Part 4. 시각화 도구

1. Matplotlib - 기본 그래프 도구

1-1 선 그래프

선 그래프(line plot)는 연속하는 데이터 값들을 직선 또는 곡선으로 연결하여 데이터 값 사이의 관계를 나타낸다. 특히 시계열 데이터와 같이 연속적인 값의 변화와 패턴을 파악하는데 적합하다.

전출지별	전입지별	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979
		이동자수 (명)	이동자수 (명)	이동자수 (명)	이동자수 (명)	이동자수 (명)	이동자수 (명)	이동자수 (명)	이동자수 (명)	이동자수 (명)	이동지
전국	전국	4,046,536	4,210,164	3,687,938	4,860,418	5,297,969	9,011,440	6,773,250	7,397,623	8,410,276	7
	서울특별시	1,742,813	1,671,705	1,349,333	1,831,858	2,050,392	3,396,662	2,756,510	2,893,403	3,307,439	2
	부산광역시	448,577	389,797	362,202	482,061	680,984	805,979	724,664	785,117	895,337	
	대구광역시	-	-	-	-	-	-	-	-	-	
	인천광역시	-	-	-	-	-	-	-	-	-	
	광주광역시	-	-	-	-	-	-	-	-	-	
	대전광역시	-	-	-	-	-	-	-	-	-	
	울산광역시	-	-	-	-	-	-	-	-	-	
	세종특별자치시	-	-	-	-	-	-	-	-	-	
	경기도	452,060	533,375	418,045	542,224	579,194	1,006,818	770,036	880,019	985,879	
	강원도	160,959	183,964	161,637	241,663	184,385	326,365	239,946	246,070	264,577	

[그림 4-1] 시도별 전출입 인구수(시도간 인구 이동)

• 기본 사용법

데이터 시각화에 사용할 matplotlib.pyplot 모듈을 "as plt"와 같이 약칭 plt로 임포트한다. 그리고 시도 간 인구 이동 데이터셋을 가져와 데이터프레임으로 변환한다.

〈예제 4-1〉 선 그래프 (File : example/part4/4.1_matplotlib_line1.py)

```
1  # -*- coding: utf-8 -*-
2
3  # 라이브러리 불러오기
4  import pandas as pd
5  import matplotlib.pyplot as plt
6
7  # Excel 데이터를 데이터프레임으로 변환
8  df = pd.read_excel('시도별 전출입 인구수.xlsx', fillna=0, header=0)
```

〈IPython 실행〉 코드 1~8라인을 부분 실행한 후 IPython 콘솔에서 head 메소드 실행

```
In [2]: df.head()
Out[2]:
  전출지별 전입지별      1970  ...      2015      2016      2017
0 전출지별 전입지별 이동자수 (명)  ... 이동자수 (명) 이동자수 (명) 이동자수 (명)
1 전국     전국      4046536  ...   7755286   7378430   7154226
2 NaN      서울특별시  1742813  ...   1589431   1515602   1472937
3 NaN      부산광역시  448577   ...   507031    459015    439073
4 NaN      대구광역시  -        ...   351424    328228    321182

[5 rows x 50 columns]
```

〈예제 4-1〉 선 그래프 (File: example/part4/4.1_matplotlib_line1.py(이어서 계속))

```
~  ~~~ 생략 ~~~

10 # 누락값(NaN)을 앞 데이터로 채움(엑셀 양식 병합 부분)
11 df = df.fillna(method='ffill')
12
13 # 서울에서 다른 지역으로 이동한 데이터만 추출하여 정리
14 mask = (df['전출지별'] == '서울특별시') & (df['전입지별'] != '서울특별시')
15 df_seoul = df[mask]
16 df_seoul = df_seoul.drop(['전출지별'], axis=1)
17 df_seoul.rename({'전입지별':'전입지'}, axis=1, inplace=True)
18 df_seoul.set_index('전입지', inplace=True)
```

〈IPython 실행〉 코드 10~18라인을 부분 실행한 후 IPython 콘솔에서 head 메소드 실행

```
In [3]: df_seoul.head(8)
Out[3]:
              1970     1971     1972  ...     2015     2016     2017
전입지                                ...
전국       1448985  1419016  1210559  ...  1726687  1655859  1571423
부산광역시   11568    11130    11768  ...    17009    15062    14484
대구광역시       -        -        -  ...    10191     9623     8891
인천광역시       -        -        -  ...    44915    43745    40485
광주광역시       -        -        -  ...     9216     8354     7932
대전광역시       -        -        -  ...    13453    12619    11815
울산광역시       -        -        -  ...     5950     5102     4260
세종특별자치시   -        -        -  ...     7550     5943     5813

[8 rows x 48 columns]
```

1. Matplotlib - 기본 그래프 도구

〈예제 4-1〉 선 그래프 (File: example/part4/4.1_matplotlib_line1.py(이어서 계속))

```
~  ~~~ 생략 ~~~

20  # 서울에서 경기도로 이동한 인구 데이터 값만 선택
21  sr_one = df_seoul.loc['경기도']
```

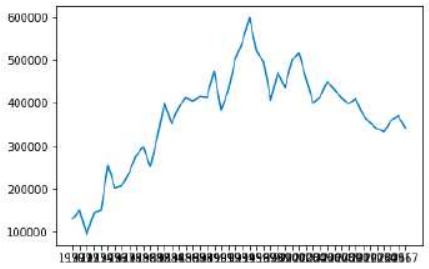
〈IPython 실행〉 코드 20~21라인을 부분 실행한 후 IPython 콘솔에서 head 메소드 실행

```
In [4]: sr_one.head()
Out[4]:
1970    130149
1971    150313
1972     93333
1973    143234
1974    149045
Name: 경기도, dtype: object
```

〈예제 4-1〉 선 그래프 (File: example/part4/4.1_matplotlib_line1.py(이어서 계속))

```
~  ~~~ 생략 ~~~

23  # x, y축 데이터를 plot 함수에 입력
24  plt.plot(sr_one.index, sr_one.values)
```

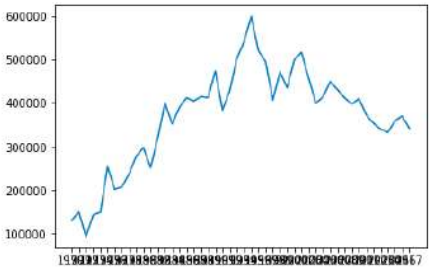
〈실행 결과〉 코드 23~24라인을 부분 실행



〈예제 4-1〉 선 그래프 (File: example/part4/4.1_matplotlib_line1.py(이어서 계속))

```
~  ~~~ 생략 ~~~

26  # 판다스 객체를 plot 함수에 입력
27  plt.plot(sr_one)
```

〈실행 결과〉 코드 26~27라인을 부분 실행



1. Matplotlib - 기본 그래프 도구

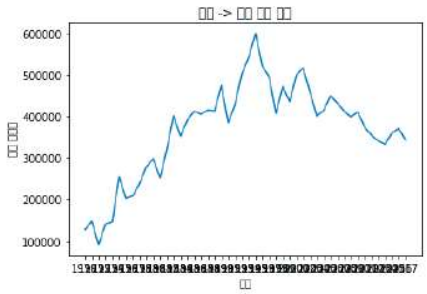
• 차트 제목, 축 이름 추가

그래프 객체에 차트 제목을 추가할 때는 `title()` 함수를 사용한다. x축 이름은 `xlabel()` 함수를 이용하고, y축 이름은 `ylabel()` 함수를 활용하여 추가한다.

〈예제 4-2〉 차트 제목, 축 이름 추가 (File: example/part4/4.2_matplotlib_line2.py)

```
~  ~~ 생략 (예제 4-1과 동일) ~~

# 서울에서 경기도로 이동한 인구 데이터 값만 선택
sr_one = df_seoul.loc['경기도']

# x, y축 데이터를 plot 함수에 입력
plt.plot(sr_one.index, sr_one.values)

# 차트 제목 추가
plt.title('서울 -> 경기 인구 이동')

# 축 이름 추가
plt.xlabel('기간')
plt.ylabel('이동 인구수')

plt.show()  # 변경사항 저장하고 그래프 출력
```

〈실행 결과〉 코드 전부 실행



• Matplotlib 한글 폰트 오류 해결

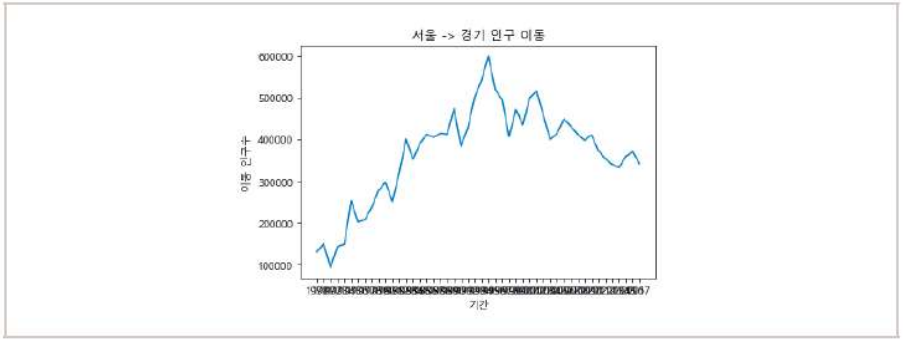
〈예제 4-3〉 한글 폰트 오류 해결 (File: example/part4/4.3_matplotlib_hangul.py)

한글 폰트 오류 해결 코드

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

# 라이브러리 불러오기
import pandas as pd
import matplotlib.pyplot as plt

# matplotlib 한글 폰트 오류 문제 해결
from matplotlib import font_manager, rc
font_path = "./malgun.ttf"   # 폰트 파일 위치
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc('font', family=font_name)

# Excel 데이터를 데이터프레임으로 변환
df = pd.read_excel('시도별 전출입 인구수.xlsx', fillna=0, header=0)

# 전출지별에서 누락값(NaN)을 앞 데이터로 채움(엑셀 양식 병합 부분)
df = df.fillna(method='ffill')

# 서울에서 다른 지역으로 이동한 데이터만 추출하여 정리
mask = (df['전출지별'] == '서울특별시') & (df['전입지별'] != '서울특별시')
df_seoul = df[mask]
df_seoul = df_seoul.drop(['전출지별'], axis=1)
df_seoul.rename({'전입지별':'전입지'}, axis=1, inplace=True)
df_seoul.set_index('전입지', inplace=True)

# 서울에서 경기도로 이동한 인구 데이터 값만 선택
sr_one = df_seoul.loc['경기도']

# x, y축 데이터를 plot 함수에 입력
plt.plot(sr_one.index, sr_one.values)

# 차트 제목 추가
plt.title('서울 -> 경기 인구 이동')

# 축 이름 추가
plt.xlabel('기간')
plt.ylabel('이동 인구수')

plt.show()  # 변경사항 저장하고 그래프 출력
```

1. Matplotlib - 기본 그래프 도구

〈실행 결과〉 코드 전부 실행



● **그래프 꾸미기**

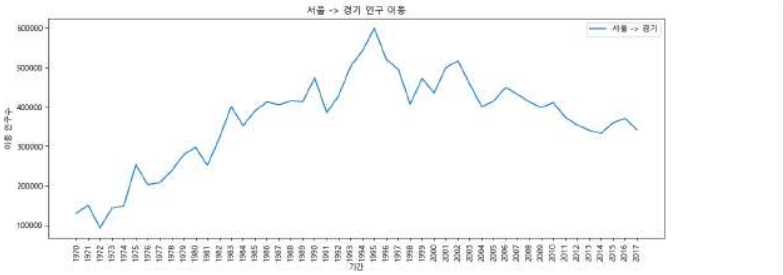
x축 눈금 라벨의 글씨가 서로 겹쳐 잘 보이지 않는 문제를 해결하는 방법을 알아보자. 이것은 눈금 라벨이 들어갈 만한 충분한 여유 공간이 없어서 발생하는 문제이다.

다음의 예제는 글씨가 들어갈 수 있는 공간을 확보하기 위해 두 가지 방법을 적용한다. 첫째, 공간을 만들기 위해 `figure()` 함수로 그림틀의 가로 사이즈를 더 크게 설정한다. 둘째, `xticks()` 함수를 활용하여 x축 눈금 라벨을 반시계 방향으로 90° 회전하여 글씨가 서로 겹치지 않게 만든다.

〈예제 4-4〉 그래프 꾸미기 (File: example/part4/4.4_matplotlib_line3.py)

```
~  ~~ 생략 (예제 4-3과 동일) ~~

# 서울에서 경기도로 이동한 인구 데이터 값만 선택
sr_one = df_seoul.loc['경기도']

# 그림 사이즈 지정(가로 14인치, 세로 5인치)
plt.figure(figsize=(14, 5))

# x축 눈금 라벨 회전하기
plt.xticks(rotation='vertical')

# x, y축 데이터를 plot 함수에 입력
plt.plot(sr_one.index, sr_one.values)

plt.title('서울 -> 경기 인구 이동')  # 차트 제목
plt.xlabel('기간')                  # x축 이름
plt.ylabel('이동 인구수')           # y축 이름

plt.legend(labels=['서울 -> 경기'], loc='best')        # 범례 표시

plt.show()  # 변경사항 저장하고 그래프 출력
```

〈실행 결과〉 코드 전부 실행



다음은 Matplotlib의 스타일 서식 지정에 대해 알아보자. 색, 폰트 등 디자인적 요소를 사전에 지정된 스타일로 빠르게 일괄 변경한다. 단, 스타일 서식을 지정하는 것은 Matplotlib 실행 환경 설정을 변경하는 것이므로 다른 파일을 실행할 때도 계속 적용되는 점에 유의한다.

〈예제 4-5〉의 30라인에서 `'ggplot'`이라는 스타일 서식을 지정한다. 36라인의 x축 눈금 라벨을 지정하는 `xticks()` 함수에 `size=10` 옵션을 추가하여 폰트 크기를 10으로 설정한다. 39라인의 `plot()` 함수에 `marker='o'` 옵션을 추가하면 원 모양의 점을 마커로 표시한다. `markersize=10` 은 마커 사이즈를 10으로 설정하는 옵션이다.

〈예제 4-5〉 스타일 서식 지정 등 (File: example/part4/4.5_matplotlib_line4.py)

