Part 3. 데이터 살펴보기

1. 데이터프레임의 구조

- **UCI 자동차 연비(auto mpg) 데이터셋**

  : 연비, 실린더 수, 배기량, 출력, 차중, 가속능력, 출시년도, 제조국, 모델명에 관한 데이터 398 개로 구성

No.	속성(attributes)		데이터 상세(범위)
1	mpg	연비	연속 값
2	cylinders	실린더 수	이산 값(예시: 3, 4, 6, 8)
3	displacement	배기량	연속 값
4	horsepower	출력	연속 값
5	weight	차중	연속 값
6	acceleration	가속능력	연속 값
7	model_year	출시년도	이산 값(예: 70, 71, 80, 81)
8	origin	제조국	이산 값(예: 1(USA), 2(EU), 3(JPN))
9	name	모델명	문자열

[표 3-1] UCI 데이터셋 – "auto mpg" 상세 항목

1-1 데이터 내용 미리보기

head() 메소드 인자로 정수 n을 전달하면 처음 n개의 행을 보여준다. 반면 tail() 메소드에 정수 n을 입력하면 마지막 n개의 행을 보여준다. 한편 정수 n을 입력하지 않고 head()와 같이 입력하면 처음 또는 마지막 5개 행을 보여준다(디폴트값: n=5).

- 앞부분 미리보기: `DataFrame 객체.head(n)`
- 뒷부분 미리보기: `DataFrame 객체.tail(n)`

〈예제 3-1〉 데이터 살펴보기 (File: example/part3/3.1_exploratory_analysis.py)

```
1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4
5  # read_csv() 함수로 df 생성
6  df = pd.read_csv('./auto-mpg.csv', header=None)
7
8  # 열 이름 지정
9  df.columns = ['mpg','cylinders','displacement','horsepower','weight',
10                'acceleration','model year','origin','name']
11
12 # 데이터프레임 df의 내용을 일부 확인
13 print(df.head())       # 처음 5개 행
14 print('\n')
15 print(df.tail())       # 마지막 5개 행
```

〈실행 결과〉 코드 1~15라인을 부분 실행

```
    mpg  cylinders  ...  origin                   name
0  18.0          8  ...       1  chevrolet chevelle malibu
1  15.0          8  ...       1          buick skylark 320
2  18.0          8  ...       1         plymouth satellite
3  16.0          8  ...       1              amc rebel sst
4  17.0          8  ...       1                ford torino

[5 rows x 9 columns]


      mpg  cylinders  ...  origin             name
393  27.0          4  ...       1  ford mustang gl
394  44.0          4  ...       2        vw pickup
395  32.0          4  ...       1    dodge rampage
396  28.0          4  ...       1      ford ranger
397  31.0          4  ...       1       chevy s-10

[5 rows x 9 columns]
```

1. 데이터프레임의 구조

1-2 데이터 요약 정보 확인하기

• 데이터프레임의 크기(행, 열)

데이터프레임 클래스의 shape 속성은 행과 열의 개수를 투플 형태로 보여준다. 예제에서 변수 df에 저장된 데이터프레임의 크기(행의 개수, 열의 개수)를 확인하려면 df.shape라고 입력한다.

데이터프레임의 크기 확인: `DataFrame 객체.shape`

〈예제 3-1〉 데이터 살펴보기 (File: example/part3/3.1_exploratory_analysis.py(이어서 계속))

```
~  ~~~ 생략 ~~~

17  # df의 모양과 크기 확인: (행의 개수, 열의 개수)를 투플로 반환
18  print(df.shape)
```

〈실행 결과〉 코드 17~18라인을 부분 실행

```
(398, 9)
```

• 데이터프레임의 기본 정보

info() 메소드를 데이터프레임에 적용하면 데이터프레임에 관한 기본 정보를 화면에 출력한다. 클래스 유형, 행 인덱스의 구성, 열 이름의 종류와 개수, 각 열의 자료형과 개수, 메모리 할당량에 관한 정보가 포함된다. 데이터프레임 df의 기본 정보를 확인하는 명령은 df.info()이다.

