BITAmin

5주차 정규 Session

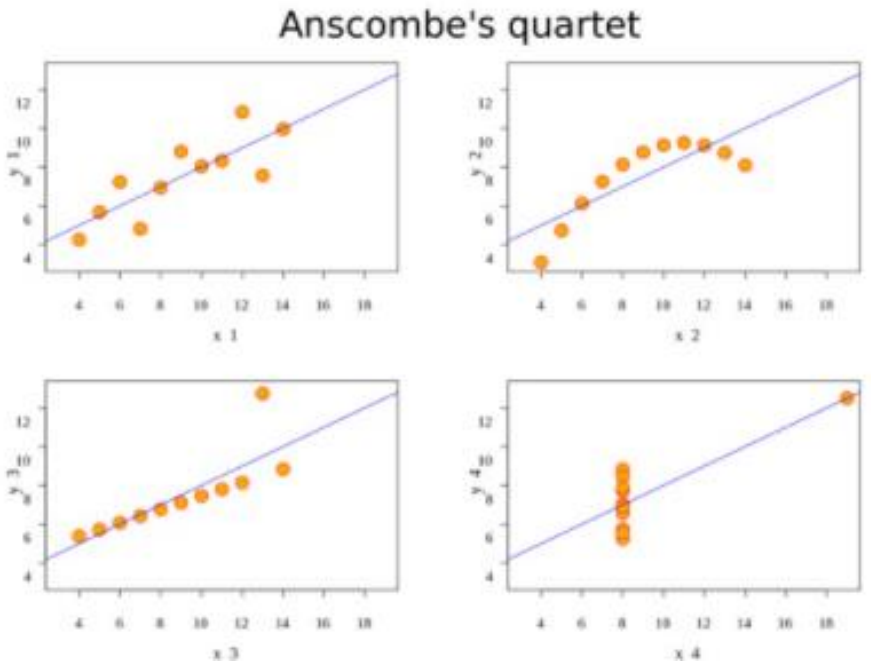
비타민 6기 3조

강혜연 이민준 정세영 정인영

CONTENTS

01. 데이터 시각화 중요성

향상된 통찰력

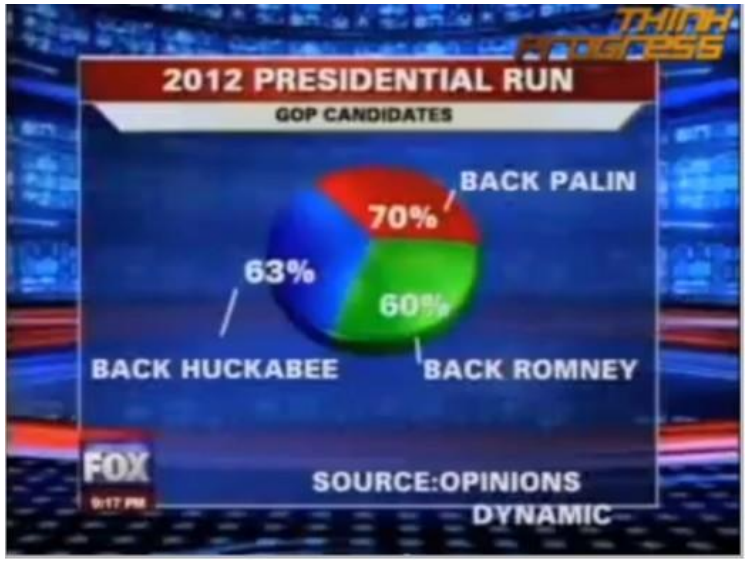


신속한 의사결정

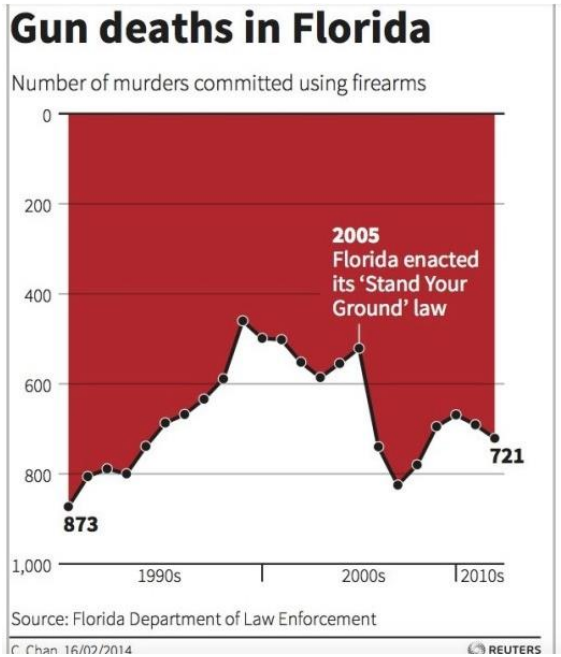
인사이트 도출 ➡ 의사결정

01. 일반적인 데이터 시각화 실수

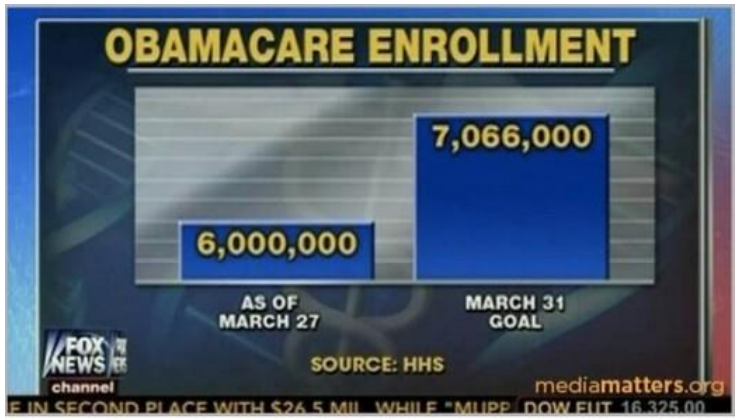
전체 합



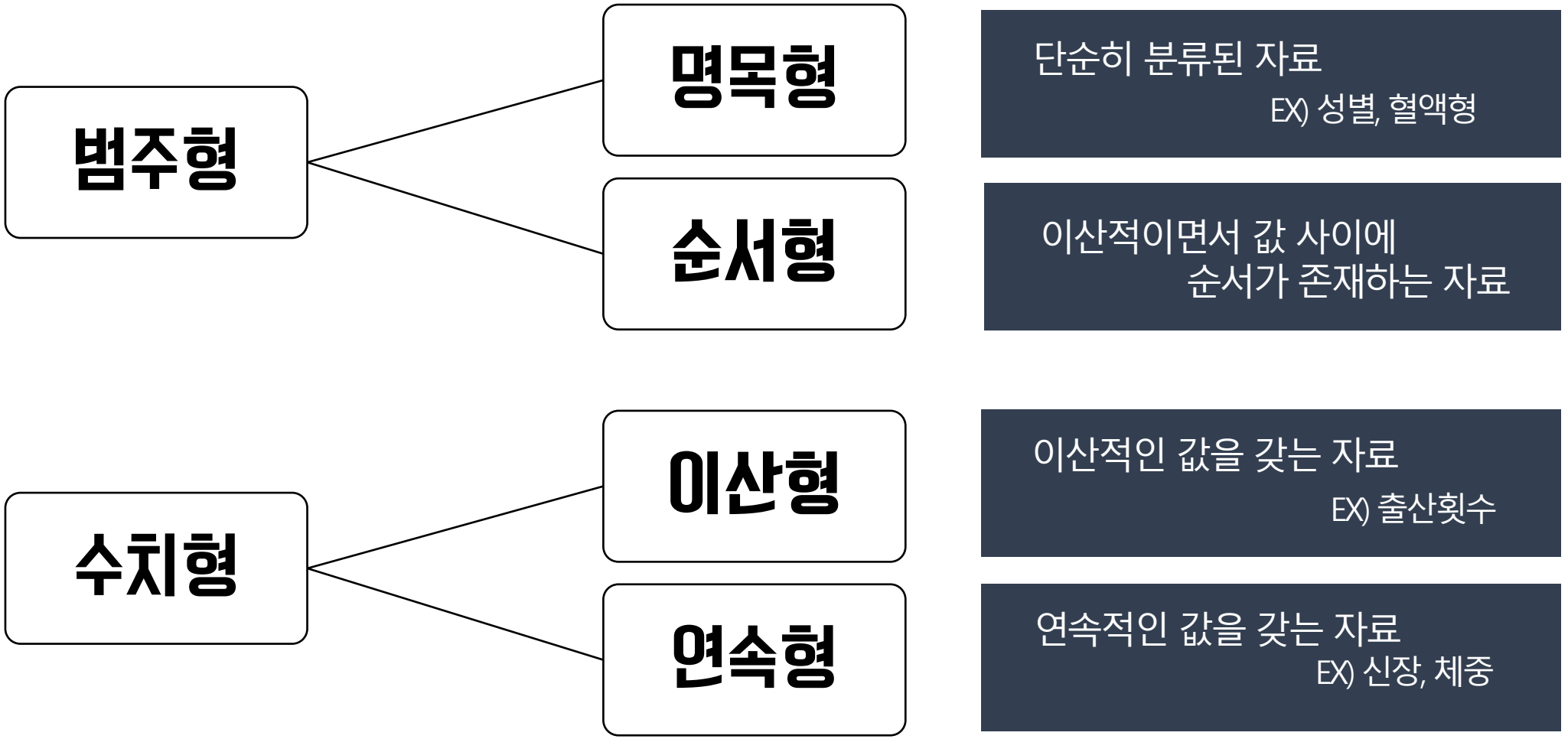
Y축의 위치



Y축의 유무



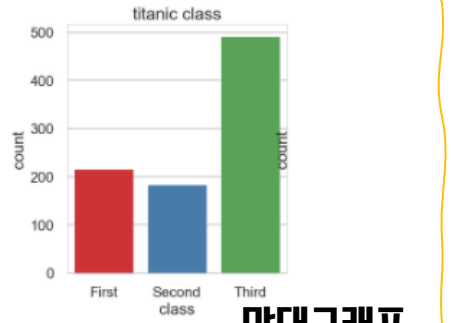
02. 데이터 타입


범주형
명목형
순서형
수치형
이산형
연속형
단순히 분류된 자료
EX) 성별, 혈액형
이산적이면서 값 사이에
순서가 존재하는 자료
이산적인 값을 갖는 자료
EX) 출산횟수
연속적인 값을 갖는 자료
EX) 신장, 체중

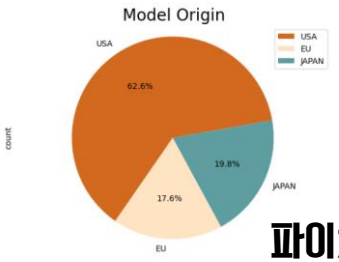
02. 그래프 타입

1차원

범주형

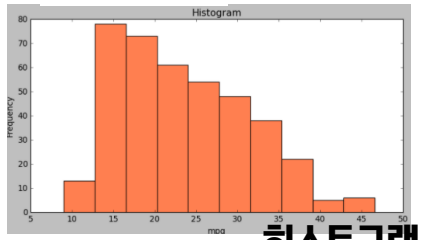


막대그래프

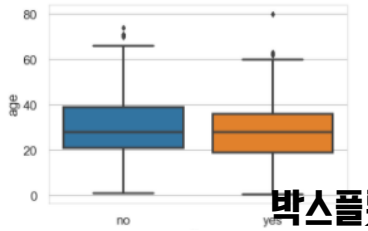


파이차트

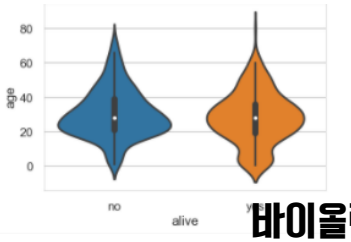
연속형



히스토그램



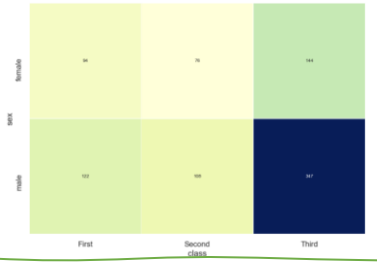
박스플롯



바이올린

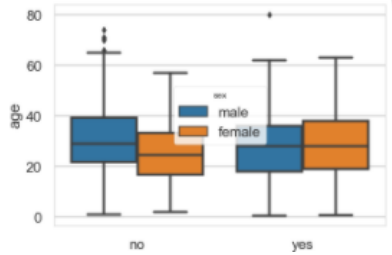
2차원

범주&범주



히트맵

범주&연속

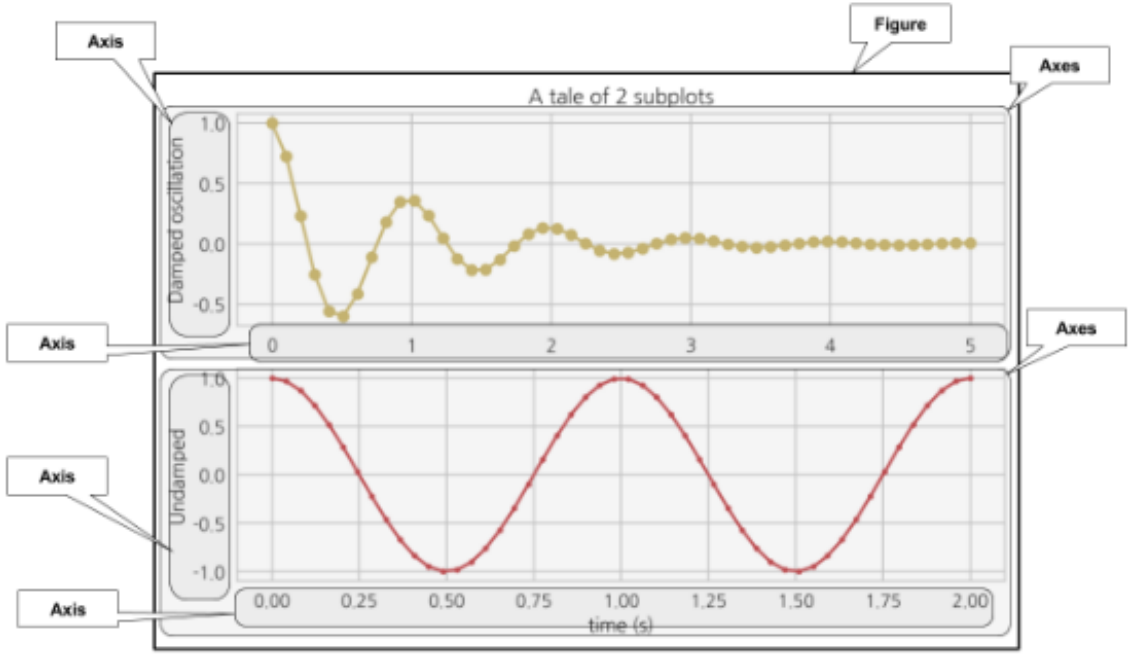


박스플롯

연속&연속

산점도

03. Matplotlib 기본 그래프 구조



Figure

위 전체 그림이 Figure
Figure은 파생축과 title, figure, lengend를 보여줌

Axes

플롯으로, 데이터 공간이 있는 이미지의 영역
축의 데이터의 범위는 set_xlim(), set_ylim()와 같은 메소드로 제어

Axis

축은 숫자라인과 유사한 오브젝트
그래프의 한계 세팅 / tick과 tick label을 만드는 것 처리

03. Matplotlib 기본 그래프 구조 – 선그래프

기본 사용법

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

#Excel 데이터를 데이터프레임으로 변환
df = pd.read_excel("시도별 전출입 인구수.xlsx", header=0)
df.head(5)
```

