈도우의 메모장에서 파일을 저장하면 인코딩 형식이 cp949 가 된다.
  pandas의 read_csv()로 csv파일을 읽을때는 텍스트 파일의 인코딩 옵션을 설정해야된다.
  텍스트 파일이 utf-8로 인코딩 되어 있으면, encoding = 'utf8'
  텍스트 파일이 cp949로 인코딩 되어 있으면, encoding = 'cp949'

Pandas로 CSV 파일 읽기

- **Pandas로 CSV 파일 읽기**

- **pd_csv_read.py**

```
import pandas as pd

# sea_rain1.csv 파일을 DataFrame 데이터로 읽어온다.
df = pd.read_csv('sea_rain1.csv', encoding='utf-8')
print(df)
```

Pandas로 CSV 파일 저장

- **Pandas로 CSV 파일 저장**

  pandas 는 표형식의 데이터 파일을 DataFrame 형식의 데이터로 읽어오는 방법과, DataFrame 형식의 데이터를 표형식으로 파일로 저장할 수 있는 방법을 제공한다.

- Pandas로 CSV(Comma Separated Value)파일 저장

  DataFrame.to_csv(filename, encoding='utf8')

  filename은 csv 파일 저장 경로를 포함 할 수 있다. (상대경로, 절대경로)
  인코딩 방식을 지정하지 않으면, 인코딩 방식은 utf-8 이 된다.

Pandas로 CSV 파일 저장

- **Pandas로 CSV 파일 저장**

- **pd_csv_write.py**

```
import pandas as pd

df_WH = pd.DataFrame({'Weight':[62, 67, 55, 74],
                      'Height':[165, 177, 160, 180]},
                       index=['ID_1', 'ID_2', 'ID_3', 'ID_4'])

df_WH.index.name = 'User'          # index 이름을 User 로 변경
print(df_WH)
```

Pandas로 CSV 파일 저장

❖ **Pandas로 CSV 파일 저장**

```
# 체질량 지수(BMI) = W / H * H
# 키의 경우 입력된 데이터가 cm 단위여서, m 단위로 변경하기 위해서 100으로
# 나눈다.
bmi = df_WH['Weight']/(df_WH['Height']/100)**2
print(bmi)                                  # bmi 정보 출력

# bmi 정보를 df_WH 데이터 프레임에 추가한다.
df_WH['BMI'] = bmi

print(df_WH)                                # df_WH 데이터프레임에 BMI 열이 추가됨

# bmi.csv 파일로 저장
df_WH.to_csv('bmi.csv', encoding='utf8')      # bmi.csv 파일 생성됨
```