데이터프레임의 기본 정보 출력: `DataFrame 객체.info()`

〈예제 3-1〉 데이터 살펴보기 (File: example/part3/3.1_exploratory_analysis.py(이어서 계속))

```
~  ~~~ 생략 ~~~

21  # 데이터프레임 df의 내용 확인
22  print(df.info())
```

〈실행 결과〉 코드 21~22라인을 부분 실행

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
mpg             398 non-null float64
cylinders       398 non-null int64
displacement    398 non-null float64
horsepower      398 non-null object
weight          398 non-null float64
acceleration    398 non-null float64
model year      398 non-null int64
origin          398 non-null int64
name            398 non-null object
dtypes: float64(4), int64(3), object(2)
memory usage: 24.9+ KB
None
```

• 데이터프레임의 기술 통계 정보 요약

데이터프레임에 describe() 메소드를 적용하면, 산술(숫자) 데이터를 갖는 열에 대한 주요 기술 통계 정보(평균, 표준편차, 최대값, 최소값, 중간값 등)를 요약하여 출력한다.

데이터프레임의 기술 통계 정보 요약: `DataFrame 객체.describe()`

〈예제 3-1〉 데이터 살펴보기 (File: example/part3/3.1_exploratory_analysis.py(이어서 계속))

```
~  ~~~ 생략 ~~~

33  # 데이터프레임 df의 기술 통계 정보 확인
34  print(df.describe())
35  print('\n')
36  print(df.describe(include='all'))
```

〈실행 결과〉 코드 33~36라인을 부분 실행

```
              mpg     cylinders  ...  model year      origin
count  398.000000  398.000000  ...  398.000000  398.000000
mean    23.514573    5.454774  ...   76.010050    1.572864
std      7.815984    1.701004  ...    3.697627    0.802055
min      9.000000    3.000000  ...   70.000000    1.000000
25%     17.500000    4.000000  ...   73.000000    1.000000
50%     23.000000    4.000000  ...   76.000000    1.000000
75%     29.000000    8.000000  ...   79.000000    2.000000
max     46.600000    8.000000  ...   82.000000    3.000000

[8 rows x 7 columns]
```

1. 데이터프레임의 구조

```
              mpg     cylinders     ...          origin          name
count  398.000000    398.000000     ...      398.000000           398
unique         NaN           NaN     ...             NaN           305
top            NaN           NaN     ...             NaN     ford pinto
freq           NaN           NaN     ...             NaN             6
mean     23.514573      5.454774     ...        1.572864           NaN
std       7.815984      1.701004     ...        0.802055           NaN
min       9.000000      3.000000     ...        1.000000           NaN
25%      17.500000      4.000000     ...        1.000000           NaN
50%      23.000000      4.000000     ...        1.000000           NaN
75%      29.000000      8.000000     ...        2.000000           NaN
max      46.600000      8.000000     ...        3.000000           NaN

[11 rows x 9 columns]
```

1-3 데이터 개수 확인

- 각 열의 데이터 개수

〈예제 3-2〉 데이터 개수 확인 (File: example/part3/3.2_exploratory_analysis2.py)

```
1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4
5  # read_csv() 함수로 df 생성
6  df = pd.read_csv('./auto-mpg.csv', header=None)
7
8  # 열 이름 지정
9  df.columns = ['mpg','cylinders','displacement','horsepower','weight',
10               'acceleration','model year','origin','name']
11
12 # 데이터프레임 df의 각 열이 가지고 있는 원소 개수 확인
13 print(df.count())
14 print('\n')
15
16 # df.count()가 반환하는 객체 타입 출력
17 print(type(df.count()))
```

〈실행 결과〉 코드 1~17라인을 부분 실행

```
mpg             398
cylinders       398
displacement    398
horsepower      398
weight          398
acceleration    398
model year      398
origin          398
name            398
dtype: int64


<class 'pandas.core.series.Series'>
```

- 각 열의 고유값 개수

열 데이터의 고유값 개수: `DataFrame 객체["열 이름"].value_counts()`

〈예제 3-2〉 데이터 개수 확인 (File: example/part3/3.2_exploratory_analysis2.py(이어서 계속))

```
~  ~~~ 생략 ~~~

20 # 데이터프레임 df의 특정 열이 가지고 있는 고유값 확인
21 unique_values = df['origin'].value_counts()
22 print(unique_values)
23 print('\n')
24
25 # value_counts 메소드가 반환하는 객체 타입 출력
26 print(type(unique_values))
```

〈실행 결과〉 코드 20~26라인을 부분 실행