```
~  ~~ 생략 (예제 4-4와 동일) ~~

# 서울에서 경기도로 이동한 인구 데이터 값만 선택
sr_one = df_seoul.loc['경기도']

# 스타일 서식 지정
plt.style.use('ggplot')

# 그림 사이즈 지정
plt.figure(figsize=(14, 5))
```

1. Matplotlib - 기본 그래프 도구

```
35  # x축 눈금 라벨 회전하기
36  plt.xticks(size=10, rotation='vertical')
37
38  # x, y축 데이터를 plot 함수에 입력
39  plt.plot(sr_one.index, sr_one.values, marker='o', markersize=10)    # 마커 표시 추가
40
41  plt.title('서울 -> 경기 인구 이동', size=30)    # 차트 제목
42  plt.xlabel('기간', size=20)                    # x축 이름
43  plt.ylabel('이동 인구수', size=20)             # y축 이름
44
45  plt.legend(labels=['서울 -> 경기'], loc='best', fontsize=15)          # 범례 표시
46
47  plt.show()  # 변경사항 저장하고 그래프 출력
```

〈실행 결과〉 코드 전부 실행

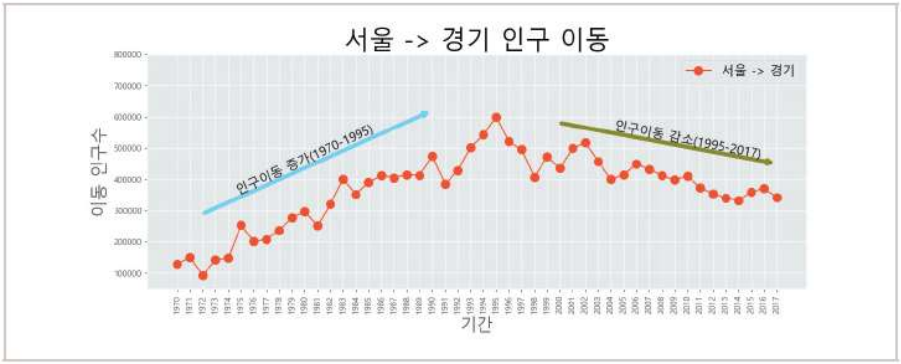
그래프에 대한 설명을 덧붙이는 주석에 대해 알아보자. annotate() 함수를 사용한다. 주석 내용(텍스트)을 넣을 위치와 정렬 방법 등을 annotate() 함수에 함께 전달한다. arrowprops 옵션을 사용하면 텍스트 대신 화살표가 표시된다. 화살표 스타일, 시작점과 끝점의 좌표를 입력한다. 〈예제 4-7〉에서는 주석을 넣을 여백 공간을 충분히 확보하기 위해 ylim() 함수를 사용하여 y축 범위를 먼저 늘려준다. 그리고 annotate() 함수로 화살표와 텍스트 위치를 잡아서 배치한다. 위치를 나타내는 (x, y) 좌표에서 x값은 인덱스 번호를 사용한다. y 좌표값에 들어갈 인구수 데이터는 숫자값이므로 그대로 사용할 수 있다. 53라인의 xy = (20, 620000)은 인덱스 번호 20을 x값으로 하고 620000명을 y값으로 한다는 뜻이다.

〈예제 4-7〉 matplotlib 스타일 리스트 출력 (File: example/part4/4.7_matplotlib_annotate.py)

```
~  ~~ 생략 (예제 4-5와 동일) ~~

48  # y축 범위 지정(최소값, 최대값)
49  plt.ylim(50000, 800000)
50
51  # 주석 표시 - 화살표
52  plt.annotate('',
53               xy=(20, 620000),        # 화살표의 머리 부분(끝점)
54               xytext=(2, 290000),     # 화살표의 꼬리 부분(시작점)
55               xycoords='data',        # 좌표체계
56               arrowprops=dict(arrowstyle='->', color='skyblue', lw=5),  # 화살표 서식
57               )
58
59  plt.annotate('',
60               xy=(47, 450000),        # 화살표의 머리 부분(끝점)
61               xytext=(30, 580000),    # 화살표의 꼬리 부분(시작점)
62               xycoords='data',        # 좌표체계
63               arrowprops=dict(arrowstyle='->', color='olive', lw=5),    # 화살표 서식
64               )
65
66  # 주석 표시 - 텍스트
67  plt.annotate('인구 이동 증가(1970-1995)',   # 텍스트 입력
68               xy=(10, 550000),            # 텍스트 위치 기준점
69               rotation=25,                # 텍스트 회전 각도
70               va='baseline',              # 텍스트 상하 정렬
71               ha='center',                # 텍스트 좌우 정렬
72               fontsize=15,                # 텍스트 크기
73               )
74
75  plt.annotate('인구 이동 감소(1995-2017)',   # 텍스트 입력
76               xy=(40, 560000),            # 텍스트 위치 기준점
77               rotation=-11,               # 텍스트 회전 각도
78               va='baseline',              # 텍스트 상하 정렬
79               ha='center',                # 텍스트 좌우 정렬
80               fontsize=15,                # 텍스트 크기
81               )
82
83  plt.show()  # 변경사항 저장하고 그래프 출력
```

1. Matplotlib - 기본 그래프 도구

〈실행 결과〉 코드 전부 실행



● **화면 분할하여 그래프 여러 개 그리기 – axe 객체 활용**

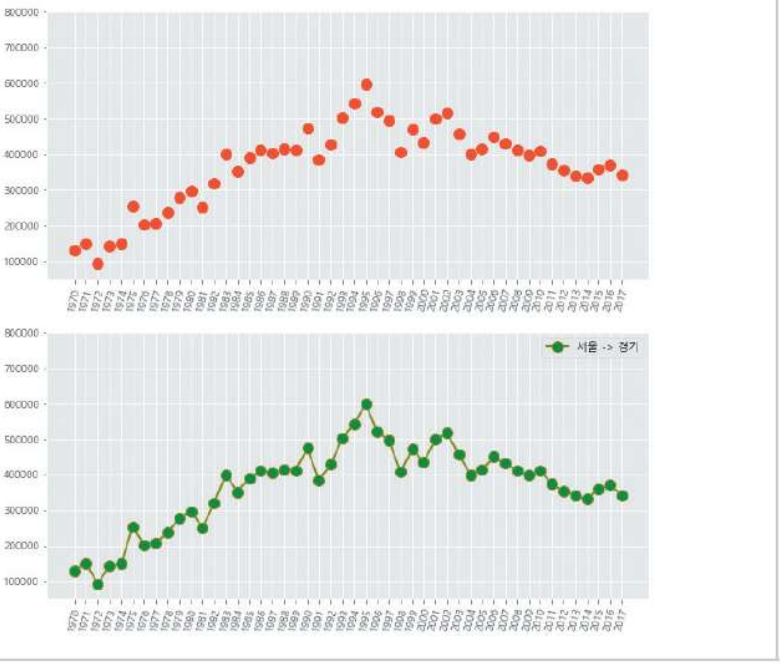
화면을 여러 개로 분할하고 분할된 각 화면에 서로 다른 그래프를 그리는 방법이다. 여러 개의 axe 객체를 만들고, 분할된 화면마다 axe 객체를 하나씩 배정한다. axe 객체는 각각 서로 다른 그래프를 표현할 수 있다. 한 화면에서 여러 개의 그래프를 비교하거나 다양한 정보를 동시에 보여줄 때 사용하면 좋다. 단, axe 객체를 1개만 생성하는 경우에는 하나의 그래프만 표시된다.

figure() 함수를 사용하여 그래프를 그리는 그림틀(fig)을 만든다. figsize 옵션으로 (가로, 세로) 그림틀의 크기를 설정한다. fig 객체에 add_subplot() 메소드를 적용하여 그림틀을 여러 개로 분할한다. 이때 나눠진 각 부분을 axe 객체라고 부른다.

〈예제 4-8〉 Matplotlib 소개 (File: example/part4/4.8_matplotlib_lines1.py)

```
~  ~~ 생략 (예제 4-7과 동일) ~~

32  # 그래프 객체 생성(figure에 2개의 서브 플롯 생성)
33  fig = plt.figure(figsize=(10, 10))
34  ax1 = fig.add_subplot(2, 1, 1)
35  ax2 = fig.add_subplot(2, 1, 2)
36
37  # axe 객체에 plot 함수로 그래프 출력
38  ax1.plot(sr_one, 'o', markersize=10)
39  ax2.plot(sr_one, marker='o', markerfacecolor='green', markersize=10,
40           color='olive', linewidth=2, label='서울 -> 경기')
41  ax2.legend(loc='best')
42
43  # y축 범위 지정(최소값, 최대값)
44  ax1.set_ylim(50000, 800000)
45  ax2.set_ylim(50000, 800000)
46
47  # 축 눈금 라벨 지정 및 75도 회전
48  ax1.set_xticklabels(sr_one.index, rotation=75)
49  ax2.set_xticklabels(sr_one.index, rotation=75)
50
51  plt.show()  # 변경사항 저장하고 그래프 출력
```

〈실행 결과〉 코드 전부 실행



1. Matplotlib - 기본 그래프 도구

앞에서 그린 두 번째 그래프에 제목과 축 이름을 추가해보자. 〈예제 4-9〉에서는 그래프를 1개만 표시하기 위하여 fig.add_subplot(1, 1, 1)과 같이 axe 객체를 1개만 생성하고 변수 ax에 할당한다. 그리고 plot() 메소드를 이용하여 ax 객체에 그래프를 그린다. ax 객체에 set_title() 메소드를 적용하여 제목을 추가한다. set_xlabel() 메소드로 x축 이름을 지정하고, set_ylabel() 메소드로 y축 이름을 지정한다. tick_params() 메소드로 축 눈금 라벨의 크기를 조절한다.

〈예제 4-9〉 axe 객체 그래프 꾸미기 (File: example/part4/4.9_matplotlib_lines2.py)

```
~  ~~ 생략 (예제 4-8과 동일) ~~

# 그래프 객체 생성(figure에 1개의 서브 플롯 생성)
fig = plt.figure(figsize=(20, 5))
ax = fig.add_subplot(1, 1, 1)

# axe 객체에 plot 함수로 그래프 출력
ax.plot(sr_one, marker='o', markerfacecolor='orange', markersize=10,
        color='olive', linewidth=2, label='서울 -> 경기')
ax.legend(loc='best')

# y축 범위 지정(최소값, 최대값)
ax.set_ylim(50000, 800000)

# 차트 제목 추가
ax.set_title('서울 -> 경기 인구 이동', size=20)

# 축 이름 추가
ax.set_xlabel('기간', size=12)
ax.set_ylabel('이동 인구수', size = 12)

# 축 눈금 라벨 지정 및 75도 회전
ax.set_xticklabels(sr_one.index, rotation=75)

# 축 눈금 라벨 크기
ax.tick_params(axis="x", labelsize=10)
ax.tick_params(axis="y", labelsize=10)

plt.show()  # 변경사항 저장하고 그래프 출력
```

〈실행 결과〉 코드 전부 실행

예제 팁

선 그래프의 꾸미기 옵션

꾸미기 옵션	설명
'o'	선 그래프가 아니라 점 그래프로 표현
marker='o'	마커 모양 (예: 'o', '+', '*', '.')
markerfacecolor='green'	마커 배경색
markersize=10	마커 크기
color='olive'	선의 색
linewidth=2	선의 두께
label='서울 -> 경기'	라벨 지정

[표 3-1] 선 그래프의 꾸미기 옵션

1. Matplotlib - 기본 그래프 도구

동일한 그림(axe 객체)에 여러 개의 그래프를 추가하는 것도 가능하다. 〈예제 4-10〉에서는 서울특별시에서 충청남도, 경상북도, 강원도로 이동한 인구 변화 그래프 3개를 하나의 같은 화면에 그려본다. 먼저 각 지역에 해당하는 행을 선택하고, 동일한 axe 객체(ax)에 선 그래프로 출력하는 plot() 메소드를 3번 적용한다. 그리고 범례와 차트 제목 등을 표시한다.

〈예제 4-10〉 같은 화면에 그래프 추가 (File: example/part4/4.10_matplotlib_lines3.py)

```
~  ~~ 생략 (예제 4-9와 동일) ~~

# 서울에서 '충청남도', '경상북도', '강원도'로 이동한 인구 데이터 값만 선택
col_years = list(map(str, range(1970, 2018)))
df_3 = df_seoul.loc[['충청남도', '경상북도', '강원도'], col_years]