```
#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)
df_seoul.head(8)
```

```
#서울에서 경기도로 이동한 인구 데이터 값만 선택
sr_one = df_seoul.loc["경기도"]
sr_one.head()
```

	전출지별	전입지별	1970	1971	1972	1973	1974	1975	1976	1977	...	2008	2009
0	전출지별	전입지별	이동자수(명)	이동자수(명)	이동자수(명)	이동자수(명)	이동자수(명)	이동자수(명)	이동자수(명)	이동자수(명)	...	이동자수(명)	이동자수(명)
1	전국	전국	4046536	4210164	3687938	4860418	5297969	9011440	6773250	7397623	...	8808256	8487275
2	NaN	서울특별시	1742813	1671705	1349333	1831858	2050392	3396662	2756510	2893403	...	2025358	1873188
3	NaN	부산광역시	448577	389797	362202	482061	680984	805979	724664	785117	...	514502	519310
4	NaN	대구광역시	-	-	-	-	-	-	-	-	...	409938	398626

5 rows × 50 columns

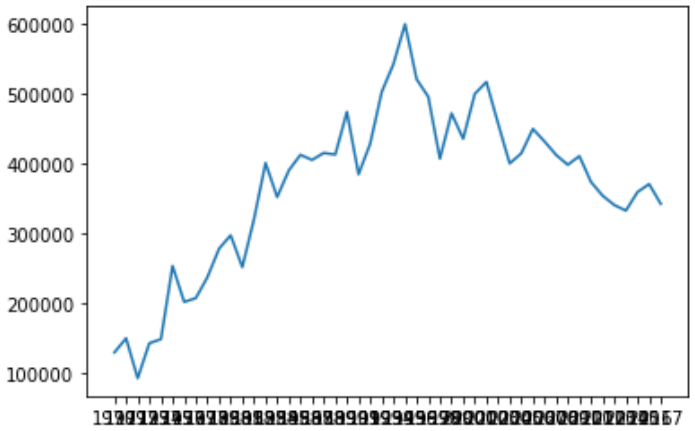
```
1970    130149
1971    150313
1972     93333
1973    143234
1974    149045
Name: 경기도, dtype: object
```

03. Matplotlib 기본 그래프 구조 – 선그래프

기본 사용법

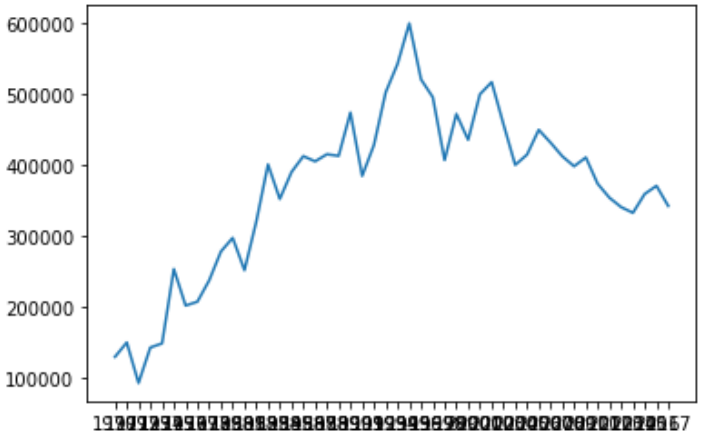
①

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



②

```
#판다스 객체를 plot함수에 입력
plt.plot(sr_one)
```



03. Matplotlib 기본 그래프 구조 – 선그래프

차트 제목, 축 이름 추가

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

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

#차트 제목 추가
① plt.title("서울 -> 경기 인구 이동")

#축 이름 추가
② plt.xlabel("기간")
③ plt.ylabel("이동 인구수")

plt.show() #변경사항 저장하고 그래프 출력
```

한글깨짐 오류 발생!

```
#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)
```

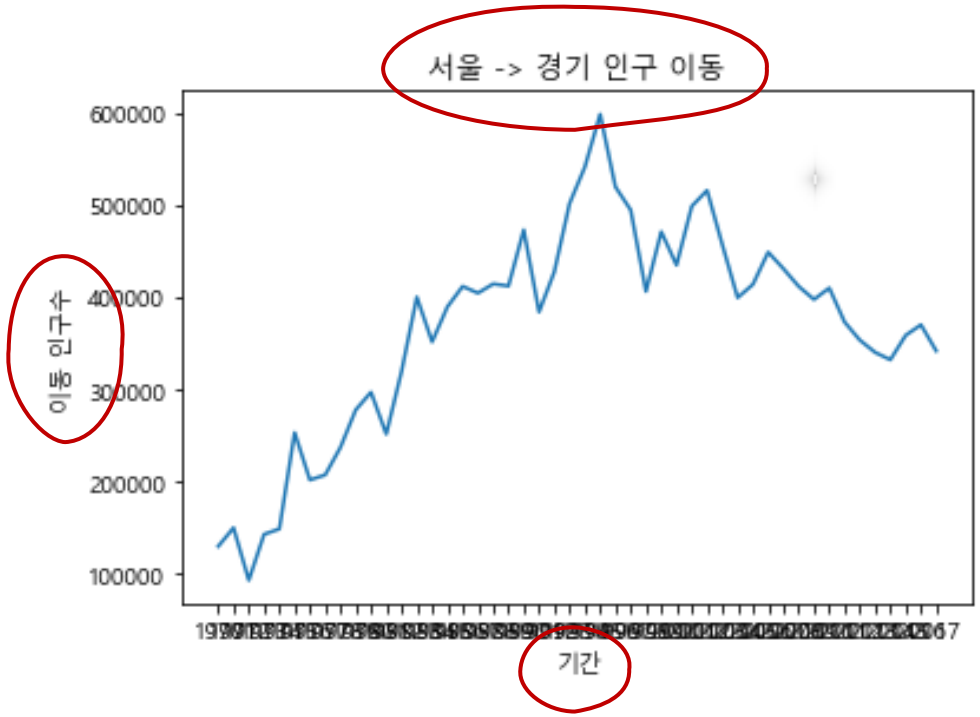
#맥의 경우 matplotlib 한글 폰트 오류 해결
from matplotlib import rc
rc("font", family="AppleGothic")

03. Matplotlib 기본 그래프 구조 - 선그래프

Matplotlib 한글 폰트 오류 해결

```
#위에 코드 그대로 가져오기
#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", header=0)

#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 plot함수에 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
```

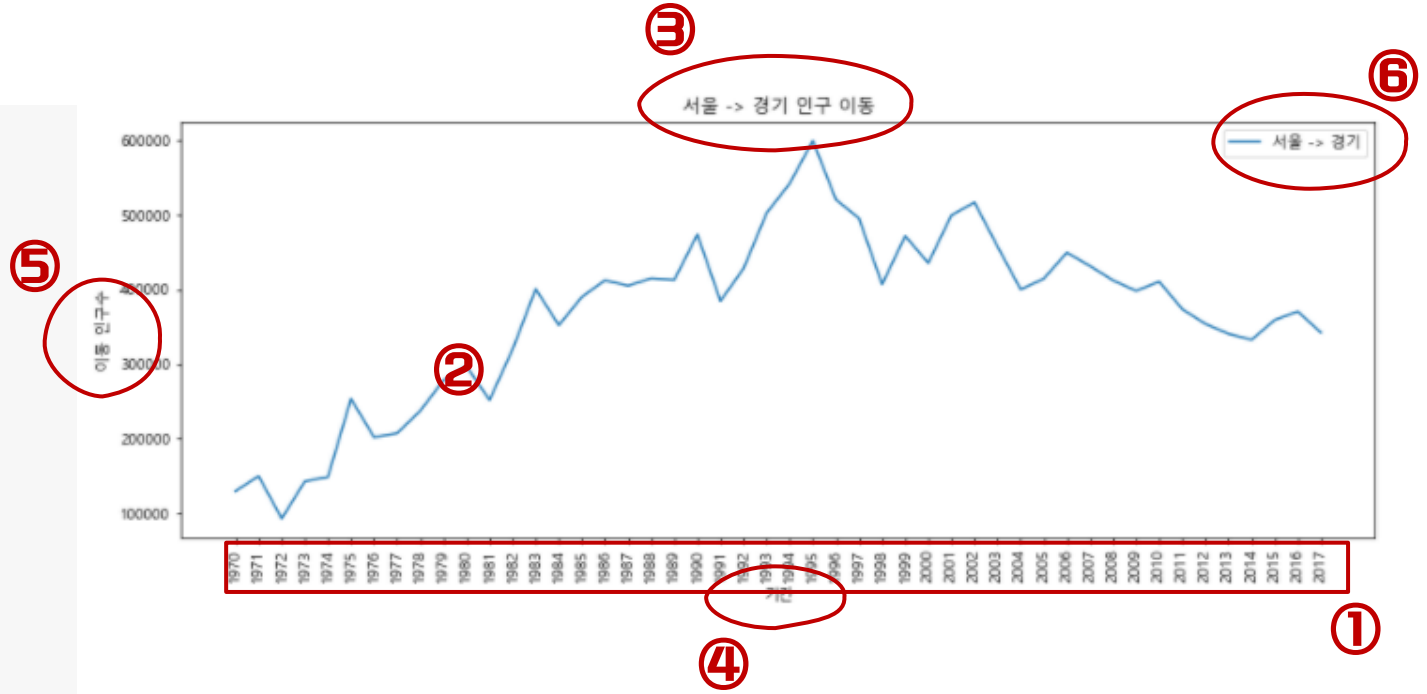
위 코드들과 동일합니다!
긁어오세요!



03. Matplotlib 기본 그래프 구조 – 선그래프

그래프 꾸미기

```
#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
① plt.xticks(rotation="vertical") X축 수직으로 회전

#x,y축 데이터를 plot함수에 입력
② plt.plot(sr_one.index, sr_one.values)

③ plt.title("서울 -> 경기 인구 이동")  #차트 제목
④ plt.xlabel("기간")  #x축 이름
⑤ plt.ylabel("이동 인구수")  #y축 이름

⑥ plt.legend(labels=["서울 -> 경기"], loc="best")  #범례 표시

plt.show() #변경사항 저장하고 그래프 출력
```



범례 TIP!!

▷ 'best' : 자동으로 계산하여 최적의 위치에 레전드를 위치시킨다.

▷ 그 외에 사용자가 직접 대략적으로 명시할 수 있는대 그 입력값은 아래와 같다.

- 'upper right', 'upper left', 'upper center'
- 'lower right', 'lower left', 'lower center',
- 'center', 'right', 'center left', 'center right'

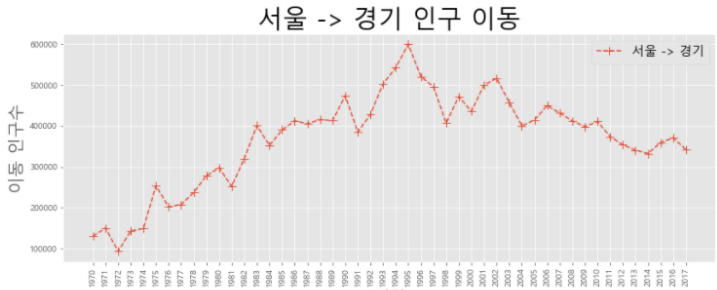
03. Matplotlib 기본 그래프 구조 – 선그래프

그래프 꾸미기

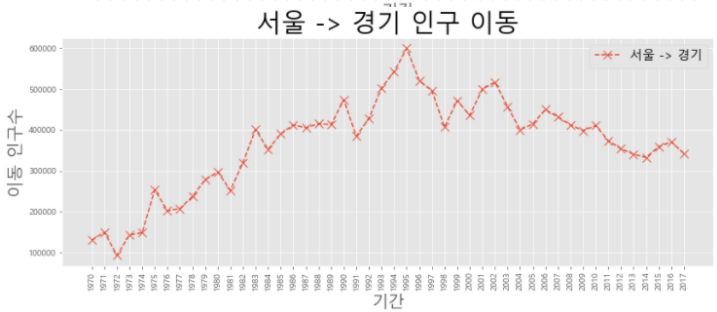
```
#서울에서 경기도로 이동한 인구 데이터 값만 선택
sr_one = df_seoul.loc["경기도"]

#스타일 서식 지정
plt.style.use("ggplot")

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

#x축 눈금 라벨 회전하기
plt.xticks(size=10, rotation="vertical")

#x, y축 데이터를 plot함수에 입력
plt.plot(sr_one.index, sr_one.values, ls="--", marker="o",
         markersize=10) #마커 표시 추가

plt.title("서울 -> 경기 인구 이동", size=30)  #차트 제목
plt.xlabel("기간", size=20)  #x축 이름
plt.ylabel("이동 인구수", size=20)  #y축 이름

plt.legend(labels=["서울 -> 경기"], loc="best", fontsize=15)  #범례 표시

plt.show()  #변경사항 저장하고 그래프 출력
```

다음 페이지 참고!

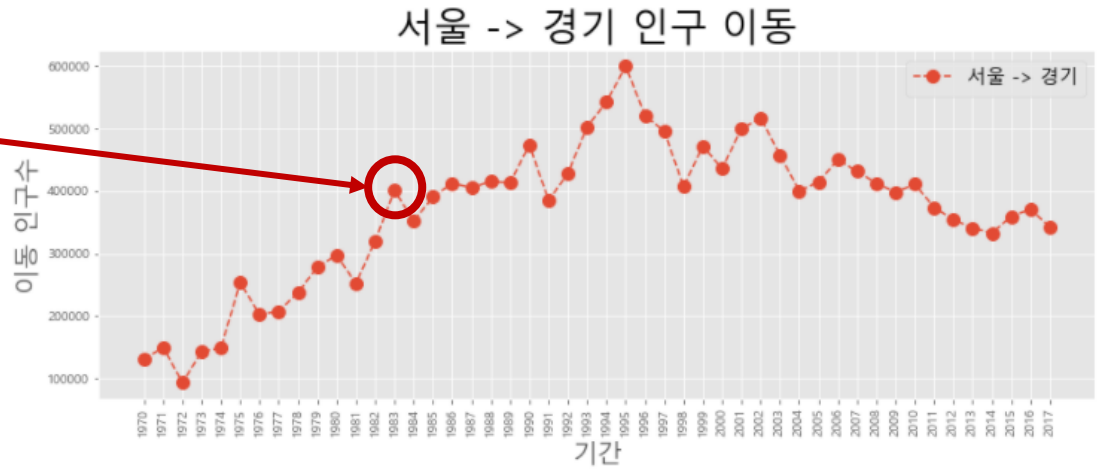
점선 표시, 원 모양 마커, 사이즈 10



marker = "x"



marker = "+"



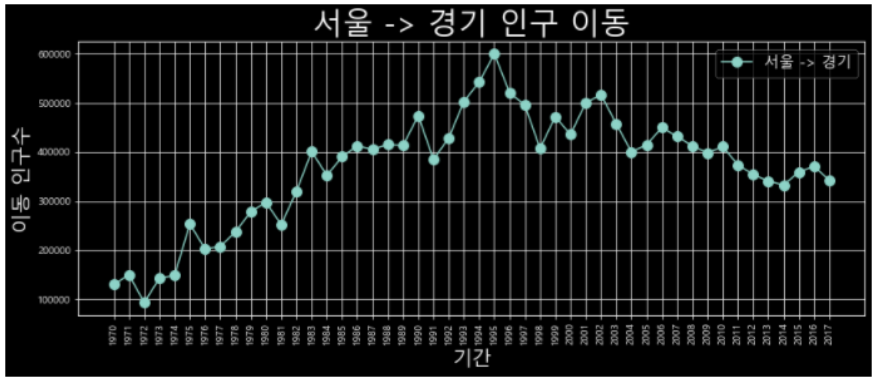
03. Matplotlib 기본 그래프 구조 – 선그래프

그래프 꾸미기 – 스타일 서식 지정

```
#라이브러리 불러오기
import matplotlib.pyplot as plt

#스타일 리스트 출력
print(plt.style.available)

#https://matplotlib.org/gallery/style_sheets/style_sheets_reference.html 에 들어가면 볼 수 있음
```

```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10']
```

plt.style.use("Solarize_Light2")



plt.style.use("dark_background")

plt.style.use("seaborn-colorblind")

03. Matplotlib 기본 그래프 구조 – 선그래프

그래프 꾸미기 – 주석표시

코드 긁어오기!