```
1    249
3     79
2     70
Name: origin, dtype: int64


<class 'pandas.core.series.Series'>
```

2. 통계함수 적용

2-1 평균값

데이터프레임에 mean() 메소드를 적용하면, 산술 데이터를 갖는 모든 열의 평균값을 각각 계산하여 시리즈 객체로 반환한다. 데이터프레임의 특정 열을 선택하여 평균값을 계산할 수도 있다.

- 모든 열의 평균값: `DataFrame 객체.mean()`
- 특정 열의 평균값: `DataFrame 객체["열 이름"].mean()`

〈예제 3-3〉 통계 함수 (File: example/part3/3.3_exploratory_analysis3.py)

```
1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4
5  # read_csv() 함수로 df 생성
6  df = pd.read_csv('./auto-mpg.csv', header=None)
7
8  # 열 이름 지정
9  df.columns = ['mpg','cylinders','displacement','horsepower','weight',
10               'acceleration','model year','origin','name']
11
12 # 평균값
13 print(df.mean())
14 print('\n')
15 print(df['mpg'].mean())
16 print(df.mpg.mean())
17 print('\n')
18 print(df[['mpg','weight']].mean())
```

〈실행 결과〉 코드 1~18라인을 부분 실행

```
mpg             23.514573
cylinders        5.454774
displacement   193.425879
weight        2970.424623
acceleration    15.568090
model year      76.010050
origin           1.572864
dtype: float64

23.514572864321615
23.514572864321615


mpg         23.514573
weight    2970.424623
dtype: float64
```

2-2 중간값

데이터프레임에 median() 메소드를 적용하면 산술 데이터를 갖는 모든 열의 중간값을 계산하여 시리즈로 반환한다. 데이터프레임의 특정 열을 선택하여 중간값을 계산할 수도 있다.

- 모든 열의 중간값: `DataFrame 객체.median()`
- 특정 열의 중간값: `DataFrame 객체["열 이름"].median()`

〈예제 3-3〉 통계 함수 (File: example/part3/3.3_exploratory_analysis3.py(이어서 계속))

```
~  ~~~ 생략 ~~~

20 # 중간값
21 print(df.median())
22 print('\n')
23 print(df['mpg'].median())
```

〈실행 결과〉 코드 20~23라인을 부분 실행

```
mpg             23.0
cylinders        4.0
displacement   148.5
weight        2803.5
acceleration    15.5
model year      76.0
origin           1.0
dtype: float64


23.0
```

2. 통계함수 적용

2-3 최대값

데이터프레임에 max() 메소드를 적용하면 데이터프레임의 각 열이 갖는 데이터 값 중에서 최대값을 계산하여 시리즈로 반환한다. 데이터프레임의 특정 열을 선택하여 계산할 수도 있다.

- 모든 열의 최대값: DataFrame 객체.max()
- 특정 열의 최대값: DataFrame 객체["열 이름"].max()

〈예제 3-3〉 통계 함수 (File: example/part3/3.3_exploratory_analysis3.py(이어서 계속))

```
~  ~~~ 생략 ~~~

25  # 최대값
26  print(df.max())
27  print('\n')
28  print(df['mpg'].max())
```

〈실행 결과〉 코드 25~28라인을 부분 실행

```
mpg                        46.6
cylinders                     8
displacement                455
horsepower                    ?
weight                     5140
acceleration               24.8
model year                   82
origin                        3
name            vw rabbit custom
dtype: object


46.6
```

2-4 최소값

데이터프레임에 min() 메소드를 적용하면 데이터프레임의 각 열이 갖는 데이터 값 중에서 최소값을 계산하여 시리즈로 반환한다. 데이터프레임의 특정 열을 선택하여 계산할 수도 있다.

- 모든 열의 최소값: DataFrame 객체.min()
- 특정 열의 최소값: DataFrame 객체["열 이름"].min()

〈예제 3-3〉 통계 함수 (File: example/part3/3.3_exploratory_analysis3.py(이어서 계속))

```
~  ~~~ 생략 ~~~

30  # 최소값
31  print(df.min())
32  print('\n')
33  print(df['mpg'].min())
```

〈실행 결과〉 코드 30~33라인을 부분 실행

```
mpg                                 9
cylinders                           3
displacement                       68
horsepower                      100.0
weight                           1613
acceleration                        8
model year                         70
origin                              1
name              amc ambassador brougham
dtype: object