# 스타일 서식 지정
plt.style.use('ggplot')

# 그래프 객체 생성(figure에 1개의 서브 플롯 생성)
fig = plt.figure(figsize=(20, 5))
ax = fig.add_subplot(1, 1, 1)

# axe 객체에 plot 함수로 그래프 출력
ax.plot(col_years, df_3.loc['충청남도',:], marker='o', markerfacecolor='green',
        markersize=10, color='olive', linewidth=2, label='서울 -> 충남')
ax.plot(col_years, df_3.loc['경상북도',:], marker='o', markerfacecolor='blue',
        markersize=10, color='skyblue', linewidth=2, label='서울 -> 경북')
ax.plot(col_years, df_3.loc['강원도',:], marker='o', markerfacecolor='red',
        markersize=10, color='magenta', linewidth=2, label='서울 -> 강원')

# 범례 표시
ax.legend(loc='best')

# 차트 제목 추가
ax.set_title('서울 -> 충남, 경북, 강원 인구 이동', size=20)

# 축 이름 추가
ax.set_xlabel('기간', size=12)
ax.set_ylabel('이동 인구수', size = 12)

# 축 눈금 라벨 지정 및 90도 회전
ax.set_xticklabels(col_years, rotation=90)

# 축 눈금 라벨 크기
ax.tick_params(axis="x", labelsize=10)
ax.tick_params(axis="y", labelsize=10)

plt.show()  # 변경사항 저장하고 그래프 출력
```

〈실행 결과〉 코드 전부 실행

이처럼 같은 axe 객체에 그래프 여러 개를 동시에 표시할 수 있다. 서울에서 서로 다른 3개 지역으로 빠져나간 인구 이동을 비교 파악하기가 쉽다. 특히 지리적으로 가까운 충남 지역으로 이동한 인구가 다른 두 지역에 비해 많은 편이다. 전반적으로 1970~80년대에는 서울에서 지방으로 전출하는 인구가 많았으나, 1990년 이후로는 줄곧 감소하는 패턴을 보이고 있다.

서울특별시에서 충청남도, 경상북도, 강원도, 전라남도 4개 지역으로 이동한 인구 변화 그래프를 그려본다. ax1~ax4까지 4개의 axe 객체를 생성한다. 각 지역에 해당하는 4개의 행을 선택하고, axe 객체에 하나씩 plot() 메소드를 적용한다. 그리고 범례와 차트 제목 등을 표시한다.

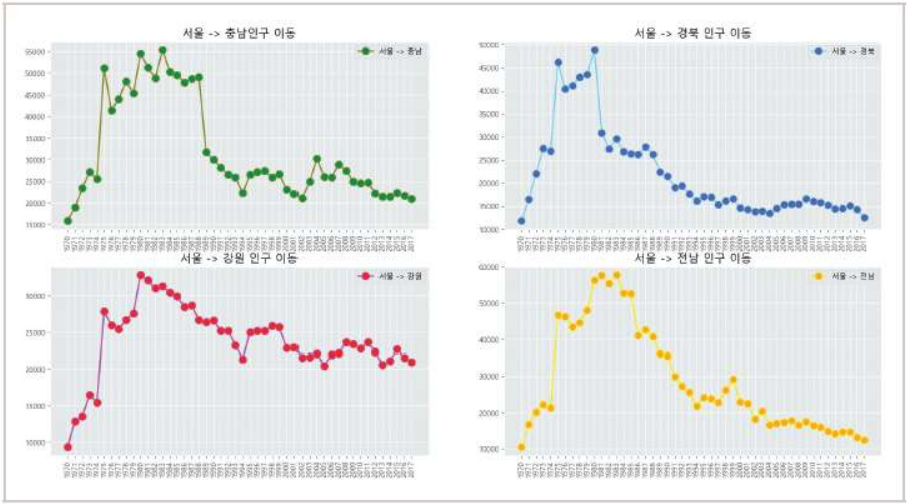
〈예제 4-11〉 화면 4분할 그래프 (File: example/part4/4.11_matplotlib_lines4.py)

```
~  ~~ 생략 (예제 4-10과 동일) ~~

# 서울에서 '충청남도', '경상북도', '강원도', '전라남도'로 이동한 인구 데이터 값만 선택
col_years = list(map(str, range(1970, 2018)))
df_4 = df_seoul.loc[['충청남도', '경상북도', '강원도', '전라남도'], col_years]

```

1. Matplotlib - 기본 그래프 도구

```
30  # 스타일 서식 지정
31  plt.style.use('ggplot')
32
33  # 그래프 객체 생성(figure에 1개의 서브 플롯 생성)
34  fig = plt.figure(figsize=(20, 10))
35  ax1 = fig.add_subplot(2, 2, 1)
36  ax2 = fig.add_subplot(2, 2, 2)
37  ax3 = fig.add_subplot(2, 2, 3)
38  ax4 = fig.add_subplot(2, 2, 4)
39
40  # axe 객체에 plot 함수로 그래프 출력
41  ax1.plot(col_years, df_4.loc['충청남도',:], marker='o', markerfacecolor='green',
42          markersize=10, color='olive', linewidth=2, label='서울 -> 충남')
43  ax2.plot(col_years, df_4.loc['경상북도',:], marker='o', markerfacecolor='blue',
44          markersize=10, color='skyblue', linewidth=2, label='서울 -> 경북')
45  ax3.plot(col_years, df_4.loc['강원도',:], marker='o', markerfacecolor='red',
46          markersize=10, color='magenta', linewidth=2, label='서울 -> 강원')
47  ax4.plot(col_years, df_4.loc['전라남도',:], marker='o', markerfacecolor='orange',
48          markersize=10, color='yellow', linewidth=2, label='서울 -> 전남')
49
50  # 범례 표시
51  ax1.legend(loc='best')
52  ax2.legend(loc='best')
53  ax3.legend(loc='best')
54  ax4.legend(loc='best')
55
56  # 차트 제목 추가
57  ax1.set_title('서울 -> 충남 인구 이동', size=15)
58  ax2.set_title('서울 -> 경북 인구 이동', size=15)
59  ax3.set_title('서울 -> 강원 인구 이동', size=15)
60  ax4.set_title('서울 -> 전남 인구 이동', size=15)
61
62  # 축 눈금 라벨 지정 및 90도 회전
63  ax1.set_xticklabels(col_years, rotation=90)
64  ax2.set_xticklabels(col_years, rotation=90)
65  ax3.set_xticklabels(col_years, rotation=90)
66  ax4.set_xticklabels(col_years, rotation=90)
67
68  plt.show()  # 변경사항 저장하고 그래프 출력
```

〈실행 결과〉 코드 전부 실행



1-2 면적 그래프

면적 그래프(area plot)는 각 열의 데이터를 선 그래프로 구현하는데, 선 그래프와 x축 사이의 공간에 색이 입혀진다. 색의 투명도(alpha)는 기본값 0.5로 투과되어 보인다(투명도: 0~1 범위).
선 그래프를 그리는 plot() 메소드에 kind='area' 옵션을 추가하면 간단하게 그릴 수 있다.
그래프를 누적할지 여부를 설정할 수 있는데, 기본값은 stacked=True 옵션이다.

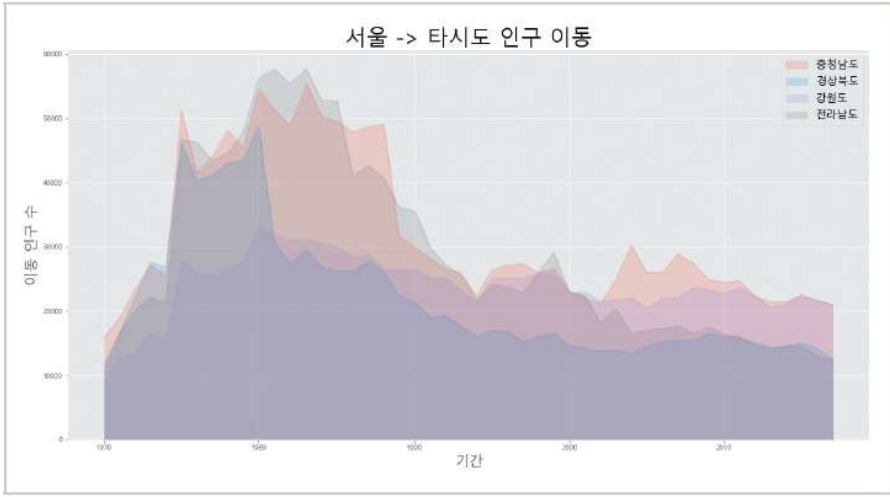
〈예제 4-13〉 면적 그래프(stacked=False) 그리기 (File: example/part4/4.13_matplotlib_area1.py)

```
~  ~~ 생략 (예제 4-11과 동일) ~~

19  # 서울에서 다른 지역으로 이동한 데이터만 추출하여 정리
20  mask = (df['전출지별'] == '서울특별시') & (df['전입지별'] != '서울특별시')
21  df_seoul = df[mask]
22  df_seoul = df_seoul.drop(['전출지별'], axis=1)
23  df_seoul.rename({'전입지별':'전입지'}, axis=1, inplace=True)
24  df_seoul.set_index('전입지', inplace=True)
25
26  # 서울에서 '충청남도', '경상북도', '강원도', '전라남도'로 이동한 인구 데이터 값만 선택
27  col_years = list(map(str, range(1970, 2018)))
```

1. Matplotlib - 기본 그래프 도구

```
28  df_4 = df_seoul.loc[['충청남도', '경상북도', '강원도', '전라남도'], col_years]
29  df_4 = df_4.transpose()
30
31  # 스타일 서식 지정
32  plt.style.use('ggplot')
33
34  # 데이터프레임의 인덱스를 정수형으로 변경(x축 눈금 라벨 표시)
35  df_4.index = df_4.index.map(int)
36
37  # 면적 그래프 그리기
38  df_4.plot(kind='area', stacked=False, alpha=0.2, figsize=(20, 10))
39
40  plt.title('서울 -> 타시도 인구 이동', size=30)
41  plt.ylabel('이동 인구 수', size=20)
42  plt.xlabel('기간', size=20)
43  plt.legend(loc='best', fontsize=15)
44
45  plt.show()
```

〈실행 결과〉 코드 전부 실행

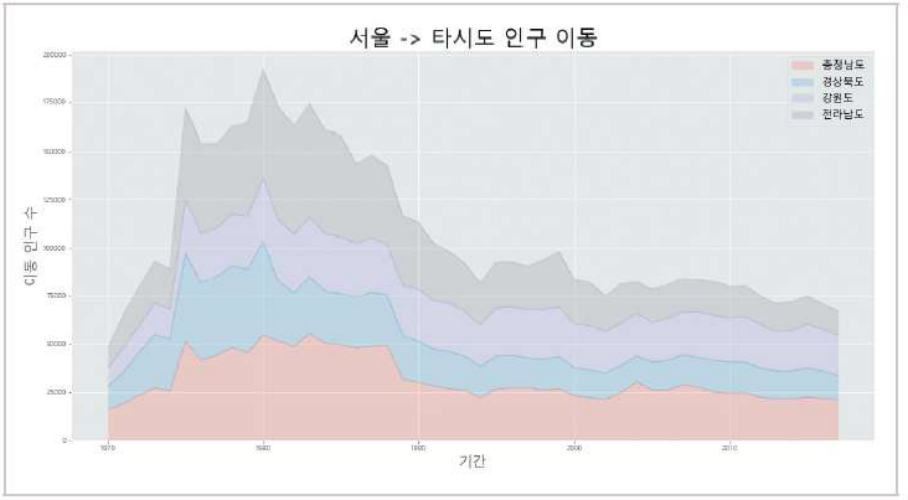


이번에는 stacked=True 옵션을 지정하여 선 그래프들이 서로 겹치지 않고 위 아래로 데이터가 누적(stacked)되는 면적 그래프를 그려본다.

〈예제 4-14〉 면적 그래프(stacked=False) 그리기 (File: example/part4/4.14_matplotlib_area2.py)

```
 -  ~~ 생략 (예제 4-13과 동일) ~~

34  # 데이터프레임의 인덱스를 정수형으로 변경(x축 눈금 라벨 표시)
35  df_4.index = df_4.index.map(int)
36
37  # 면적 그래프 그리기
38  df_4.plot(kind='area', stacked=True, alpha=0.2, figsize=(20, 10))
39
40  plt.title('서울 -> 타시도 인구 이동', size=30)
41  plt.ylabel('이동 인구 수', size=20)
42  plt.xlabel('기간', size=20)
43  plt.legend(loc='best', fontsize=15)
44
45  plt.show()
```

〈실행 결과〉 코드 전부 실행



앞에서 plot() 메소드로 생성한 그래프는 axe 객체이다. 다음과 같이 axe 객체(ax)의 세부적인 요소를 설정할 수 있다. axe 객체의 속성을 이용하여 제목, 축 이름 등을 설정한다.

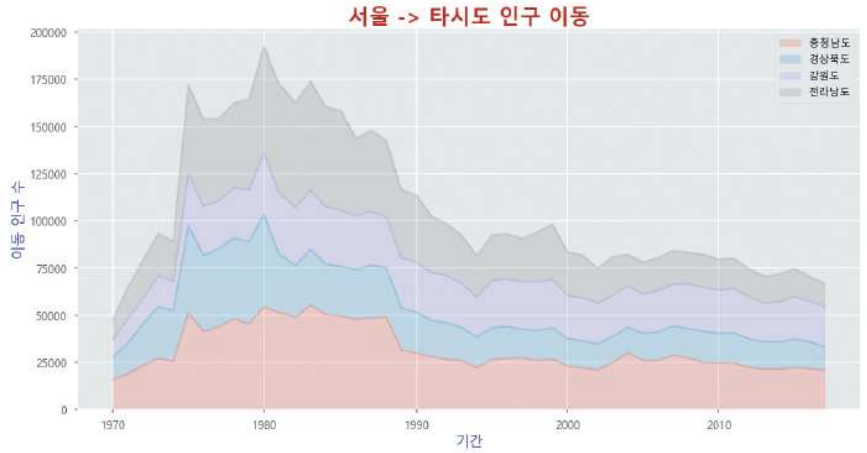
1. Matplotlib - 기본 그래프 도구

〈예제 4-15〉 axes 객체 속성 변경하기 (File: example/part4/4.15_matplotlib_area3.py)

