판다스(pandas)

- *핵심구조*
  - Series(시리즈) : 1차원 구조를 가진 하나의 열. 첫글자 대문자
  - DataFrame(데이터 프레임) : 복수의 열을 가진 2차원 데이터. 대소문자 구분
- *잘하는 일*
  - 데이터 불러오기 및 저장하기
  - 데이터 보기 및 검사
  - 필터, 정렬 및 그룹화
  - 데이터 정제

판다스 데이터프레임 만들기

In [ ]:
```
import pandas as pd
import numpy as np

df = pd.DataFrame({'이름':['김수안','김수정','박동윤','강이안','강지안'],
                   '나이' : [19, 23, 22,19,16],
                   '성별' : ['여','여','남','여','남'],
                   '평점' : [4.35, 4.23, 4.45, 4.37, 4.25]})
df
```

csv 파일 불러오기

In [ ]:
```
df2 = pd.read_csv('data/countries.csv')
df2
```

In [ ]:
```
# 첫번째 열을 인덱스로 사용하려면,
df3 = pd.read_csv('data/countries.csv', index_col=0)
df3
```

열을 기준으로 데이터 선택하기

In [ ]:
```
# 특정 열만 선택시 대괄호([])에 열이름 넣으면 됨

print(df2['population'])
print()
print(df3['population'])    # 인덱스 컬럼 지정한 데이터프레임
```

In [ ]:
```
# 두개의 열만 선택시 컬럼명을 리스트에 넣어서 전달 함.

df3[ ['area','population'] ]
```

데이터 가시화

- plot(kind='bar'...)메소드 이용.
- kind속성 종류 : bar, barh, line, pie, hist, box, scatter 등
- 그외 linestyle, marker, color 속성 등

In [ ]:
```
# 각 국가의 인구만 추출하여 막대 그래프로 그리기

import matplotlib.pyplot as plt

df3['population'].plot(kind='bar', color=('b', 'g', 'r','m', 'k'))

# df3['population'].plot(kind='pie', autopct='%.2f%%')

# df3['population'].plot(kind='line', linestyle='--', marker='o', color='b')

plt.show()
```

In [ ]:
```
# weather.csv 파일 불어와서 평균풍속의 히스토그램 그리기

weather = pd.read_csv('data/weather.csv') # 한글데이터가 있어서 오류생김

# weather = pd.read_csv('data/weather.csv', encoding = 'CP949') #한글 인코딩 문자를
# weather = pd.read_csv('data/weather.csv', index_col = 0, encoding = 'CP949') # 첫번

weather
```

In [ ]:
```
# 평균풍속으로 히스토그램 그리기

weather['평균 풍속(m/s)'].plot(kind='hist',bins=33)

plt.show()
```

판다스에서도 슬라이싱으로 행 추출

In [ ]:
```
print(weather.head()) #처음 5행만
print(weather.tail()) #마지막 5행만
```

In [ ]:
```
weather[::30] # 처음부터 끝까지 데이터 중 30일 간격으로 행 추출
```

In [ ]:
```
print(df3)
print()
print(df3['population'][:3])
print()
print(df3.loc['US','capital'])
print(df3['capital'].loc['US'])
```

새로운 열(컬럼) 생성하기

In [ ]:
```
# 인구밀도(인구수/면적) 구하여 새로운 열로 추가하기

df3['density'] = df3['population'] / df3['area']
df3
```

데이터 간편 분석 - **describe()** 함수

In [ ]:
```
weather.describe()
```

In [ ]:
```
# 일부 데이터 집계 분석도 다양하게 구할 수 있음

print(weather.mean())     # 모든 열(컬럼)의 평균 구하기
print()
print(weather['평균기온(° C)'].mean())   # 평균기온 열만 평균 구하기
```

```
print()
print(weather.mean()[['평균기온(° C)','평균 풍속(m/s)']])   # 평균기온, 평균풍속 열만
```

In [ ]:
```
# 열의 갯수 확인하기

weather.count() # 같지않음. 결측치가 존재함.
```

weather 데이터의 바람세기 분석하기

- 몇월의 바람이 가장 강한가?

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt

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

monthly = [None for i in range(12) ]    # 변수 초기화
monthly_wind = [0 for i in range(12)]   # 변수 초기화
print(monthly, monthly_wind)

weather['month'] = pd.DatetimeIndex(weather['일시']).month  #일시 컬럼에서 월만 추출
print(weather)

for x in range(12):
    monthly[x] = weather[weather['month']== x+1]          #달별로 분리
    monthly_wind[x] = monthly[x].mean()['평균 풍속(m/s)'] #달별로 평균풍속 평균내기

# print(monthly)
# print(monthly_wind)

plt.plot(monthly_wind, 'r--o')
# plt.xticks(range(12),range(1,13))  #x축에 틱이름
plt.show()
```

데이터를 특정한 값에 기반하여 묶는 기능. 그루핑 - groupby() 함수

In [ ]:
```
# groupby() 함수를 이용하기

weather

# means = weather.groupby('month').mean()
# means

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

In [ ]:
```
#  groupby()함수를 이용하여 바람세기 분석하기

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

weather = pd.read_csv('data/weather.csv', encoding='CP949')
weather['month'] = pd.DatetimeIndex(weather['일시']).month  #일시 컬럼에서 월만 추출

means = weather.groupby('month').mean()  # 한줄로 해결됨!
# means
```

```
means['평균 풍속(m/s)'].plot(kind = 'bar')

plt.show()
```