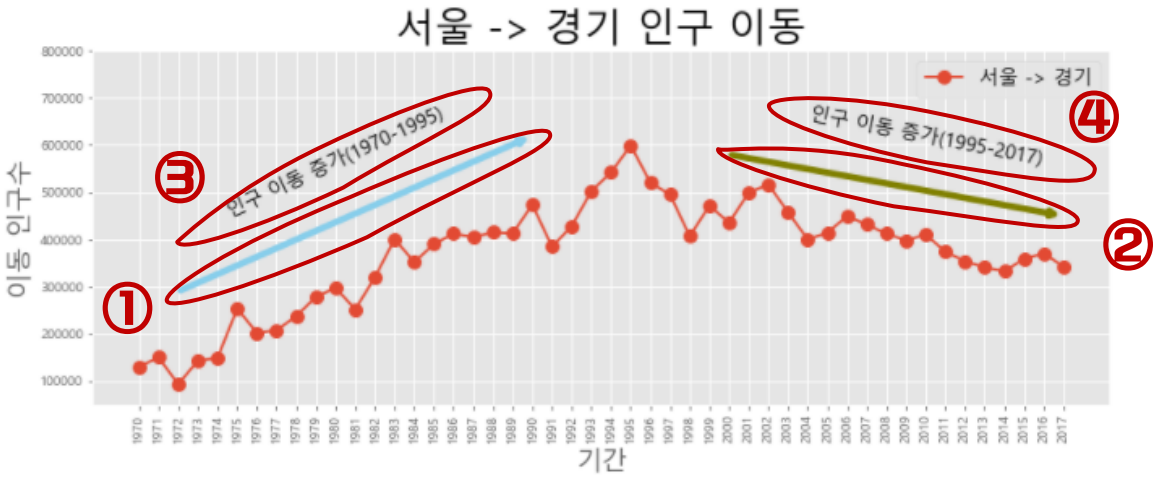
```
#서울에서 경기도로 이동한 인구 데이터 값만 선택
sr_one = df_seoul.loc["경기도"]

#스타일 서식 지정
plt.style.use("ggplot")

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

#x축 눈금 라벨 회전하기
plt.xticks(size=10, rotation="vertical")

#x, y축 데이터를 plot함수에 입력
plt.plot(sr_one.index, sr_one.values, marker="o", markersize=10)  #마커 표시 추가

plt.title("서울 -> 경기 인구 이동", size=30)  #차트 제목
plt.xlabel("기간", size=20)  #x축 이름
plt.ylabel("이동 인구수", size=20)  #y축 이름

plt.legend(labels=["서울 -> 경기"], loc="best", fontsize=15)  #범례 표시
```

```
#y축 범위 지정(최솟값, 최대값)
plt.ylim(50000, 800000)

#주석 표시 - 화살표
plt.annotate(
    "",
    xy=(20, 620000),  #화살표 머리 부분(끝점)
    xytext=(2, 290000),  #화살표 꼬리 부분(시작점)
    xycoords="data",  #좌표체계
    arrowprops=dict(arrowstyle="->", color="skyblue", lw=5),  #화살표 서식
)

plt.annotate(
    "",
    xy=(47, 450000),  #화살표 머리 부분(끝점)
    xytext=(30, 580000),  #화살표 꼬리 부분(시작점)
    xycoords="data",  #좌표체계
    arrowprops=dict(arrowstyle="->", color="olive", lw=5),  #화살표 서식
)
```

```
#주석 표시 - 텍스트
plt.annotate(
    "인구 이동 증가(1970-1995)",  #텍스트 입력
    xy=(10, 450000),  #텍스트 위치 기준점
    rotation=25,  #텍스트 회전 각도
    va="baseline",  #텍스트 상하 정렬
    ha="center",  #텍스트 좌우 정렬
    fontsize=15,  #텍스트 크기
)

plt.annotate(
    "인구 이동 증가(1995-2017)",  #텍스트 입력
    xy=(40, 560000),  #텍스트 위치 기준점
    rotation=-11,  #텍스트 회전 각도
    va="baseline",  #텍스트 상하 정렬
    ha="center",  #텍스트 좌우 정렬
    fontsize=15,  #텍스트 크기
)

plt.show()  #변경사항 저장하고 그래프 출력
```

annotate함수 TIP!!

▷rotation옵션 : 양의 회전 방향은 반시계방향
▷va옵션 : 'center', 'top', 'bottom', 'baseline'
(글자 위아래 세로 방향 정렬)
▷ha옵션 : 'center', 'left', 'right'
(글자 좌우 가로 방향 정렬)



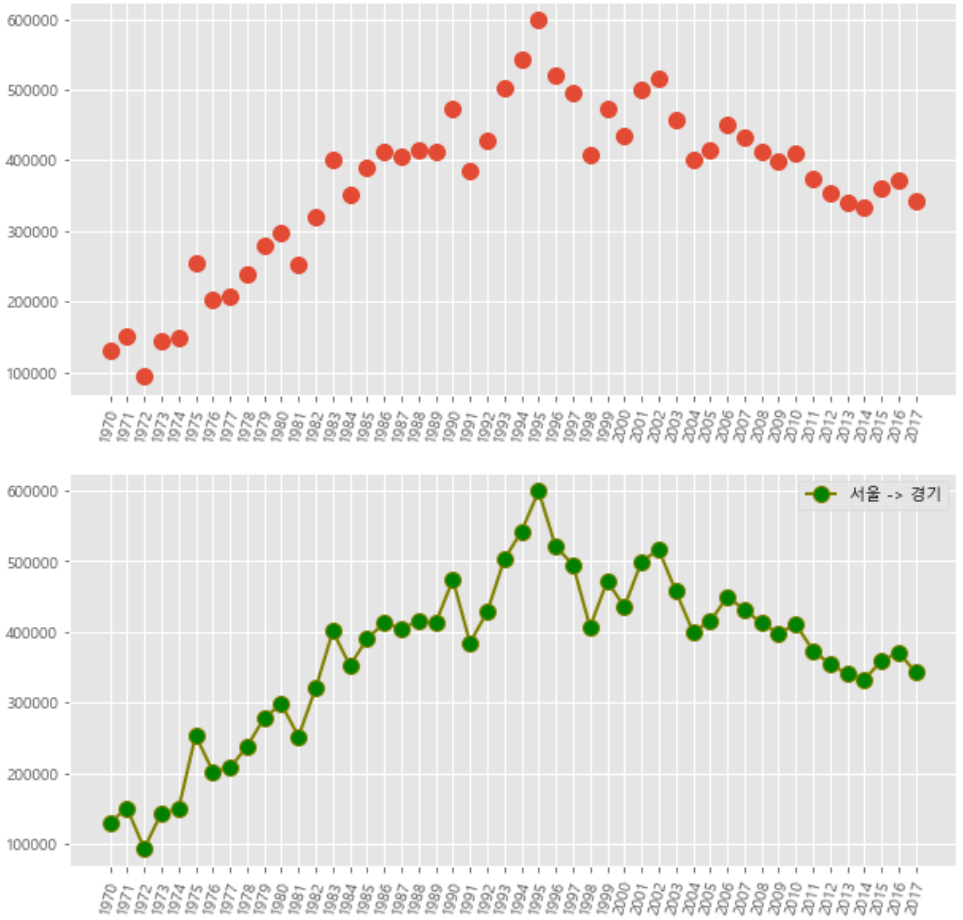
03. Matplotlib 기본 그래프 구조 - 선그래프

화면 분할하여 그래프 여러개 그리기 - axe객체 활용

```
# 그래프 객체 생성 (figure에 2개의 서브 플롯을 생성)
fig, ax = plt.subplots(2, 1, figsize=(10, 10))

# axe 객체에 plot 함수로 그래프 출력
ax[0].plot(sr_one, 'o', markersize=10)
ax[1].plot(sr_one,
           marker='o',
           markerfacecolor='green',
           markersize=10,
           color='olive',
           linewidth=2,
           label='서울 -> 경기')
ax[1].legend(loc='best')

#y축 범위 지정 (최소값, 최대값)
ax[0].set_ylim(50000, 800000)
ax[1].set_ylim(50000, 800000)

# 축 눈금 라벨 지정 및 75도 회전
ax[0].set_xticklabels(sr_one.index, rotation=75)
ax[1].set_xticklabels(sr_one.index, rotation=75)

plt.show()  # 변경사항 저장하고 그래프 출력
```

subplots(행크기, 열크기, figsize(가로, 세로))

글자 반시계 방향 75도 회전



03. Matplotlib 기본 그래프 구조 – 선그래프

axe객체 그래프 그리기

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

#axe객체에 plot함수로 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(최솟값, 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
```

① ax.set_ylim(50000, 800000)

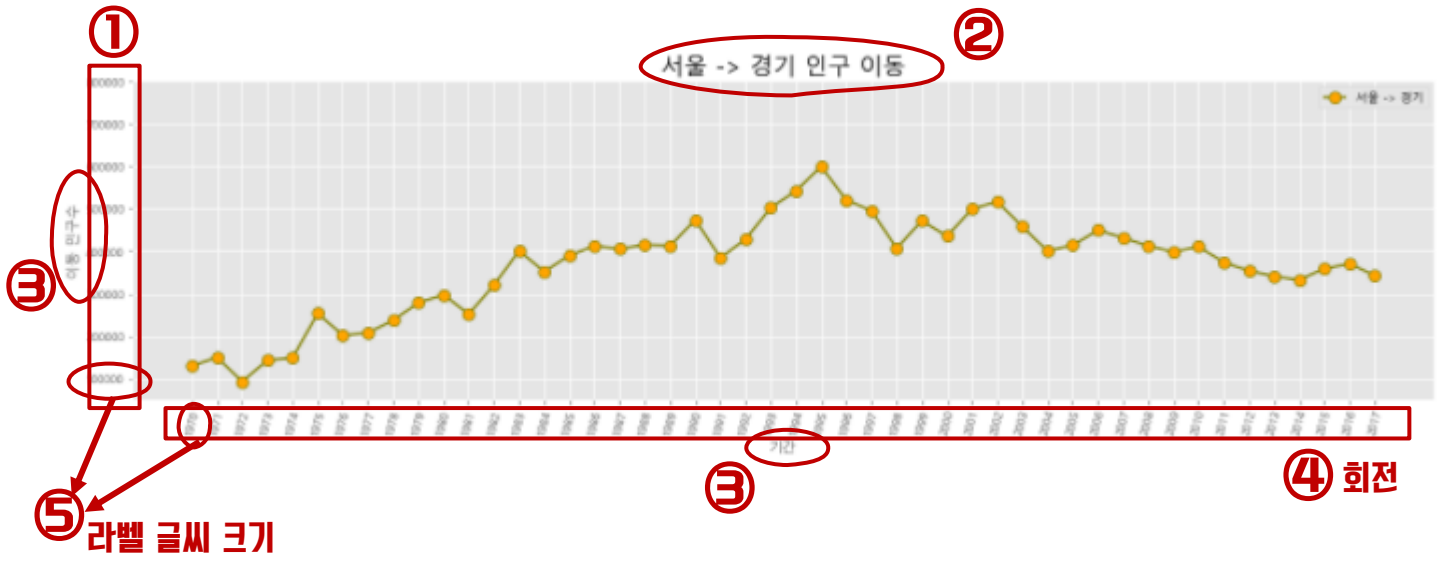
② ax.set_title("서울 -> 경기 인구 이동", size=20)

③ ax.set_xlabel("기간", size=12)
ax.set_ylabel("이동 인구수", size=12)

④ ax.set_xticklabels(sr_one.index, rotation=75)

⑤ ax.tick_params(axis="x", labelsize=10)
ax.tick_params(axis="y", labelsize=10)

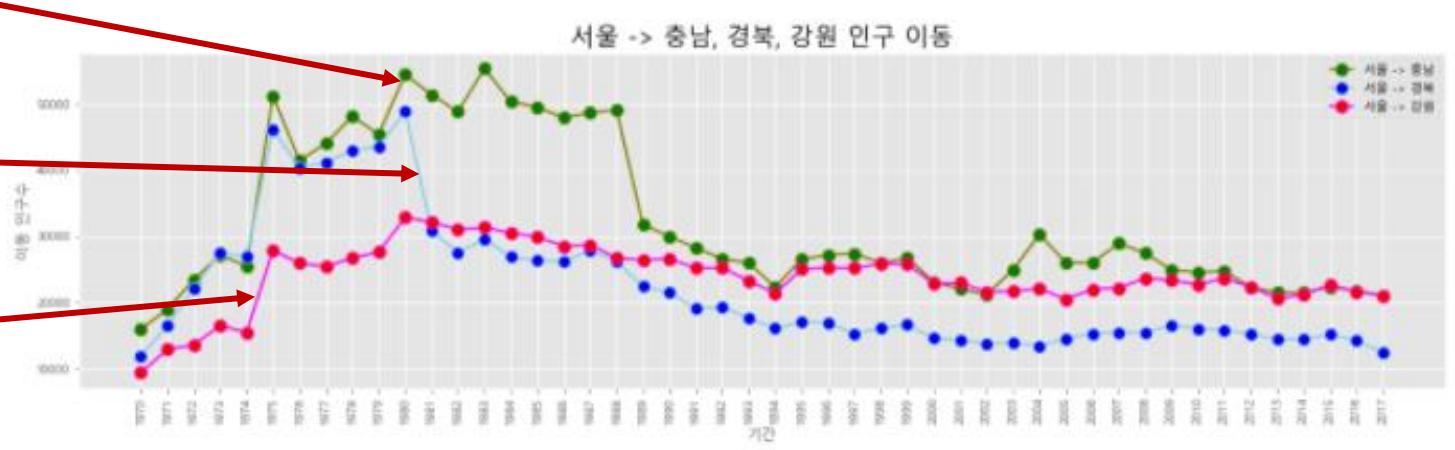
add_subplot(행크기, 열크기, 서브플롯 순서)



03. Matplotlib 기본 그래프 구조 – 선그래프

같은 화면에 그래프 추가

```
#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(figure 1개 서브 플롯 생성)
fig = plt.figure(figsize=(20, 5))
ax = fig.add_subplot(1, 1, 1)