```
~  ~~ 생략 (예제 4-14와 동일) ~~

34  # 데이터프레임의 인덱스를 정수형으로 변경(x축 눈금 라벨 표시)
35  df_4.index = df_4.index.map(int)
36
37  # 면적 그래프 axe 객체 생성
38  ax = df_4.plot(kind='area', stacked=True, alpha=0.2, figsize=(20, 10))
39  print(type(ax))
40
41  # axe 객체 설정 변경
42  ax.set_title('서울 -> 타시도 인구 이동', size=30, color='brown', weight='bold')
43  ax.set_ylabel('이동 인구 수', size=20, color='blue')
44  ax.set_xlabel('기간', size=20, color='blue')
45  ax.legend(loc='best', fontsize=15)
46
47  plt.show()
```

〈실행 결과〉 코드 전부 실행

```
<class 'matplotlib.axes._subplots.AxesSubplot'>
```



1-3 막대 그래프

막대 그래프(bar plot)는 데이터 값의 크기에 비례하여 높이를 갖는 직사각형 막대로 표현한다. 막대 높이의 상대적 길이 차이를 통해 값의 크고 작음을 설명한다. 세로형과 가로형 막대 그래프 두 종류가 있다. 다만, 세로형의 경우 정보 제공 측면에서 보면 선 그래프와 큰 차이가 없다.

세로형 막대 그래프는 시간적으로 차이가 나는 두 점에서 데이터 값의 차이를 잘 설명한다. 즉, 시계열 데이터를 표현하는데 적합하다. `plot()` 메소드에 `kind='bar'` 옵션을 입력한다.

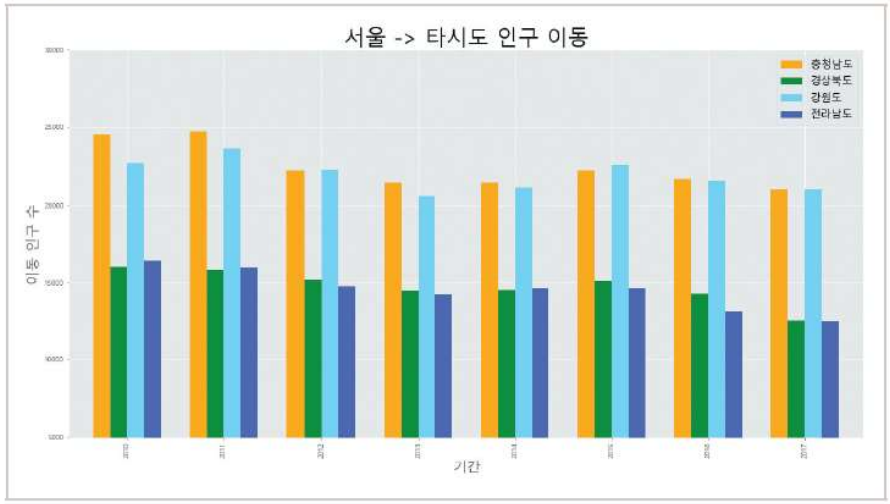
2010~2017년에 해당하는 데이터를 추출하기 위해, 27라인의 `col_years` 변수에 저장하는 값의 범위를 변경한다. `plot()` 메소드의 `color` 옵션을 추가하여 막대 색상을 다르게 설정한다.

〈예제 4-16〉 세로형 막대 그래프 (File: example/part4/4.16_matplotlib_bar1.py)

```
~  ~~ 생략 (예제 4-15와 동일) ~~

26  # 서울에서 '충청남도', '경상북도', '강원도', '전라남도'로 이동한 인구 데이터 값만 선택
27  col_years = list(map(str, range(2010, 2018)))
28  df_4 = df_seoul.loc[['충청남도', '경상북도', '강원도', '전라남도'], col_years]
29  df_4 = df_4.transpose()
30
31  # 스타일 서식 지정
32  plt.style.use('ggplot')
33
34  # 데이터프레임의 인덱스를 정수형으로 변경(x축 눈금 라벨 표시)
35  df_4.index = df_4.index.map(int)
36
37  # 막대 그래프 그리기
38  df_4.plot(kind='bar', figsize=(20, 10), width=0.7,
39            color=['orange', 'green', 'skyblue', 'blue'])
40
41  plt.title('서울 -> 타시도 인구 이동', size=30)
42  plt.ylabel('이동 인구 수', size=20)
43  plt.xlabel('기간', size=20)
44  plt.ylim(5000, 30000)
45  plt.legend(loc='best', fontsize=15)
46
47  plt.show()
```

1. Matplotlib - 기본 그래프 도구

〈실행 결과〉 코드 전부 실행

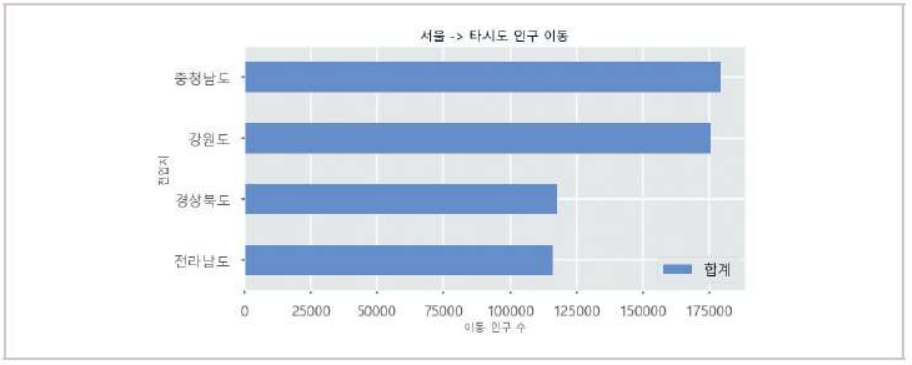


가로형 막대 그래프는 각 변수 사이 값의 크기 차이를 설명하는데 적합하다. plot() 메소드의 옵션으로 kind='barh'를 입력한다. 다음의 예제에서는 2010~2017년의 기간 동안 서울에서 각 시도로 이동한 인구의 합계를 구하여 시도별로 비교하는 그래프를 그린다.

〈예제 4-17〉 가로형 막대 그래프 (File: example/part4/4.17_matplotlib_barh1.py)

```
~  ~~ 생략 (예제 4-16과 동일) ~~

# 서울에서 '충청남도', '경상북도', '강원도', '전라남도'로 이동한 인구 데이터 값만 선택
col_years = list(map(str, range(2010, 2018)))
df_4 = df_seoul.loc[['충청남도', '경상북도', '강원도', '전라남도'], col_years]

# 2010-2017년 이동 인구 수를 합계하여 새로운 열로 추가
df_4['합계'] = df_4.sum(axis=1)

# 가장 큰 값부터 정렬
df_total = df_4[['합계']].sort_values(by='합계', ascending=True)

# 스타일 서식 지정
plt.style.use('ggplot')

# 수평 막대 그래프 그리기
df_total.plot(kind='barh', color='cornflowerblue', width=0.5, figsize=(10, 5))

plt.title('서울 -> 타시도 인구 이동')
plt.ylabel('전입지')
plt.xlabel('이동 인구 수')

plt.show()
```

〈실행 결과〉 코드 전부 실행



2010~2017년에 이동 인구 합계를 기준으로 서울에서 충청남도로 이동한 사람이 제일 많다. 다음으로 강원도, 경상북도, 전라남도 순으로 나타난다.

예제 창고

보조 축 활용하기(2축 그래프 그리기)

지금까지 그래프를 그릴 때 y축을 한 개만 사용하였다. Excel에서 차트를 그릴 때처럼 보조 축을 추가하여 2개의 y축을 갖는 그래프를 그릴 수 있다.

남북한 발전량 데이터셋[4]을 사용하여 보조 축을 설정하는 방법을 살펴보자. 자료실에서 Excel 파일[5]을 다운로드 받아 파이썬 파일과 같은 폴더에 저장한다. 기존 축에는 막대 그래프의 값을 표시하고 보조 축에는 선 그래프의 값을 표시한다. 막대 그래프는 연도별 북한의 발전량을 나타내고 선 그래프는 북한 발전량의 전년 대비 증감률을 백분률로 나타낸다.

증감률을 계산하기 위해 rename() 메소드로 '합계' 열의 이름을 '총발전량'으로 바꾸고, shift() 메소드를 이용하여 '총발전량' 열의 데이터를 1행씩 뒤로 이동시켜서 '총발전량 - 1년' 열을 새로 생성한다. 그리고 두 열의 데이터를 이용하여 전년도 대비 변동율을 계산한 결과를 '증감률' 열에 저장한다.

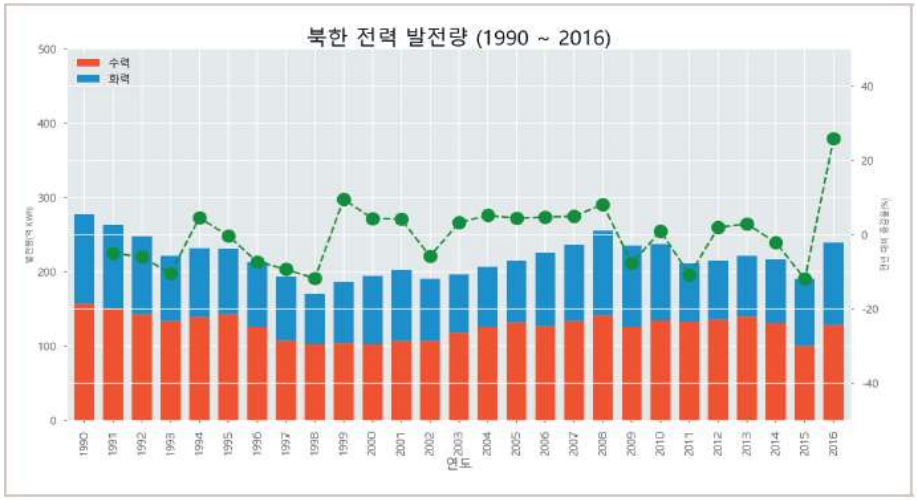
1. Matplotlib - 기본 그래프 도구

ax1 객체는 막대 그래프에 stacked=True 옵션을 지정하여, '수력', '화력' 열의 값을 아래 위로 쌓은 형태의 세로형 막대 그래프를 그린다. ax1 객체에 twinx() 메소드를 적용하여 ax1 객체의 쌍둥이 객체를 만들고, 쌍둥이 객체를 ax2 변수에 저장한다. ax2 객체에 plot() 메소드를 적용하여 선 그래프를 그린다. 그래프를 그리는데 사용할 데이터는 '증감률' 열에서 가져온다. ls='- - -' 옵션은 선 스타일(line style)을 점선으로 설정하는 명령이다.

〈예제 4-18〉 2축 그래프 그리기(File: example/part4/4.18_matplotlib_secondary_y.py)

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

# 라이브러리 불러오기
import pandas as pd
import matplotlib.pyplot as plt

# matplotlib 한글 폰트 오류 문제 해결
from matplotlib import font_manager, rc
font_path = "./malgun.ttf"                    # 폰트 파일 위치
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc('font', family=font_name)

plt.style.use('ggplot')                       # 스타일 서식 지정
plt.rcParams['axes.unicode_minus']=False      # 마이너스 부호 출력 설정

# Excel 데이터를 데이터프레임으로 변환
df = pd.read_excel('./남북한발전전력량.xlsx', convert_float=True)
df = df.loc[5:9]
df.drop('전력량(억kWh)', axis='columns', inplace=True)
df.set_index('발전 전력별', inplace=True)
df = df.T

# 증감률(변동률) 계산
df = df.rename(columns={'합계':'총발전량'})
df['총발전량 - 1년'] = df['총발전량'].shift(1)
df['증감률'] = ((df['총발전량']/df['총발전량 - 1년']) - 1) * 100

# 2축 그래프 그리기
ax1 = df[['수력','화력']].plot(kind='bar', figsize=(20, 10), width=0.7, stacked=True)
ax2 = ax1.twinx()
ax2.plot(df.index, df.증감률, ls='--', marker='o', markersize=20,
         color='red', label='전년대비 증감률(%)')

ax1.set_ylim(0, 500)
ax2.set_ylim(-50, 50)
ax1.set_xlabel('연도', size=20)
ax1.set_ylabel('발전량(억 KWh)')
ax2.set_ylabel('전년 대비 증감률(%)')

plt.title('북한 전력 발전량 (1990~2016)', size=30)
ax1.legend(loc='upper left')

plt.show()
```

<실행 결과> 코드 전부 실행



2015년 수력 발전량이 일시적으로 급감한 사실이 있다. 기사를 검색해 보면 2015년에 북한의 가뭄이 심각했다는 뉴스를 찾아볼 수 있다.

