
파이썬 기반
빅데이터 처리
및 분석기술
Crayon

이번 시간에는

2 Pandas 심화 (2)

- 판다스 내장 그래픽 활용
- 중복 데이터 처리

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 그래프 활용

- 그래프는 데이터의 의미를 전달하는데 효과적
- 시각화는 데이터의 분포와 패턴 파악에 용의함

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 그래프를 이용한 시각화

- Matplotlib 라이브러리의 기능을 일부 내장하여 별도의 임포트없이 간단히 그래프 활용 가능
- 시리즈 또는 데이터프레임 객체에 plot() 시각화 내장 함수를 이용

kind 값	설명	kind 값	설명
'line'	선 그래프	'box'	박스 플롯
'bar'	수직 막대 그래프	'scatter'	산점도 그래프
'barh'	수평 막대 그래프	'area'	면적 그래프
'hist'	히스토그램	'pie'	파이 그래프

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 한글 폰트 설정

- 한글 폰트(나눔) 설치

```
!apt -qq -y install fonts-nanum
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 한글 폰트 설정

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
fontpath = '/usr/share/fonts/truetype/nanum/NanumBarunGothic.ttf'
font = fm.FontProperties(fname=fontpath, size=9)
plt.rc('font', family="NanumBarunGothic")
mpl.font_manager._rebuild()
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 한글 폰트 설정

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
fontpath = '/usr/share/fonts/truetype/nanum/NanumBarunGothic.ttf'
font = fm.FontProperties(fname=fontpath, size=9)
plt.rc('font', family="NanumBarunGothic")
mpl.font_manager._rebuild()
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 한글 폰트 설정

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
fontpath = '/usr/share/fonts/truetype/nanum/NanumBarunGothic.ttf'
font = fm.FontProperties(fname=fontpath, size=9)
plt.rc('font', family="NanumBarunGothic")
mpl.font_manager._rebuild()
```

- 나눔 폰트 → 기본 폰트
- 에러 발생 시, colab 메뉴 런타임 → 런타임 다시 시작

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 선 그래프

- 옵션 없이 plot() 함수를 적용하면 기본 선 그래프를 그림

```
import pandas as pd
data = pd.read_csv('/content/drive/My Drive/일별평균대기오염도_2018_중구.csv', header=None, encoding='euc-kr')
data.columns=['측정일시','이산화질소농도','오존농도','이산화탄소농도','아황산가스','미세먼지','초미세먼지']
p_data = data.iloc[:,5:]
p_data.plot()
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 선 그래프

- 옵션 없이 plot() 함수를 적용하면 기본 선 그래프를 그림

```
import pandas as pd
data = pd.read_csv('/content/drive/My Drive/일별평균대기오염도_2018_중구.csv', header=None, encoding='euc-kr')
data.columns=['측정일시','이산화질소농도','오존농도','이산화탄소농도','아황산가스','미세먼지','초미세먼지']
p_data = data.iloc[:,5:]
p_data.plot()
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 선 그래프

- 옵션 없이 plot() 함수를 적용하면 기본 선 그래프를 그림

```
import pandas as pd
data = pd.read_csv('/content/drive/My Drive/일별평균대기오염도_2018_중구.csv', header=None, encoding='euc-kr')
data.columns=['측정일시','이산화질소농도','오존농도','이산화탄소농도','아황산가스','미세먼지','초미세먼지']
p_data = data.iloc[:,5:]
p_data.plot()
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 선 그래프

- 옵션 없이 plot() 함수를 적용하면 기본 선 그래프를 그림

```
import pandas as pd
data = pd.read_csv('/content/drive/My Drive/일별평균대기오염도_2018_중구.csv', header=None, encoding='euc-kr')
data.columns=['측정일시','이산화질소농도','오존농도','이산화탄소농도','아황산가스','미세먼지','초미세먼지']
p_data = data.iloc[:,5:]
p_data.plot()
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 선 그래프

- 옵션 없이 plot() 함수를 적용하면 기본 선 그래프를 그림

```
import pandas as pd
data = pd.read_csv('/content/drive/My Drive/일별평균대기오염도_2018_중구.csv', header=None, encoding='euc-kr')
data.columns=['측정일시','이산화질소농도','오존농도','이산화탄소농도','아황산가스','미세먼지','초미세먼지']
p_data = data.iloc[:,5:]
p_data.plot()
```