9.0
```

2-5 표준편차

데이터프레임에 std() 메소드를 적용하면 산술 데이터를 갖는 열의 표준편차를 계산하여 시리즈로 반환한다. 데이터프레임의 특정 열을 선택하여 계산할 수도 있다.

- 모든 열의 표준편차: DataFrame 객체.std()
- 특정 열의 표준편차: DataFrame 객체["열 이름"].std()

〈예제 3-3〉 통계 함수 (File: example/part3/3.3_exploratory_analysis3.py(이어서 계속))

```
~  ~~~ 생략 ~~~

35  # 표준편차
36  print(df.std())
37  print('\n')
38  print(df['mpg'].std())
```

2. 통계함수 적용

〈실행 결과〉 코드 35~38라인을 부분 실행

```
mpg              7.815984
cylinders        1.701004
displacement   104.269838
weight         846.841774
acceleration     2.757689
model year       3.697627
origin           0.802055
dtype: float64


7.815984312565782
```

2-6 상관계수

데이터프레임에 corr() 메소드를 적용하면 두 열 간의 상관계수를 계산한다. 산술 데이터를 갖는 모든 열에 대하여 2개씩 서로 짝을 짓고, 각각의 경우에 대하여 상관계수를 계산한다.

- 모든 열의 상관계수: `DataFrame 객체.corr()`
- 특정 열의 상관계수: `DataFrame 객체[열 이름의 리스트].corr()`

〈예제 3-3〉 통계 함수 (File: example/part3/3.3_exploratory_analysis3.py(이어서 계속))

```
~ ~~~ 생략 ~~~

40 # 상관계수
41 print(df.corr())
42 print('\n')
43 print(df[['mpg','weight']].corr())
```

〈실행 결과〉 코드 40~43라인을 부분 실행

```
                mpg  cylinders  ...  model year    origin
mpg        1.000000  -0.775396  ...    0.579267  0.563450
cylinders -0.775396   1.000000  ...   -0.348746 -0.562543
displacement -0.804203  0.950721  ...   -0.370164 -0.609409
weight     -0.831741   0.896017  ...   -0.306564 -0.581024
acceleration  0.420289  -0.505419  ...    0.288137  0.205873
model year  0.579267  -0.348746  ...    1.000000  0.180662
origin     0.563450  -0.562543  ...    0.180662  1.000000

[7 rows x 7 columns]


                mpg      weight
mpg        1.000000   -0.831741
weight    -0.831741    1.000000
```

3. 판다스 내장 그래프 도구 활용

그래프를 이용한 시각화 방법은 데이터의 분포와 패턴을 파악하는데 크게 도움이 된다.

시리즈 또는 데이터프레임 객체에 plot() 메소드를 적용하고, kind 옵션으로 그래프의 종류를 선택한다.

kind 옵션	설명	kind 옵션	설명
'line'	선 그래프	'kde'	커널 밀도 그래프
'bar'	수직 막대 그래프	'area'	면적 그래프
'barh'	수평 막대 그래프	'pie'	파이 그래프
'his'	히스토그램	'scatter'	산점도 그래프
'box'	박스 플롯	'hexbin'	고밀도 산점도 그래프

[표 3-3] 판다스 내장 plot( ) 메소드 – 그래프 종류

• 선 그래프

데이터프레임(또는 시리즈) 객체에 plot() 메소드를 적용할 때, 다른 옵션을 추가하지 않으면 가장 기본적인 선 그래프를 그린다.

선 그래프: DataFrame 객체.plot()

〈예제 3-4〉 선 그래프 그리기 (File: example/part3/3.4_df_plot.py)

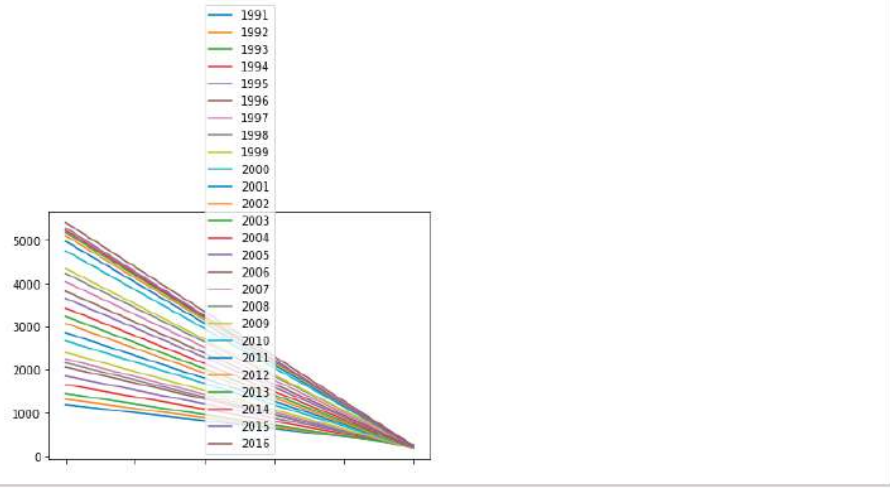
```
1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4
5  df = pd.read_excel('./남북한발전전력량.xlsx')  # 데이터프레임 변환
6
7  df_ns = df.iloc[[0, 5], 3:]                  # 남한, 북한 발전량 합계 데이터만 추출
8  df_ns.index = ['South','North']              # 행 인덱스 변경
9  df_ns.columns = df_ns.columns.map(int)       # 열 이름의 자료형을 정수형으로 변경
10 print(df_ns.head())
11 print('\n')
12
13 # 선 그래프 그리기
14 df_ns.plot()
```

〈실행 결과〉 코드 1~14라인을 부분 실행