1-4 히스토그램

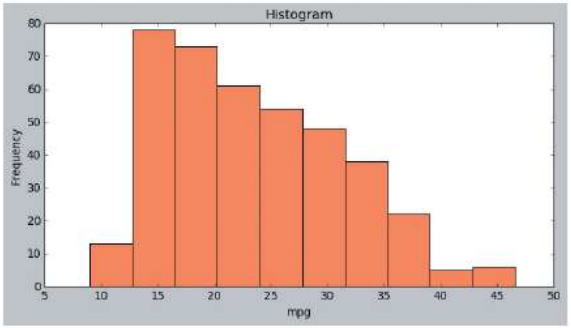
히스토그램(histogram)은 변수가 하나인 단변수 데이터의 빈도수를 그래프로 표현한다. x축을 같은 크기의 여러 구간으로 나누고 각 구간에 속하는 데이터 값의 개수(빈도)를 y축에 표시한다. 구간을 나누는 간격의 크기에 따라 빈도가 달라지고 히스토그램의 모양이 변한다.

1. Matplotlib - 기본 그래프 도구

〈예제 4-19〉 히스토그램 (File: example/part4/4.19_matplotlib_hist.py)

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

# 라이브러리 불러오기
import pandas as pd
import matplotlib.pyplot as plt

plt.style.use('classic')   # 스타일 서식 지정

# 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']

# 연비(mpg) 열에 대한 히스토그램 그리기
df['mpg'].plot(kind='hist', bins=10, color='coral', figsize=(10, 5))

# 그래프 꾸미기
plt.title('HIstogram')
plt.xlabel('mpg')
plt.show()
```

〈실행 결과〉 코드 전부 실행



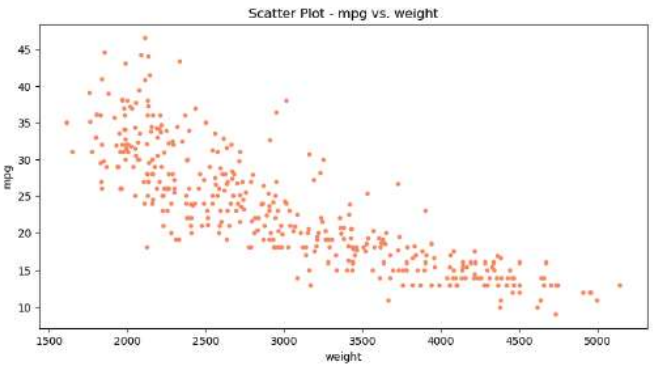
1-5 산점도

plot() 메소드에 kind='scatter' 옵션을 사용하여 산점도를 그린다. x='weight' 옵션을 사용하여 x축에 위치할 변수(데이터프레임의 열)를 선택한다. 마찬가지로 y='mpg' 옵션을 지정하여 'mpg' 열을 y축에 놓을 변수로 선택한다. 점의 색상(c)과 크기(s)를 설정하는 옵션을 추가한다.

〈예제 4-20〉 산점도 (File: example/part4/4.20_matplotlib_scatter.py)

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

# 라이브러리 불러오기
import pandas as pd
import matplotlib.pyplot as plt

plt.style.use('default')   # 스타일 서식 지정

# 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']

# 연비(mpg)와 차중(weight) 열에 대한 산점도 그리기
df.plot(kind='scatter', x='weight', y='mpg', c='coral', s=10, figsize=(10, 5))
```

〈실행 결과〉 코드 전부 실행



1. Matplotlib - 기본 그래프 도구

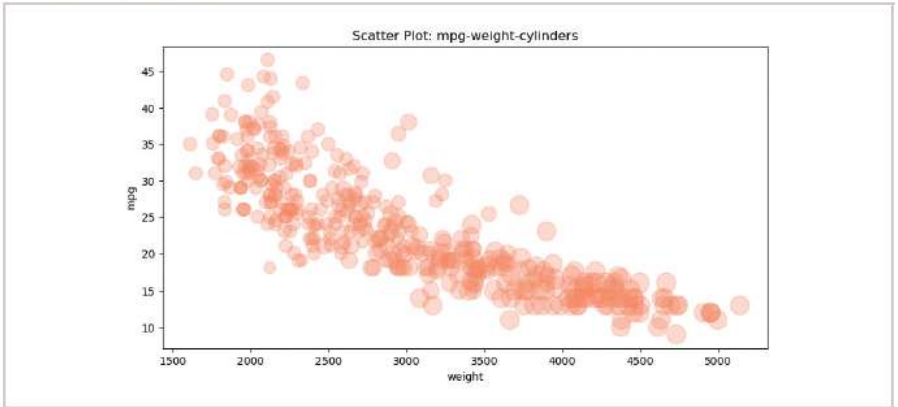
앞에서 자동차 무게와 연비 사이의 관계를 산점도로 표현하였다. 여기에 새로운 변수를 추가해서 점의 크기 또는 색상으로 표현할 수 있다. 여기서는 3번째 변수로 실린더 개수('cylinders' 열)를 추가해 보자.

실린더 개수를 나타내는 정수를 그대로 쓰는 대신, 해당 열의 최대값 대비 상대적 크기를 나타내는 비율을 계산하여 cylinders_size 변수에 저장한다. cylinders_size는 0~1 범위의 실수 값의 배열(시리즈)이다. 점의 크기를 정하는 s 옵션에 cylinders_size를 입력하여 값의 크기에 따라 점의 크기를 값에 따라 다르게 표시한다. 이처럼 점의 크기에 변화를 주면 모양이 비눗방울 같다고 해서 버블(bubble) 차트라고 부르기도 한다.

〈예제 4-21〉 버블 차트 (File: example/part4/4_21_matplotlib_bubble.py)

```
 ~  ~~ 생략 (예제 4-20과 동일) ~~

16  # cylinders 개수의 상대적 비율을 계산하여 시리즈 생성
17  cylinders_size = df.cylinders/df.cylinders.max() * 300
18
19  # 3개의 변수로 산점도 그리기
20  df.plot(kind='scatter', x='weight', y='mpg', c='coral', figsize=(10, 5),
21          s=cylinders_size, alpha=0.3)
22  plt.title('Scatter Plot: mpg-weight-cylinders')
23  plt.show()
```

〈실행 결과〉 코드 전부 실행



1-6 파이 차트

파이 차트(pie chart)는 원을 파이 조각처럼 나누어서 표현한다. 조각의 크기는 해당 변수에 속하는 데이터 값의 크기에 비례한다. plot() 메소드에 kind='pie' 옵션을 사용하여 그린다.

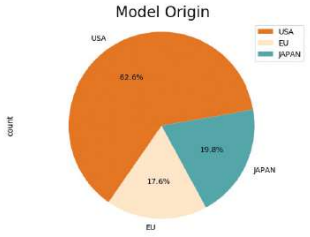
〈예제 4-23〉 파이 차트 (File: example/part4/4_23_matplotlib_pie.py)

```
1  # -*- coding: utf-8 -*-
2
3  # 라이브러리 불러오기
4  import pandas as pd
5  import matplotlib.pyplot as plt
6
7  plt.style.use('default')   # 스타일 서식 지정
8
9  # read_csv() 함수로 df 생성
10 df = pd.read_csv('./auto-mpg.csv', header=None)
11
12 # 열 이름 지정
13 df.columns = ['mpg','cylinders','displacement','horsepower','weight',
14               'acceleration','model year','origin','name']
15
16 # 데이터 개수 카운트를 위해 값 1을 가진 열 추가
17 df['count'] = 1
18 df_origin = df.groupby('origin').sum()     # origin 열을 기준으로 그룹화, 합계 연산
19 print(df_origin.head())                    # 그룹 연산 결과 출력
20
21 # 제조국가(origin) 값을 실제 지역명으로 변경
22 df_origin.index = ['USA', 'EU', 'JPN']
23
24 # 제조국가(origin) 열에 대한 파이 차트 그리기 - count 열 데이터 사용
25 df_origin['count'].plot(kind='pie',
26                         figsize=(7, 5),
27                         autopct='%1.1f%%',    # 퍼센트 % 표시
28                         startangle=10,    # 파이 조각을 나누는 시작점(각도 표시)
29                         colors=['chocolate', 'bisque', 'cadetblue']    # 색상 리스트
30                         )
31
32 plt.title('Model Origin', size=20)
33 plt.axis('equal')    # 파이 차트의 비율을 같게(원에 가깝게) 조정
34 plt.legend(labels=df_origin.index, loc='upper right')   # 범례 표시
35 plt.show()
```

1. Matplotlib - 기본 그래프 도구

〈실행 결과〉 코드 전부 실행

```
          mpg  cylinders  displacement  ...  acceleration  model year  count
origin                                  ...
1      5000.8       1556       61229.5  ...        3743.4       18827    249
2      1952.4        291        7640.0  ...        1175.1        5307     70
3      2405.6        324        8114.0  ...        1277.6        6118     79

[3 rows x 7 columns]
```

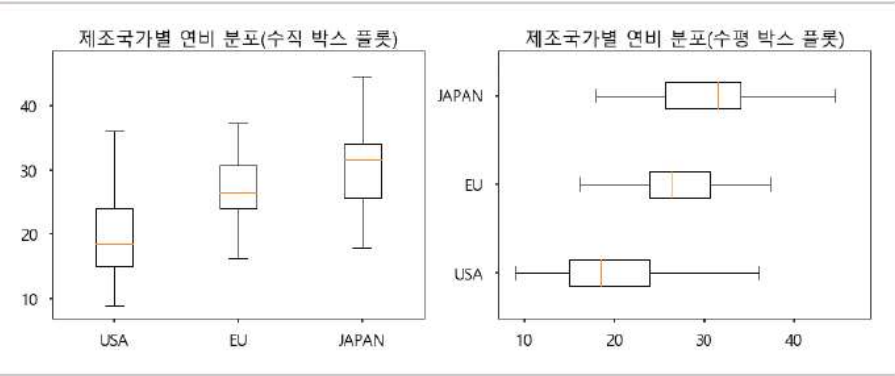


1-7 박스 플롯

〈예제 4-24〉 박스 플롯 (File: example/part4/4.24_matplotlib_box.py)

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

# 라이브러리 불러오기
import pandas as pd
import matplotlib.pyplot as plt

# matplotlib 한글 폰트 오류 문제 해결
from matplotlib import font_manager, rc
font_path = "./malgun.ttf"   # 폰트 파일 위치
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc('font', family=font_name)

plt.style.use('seaborn-poster')             # 스타일 서식 지정
plt.rcParams['axes.unicode_minus']=False    # 마이너스 부호 출력 설정

# 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']

# 그래프 객체 생성(figure에 2개의 서브 플롯 생성)
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

# axe 객체에 boxplot 메소드로 그래프 출력
ax1.boxplot(x=[df[df['origin']==1]['mpg'],
               df[df['origin']==2]['mpg'],
               df[df['origin']==3]['mpg']],
            labels=['USA', 'EU', 'JAPAN'])

ax2.boxplot(x=[df[df['origin']==1]['mpg'],
               df[df['origin']==2]['mpg'],
               df[df['origin']==3]['mpg']],
            labels=['USA', 'EU', 'JAPAN'],
            vert=False)

ax1.set_title('제조국가별 연비 분포(수직 박스 플롯)')
ax2.set_title('제조국가별 연비 분포(수평 박스 플롯)')

plt.show()
```

〈실행 결과〉 코드 전부 실행



2. Seaborn 라이브러리 - 고급 그래프 도구

Seaborn은 Matplotlib의 기능과 스타일을 확장한 파이썬 시각화 도구의 고급 버전이다. 비교적 단순한 인터페이스를 제공하기 때문에 초보자도 어렵지 않게 배울 수 있다.
먼저 Seaborn 라이브러리를 설치해야 한다. 아나콘다 배포판을 사용하는 경우 기본으로 설치되기 때문에 추가 설치를 하지 않아도 된다. 한편 파이썬 실행 파일에 Seaborn 라이브러리를 임포트할 때는 'sns'라는 약칭을 주로 사용한다.

• 데이터셋 가져오기

Seaborn 라이브러리에서 제공하는 'titanic' 데이터셋을 사용한다. Seaborn의 load_dataset() 함수를 사용하여 데이터프레임으로 가져온다. 모두 891명의 탑승객 정보가 담겨 있다.

〈예제 4-25〉 titanic 데이터셋 (File: example/part4/4.25_seaborn_dataset.py)

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

# 라이브러리 불러오기
import seaborn as sns

# titanic 데이터셋 가져오기
titanic = sns.load_dataset('titanic')

# titanic 데이터셋 살펴보기
print(titanic.head())
print('\n')
print(titanic.info())
```

〈실행 결과〉 코드 전부 실행

```
   survived  pclass     sex   age  ...  deck  embark_town  alive  alone
0         0       3    male  22.0  ...   NaN  Southampton     no  False
1         1       1  female  38.0  ...     C    Cherbourg    yes  False
2         1       3  female  26.0  ...   NaN  Southampton    yes   True
3         1       1  female  35.0  ...     C  Southampton    yes  False
4         0       3    male  35.0  ...   NaN  Southampton     no   True

[5 rows x 15 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
survived       891 non-null int64
pclass         891 non-null int64
sex            891 non-null object
age            714 non-null float64
sibsp          891 non-null int64
parch          891 non-null int64
fare           891 non-null float64
embarked       889 non-null object
class          891 non-null category
who            891 non-null object
adult_male     891 non-null bool
deck           203 non-null category
embark_town    889 non-null object
alive          891 non-null object
alone          891 non-null bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 63.0+ KB
None
```

• 회귀선이 있는 산점도

regplot() 함수는 서로 다른 2개의 연속 변수 사이의 산점도를 그리고 선형회귀분석에 의한 회귀선을 함께 나타낸다. fit_reg=False 옵션을 설정하면 회귀선을 안 보이게 할 수 있다. 다음의 예제에서 왼쪽 그래프는 선형회귀선을 표시하고 오른쪽 그래프에는 표시하지 않는다.