#axe객체에 plot함수로 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
```



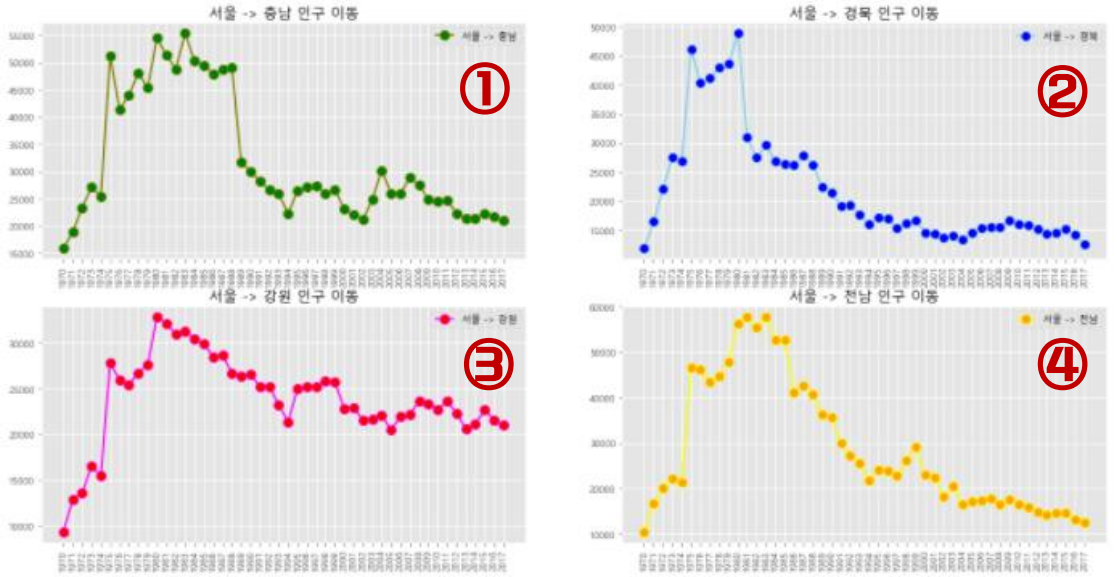
03. Matplotlib 기본 그래프 구조 – 선그래프

화면 4분할 그래프

```
#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]

#스타일 서식 지정
plt.style.use("ggplot")

#그래프 객체 생성(figure 1개 서브 플롯 생성)
fig = plt.figure(figsize=(20, 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)
```

```
#axe객체에 plot함수로 그래프 출력
ax1.plot(col_years,
         df_4.loc["충청남도", :],
         marker="o",
         markerfacecolor="green",
         markersize=10,
         color="olive",
         linewidth=2,
         label="서울 -> 충남")
ax2.plot(col_years,
         df_4.loc["경상북도", :],
         marker="o",
         markerfacecolor="blue",
         markersize=10,
         color="skyblue",
         linewidth=2,
         label="서울 -> 경북")
ax3.plot(col_years,
         df_4.loc["강원도", :],
         marker="o",
         markerfacecolor="red",
         markersize=10,
         color="magenta",
         linewidth=2,
         label="서울 -> 강원")
ax4.plot(col_years,
         df_4.loc["전라남도", :],
         marker="o",
         markerfacecolor="orange",
         markersize=10,
         color="yellow",
         linewidth=2,
         label="서울 -> 전남")
```

```
#범례 표시
ax1.legend(loc="best")
ax2.legend(loc="best")
ax3.legend(loc="best")
ax4.legend(loc="best")

#차트 제목 추가
ax1.set_title("서울 -> 충남 인구 이동", size=15)
ax2.set_title("서울 -> 경북 인구 이동", size=15)
ax3.set_title("서울 -> 강원 인구 이동", size=15)
ax4.set_title("서울 -> 전남 인구 이동", size=15)

#축 눈금 라벨 지정 및 90도 회전
ax1.set_xticklabels(col_years, rotation=90)
ax2.set_xticklabels(col_years, rotation=90)
ax3.set_xticklabels(col_years, rotation=90)
ax4.set_xticklabels(col_years, rotation=90)

plt.show()  #변경사항 저장하고 그래프 출력
```



03. Matplotlib 기본 그래프 구조 - 선그래프

Matplotlib에서 사용할 수 있는 색의 종류

```
#라이브러리 불러오기
import matplotlib

#컬러 정보를 담을 빈 딕셔너리 생성
colors = {}

#컬러 이름과 헥사코드 확인하여 딕셔너리에 저장
for name, hex in matplotlib.colors.cnames.items():
    colors[name] = hex

#딕셔너리 출력
print(colors)
```

```
{'aliceblue': '#F0F8FF', 'antiquewhite': '#FAEBD7', 'aqua': '#00FFFF', 'aquamarine': '#7FFFD4', 'a
zure': '#F0FFFF', 'beige': '#F5F5DC', 'bisque': '#FFE4C4', 'black': '#000000', 'blanchedalmond':
'#FFEBCD', 'blue': '#0000FF', 'blueviolet': '#8A2BE2', 'brown': '#A52A2A', 'burlywood': '#DEB887',
'cadetblue': '#5F9EA0', 'chartreuse': '#7FFF00', 'chocolate': '#D2691E', 'coral': '#FF7F50', 'corn
flowerblue': '#6495ED', 'cornsilk': '#FFF8DC', 'crimson': '#DC143C', 'cyan': '#00FFFF', 'darkblu
e': '#00008B', 'darkcyan': '#008B8B', 'darkgoldenrod': '#B8860B', 'darkgray': '#A9A9A9', 'darkgree
n': '#006400', 'darkgrey': '#A9A9A9', 'darkkhaki': '#BDB76B', 'darkmagenta': '#8B008B', 'darkolive
```

.
.
.

03. 면적 그래프

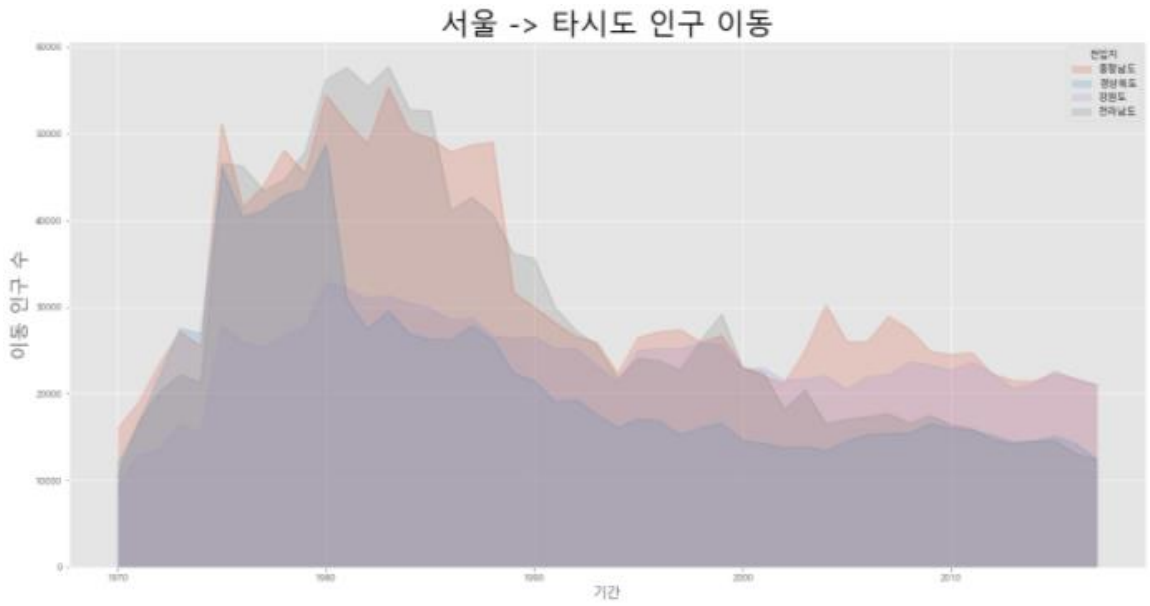
면적 그래프 그리기

```
#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 "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]
df_4 = df_4.transpose()

#스타일 서식 지정
plt.style.use("ggplot")

#데이터프레임의 인덱스 정수형의 변경(x축 눈금 라벨 표시)
df_4.index = df_4.index.map(int)

#면적 그래프 그리기
df_4.plot(kind="area", stacked=False, alpha=0.2, figsize=(20, 10))

plt.title("서울 -> 타시도 인구 이동", size=30)
plt.ylabel("이동 인구 수", size=20)
plt.xlabel("기간", fontsize=15)

plt.show()
```

면적그래프 옵션

stacked : False 누적X

alpha : 투명도 0~1 범위(기본값 0.5)



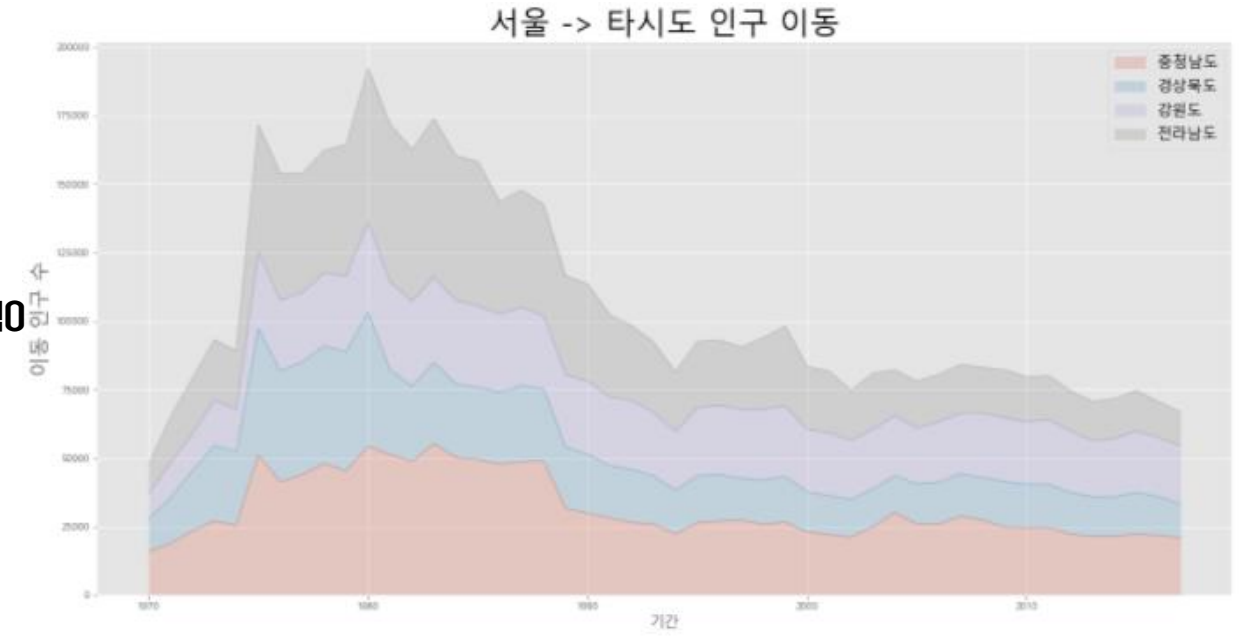
03. 면적 그래프

면적 그래프 그리기

```
#데이터프레임의 인덱스를 정수형으로 변경(x축 눈금 라벨 표시)
df_4.index = df_4.index.map(int)

#면적 그래프 그리기
df_4.plot(kind="area", stacked=True, alpha=0.2, figsize=(20, 10))

plt.title("서울 -> 타시도 인구 이동", size=30)
plt.ylabel("이동 인구 수", size=20)
plt.xlabel("기간", fontsize=15)
plt.legend(loc="best", fontsize=15)

plt.show()
```

앞 코드 긁어와서 이 부분만 수정

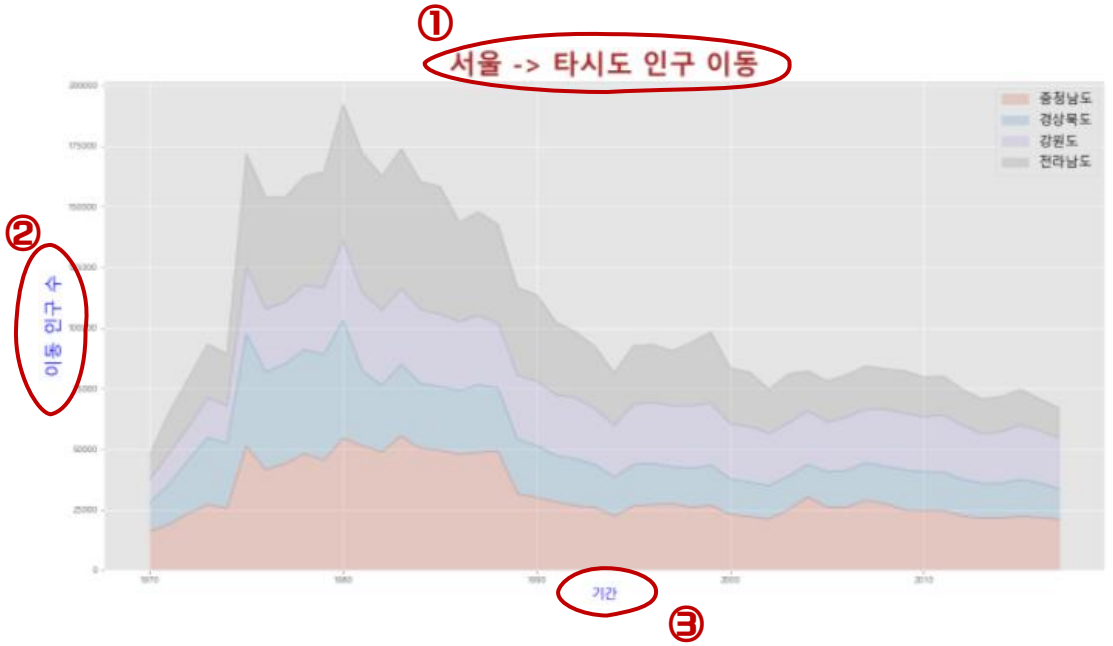
stacked : True 누적O



03. 면적 그래프

면적 그래프 그리기 – axe 객체 속성 변경

```
#데이터프레임의 인덱스를 정수형으로 변경(x축 눈금 라벨 표시)
df_4.index = df_4.index.map(int)

#면적 그래프 axe 객체 생성
ax = df_4.plot(kind="area", stacked=True, alpha=0.2, figsize=(20, 10))