```
        1991  1992  1993  1994  1995 ...  2012  2013  2014  2015  2016
South  1186  1310  1444  1650  1847 ...  5096  5171  5220  5281  5404
North   263   247   221   231   230 ...   215   221   216   190   239

[2 rows x 26 columns]


Out[1]: <matplotlib.axes._subplots.AxesSubplot at 0xa244e50>
```



앞의 그래프 표현은 어색하다. 시간의 흐름에 따른 연도별 발전량 변화 추이를 보기 위해서는 연도 값을 x축에 표시해야 하는 것이 적절하다. 다음 예제에서 x축, y 축 값을 서로 바꿔서 출력한다.

3. 판다스 내장 그래프 도구 활용

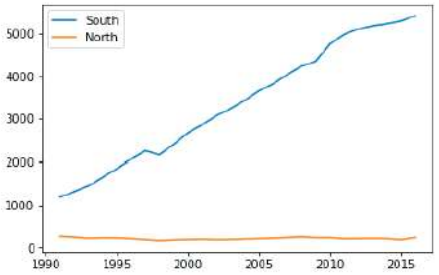
〈예제 3-4〉 선 그래프 그리기 (File: example/part3/3.4_df_plot.py(이어서 계속))

```
~  ~~~ 생략 ~~~
16  # 행, 열 전치하여 다시 그리기
17  tdf_ns = df_ns.T
18  print(tdf_ns.head())
19  print('\n')
20  tdf_ns.plot()
```

〈실행 결과〉 코드 16~20라인을 부분 실행

```
      South  North
1991   1186    263
1992   1310    247
1993   1444    221
1994   1650    231
1995   1847    230


Out[2]: <matplotlib.axes._subplots.AxesSubplot at 0x17e6bb0>
```



● 막대 그래프

plot() 메소드로 선 그래프가 아닌 다른 종류의 그래프를 그리려면, kind 옵션에 그래프 종류를 지정한다. 막대 그래프를 예로 들면, 다음과 같이 kind='bar' 옵션을 추가하는 방식이다.

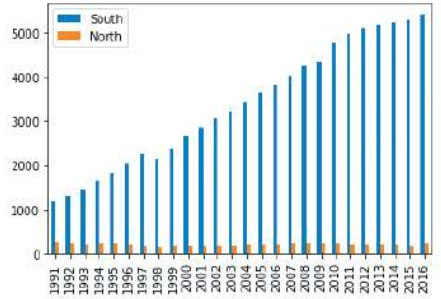
```
막대 그래프: DataFrame 객체.plot(kind='bar')
```

〈예제 3-5〉 막대 그래프 (File: example/part3/3.5_df_plot_bar.py)

```
1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4
5  df = pd.read_excel('./남북한발전전력량.xlsx')  # 데이터프레임 변환
6
7  df_ns = df.iloc[[0, 5], 3:]                  # 남한, 북한 발전량 합계 데이터만 추출
8  df_ns.index = ['South','North']              # 행 인덱스 변경
9  df_ns.columns = df_ns.columns.map(int)       # 열 이름의 자료형을 정수형으로 변경
10
11 # 행, 열 전치하여 막대 그래프 그리기
12 tdf_ns = df_ns.T
13 print(tdf_ns.head())
14 print('\n')
15 tdf_ns.plot(kind='bar')
```

〈실행 결과〉 코드 전부 실행