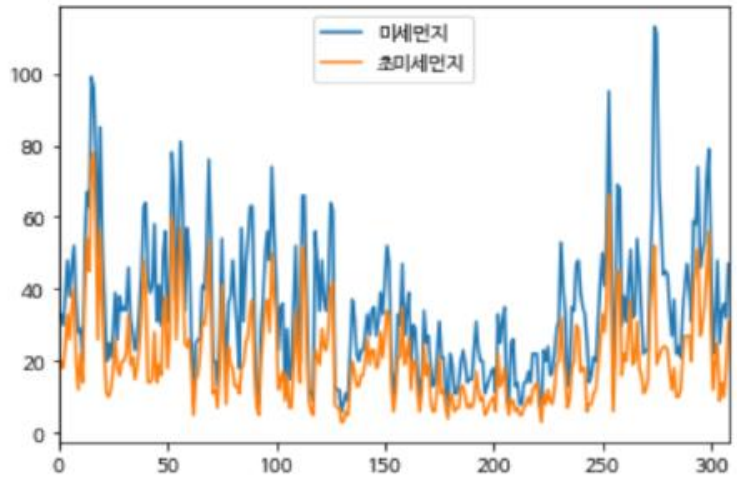
2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 선 그래프

- 옵션 없이 plot() 함수를 적용하면 기본 선 그래프를 그림

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc2a4f64e0>
```



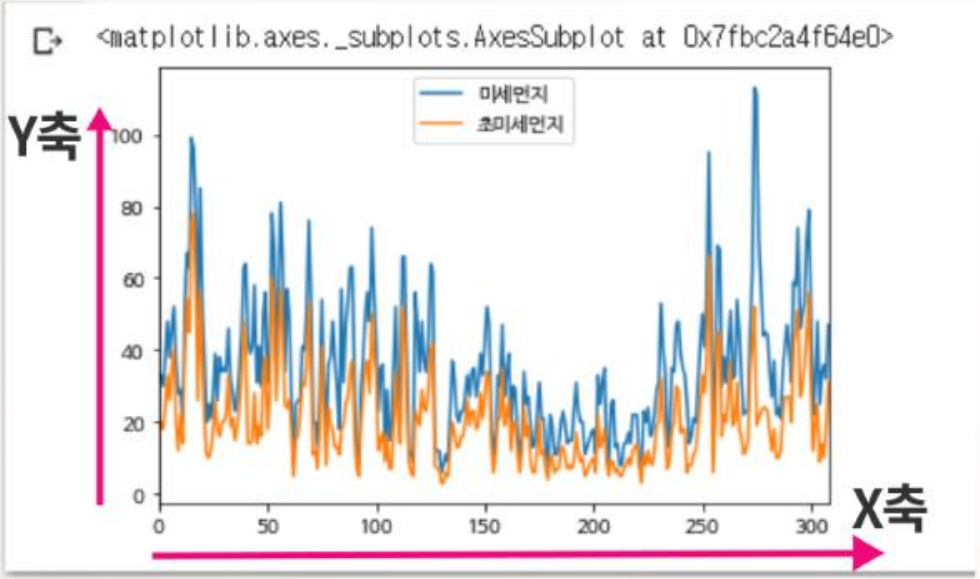
2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 선 그래프

- 옵션 없이 plot() 함수를 적용하면 기본 선 그래프를 그림

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc2a4f64e0>
```



2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 막대 그래프

- kind='bar' 옵션 적용

```
b_data = data.iloc[0:30, 5:]
b_data.plot(kind='bar')
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 막대 그래프

- kind='bar' 옵션 적용

```
b_data = data.iloc[0:30, 5:]
b_data.plot(kind='bar')
```

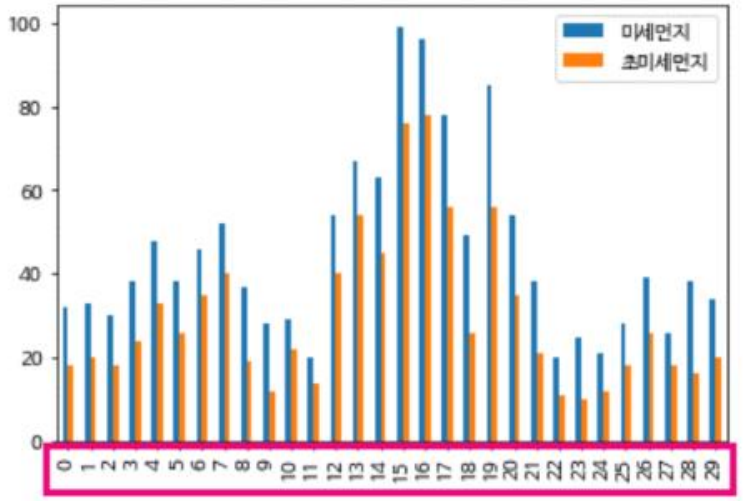
2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 막대 그래프

- kind='bar' 옵션 적용

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc29ba5828>
```