#axe 객체 설정 변경
① ax.set_title("서울 -> 타시도 인구 이동", size=30, color="brown", weight="bold")
② ax.set_ylabel("이동 인구 수", size=20, color="blue")
③ ax.set_xlabel("기간", fontsize=15, color="blue")
ax.legend(loc="best", fontsize=15)

plt.show()
```



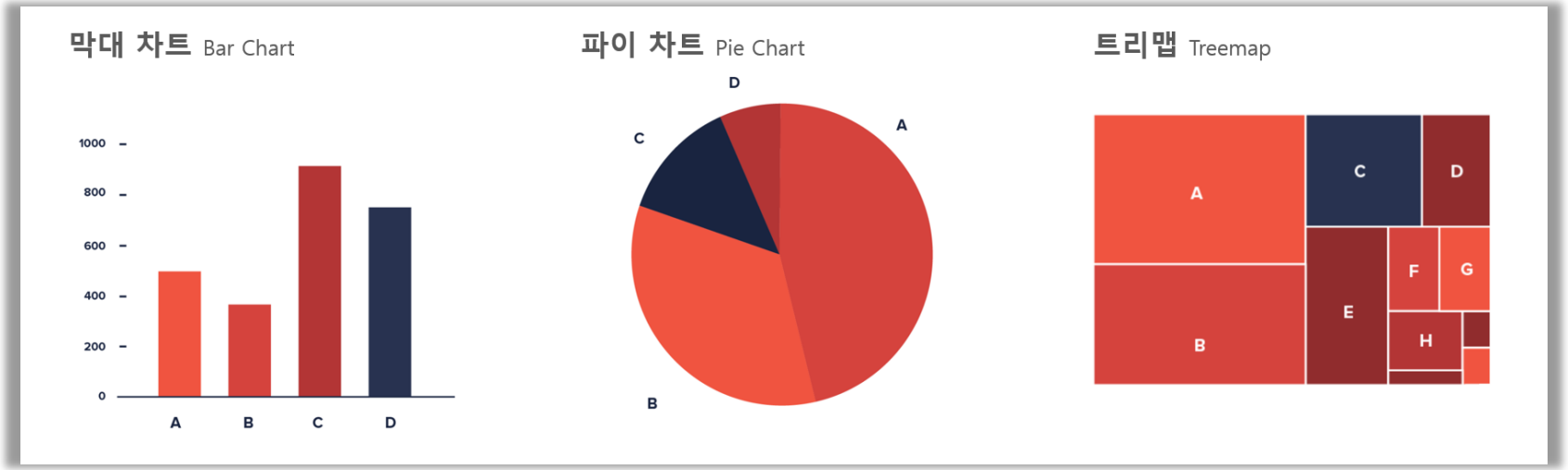
03. 범주형 자료 그래프

범주형 자료

값의 연속적인 자료가 아닌, 몇 개의 범주 또는 항목 형태로 나타나는 자료

관측값의 크기가 아닌, 자료가 갖는 범주의 종류와 각 범주가 나타나는 횟수에 집중

순위형 자료(ex.평점, 선호도)와 명목형 자료(ex. 혈액형 성별)

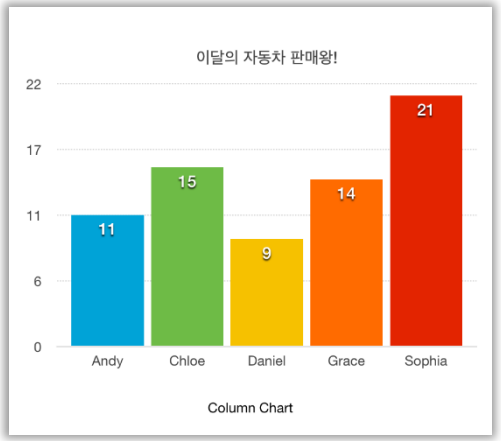


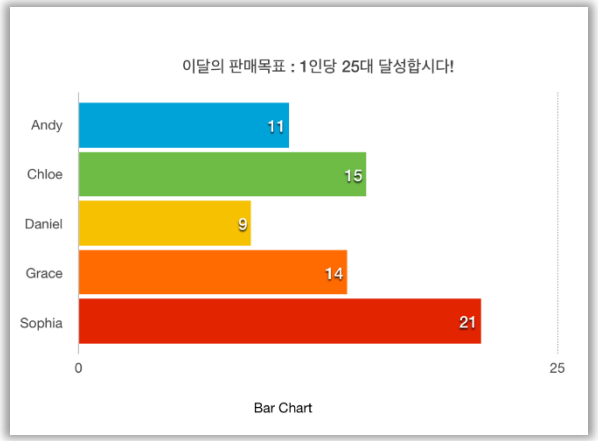
03. 범주형 자료 그래프

막대 그래프

각 범주의 데이터 값의 크기에 비례하여 직사각형 막대로 표현

막대 높이의 상대적 길이 차이를 통해 크고 작음을 설명





시계열 데이터 설명하기 적합

kind = `bar'

각 변수 사이 값의 크기 차이를 설명하는 데 적합

kind = `barh'

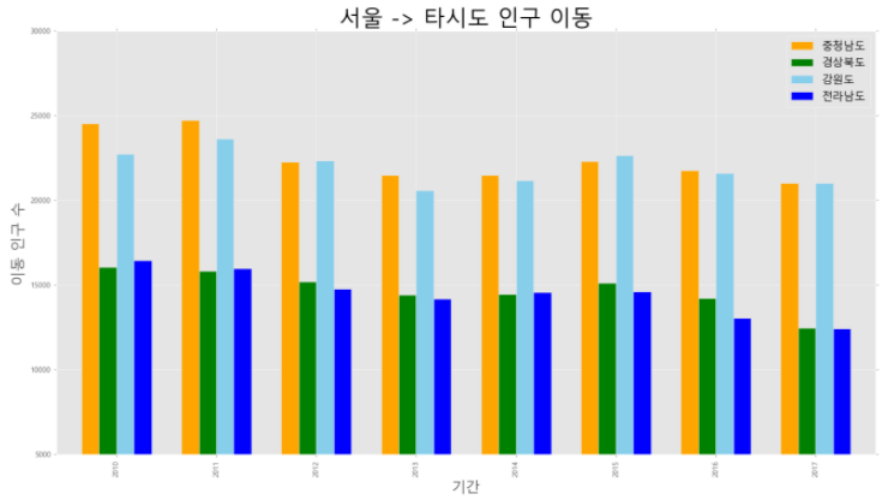
03. 일변량 범주형 변수 – 막대그래프(bar chart)

세로형 막대 그래프

df_4.plot(kind=`bar', figsize=(20,10), width=0.7, color=[`orange',`green',`skyblue',`blue'])

```
# 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]
df_4 = df_4.transpose()

# 스타일 서식 지정
plt.style.use('ggplot')

# 데이터프레임의 인덱스를 정수형으로 변경 (x축 눈금 라벨 표시)
df_4.index = df_4.index.map(int)

# 막대 그래프 그리기
df_4.plot(kind='bar', figsize=(20, 10), width=0.7,
          color=['orange', 'green', 'skyblue', 'blue'])

plt.title('서울 -> 타시도 인구 이동', size=30)
plt.ylabel('이동 인구 수', size=20)
plt.xlabel('기간', size=20)
plt.ylim(5000, 30000)
plt.legend(loc='best', fontsize=15)

plt.show()
```

– 막대그래프 세로형 지정 : kind=`bar'



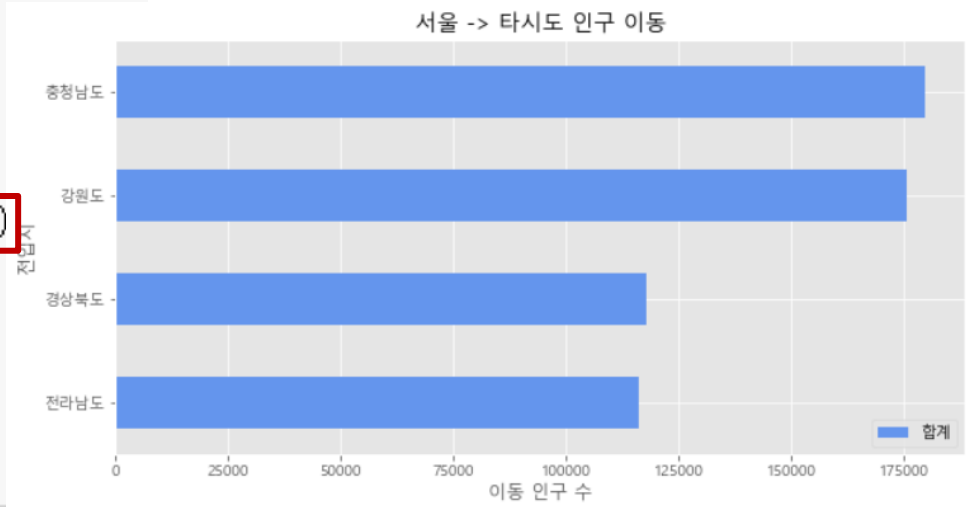
03. 일변량 범주형 변수 – 막대그래프(bar chart)

가로형 막대 그래프

df_total.plot(kind=`barh`, color=`cornflowerblue`, width=0.5, figsize=(10,5))

```
# 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()
```

– 막대그래프 가로형 지정 : kind=`barh`



03. 일변량 범주형 변수 – 막대그래프(bar chart)

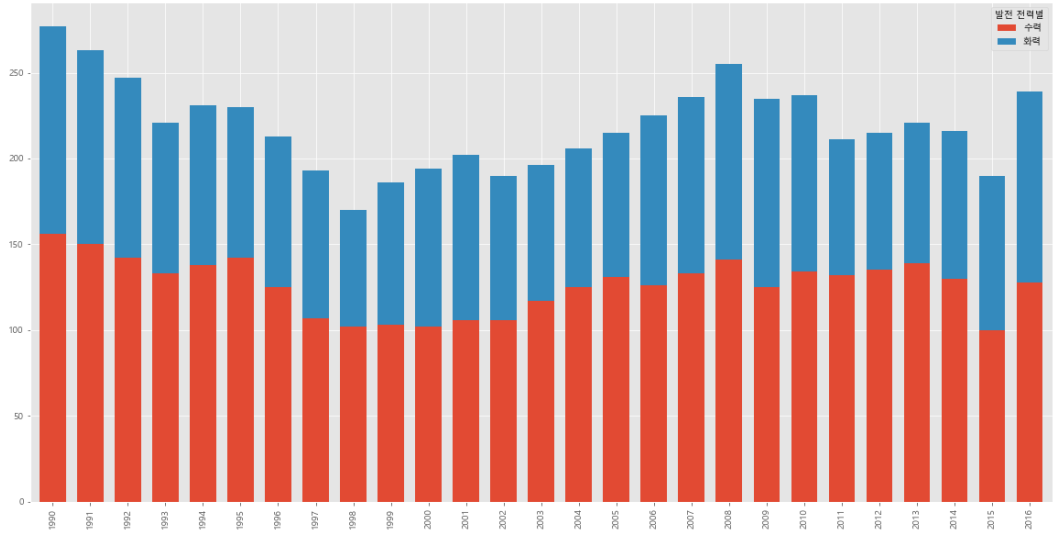
누적 그래프①

df[['수력','화력']].plot(kind='bar', figsize=(20, 10), width=0.7, stacked=True)

```
# -*- 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 = "c:/Windows/Fonts/malgun.ttf"   #폰트파일의 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 데이터프레임 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

# 누적 그래프 그리기
ax1 = df[['수력','화력']].plot(kind='bar', figsize=(20, 10), width=0.7, stacked=True)

plt.show()
```

판다스의
데이터 프레임 안의 함수

- df[[`항목1`,`항목2`]].plot
- stacked=True



03. 일변량 범주형 변수 – 막대그래프(bar chart)

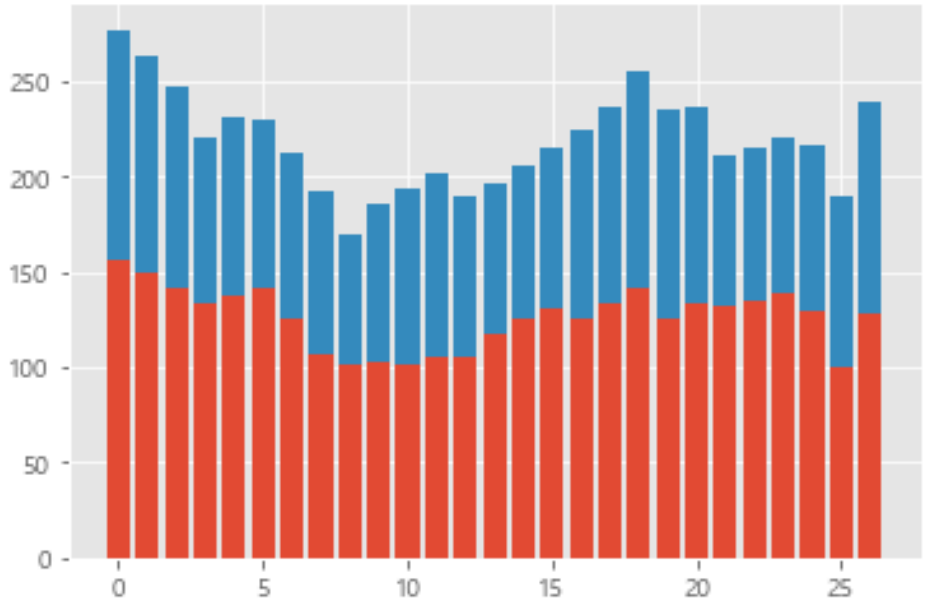
누적 그래프②

plt.bar(range(len(df['수력'])), df['수력'])
plt.bar(range(len(df['화력'])), df['화력'], bottom=df['수력'])

```
23  # 누적 그래프 그리기
24  plt.bar(range(len(df['수력'])), df['수력'])
25  plt.bar(range(len(df['화력'])), df['화력'], bottom=df['수력'])
26
27  plt.show()
```

Matplotlib의 함수

상단 그래프와 하단 그래프를 구분해야 하기 때문에
plt.legend(loc="best") 을 달아주자.



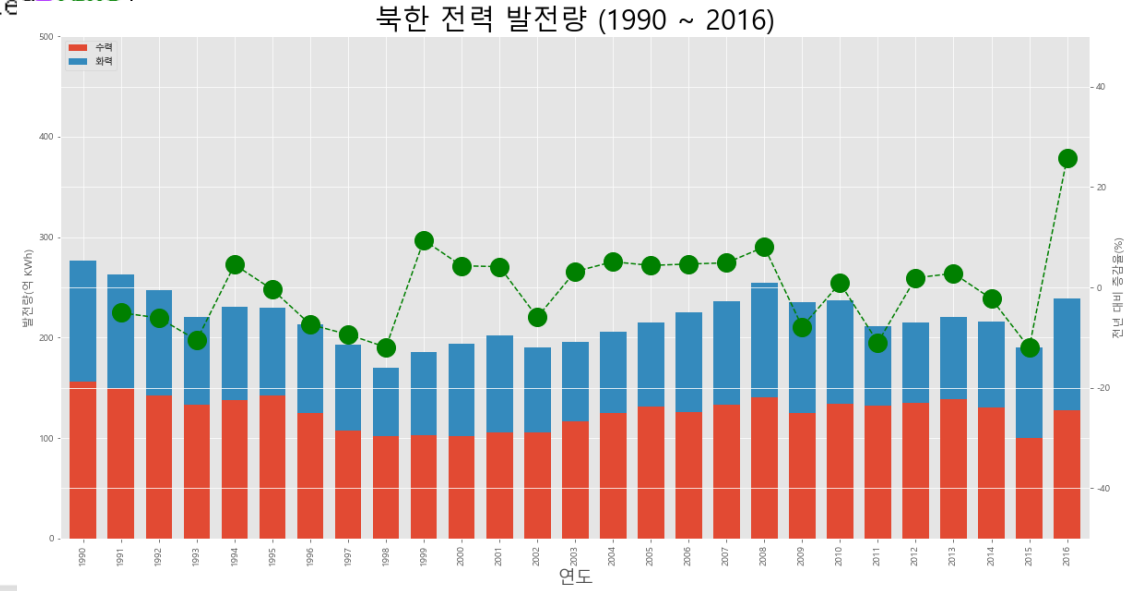
03. 일변량 범주형 변수 – 막대그래프(bar chart)

2축 그래프

보조 축을 추가하여 2개의 축을 갖는 그래프

- 쌍둥이 객체 만들기 : ax1.twinx()