```
      South  North
1991   1186    263
1992   1310    247
1993   1444    221
1994   1650    231
1995   1847    230
```



3. 판다스 내장 그래프 도구 활용

• 히스토그램

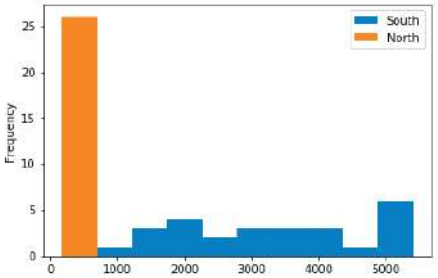
이번에는 남북한 발전량에 관한 히스토그램을 그린다. plot() 메소드에 kind='hist' 옵션을 넣는다.

```
히스토그램: DataFrame 객체.plot(kind='hist')
```

〈예제 3-6〉 히스토그램 (File: example/part3/3.6_df_plot_hist.py)

```
1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4
5  df = pd.read_excel('./남북한발전전력량.xlsx')  # 데이터프레임 변환
6
7  df_ns = df.iloc[[0, 5], 3:]                  # 남한, 북한 발전량 합계 데이터만 추출
8  df_ns.index = ['South','North']              # 행 인덱스 변경
9  df_ns.columns = df_ns.columns.map(int)       # 열 이름의 자료형을 정수형으로 변경
10
11 # 행, 열 전치하여 히스토그램 그리기
12 tdf_ns = df_ns.T
13 tdf_ns.plot(kind='hist')
```

〈실행 결과〉 코드 전부 실행



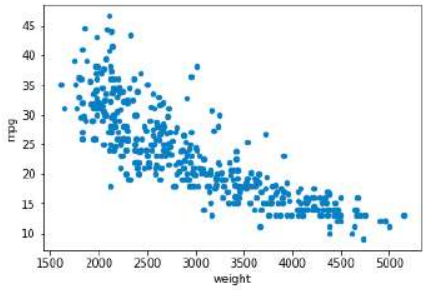
• 산점도

UCI 자동차 연비 데이터셋[5]을 이용하여 두 변수의 관계를 나타내는 산점도를 그린다. 자료실에서 CSV 파일[6]을 다운로드 받는다. plot() 메소드에 kind='scatter' 옵션을 넣고, 데이터프레임의 열 중에서 서로 비교할 두 변수를 선택한다. x축에 차량의 무게 데이터를 갖는 'weight' 열을 지정하고, y축에는 연비를 나타내는 'mpg' 열을 지정한다.

〈예제 3-7〉 산점도 (File: example/part3/3.7_df_plot_scatter.py)

```
1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4
5  # read_csv() 함수로 df 생성
6  df = pd.read_csv('./auto-mpg.csv', header=None)
7
8  # 열 이름 지정
9  df.columns = ['mpg','cylinders','displacement','horsepower','weight',
10               'acceleration','model year','origin','name']
11
12 # 2개의 열을 선택하여 산점도 그리기
13 df.plot(x='weight',y='mpg', kind='scatter')
```

〈실행 결과〉 코드 전부 실행



3. 판다스 내장 그래프 도구 활용

● **박스 플롯**

박스 플롯은 특정 변수의 데이터 분포와 분산 정도에 대한 정보를 제공한다. plot() 메소드에 kind='box' 옵션을 입력한다. 예제에서 연비('mpg' 열) 데이터는 10~45 범위에 넓게 분포되어 있다. 또한 'o' 표시의 이상값(outlier)도 확인된다. 반면, 실린더 개수('cylinders' 열)는 10 미만의 좁은 범위에 몰려 있다. 이처럼 각 변수들의 데이터가 퍼져 있는 정도를 확인할 때 쓰인다.

〈예제 3-8〉 박스 플롯 (File: example/part3/3.8_df_plot_boxplot.py)

```
# -*- coding: utf-8 -*-

import pandas as pd

# read_csv() 함수로 df 생성
df = pd.read_csv('./auto-mpg.csv', header=None)

# 열 이름 지정
df.columns = ['mpg','cylinders','displacement','horsepower','weight',
              'acceleration','model year','origin','name']

# 열을 선택하여 박스 플롯 그리기
df[['mpg','cylinders']].plot(kind='box')
```

〈실행 결과〉 코드 전부 실행