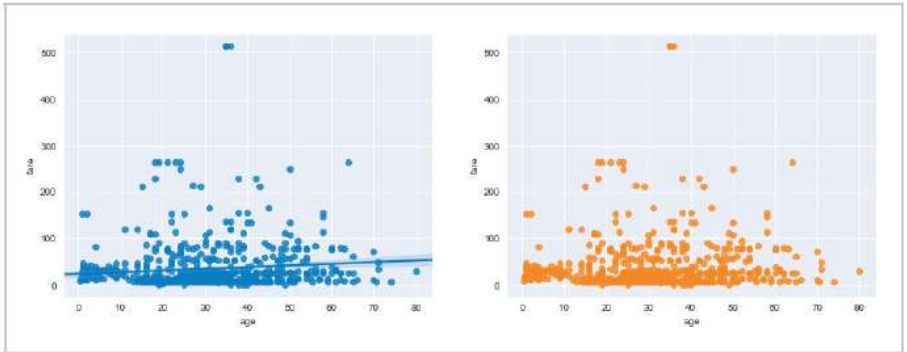
〈예제 4-26〉 회귀선이 있는 산점도 (File: example/part4/4.26_seaborn_regplot.py)

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

# 라이브러리 불러오기
import matplotlib.pyplot as plt
import seaborn as sns

# seaborn 제공 데이터셋 가져오기
titanic = sns.load_dataset('titanic')

# 스타일 테마 설정(5가지: darkgrid, whitegrid, dark, white, ticks)
sns.set_style('darkgrid')
```

2. Seaborn 라이브러리 - 고급 그래프 도구

```
13  # 그래프 객체 생성(figure에 2개의 서브 플롯 생성)
14  fig = plt.figure(figsize=(15, 5))
15  ax1 = fig.add_subplot(1, 2, 1)
16  ax2 = fig.add_subplot(1, 2, 2)
17
18  # 그래프 그리기 - 선형회귀선 표시(fit_reg=True)
19  sns.regplot(x='age',       # x축 변수
20              y='fare',      # y축 변수
21              data=titanic,  # 데이터
22              ax=ax1)        # axe 객체 - 1번째 그래프
23
24  # 그래프 그리기 - 선형회귀선 미표시(fit_reg=False)
25  sns.regplot(x='age',       # x축 변수
26              y='fare',      # y축 변수
27              data=titanic,  # 데이터
28              ax=ax2,        # axe 객체 - 2번째 그래프
29              fit_reg=False) # 회귀선 미표시
30
31  plt.show()
```

〈실행 결과〉 코드 전부 실행



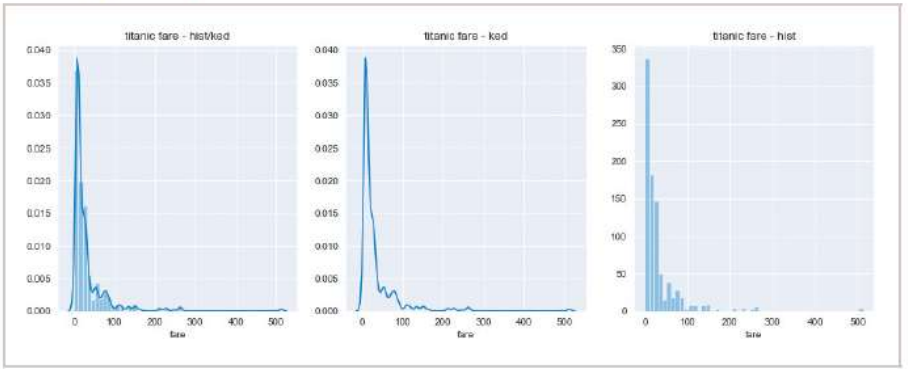
● **히스토그램/커널 밀도 그래프**

단변수(하나의 변수) 데이터의 분포를 확인할 때 `distplot()` 함수를 이용한다. 기본값으로 히스토그램과 커널 밀도 함수[8]를 그래프로 출력한다.

〈예제 4-27〉 히스토그램/커널밀도함수 (File: example/part4/4.27_seaborn_distplot.py)

```
~  ~~ 생략 (예제 4-26과 동일) ~~

13  # 그래프 객체 생성(figure에 3개의 서브 플롯 생성)
14  fig = plt.figure(figsize=(15, 5))
15  ax1 = fig.add_subplot(1, 3, 1)
16  ax2 = fig.add_subplot(1, 3, 2)
17  ax3 = fig.add_subplot(1, 3, 3)
18
19  # 기본값
20  sns.distplot(titanic['fare'], ax=ax1)
21
22  # hist=False
23  sns.distplot(titanic['fare'], hist=False, ax=ax2)
24
25  # kde=False
26  sns.distplot(titanic['fare'], kde=False, ax=ax3)
27
28  # 차트 제목 표시
29  ax1.set_title('titanic fare - hist/ked')
30  ax2.set_title('titanic fare - ked')
31  ax3.set_title('titanic fare - hist')
32
33  plt.show()
```

〈실행 결과〉 코드 전부 실행



2. Seaborn 라이브러리 - 고급 그래프 도구

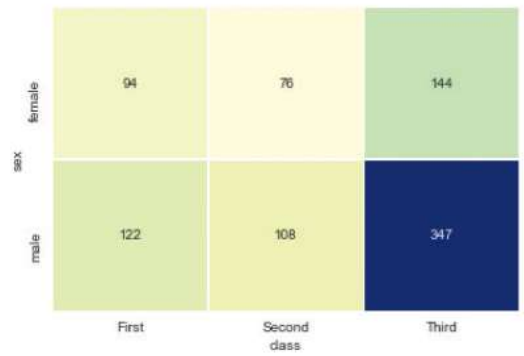
• 히트맵

Seaborn 라이브러리는 히트맵(heatmap)을 그리는 heatmap() 메소드를 제공한다. 2개의 범주형 변수를 각각 x, y축에 놓고 데이터를 매트릭스 형태로 분류한다. 데이터프레임을 피벗테이블로 정리할 때 한 변수('sex' 열)를 행 인덱스로 나머지 변수('class' 열)를 열 이름으로 설정한다. aggfunc='size' 옵션은 데이터 값의 크기를 기준으로 집계한다는 뜻이다. 피벗테이블에 대해서는 [Part 6]에서 자세히 알아본다.

〈예제 4-28〉 히트맵 (File: example/part4/4.28_seaborn_heatmap.py)

```
~  ~~ 생략 (예제 4-27과 동일) ~~

13  # 피벗테이블로 범주형 변수를 각각 행, 열로 재구분하여 정리
14  table = titanic.pivot_table(index=['sex'], columns=['class'], aggfunc='size')
15
16  # 히트맵 그리기
17  sns.heatmap(table,                  # 데이터프레임
18              annot=True, fmt='d',    # 데이터 값 표시 여부, 정수형 포맷
19              cmap='YlGnBu',          # 컬러 맵
20              linewidth=.5,           # 구분 선
21              cbar=False)             # 컬러 바 표시 여부
22
23  plt.show()
```

〈실행 결과〉 코드 전부 실행



• 범주형 데이터의 산점도

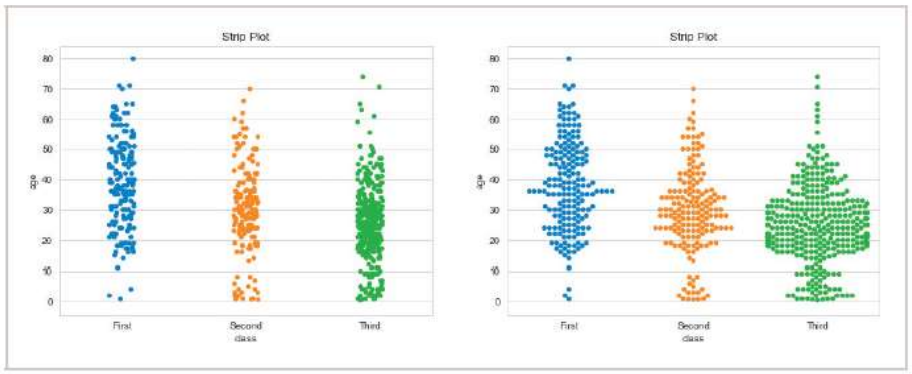
범주형 변수에 들어 있는 각 범주별 데이터의 분포를 확인하는 방법이다. stripplot() 함수와 swarmplot() 함수를 사용할 수 있다. swarmplot() 함수는 데이터의 분산까지 고려하여, 데이터 포인트가 서로 중복되지 않도록 그린다.

〈예제 4-29〉 범주형 데이터의 산점도 (File: example/part4/4.29_seaborn_scatter.py)

```
1   # -*- coding: utf-8 -*-
2
3   # 라이브러리 불러오기
4   import matplotlib.pyplot as plt
5   import seaborn as sns
6
7   # Seaborn 제공 데이터셋 가져오기
8   titanic = sns.load_dataset('titanic')
9
10  # 스타일 테마 설정(5가지: darkgrid, whitegrid, dark, white, ticks)
11  sns.set_style('whitegrid')
12
13  # 그래프 객체 생성(figure에 2개의 서브 플롯 생성)
14  fig = plt.figure(figsize=(15, 5))
15  ax1 = fig.add_subplot(1, 2, 1)
16  ax2 = fig.add_subplot(1, 2, 2)
17
18  # 이산형 변수의 분포 - 데이터 분산 미고려
19  sns.stripplot(x="class",        # x축 변수
20                y="age",          # y축 변수
21                data=titanic,     # 데이터셋 - 데이터프레임
22                ax=ax1)           # axe 객체 - 1번째 그래프
23
24  # 이산형 변수의 분포 - 데이터 분산 고려(중복 X)
25  sns.swarmplot(x="class",        # x축 변수
26                y="age",          # y축 변수
27                data=titanic,     # 데이터셋 - 데이터프레임
28                ax=ax2)           # axe 객체 - 2번째 그래프
29
30  # 차트 제목 표시
31  ax1.set_title('Strip Plot')
32  ax2.set_title('Strip Plot')
33
34  plt.show()
```

2. Seaborn 라이브러리 - 고급 그래프 도구

〈실행 결과〉 코드 전부 실행



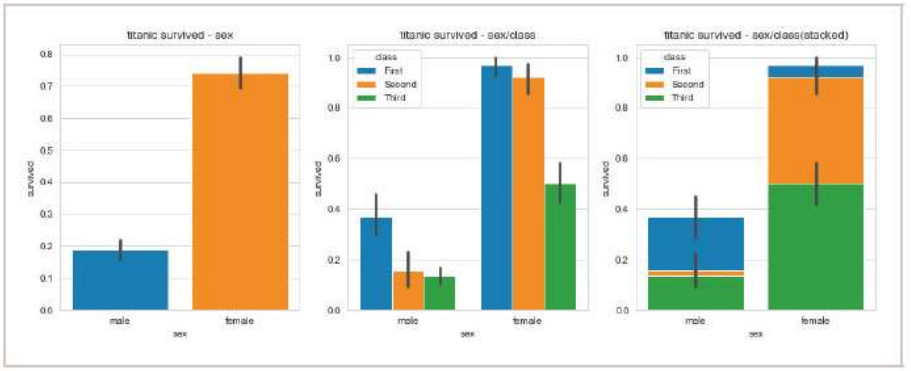
• 막대 그래프

막대 그래프를 그리는 barplot() 함수를 소개한다. 3개의 axe 객체(서브 플롯)을 만들고, 옵션에 변화를 주면서 차이를 살펴보자. x축, y축에 변수 할당, x축, y축에 변수 할당하고 hue 옵션 추가, x축, y축에 변수 할당하고 hue 옵션을 추가하여 누적 출력 순으로 실행한다.

〈예제 4-30〉 막대 그래프 (File: example/part4/4.30_seaborn_bar.py)

```
~  ~~ 생략 (예제 4-29와 동일) ~~

13  # 그래프 객체 생성(figure에 3개의 서브 플롯 생성)
14  fig = plt.figure(figsize=(15, 5))
15  ax1 = fig.add_subplot(1, 3, 1)
16  ax2 = fig.add_subplot(1, 3, 2)
17  ax3 = fig.add_subplot(1, 3, 3)
18
19  # x축, y축에 변수 할당
20  sns.barplot(x='sex', y='survived', data=titanic, ax=ax1)
21
22  # x축, y축에 변수 할당하고 hue 옵션 추가
23  sns.barplot(x='sex', y='survived', hue='class', data=titanic, ax=ax2)
24
25  # x축, y축에 변수 할당하고 hue 옵션을 추가하여 누적 출력
26  sns.barplot(x='sex', y='survived', hue='class', dodge=False, data=titanic, ax=ax3)
27
28  # 차트 제목 표시
29  ax1.set_title('titanic survived - sex')
30  ax2.set_title('titanic survived - sex/class')
31  ax3.set_title('titanic survived - sex/class(stacked)')
32
33  plt.show()
```

〈실행 결과〉 코드 전부 실행



• 빈도 그래프

각 범주에 속하는 데이터의 개수를 막대 그래프로 나타내는 countplot() 함수를 소개한다. 예제를 통해 3개의 서브 플롯을 비교한다. "기본 설정, hue 옵션 추가, 축 방향으로 hue 변수를 분리하지 않고 위로 쌓아 올리는 누적 그래프로 출력" 순으로 실행한다. 그래프 색 구성을 다르게 하려면 palette 옵션을 변경하여 적용한다.

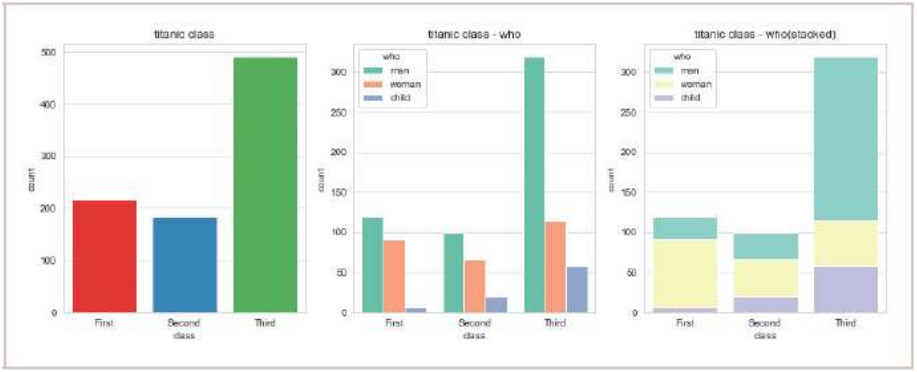
〈예제 4-31〉 빈도 그래프 (File: example/part4/4.31_seaborn_count.py)