2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 히스토그램

- kind='hist' 옵션 적용

```
h_data = data.iloc[0:30, 5]
h_data.plot(kind='hist')
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 히스토그램

- kind='hist' 옵션 적용

```
h_data = data.iloc[0:30, 5]
h_data.plot(kind='hist')
```

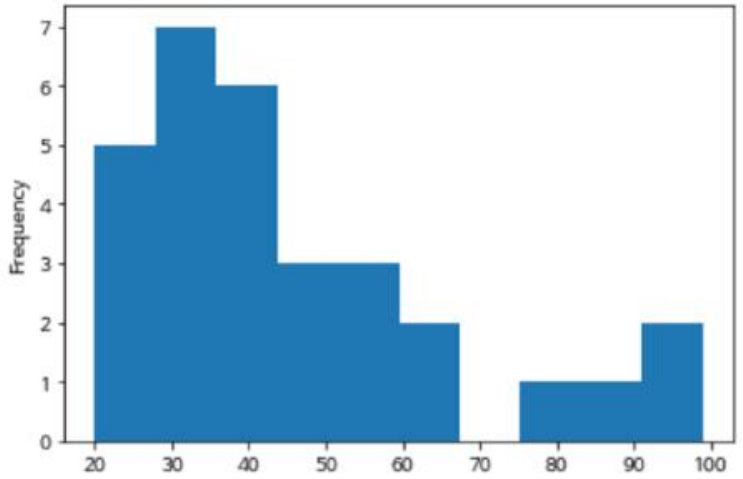
2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 히스토그램

- kind='hist' 옵션 적용

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc295f1d30>
```



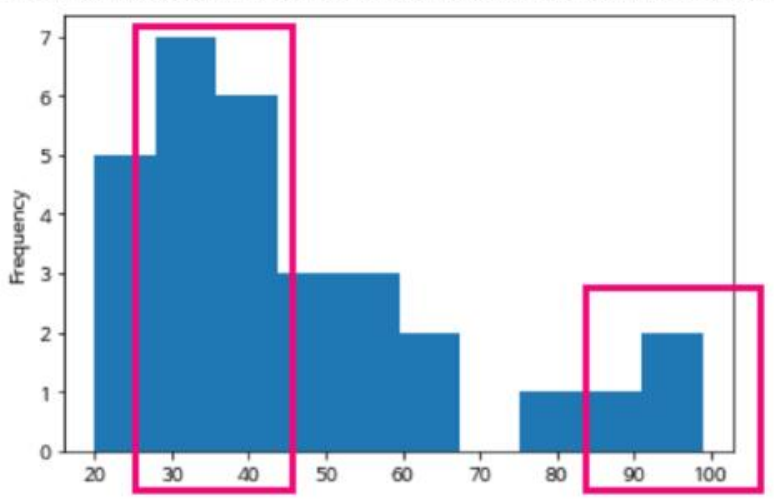
2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 히스토그램

- kind='hist' 옵션 적용

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc295f1d30>
```



2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 산점도

- x, y 축으로 이루어진 그래프에 두 변수의 값을 점으로 나타낸 그래프
- 두 변수 사이의 관계를 시각적으로 나타냄
- kind='scatter' 옵션 적용

```
data.plot(x='미세먼지', y='초미세먼지', kind='scatter')
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 산점도

- x, y 축으로 이루어진 그래프에 두 변수의 값을 점으로 나타낸 그래프
- 두 변수 사이의 관계를 시각적으로 나타냄
- kind='scatter' 옵션 적용