```
23  # 증감율(변동률) 계산
24  df = df.rename(columns={'합계':'총발전량'})
25  df['총발전량 - 1년'] = df['총발전량'].shift(1)
26  df['증감율'] = ((df['총발전량'] / df['총발전량 - 1년']) - 1) * 100
27
28  # 2축 그래프 그리기
29  ax1 = df[['수력','화력']].plot(kind='bar', figsize=(20, 10), width=0.7, stacked=True)
30  ax2 = ax1.twinx()
31  ax2.plot(df.index, df.증감율, ls='--', marker='o', markersize=20,
32           color='green', label='전년대비 증감율(%)')
33
34  ax1.set_ylim(0, 500)
35  ax2.set_ylim(-50, 50)
36
37  ax1.set_xlabel('연도', size=20)
38  ax1.set_ylabel('발전량(억 KWh)')
39  ax2.set_ylabel('전년 대비 증감율(%)')
40
41  plt.title('북한 전력 발전량 (1990 ~ 2016)', size=30)
42  ax1.legend(loc='upper left')
43
44  plt.show()
```

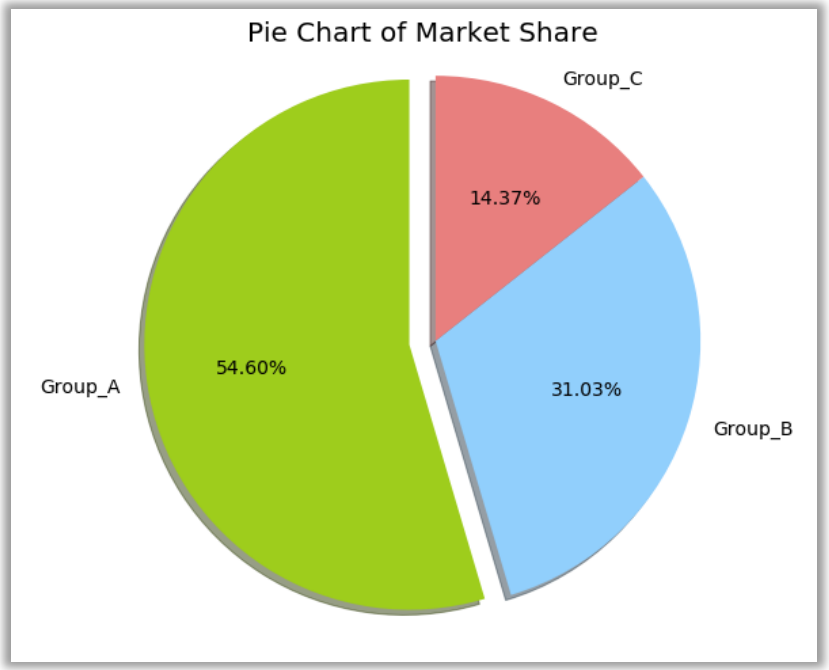


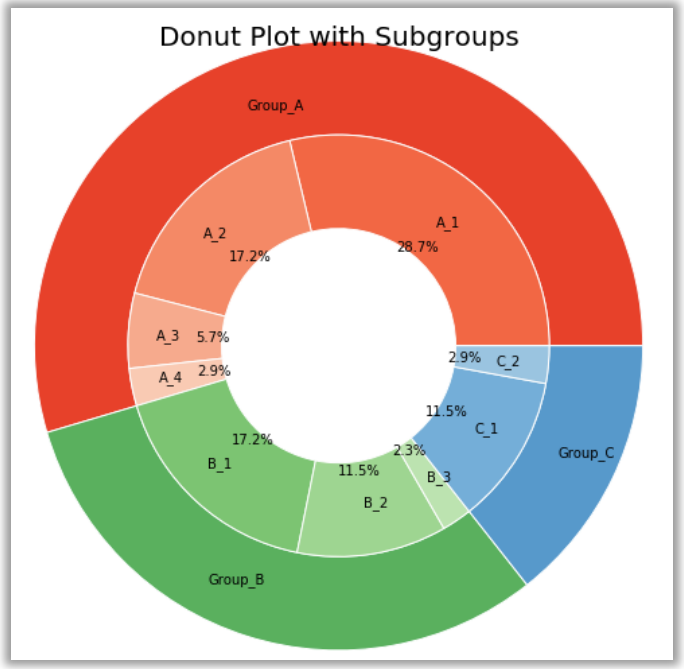
03. 범주형 자료 그래프

파이 차트

원을 파이 조각처럼 나누어 표현한 그래프

조각의 크기는 해당 변수의 데이터 크기에 비례





03. 일변량 범주형 변수 – 원 그래프(pie chart)

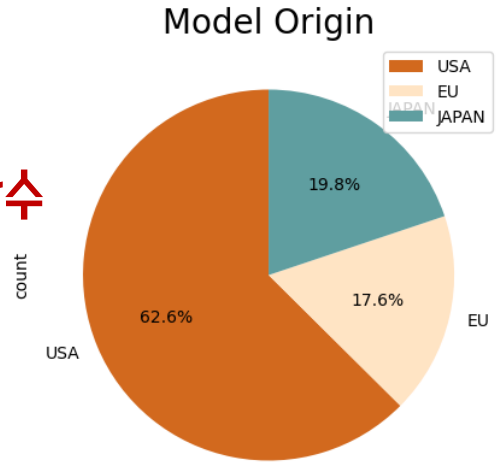
파이 차트①

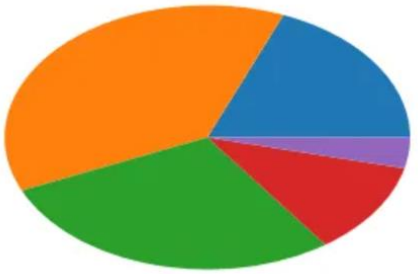
df_origin['count'].plot
(kind='pie', figsize=(7, 5), autopct='%1.1f%%', startangle=90, colors=['chocolate', 'bisque', 'cadetblue'])

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

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

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

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

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

# 데이터 개수 카운트를 위해 값 1을 가진 열을 추가
df['count'] = 1
df_origin = df.groupby('origin').sum()   # origin 열을 기준으로 그룹화, 합계 연산
print(df_origin.head())                  # 그룹 연산 결과 출력

# 제조국가(origin) 값을 실제 지역명으로 변경
df_origin.index = ['USA', 'EU', 'JAPAN']

# 제조국가(origin) 열에 대한 파이 차트 그리기 - count 열 데이터 사용
df_origin['count'].plot(kind='pie',
                        figsize=(7, 5),
                        autopct='%1.1f%%',   # 퍼센트 % 표시
                        startangle=90,       # 파이 조각을 나누는 시작점(각도 표시)
                        colors=['chocolate', 'bisque', 'cadetblue']    # 색상 리스트
                        )

plt.title('Model Origin', size=20)
plt.legend(labels=df_origin.index, loc='upper right')   # 범례 표시
plt.show()
```

판다스의
데이터 프레임 안의 함수

- 파이차트 : .plot(kind=`pie')
- 완전한 원형으로 : plt.axis(`equal')
- 파이 조각의 시작점 : startangle





plt.axis(`equal' 지정 X)

03. 일변량 범주형 변수 – 원 그래프(pie chart)

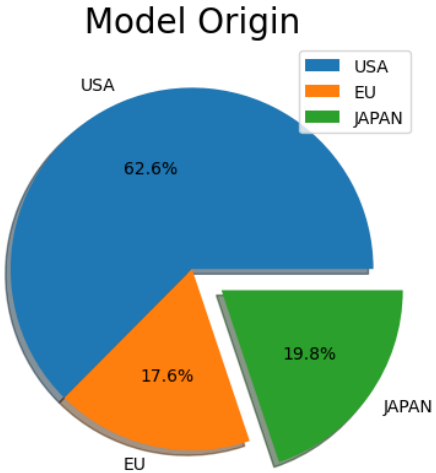
파이 차트②

plt.pie(df_origin['count'],explode=None, labels=None,
colors=None, autopct=None, shadow=False)

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

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

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

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

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

# 데이터 개수 카운트를 위해 값 1을 가진 열을 추가
df['count'] = 1
df_origin = df.groupby('origin').sum()   # origin 열을 기준으로 그룹화, 합계 연산
print(df_origin.head())                  # 그룹 연산 결과 출력

# 제조국가(origin) 값을 실제 지역명으로 변경
df_origin.index = ['USA', 'EU', 'JAPAN']

# 제조국가(origin) 열에 대한 파이 차트 그리기 - count 열 데이터 사용
plt.pie(df_origin['count'],explode= [0,0,0.2], labels=df_origin.index,
    colors=None, autopct='%1.1f%%', shadow=True)


plt.title('Model Origin', size=20)
plt.legend(labels=df_origin.index, loc='upper right')   # 범례 표시
plt.show()
```

Matplotlib의 함수

- 파이차트 : .pie(data, (옵션들))
- 파이조각 돌출 : explode
- 파이차트 그림자 : shadow



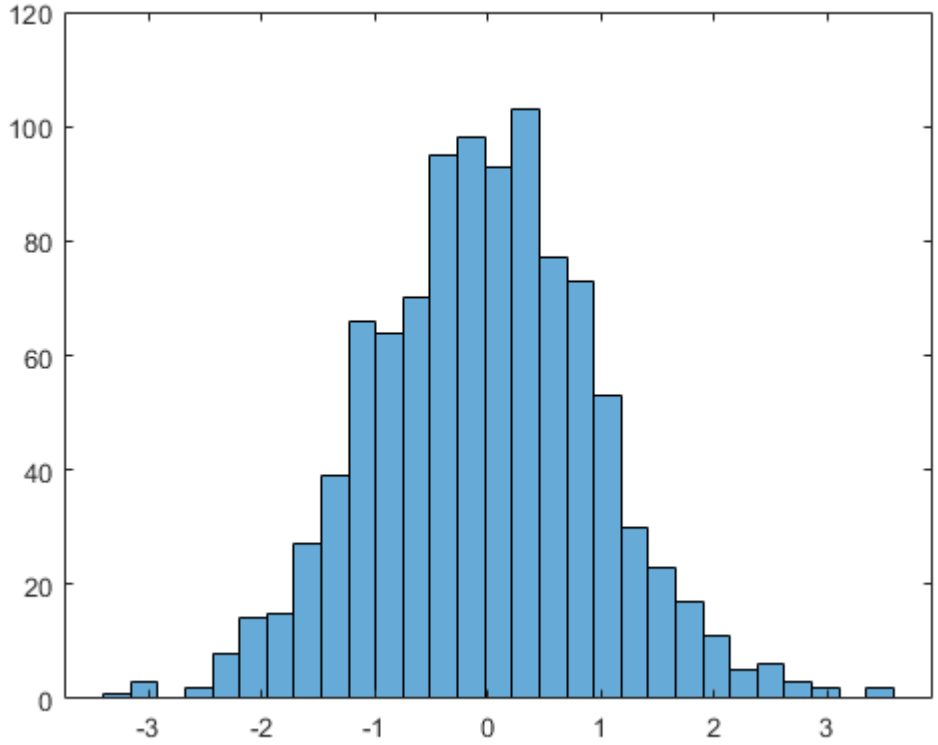
전체 대비 백분율 : autopct

- `%0.2f' : 소수점 두번째자리까지 보여줌
- '%0.2f%%' : 소숫점 두번째자리까지 보여주고 뒤에 %
- `%d%% ' : 가장 가까운 정수로 반올림하고 맨 뒤에 %

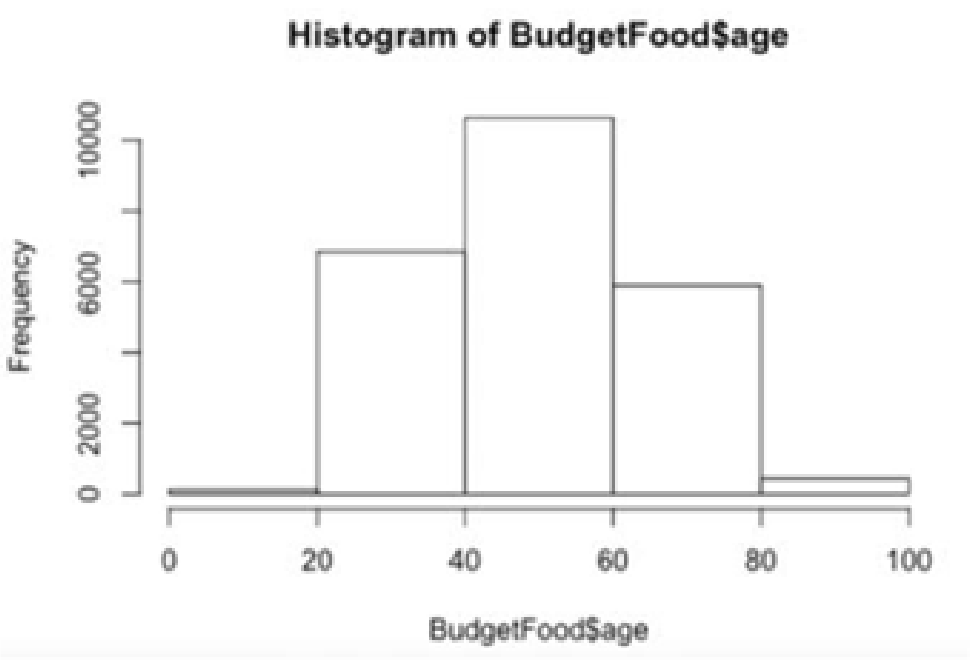
03. 일변량 연속형 변수 – 히스토그램(histogram)

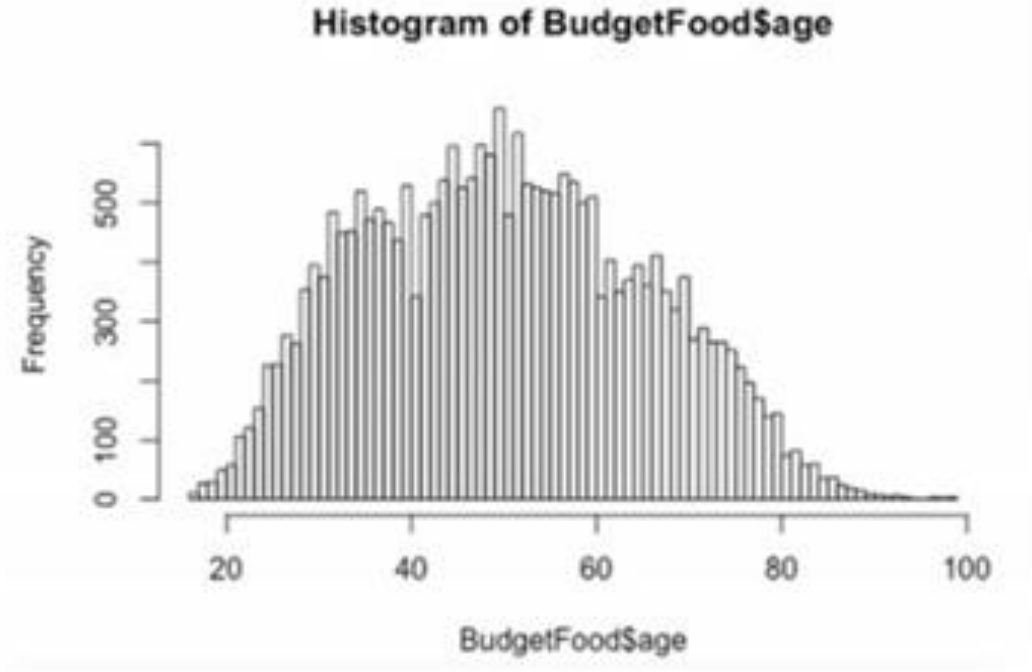
* 히스토그램이란?

- 단변수 데이터의 빈도수를 표현
- 주로 연속형 데이터의 분포를 파악할 때 많이 사용
- X축 : 데이터의 전범위를 같은 크기의 여러 구간으로 구분
- Y축 : 각 구간에 속한 데이터의 빈도수 표현
- 구간 간격 설정이 중요!!(구간 간격 크기에 따라 모양이 달라짐 적절한 구간 설정 중요)



03. 일변량 연속형 변수 – 히스토그램(histogram)



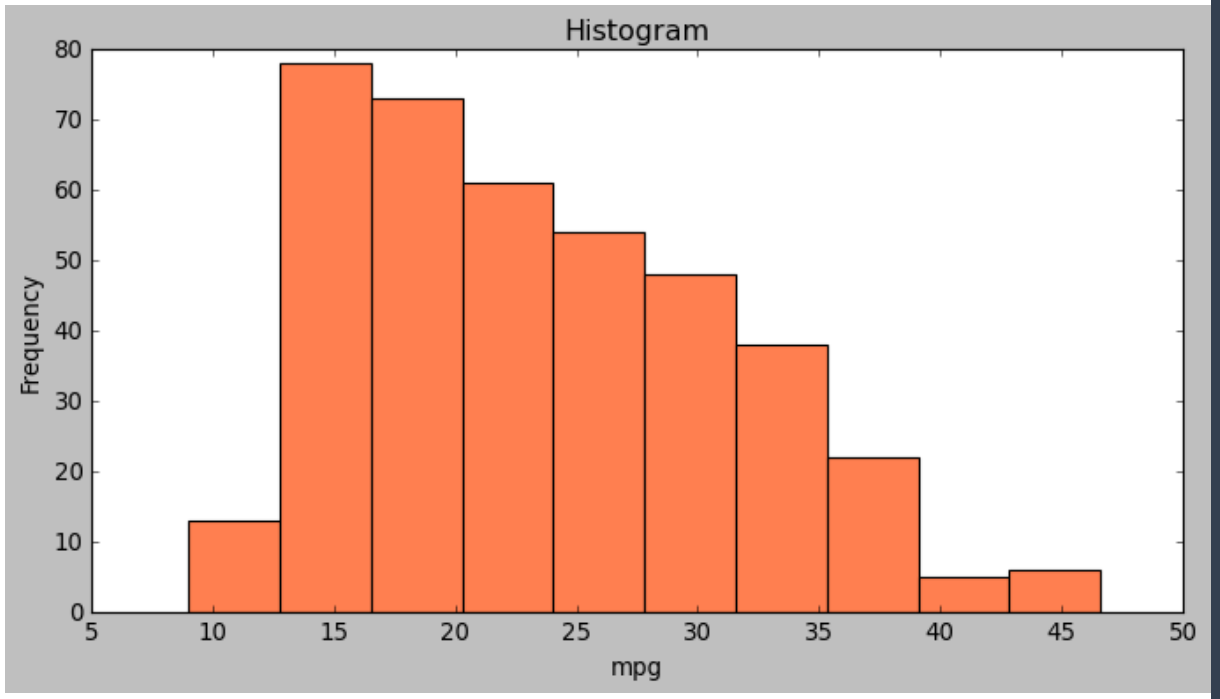


같은 변수, 같은 히스토그램이지만 구간 수에 따라 확연히 다른 그래프가 그려짐을 볼 수 있다!!

03. 일변량 연속형 변수 – 히스토그램(histogram)

* df.plot(kind=`hist', bins=10, …) or df.plot.hist(bins=10,…)

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

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

# 열 이름을 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, # 구간 수 설정
               color='coral', figsize=(10, 5))

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



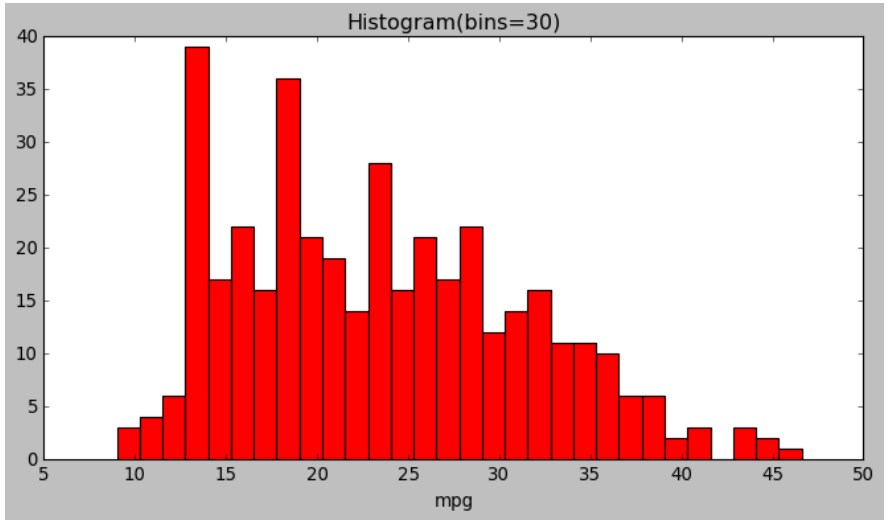
- 데이터(dataframe or series).plot(kind = `hist',…)가 기본 형태
- bins : 구간의 개수 지정(default는 10)
- 좌편향된 그래프임을 알 수 있다

03. 일변량 연속형 변수 – 히스토그램(histogram)

* 구간값 변화에 따른 비교(plt.hist() 사용)

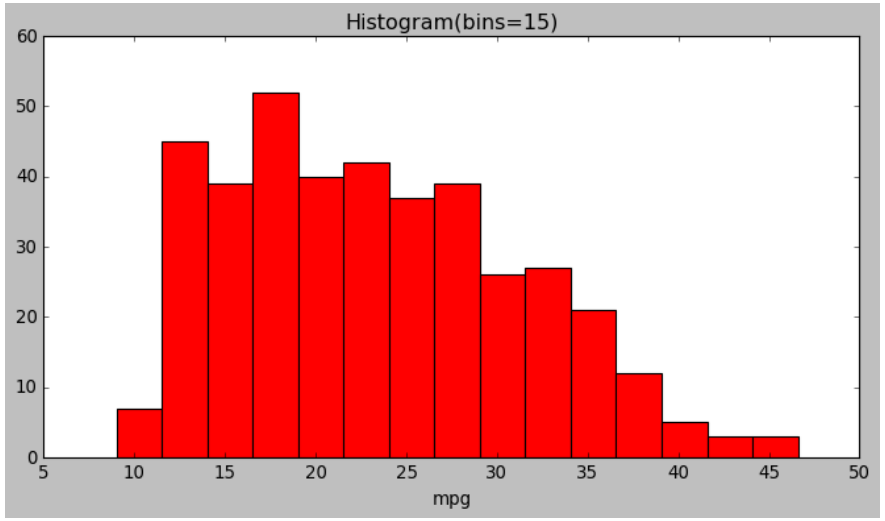
```
plt.figure(figsize=(10,5))
plt.hist(df['mpg'], bins = 30, color='red')