조건에 맞게 필터링하기

- 넘파이 논리적 인덱싱 방식과 동일

In [ ]:
```
# 최대 풍속이 10 이상인 행만 추출하기

# weather

weather[weather['최대 풍속(m/s)']>= 10.0]
```

데이터 정제하기

- 결손값(결측치) 찾기 함수 : isna() 함수
- 결측치 삭제 함수 : dropna() 함수
- 결측치를 새로운 값으로 대체 함수 : fillna() 함수

In [ ]:
```
# 결측치 찾아보기

weather[weather['평균 풍속(m/s)'].isna()]  # 2012-02-11외 여러개 결측치 있음

# missing_data = weather[weather['평균 풍속(m/s)'].isna()]
# missing_data
```

In [ ]:
```
# 결측치 있는 행 삭제하기

# 12-2-11일 데이터 있는지 우선 확인하기
weather[weather['일시'] == '2012-02-11']

# weather.dropna(axis= 0, how='any', inplace= True) # 행으로 결측치가 하나라도 있으면

# 결측치 있는지 확인하기
# weather[weather['평균 풍속(m/s)'].isna()]
```

In [ ]:
```
# 결측치를 0으로 채우기

weather = pd.read_csv('data/weather.csv', index_col= 0, encoding='CP949') #첫번째 열
weather
# weather[weather['평균 풍속(m/s)'].isna()]

# 결측치를 0으로 채우고, 원본에 반영되도록 inplace=True 하기
weather.fillna(0, inplace= True) # 결손값을 0으로 채움.

print(weather.loc['2012-02-11'])  #일시가 인덱스 열임. 12-2-11일자 행을 가져옴
```

결측치를 해당필드의 평균값으로 대체하기

In [ ]:
```
# 원본 다시 가져오기
weather = pd.read_csv('data/weather.csv', index_col= 0, encoding='CP949') #첫번째 열
weather[weather['평균 풍속(m/s)'].isna()]
```

```
# 최대풍속 열의 결측치를 최대풍속 평균값으로 대체하기

# weather['최대 풍속(m/s)'].fillna(weather['최대 풍속(m/s)'].mean(), inplace= True)
# print(weather.loc['2012-02-11'])  # 12-2-11일자 행을 가져와서 결과 확인하기

# # 평균풍속 열의 결측치를 평균풍속 평균값으로 대체하기

# weather['평균 풍속(m/s)'].fillna(weather['평균 풍속(m/s)'].mean(), inplace= True)
# print(weather.loc['2012-02-11'])  # 12-2-11일자 행을 가져와서 결과 확인하기
```

데이터 구조 변경하기

- 기준에 따른 집계하기 : pivot() 함수
- 데이터프레임 합치기 : concat() 함수
- 데이터베이스 join방식의 데이터 병합하기 : merge() 함수
- 데이터 정렬 : sort_values() 함수

In [ ]:
```
# 피봇함수 사용해보기

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

In [ ]:
```
df_2 = df_1.pivot(index='item', columns='type', values='price')

df_2
```

In [ ]:
```
# 데이터프레임 합치기

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'] )
```

In [ ]:
```
df_1
```

In [ ]:
```
df_2
```

In [ ]:
```
df_3 = pd.concat( [df_1, df_2])
df_3
```

다양한 방법으로 **concat** 적용해보기

In [ ]:
```
pd.concat( [df_1, df_2] , axis = 1, join = 'inner' )
```

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

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

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

```
print()

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

데이터베이스 join 방식의 데이터 병합 - merge

In [ ]:
```
print(df_1)
print(df_2)
```

In [ ]:
```
df_3 = df_1.merge(df_2, how='outer', on='B')
df_3
```

In [ ]:
```
df_3 = df_1.merge(df_2, how='outer', on='C')
df_3
```

인덱스를 키로 활용하여 merge 적용해 보기

In [ ]:
```
df_1.merge(df_2, how ='outer', left_index = True, right_index = True )
```

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

In [ ]:
```
print('left outer ₩n' , df_1.merge(df_2, how='left', on='B' ) )
print()

print('right outer ₩n' ,df_1.merge(df_2, how='right', on='B' ) )
print()

print('full outer ₩n' ,df_1.merge(df_2, how='outer', on='B' ) )
print()

print('inner ₩n' ,df_1.merge(df_2, how='inner', on='B' ) )
```

정렬

In [ ]:
```
df2
```

In [ ]:
```
sorted = df2.sort_values('area', ascending=False)
print(sorted)
```

In [ ]:
```
df2.sort_values(['population', 'area'], inplace = True) # 정렬한 것을 원본파일에 반

df2
```

In [ ]:
```
df2.sort_values(['population', 'area'], ascending = False, inplace = True)

df2
```