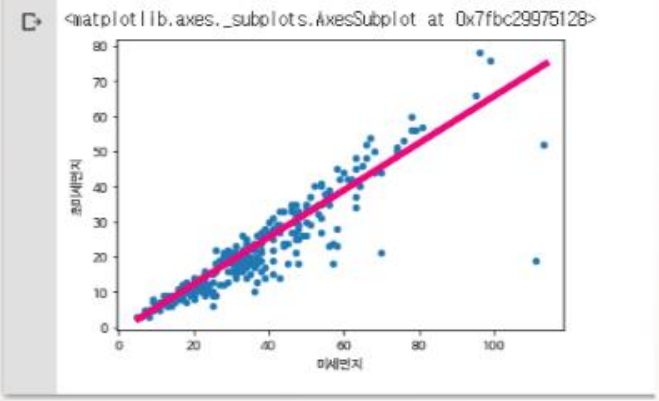
```
data.plot(x='미세먼지', y='초미세먼지', kind='scatter')
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 산점도

- X, y 축으로 이루어진 그래프에 두 변수의 값을 점으로 나타낸 그래프
- 두 변수 사이의 관계를 시각적으로 나타냄
- kind='scatter' 옵션 적용



2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 박스플롯

- 특정 변수의 데이터 분포와 분산 정도에 관한 정보 제공
- kind='box' 옵션 적용

```
data[['미세먼지', '초미세먼지']].plot(kind='box')
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 박스플롯

- 특정 변수의 데이터 분포와 분산 정도에 관한 정보 제공
- kind='box' 옵션 적용

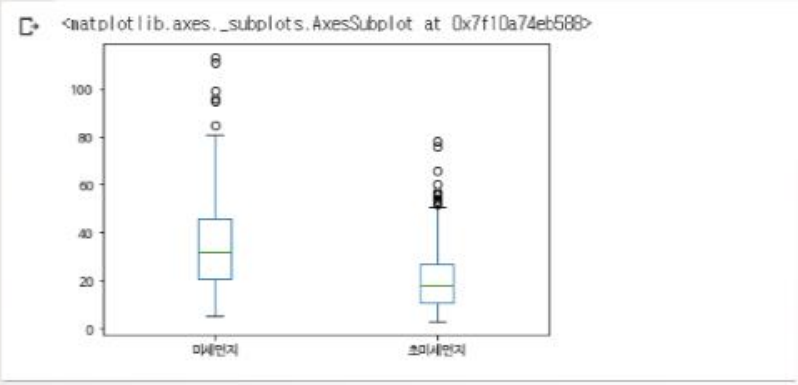
```
data[['미세먼지', '초미세먼지']].plot(kind='box')
```

2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 박스플롯

- 특정 변수의 데이터 분포와 분산 정도에 관한 정보 제공
- kind='box' 옵션 적용

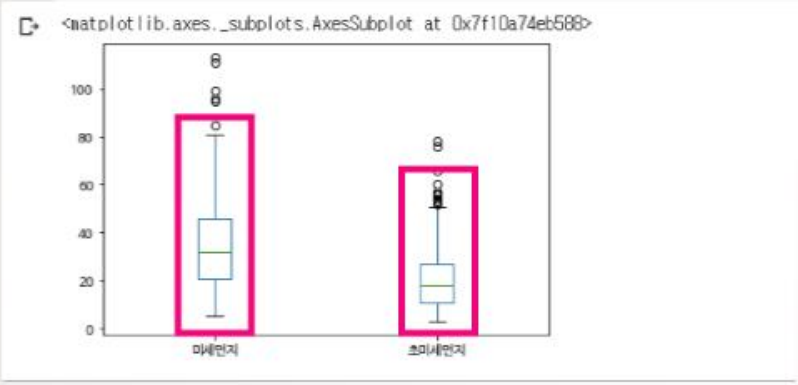


2. Pandas의 심화 (2)

1) Pandas 내장 그래프

◆ 박스플롯

- 특정 변수의 데이터 분포와 분산 정도에 관한 정보 제공
- kind='box' 옵션 적용



2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터

- 데이터프레임의 각 행 → 데이터 셋(레코드)
- 동일한 데이터 셋이 2개 이상 존재하는 경우 → 분석 결과 왜곡
- 중복 데이터가 있는 지를 확인 → 있을 경우 삭제

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 확인

- duplicated() 함수 적용
- 현재 행 vs. 이전 행 비교
  - 중복된 행이면 True
  - 처음 나오는 행이면 False

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 확인

```
dataSet = pd.DataFrame({'c1':['a', 'a', 'b', 'a', 'b'],
                        'c2':[1, 1, 1, 2, 1],
                        'c3':[1, 1, 2, 2, 2]})
dataSet_dup = dataSet.duplicated()
print(dataSet_dup)
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 확인