# 그래프 꾸미기
plt.title('Histogram(bins=30)')
plt.xlabel('mpg')
plt.show()
```

plt.hist는 함수 내부에 figsize 옵션x



- 구간수를 30으로 설정하면 너무 자세한 그래프로 인해 분포파악이 오히려 더 힘듦

```
plt.figure(figsize=(10,5))
plt.hist(df['mpg'], bins = 15, color='red')

# 그래프 꾸미기
plt.title('Histogram(bins=15)')
plt.xlabel('mpg')
plt.show()
```

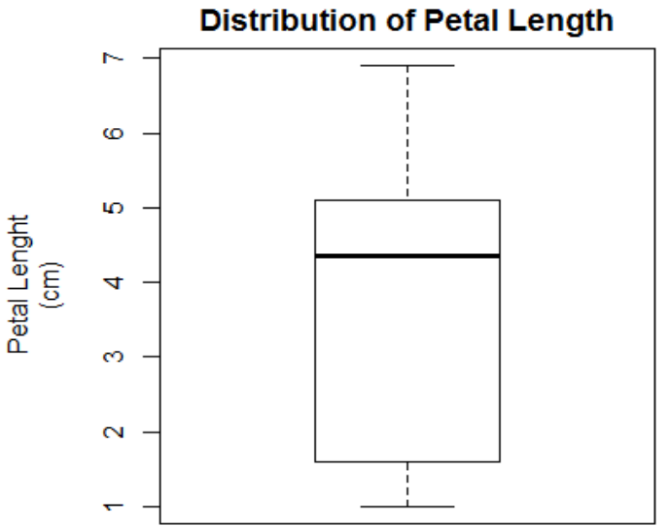


- 구간수가 10일 때보다 분포를 더 자세히 나타내면서 매끄러운 15가 적당

03. 일변량 연속형 변수 – 박스플롯(boxplot)

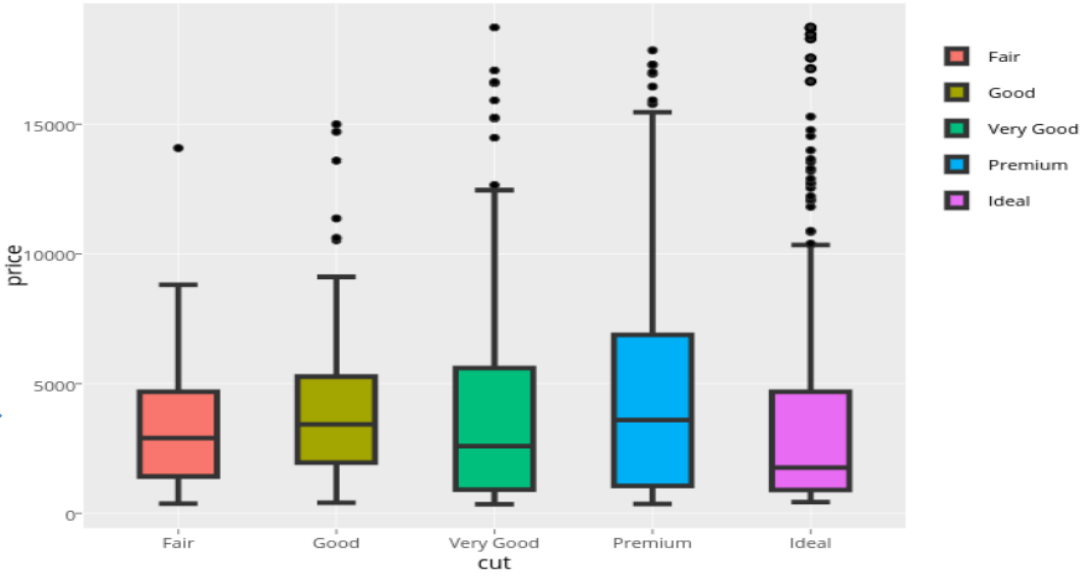
* 박스플롯(상자그림)이란?

- 5개의 요약통계량(최소값,1분위수,중앙값,3분위수, 최대값) 제공
- 분포의 대칭성,치우침(왜도),이상치 쉽게 파악 가능
- 일변량, 다변량의 경우 모두 사용
  → 여러 변수의 정보를 한번에 비교하기 편하기 때문
- 일변량의 경우 연속형 변수
- 다변량의 경우 주로 범주형 & 연속형 변수 또는 서로 다른 연속형 변수



하나의 연속형 변수

연속형 & 범주형 변수



03. 일변량 연속형 변수 – 박스플롯(boxplot)

* 박스플롯 이해를 위한 기본 통계 용어

- 중앙값(median) : 데이터의 중심(50%) 값
- 1분위수(Q1) : 데이터의 25%지점 값
- 3분위수(Q3) : 데이터의 75%지점 값
- IQR = Q3 – Q1 : 그림에서 상자의 길이
- Upper fence = Q3 + 1.5 * IQR
  Lower fence = Q1 – 1.5 * IQR
- fence를 넘어가는 값들은 이상치로 간주하고 점으로 표시

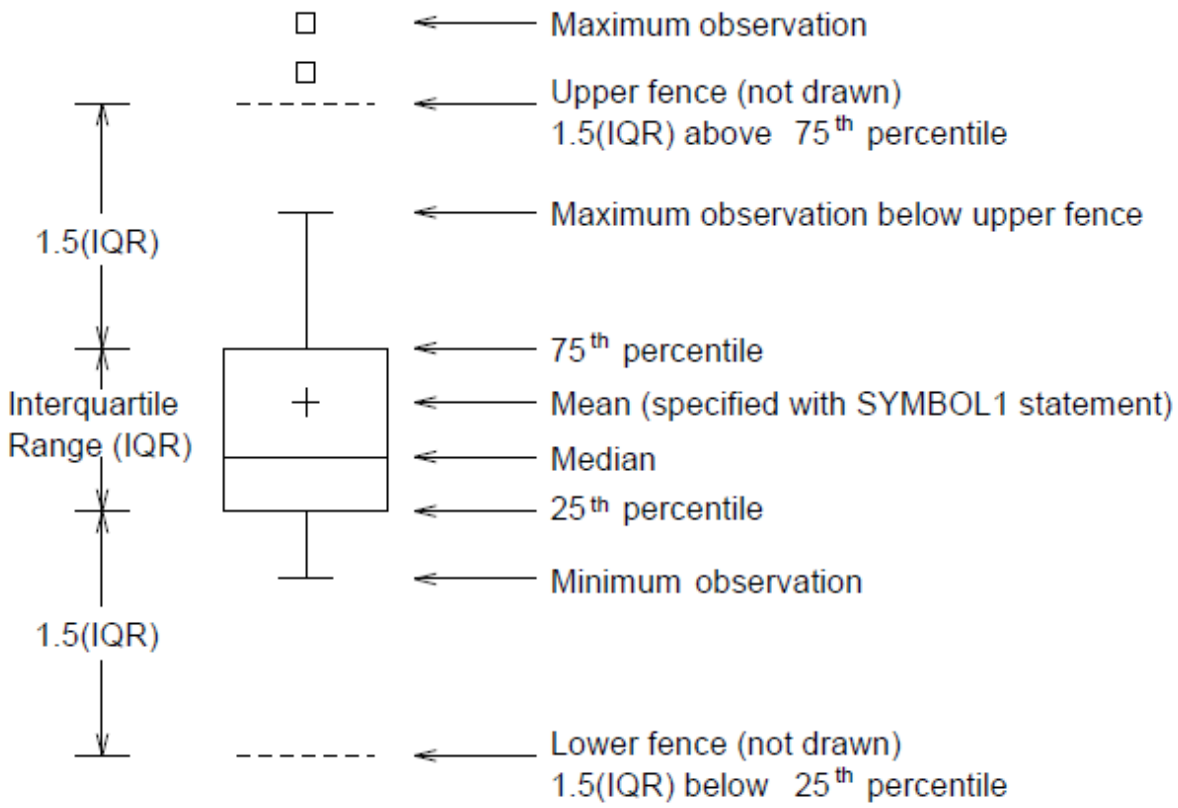


Figure 18.4. BOXSTYLE= SCHEMATIC

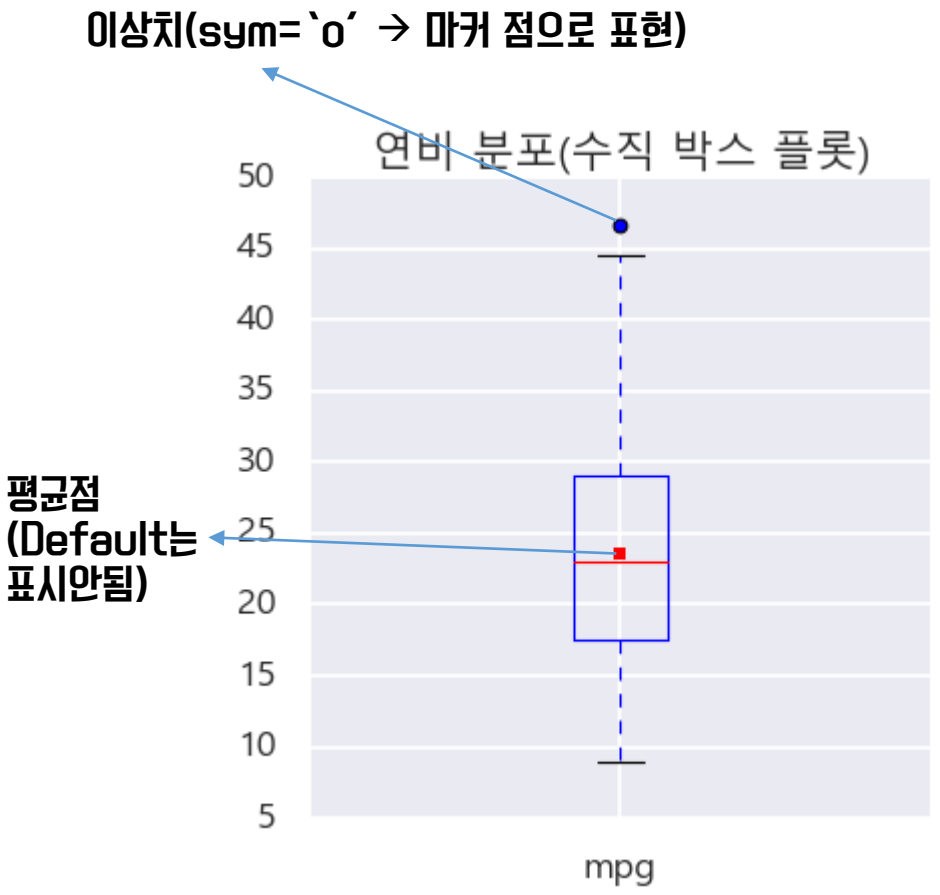
03. 일변량 연속형 변수 – 박스플롯(boxplot)

* plt.boxplot(x, sym=None, vert=True, labels=None, showmeans=False, …)

- sym : 이상치 마커 모양 지정(default는 `+`로 표현됨)
- showmeans = True : 평균값 표시