```
~  ~~ 생략 (예제 4-30과 동일) ~~

19  # 기본값
20  sns.countplot(x='class', palette='Set1', data=titanic, ax=ax1)
21
22  # hue 옵션에 'who' 추가
23  sns.countplot(x='class', hue='who', palette='Set2', data=titanic, ax=ax2)
```

2. Seaborn 라이브러리 - 고급 그래프 도구

```
# dodge=False 옵션 추가(축 방향으로 분리하지 않고 누적 그래프 출력)
sns.countplot(x='class', hue='who', palette='Set3', dodge=False, data=titanic, ax=ax3)

# 차트 제목 표시
ax1.set_title('titanic class')
ax2.set_title('titanic class - who')
ax3.set_title('titanic class - who(stacked)')

plt.show()
```

〈실행 결과〉 코드 전부 실행



• 박스 플롯/바이올린 그래프

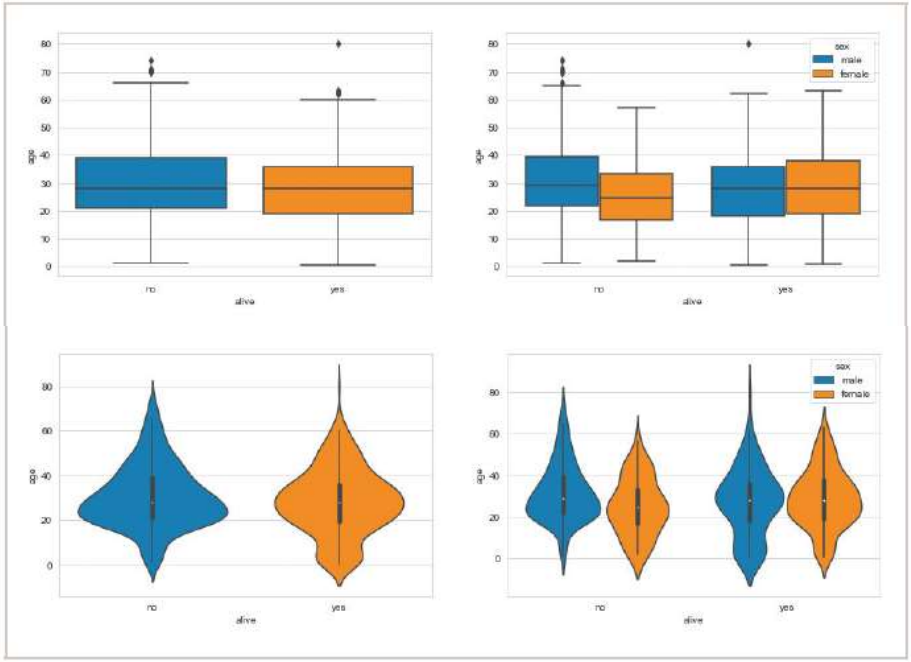
박스 플롯은 범주형 데이터 분포와 주요 통계 지표를 함께 제공한다. 다만, 박스 플롯만으로는 데이터가 퍼져 있는 분산의 정도를 정확하게 알기는 어렵기 때문에 커널 밀도 함수 그래프를 y 축 방향에 추가하여 바이올린 그래프를 그리는 경우도 있다. 박스 플롯은 `boxplot()` 함수로 그리고 바이올린 그래프는 `violinplot()` 함수로 그린다. 예제에서는 타이타닉 생존자의 분포를 파악한다. hue 변수에 `'sex'` 열을 추가하면 남녀 데이터를 구분하여 표시할 수 있다.

〈예제 4-32〉 박스 플롯/바이올린 그래프 (File: example/part4/4.32_seaborn_box_violin.py)

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

# 라이브러리 불러오기
import matplotlib.pyplot as plt
import seaborn as sns

# Seaborn 제공 데이터셋 가져오기
titanic = sns.load_dataset('titanic')

# 스타일 테마 설정(5가지: darkgrid, whitegrid, dark, white, ticks)
sns.set_style('whitegrid')

# 그래프 객체 생성(figure에 4개의 서브 플롯 생성)
fig = plt.figure(figsize=(15, 10))
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)

# 박스 플롯 - 기본값
sns.boxplot(x='alive', y='age', data=titanic, ax=ax1)

# 바이올린 그래프 - hue 변수 추가
sns.boxplot(x='alive', y='age', hue='sex', data=titanic, ax=ax2)

# 박스 플롯 - 기본값
sns.violinplot(x='alive', y='age', data=titanic, ax=ax3)

# 바이올린 그래프 - hue 변수 추가
sns.violinplot(x='alive', y='age', hue='sex', data=titanic, ax=ax4)

plt.show()
```

2. Seaborn 라이브러리 - 고급 그래프 도구

〈실행 결과〉 코드 전부 실행



• 조인트 그래프

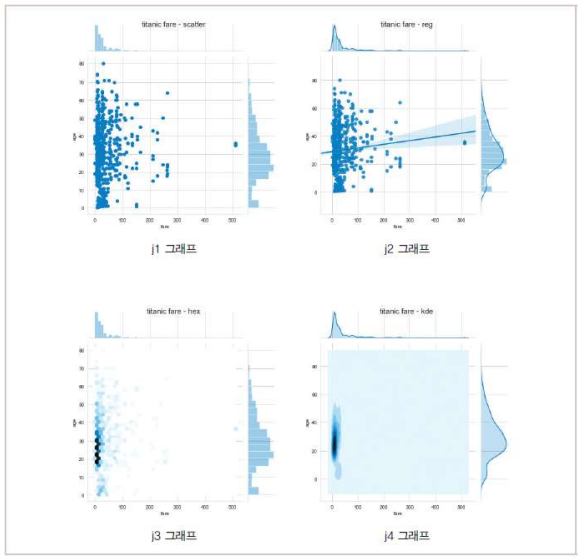
jointplot() 함수는 산점도를 기본으로 표시하고, x-y축에 각 변수에 대한 히스토그램을 동시에 보여준다. 따라서 두 변수의 관계와 데이터가 분산되어 있는 정도를 한눈에 파악하기 좋다. 예제에서는 산점도(기본값), 회귀선 추가(kind='reg'), 육각 산점도(kind='hex'), 커널 밀집 그래프(kind='kde') 순으로 조인트 그래프를 그리고 차이를 비교한다.

〈예제 4-33〉 조인트 그래프 (File: example/part4/4.33_seaborn_joint.py)

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

# 라이브러리 불러오기
import matplotlib.pyplot as plt
import seaborn as sns

# seaborn 제공 데이터셋 가져오기
titanic = sns.load_dataset('titanic')
```

```
# 스타일 테마 설정(5가지: darkgrid, whitegrid, dark, white, ticks)
sns.set_style('whitegrid')

# 조인트 그래프 - 산점도(기본값)
j1 = sns.jointplot(x='fare', y='age', data=titanic)

# 조인트 그래프 - 회귀선
j2 = sns.jointplot(x='fare', y='age', kind='reg', data=titanic)

# 조인트 그래프 - 육각 그래프
j3 = sns.jointplot(x='fare', y='age', kind='hex', data=titanic)

# 조인트 그래프 - 커널 밀집 그래프
j4 = sns.jointplot(x='fare', y='age', kind='kde', data=titanic)

# 차트 제목 표시
j1.fig.suptitle('titanic fare - scatter', size=15)
j2.fig.suptitle('titanic fare - reg', size=15)
j3.fig.suptitle('titanic fare - hex', size=15)
j4.fig.suptitle('titanic fare - kde', size=15)

plt.show()
```

〈실행 결과〉 코드 전부 실행



2. Seaborn 라이브러리 - 고급 그래프 도구

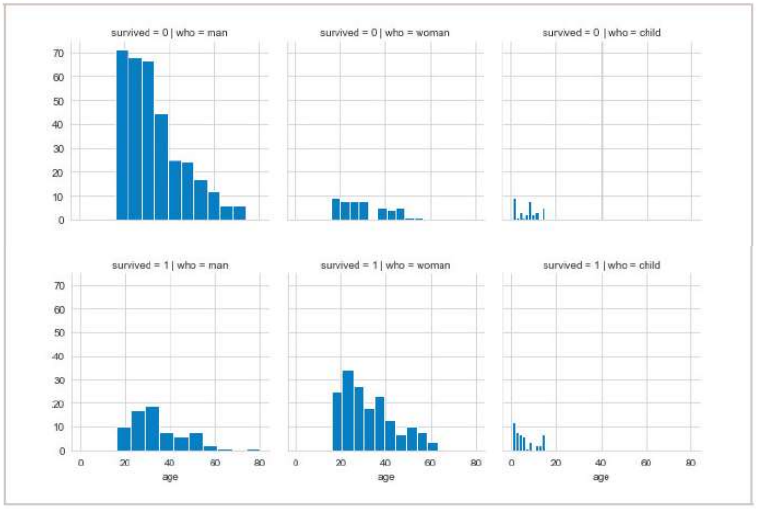
• 조건을 적용하여 화면을 그리드로 분할하기

FacetGrid() 함수는 행, 열 방향으로 서로 다른 조건을 적용하여 여러 개의 서브 플롯을 만든다. 그리고 각 서브 플롯에 적용할 그래프 종류를 map() 메소드를 이용하여 그리드 객체에 전달한다.

〈예제 4-34〉 조건에 맞게 화면 분할 (File: example/part4/4.34_seaborn_facetgrid.py)

```
1  # -*- coding: utf-8 -*-
2
3  # 라이브러리 불러오기
4  import matplotlib.pyplot as plt
5  import seaborn as sns
6
7  # Seaborn 제공 데이터셋 가져오기
8  titanic = sns.load_dataset('titanic')
9
10 # 스타일 테마 설정(5가지: darkgrid, whitegrid, dark, white, ticks)
11 sns.set_style('whitegrid')
12
13 # 조건에 따라 그리드 나누기
14 g = sns.FacetGrid(data=titanic, col='who', row='survived')
15
16 # 그래프 적용하기
17 g = g.map(plt.hist, 'age')
```

〈실행 결과〉 코드 전부 실행

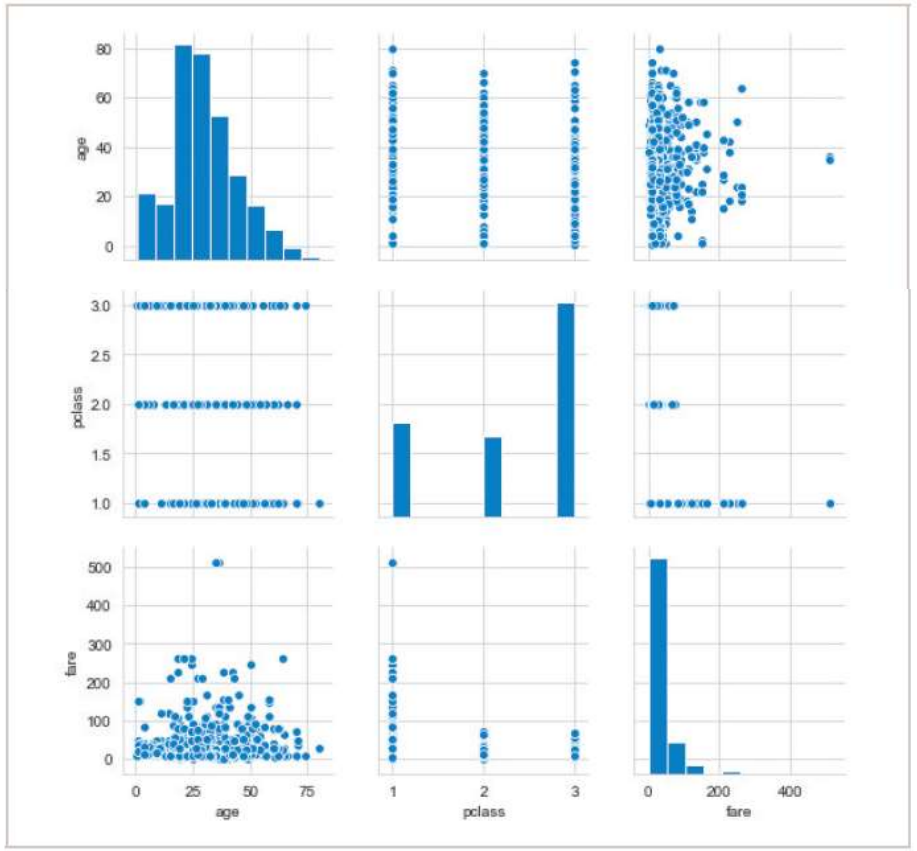


• 이변수 데이터의 분포

〈예제 4-35〉 이변수 데이터 분포 (File: example/part4/4.35_seaborn_pairplot.py)