```
dataSet = pd.DataFrame({'c1':['a', 'a', 'b', 'a', 'b'],
                        'c2':[1, 1, 1, 2, 1],
                        'c3':[1, 1, 2, 2, 2]})
dataSet_dup = dataSet.duplicated()
print(dataSet_dup)
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 확인

```
dataSet = pd.DataFrame({'c1':['a', 'a', 'b', 'a', 'b'],
                        'c2':[1, 1, 1, 2, 1],
                        'c3':[1, 1, 2, 2, 2]})
dataSet_dup = dataSet.duplicated()
print(dataSet_dup)
```

```
0    False
1     True
2    False
3    False
4     True
dtype: bool
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 확인

```
dataSet = pd.DataFrame({'c1':['a', 'a', 'b', 'a', 'b'],
                        'c2':[1, 1, 1, 2, 1],
                        'c3':[1, 1, 2, 2, 2]})
dataSet_dup = dataSet.duplicated()
print(dataSet_dup)
```

```
0    False
1     True
2    False
3    False
4     True
dtype: bool
```

	c1열	c2열	c3열
0행	'a'	1	1
1행	'a'	1	1
2행	'b'	1	2
3행	'a'	2	2
4행	'a'	1	2

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 확인

```
dataSet = pd.DataFrame({'c1':['a', 'a', 'b', 'a', 'b'],
                        'c2':[1, 1, 1, 2, 1],
                        'c3':[1, 1, 2, 2, 2]})
dataSet_dup = dataSet.duplicated()
print(dataSet_dup)
```

```
0    False
1     True
2    False
3    False
4     True
dtype: bool
```

	c1열	c2열	c3열
0행	'a'	1	1
1행	'a'	1	1
2행	'b'	1	2
3행	'a'	2	2
4행	'a'	1	2

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 열 요소값에 대해 중복 여부 판별

- 열 → 시리즈 객체
- duplicated() 함수 적용

```
c3_dup = dataSet['c3'].duplicated()
print(c3_dup)
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 열 요소값에 대해 중복 여부 판별

- 열 → 시리즈 객체
- duplicated() 함수 적용

```
c3_dup = dataSet['c3'].duplicated()
print(c3_dup)
```

```
0    False
1     True
2    False
3     True
4     True
Name: c3, dtype: bool
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 열 요소값에 대해 중복 여부 판별

- 열 → 시리즈 객체
- duplicated() 함수 적용

```
c3_dup = dataSet['c3'].duplicated()
print(c3_dup)
```

```
0    False
1     True
2    False
3     True
4     True
Name: c3, dtype: bool
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

- drop_duplicates() 함수 적용

  - 중복 행들을 제거하고 고유한 값 만을 가진 행들만 남김

- 원본까지 변경할 경우

  - inplace 옵션에 True를 지정하여 drop_duplicates() 함수 호출

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

```
dataSet2 = dataSet.drop_duplicates()
print(dataSet2)
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

```
dataSet2 = dataSet.drop_duplicates()
print(dataSet2)
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

```
dataSet2 = dataSet.drop_duplicates()
print(dataSet2)
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

```
dataSet2 = dataSet.drop_duplicates()
print(dataSet2)
```

```
  c1  c2  c3
0  a   1   1
2  b   1   2
3  a   2   2
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

```
dataSet2 = dataSet.drop_duplicates()
print(dataSet2)
```

```
  c1  c2  c3
0  a   1   1
2  b   1   2
3  a   2   2
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

- drop_duplicates() 함수 subset 옵션 적용

  - 중복 행들을 제거하고 고유한 값 만을 가진 행들만 남김

  - subset 옵션에 지정된 열을 기준으로 중복 여부 판단

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

```
dataSet_sub = dataSet.drop_duplicates(subset=['c2', 'c3'])
print(dataSet_sub)
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

```
dataSet_sub = dataSet.drop_duplicates(subset=['c2', 'c3'])
print(dataSet_sub)
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

```
dataSet_sub = dataSet.drop_duplicates(subset=['c2', 'c3'])
print(dataSet_sub)
```

```
  c1  c2  c3
0  a   1   1
2  b   1   2
3  a   2   2
```

2. Pandas의 심화 (2)

2) 중복 데이터 처리

◆ 중복 데이터 제거

```
dataSet_sub = dataSet.drop_duplicates(subset=['c2', 'c3'])
print(dataSet_sub)
```

```
  c1  c2  c3
0  a   1   1
2  b   1   2
3  a   2   2
```

이번 시간에는

2 Pandas 심화 (2)

- 판다스 내장 그래픽 활용
- 중복 데이터 처리