```
~  ~~ 생략 (예제 4-34와 동일) ~~

13 # titanic 데이터셋 중에서 분석 데이터 선택하기
14 titanic_pair = titanic[['age','pclass', 'fare']]
15
16 # 조건에 따라 그리드 나누기
17 g = sns.pairplot(titanic_pair)
```

〈실행 결과〉 코드 전부 실행



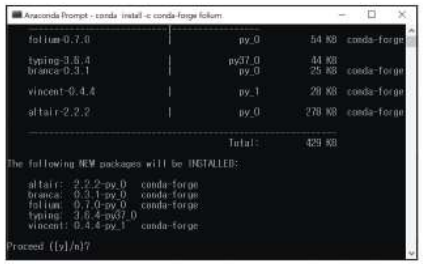
3. Folium 라이브러리 - 지도 활용

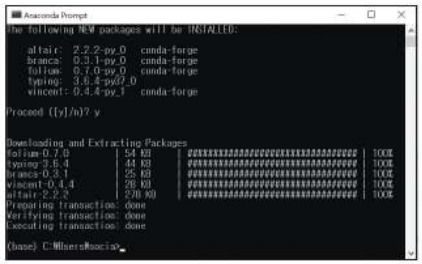
• Folium 설치하기

Folium을 사용하기 위해서는 먼저 라이브러리를 설치해야 한다. 아나콘다 배포판을 사용하는 경우에도 설치가 필요하다. 설치 방법은 비교적 간단하다.

아나콘다 배포판에서 Folium 설치 방법

아나콘다 프롬프트를 실행하고, conda install –c conda –forge folium을 입력하고 Enter를 치면 설치된다. 설치가 완료되면 아나콘다 내비게이터를 재실행시킨다.





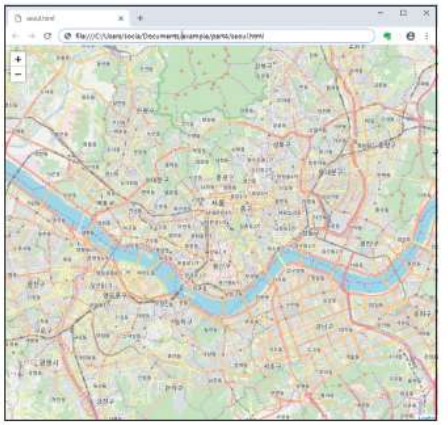
• 지도 만들기

Folium 라이브러리의 Map() 함수를 이용하면 간단하게 지도 객체를 만들 수 있다. 지도 화면은 고정된 것이 아니고 줌(zoom) 기능과 화면 이동(scroll)이 모두 가능하다.

〈예제 4-36〉 지도 만들기 (File: example/part4/4.36_folium_map.py)

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

# 라이브러리 불러오기
import folium

# 서울 지도 만들기
seoul_map = folium.Map(location=[37.55,126.98], zoom_start=12)

# 지도를 HTML 파일로 저장하기
seoul_map.save('./seoul.html')
```

〈실행 결과〉 저장된 HTML 파일 실행 (File: example/part4/seoul.html)



• 지도 스타일 적용하기

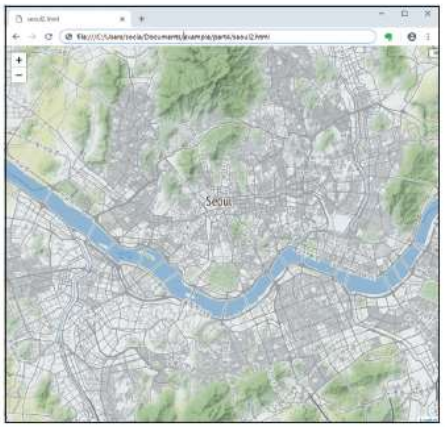
Map() 함수에 tiles 옵션을 적용하면 지도에 적용하는 스타일을 변경하여 지정할 수 있다. 다음의 예제에서는 'Stamen Terrain' 맵과 'Stamen Toner' 맵의 스타일을 비교한다.

〈예제 4-37〉 지도 스타일 적용 (File: example/part4/4.37_folium_map_tiles.py)

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

# 라이브러리 불러오기
import folium

# 서울 지도 만들기
seoul_map2 = folium.Map(location=[37.55,126.98], tiles='Stamen Terrain',
                        zoom_start=12)
seoul_map3 = folium.Map(location=[37.55,126.98], tiles='Stamen Toner',
                        zoom_start=15)

# 지도를 HTML 파일로 저장하기
seoul_map2.save('./seoul2.html')
seoul_map3.save('./seoul3.html')
```

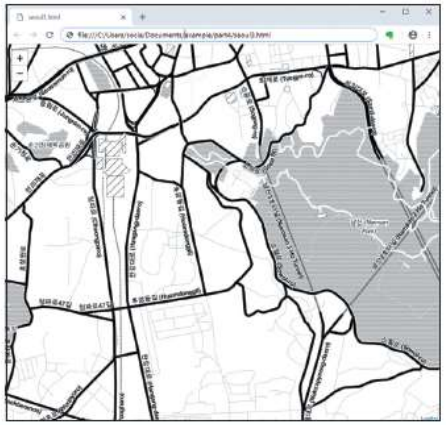
3. Folium 라이브러리 - 지도 활용

〈실행 결과〉 Stamen Terrain 적용(zoom_start=12) (File: example/part4/seoul2.html)



저장된 HTML 파일을 웹 브라우저에서 실행한다. 앞의 예제와 비교해 보면 'Stamen Terrain' 맵은 산악 지형 등의 지형이 보다 선명하게 드러난다.

〈실행 결과〉 Stamen Toner 적용(zoom_start=15) (File: example/part4/seoul3.html)



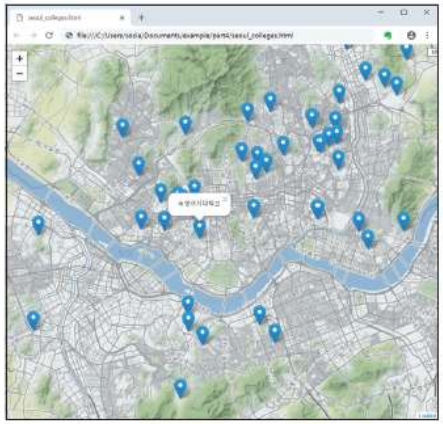
저장된 HTML 파일을 웹 브라우저에서 실행한다. 'Stamen Toner' 스타일을 적용한 맵은 흑백 스타일로 도로망을 강조해서 보여준다.

• 지도에 마커 표시하기

〈예제 4-38〉 지도에 마커 표시하기 (File: example/part4/4.38_folium_map_marker.py)

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

# 라이브러리 불러오기
import pandas as pd
import folium

# 대학교 리스트를 데이터프레임으로 변환
df = pd.read_excel('./서울지역 대학교 위치.xlsx')

# 서울 지도 만들기
seoul_map = folium.Map(location=[37.55,126.98], tiles='Stamen Terrain',
                       zoom_start=12)

# 대학교 위치 정보를 Marker로 표시
for name, lat, lng in zip(df.index, df.위도, df.경도):
    folium.Marker([lat, lng], popup=name).add_to(seoul_map)

# 지도를 HTML 파일로 저장하기
seoul_map.save('./seoul_colleges.html')
```

〈실행 결과〉 지도에 마커 표시하기 (File: example/part4/seoul_colleges.html)



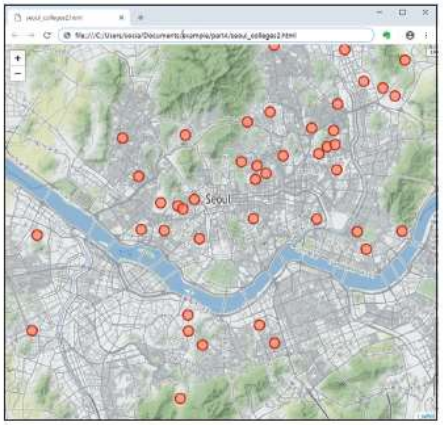
3. Folium 라이브러리 - 지도 활용

이번에는 원형 마커를 표시해 보자. 앞의 예제에서 Marker() 함수 대신에 CircleMarker() 함수를 사용한다. 원형 마커의 크기, 색상, 투명도 등을 설정할 수 있다.

〈예제 4-39〉 지도에 원형 마커 표시 (File: example/part4/4.39_folium_map_circlemarker.py)

```
~  ~~ 생략 (예제 4-38과 동일) ~~

14  # 대학교 위치 정보를 CircleMarker로 표시
15  for name, lat, lng in zip(df.index, df.위도, df.경도):
16      folium.CircleMarker([lat, lng],
17                          radius=10,          # 원의 반지름
18                          color='brown',      # 원의 둘레 색상
19                          fill=True,
20                          fill_color='coral',  # 원을 채우는 색
21                          fill_opacity=0.7,    # 투명도
22                          popup=name
23      ).add_to(seoul_map)
24
25  # 지도를 HTML 파일로 저장하기
26  seoul_map.save('./seoul_colleges2.html')
```

〈실행 결과〉 지도에 마커 표시하기 (File: example/part4/seoul_colleges.html)



● 지도 영역에 단계구분도(Choropleth Map) 표시하기

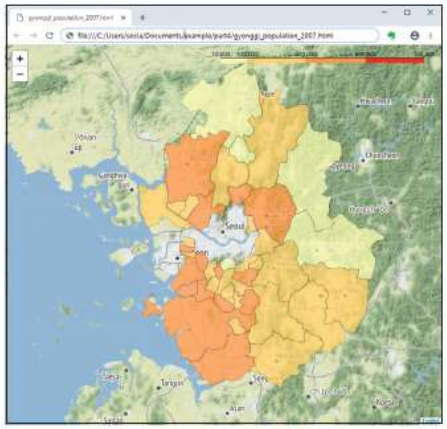
행정구역과 같이 지도 상의 어떤 경계에 둘러싸인 영역에 색을 칠하거나 음영 등으로 정보를 나타내는 시각화 방법이다.

〈예제 4-40〉 지도 영역에 단계구분도 표시하기 (File: example/part4/4.40_folium_choropleth.py)

```
1   # -*- coding: utf-8 -*-
2
3   # 라이브러리 불러오기
4   import pandas as pd
5   import folium
6   import json
7
8   # 경기도 인구변화 데이터를 불러와서 데이터프레임으로 변환
9   file_path = './경기도인구데이터.xlsx'
10  df = pd.read_excel(file_path, index_col='구분')
11  df.columns = df.columns.map(str)
12
13  # 경기도 시군구 경계 정보를 가진 geo-json 파일 불러오기
14  geo_path = './경기도행정구역경계.json'
15  try:
16      geo_data = json.load(open(geo_path, encoding='utf-8'))
17  except:
18      geo_data = json.load(open(geo_path, encoding='utf-8-sig'))
19
20  # 경기도 지도 만들기
21  g_map = folium.Map(location=[37.5502,126.982],
22                     tiles='Stamen Terrain', zoom_start=9)
23
24  # 출력할 연도 선택(2007~2017년 중에서 선택)
25  year = '2007'
26
27  # Choropleth 클래스로 단계구분도 표시하기
28  folium.Choropleth(geo_data=geo_data,              # 지도 경계
29                    data = df[year],                # 표시하려는 데이터
30                    columns = [df.index, df[year]],  # 열 지정
31                    fill_color='YlOrRd', fill_opacity=0.7, line_opacity=0.3,
32                    threshold_scale=[10000, 100000, 300000, 500000, 700000],
33                    key_on='feature.properties.name',
34                    ).add_to(g_map)
35
36  # 지도를 HTML 파일로 저장하기
37  g_map.save('./gyonggi_population_' + year + '.html')
```

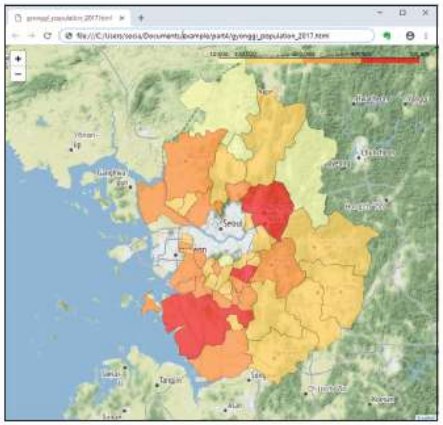
3. Folium 라이브러리 - 지도 활용

〈실행 결과〉 2007년도 경기도 인구 분포 (File: example/part4/gyonggi_population_2007.html)



25라인에 `year = '2007'`라고 입력하여, 2007년도 경기도 지역의 인구 수를 지도에 표시하였다. 웹 브라우저에서 HTML 파일을 열어 보면 동북부 지역을 제외하고 비교적 균일한 분포를 나타낸다.

〈실행 결과〉 2017년도 경기도 인구 분포 (File: example/part4/gyonggi_population_2017.html)



25라인에 `year = '2017'`라고 입력하여, 2017년도 경기도 지역의 인구 수를 지도에 표시하였다. 2007년과 비교하면 남양주, 분당, 화성(동탄) 지역의 신도시 개발과 인구 유입으로 인구가 집중되는 현상이 심화된 것을 볼 수 있다.