```
from matplotlib import font_manager, rc
font_path = "./malgun.ttf"   #폰트파일의 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']=True    # 마이너스 부호 출력 설정

# 그래프 객체 생성
fig=plt.figure(figsize=(5,5))

#  boxplot 그래프
plt.boxplot(x=df['mpg'],labels=['mpg'],sym='o',vert=True,showmeans=True)

plt.title('연비 분포(수직 박스 플롯)')
plt.show()
```

한글폰트 설정 코드는
앞에서 긁어와 주세요



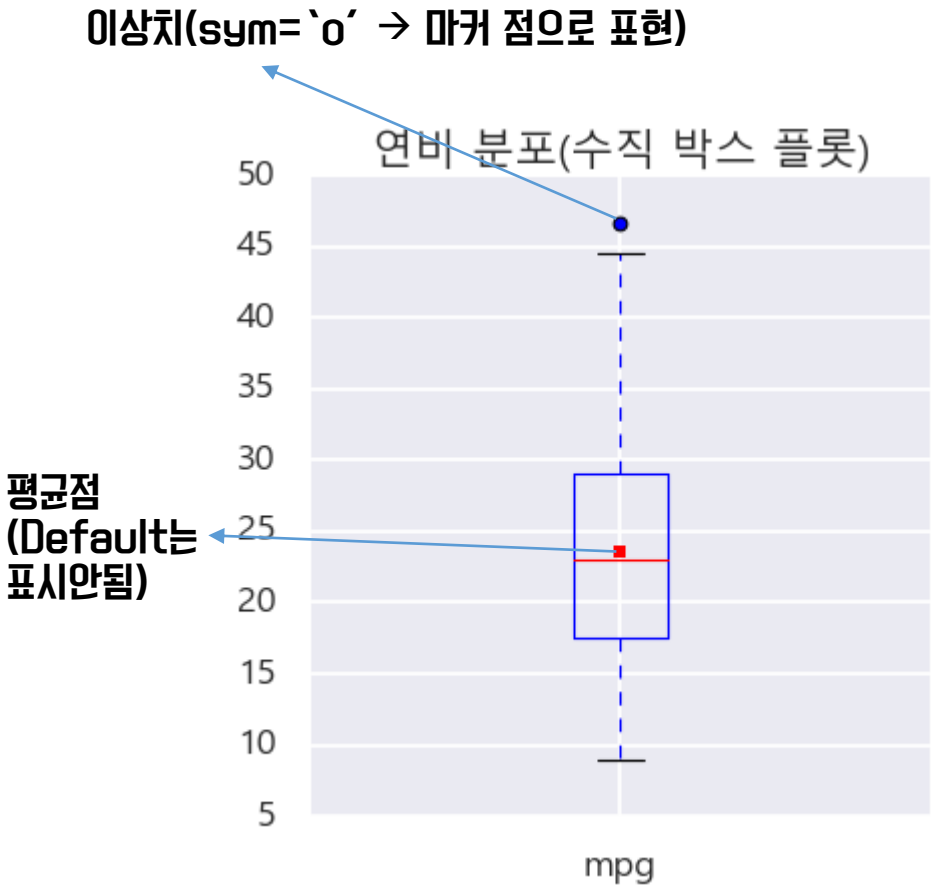
03. 일변량 연속형 변수 – 박스플롯(boxplot)

* plt.boxplot(x, sym=None, vert=True, labels=None, showmeans=False, …)

- sym : 이상치 마커 모양 지정(default는 `+`로 표현됨)
- showmeans = True : 평균값 표시

```
from matplotlib import font_manager, rc
font_path = "./malgun.ttf"   #폰트파일의 위치
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc('font', family=font_name)

plt.style.use('seaborn-poster')
plt.rcParams['axes.unicode_minus'] = False

# 그래프 객체 생성
fig = plt.figure(figsize=(15, 5))

# boxplot그래프 출력
plt.boxplot(x=df['mpg'],labels = ['mpg'],sym = 'o',vert=False,showmeans = True)

plt.title('연비 분포(수직 박스 플롯)')
plt.show()
```

요약통계량의 대략적인 값과 좌편향된 형태라는걸 대략적으로 파악 가능!

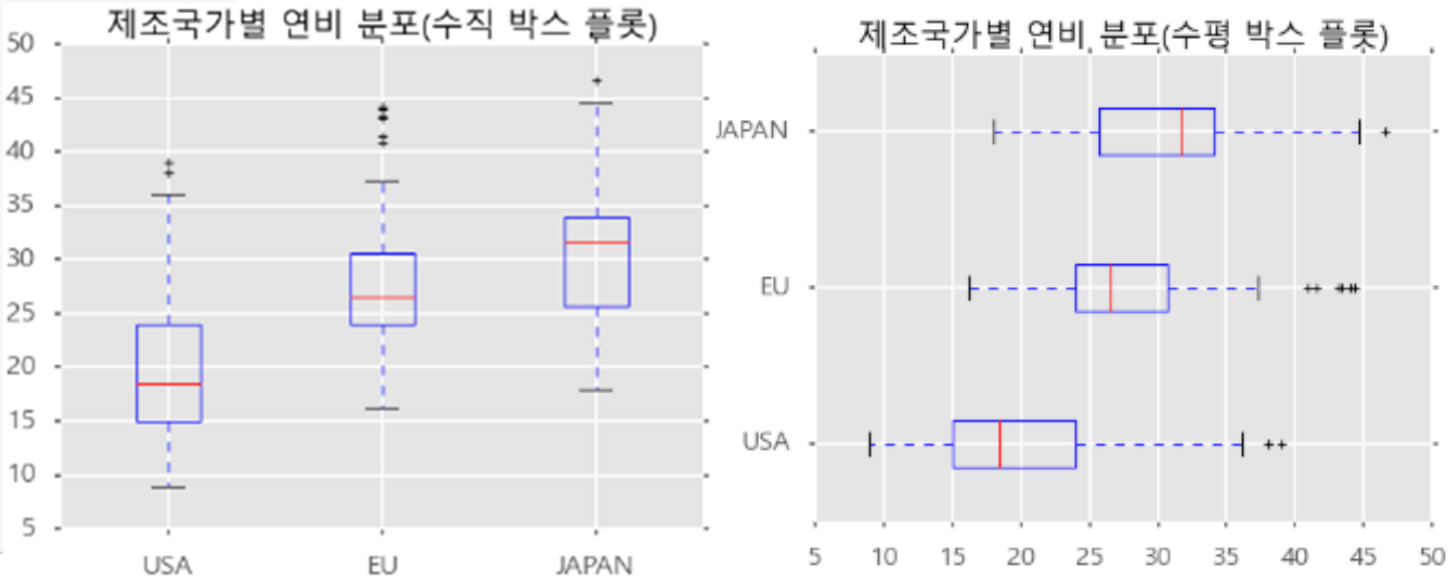


03. 범주형 & 연속형 변수 – 박스플롯(boxplot)

* plt.boxplot(x, sym=None, vert=True, labels=None, showmeans=False, …)

```
##### 한글 폰트 내용 불러와주세요!

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

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

# axe 객체에 boxplot 메서드로 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()
```

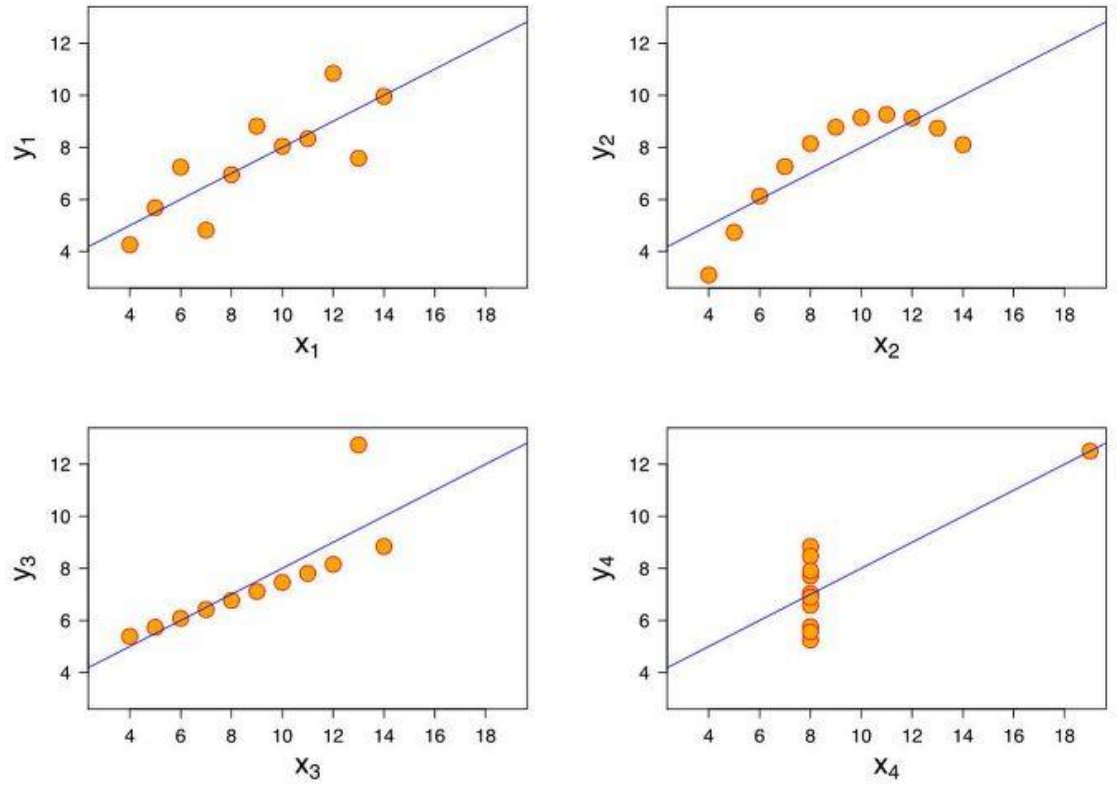
- 두 개 이상의 데이터는 리스트 형태로!
- vert = False : x와 y축 변경
- USA, EU, JAPAN 순으로 연비가 더 높음을 파악



03. 연속형 & 연속형 변수 – 산점도(scatter plot)

* 산점도(scatter plot)란?

- 주로 서로 다른 연속형 변수의 관계 표현
- 변수는 정수형(int64)이나 실수형(float64)
- 데이터 값의 좌표를 점으로 표시
- 변수들의 관계(선형,비선형 등등), 패턴의 유무 파악이 목적
- 선 그래프 학습에서 `o` 옵션을 사용한 그래프가 산점도 그래프 중 하나이다



03. 연속형 & 연속형 변수 – 산점도(scatter plot)

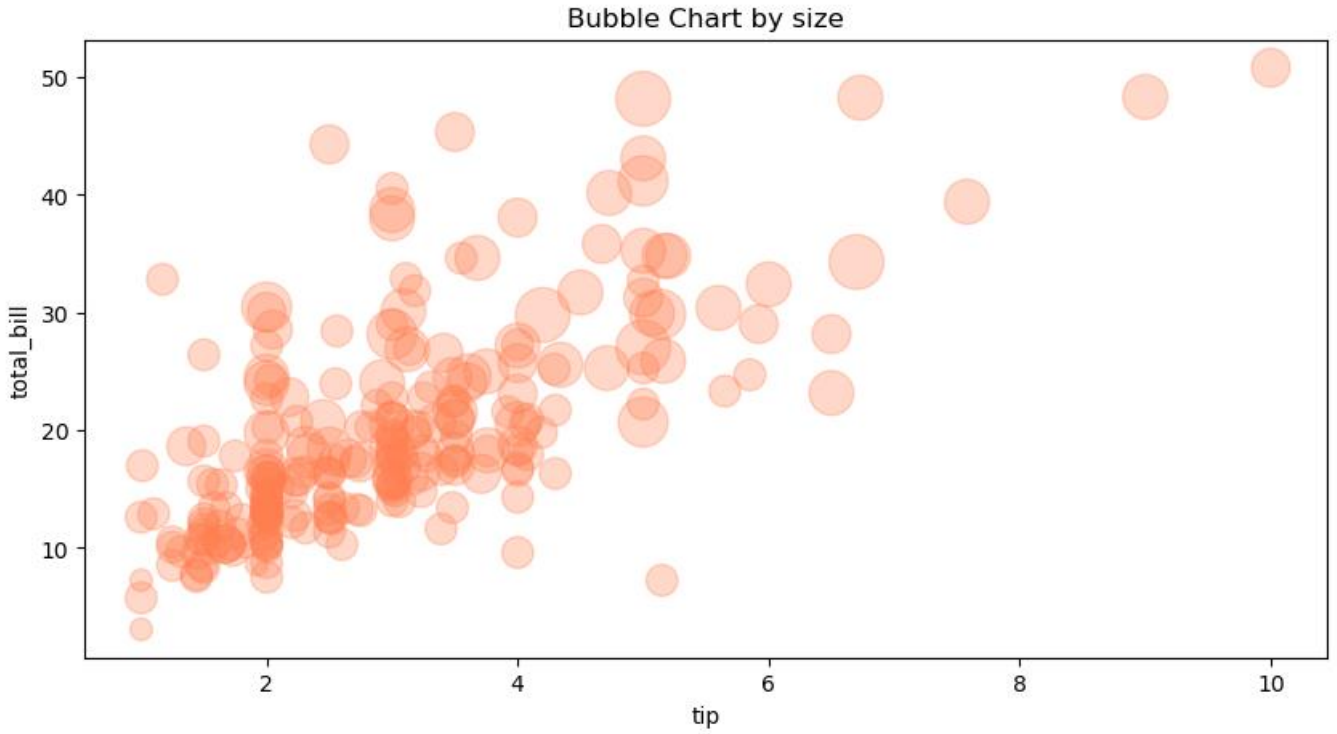
* df.plot(kind = `scatter', x, y, c=None, s=None, …)

```
plt.style.use('default')

df.plot(kind = 'scatter',x='weight',y='mpg',c='coral',s=10, figsize = (10,5))
plt.title('Scatter plot - mpg vs weight')
plt.show()
```

- kind = `scatter' 지정
- x, y : x축, y축에 들어갈 변수명 지정
- c : color 설정
- s : 점의 size 설정
- 점들의 분포가 우하향하는 경향
  → 음의 상관관계를 가진다

* 번외 - 버블차트

```
import seaborn as sns
tip = sns.load_dataset('tips') # tips 데이터 준비

# 변수 size에 따라 크기 변화주기
tip.plot(kind='scatter', x='tip', y='total_bill', c='coral', figsize=(10, 5),
         s = tip['size']*100, alpha = 0.3)

plt.title('Bubble Chart by size')
plt.show()
```



점의 투명도 설정

값의 크기에 따라 사이즈 다르게 표시 가능

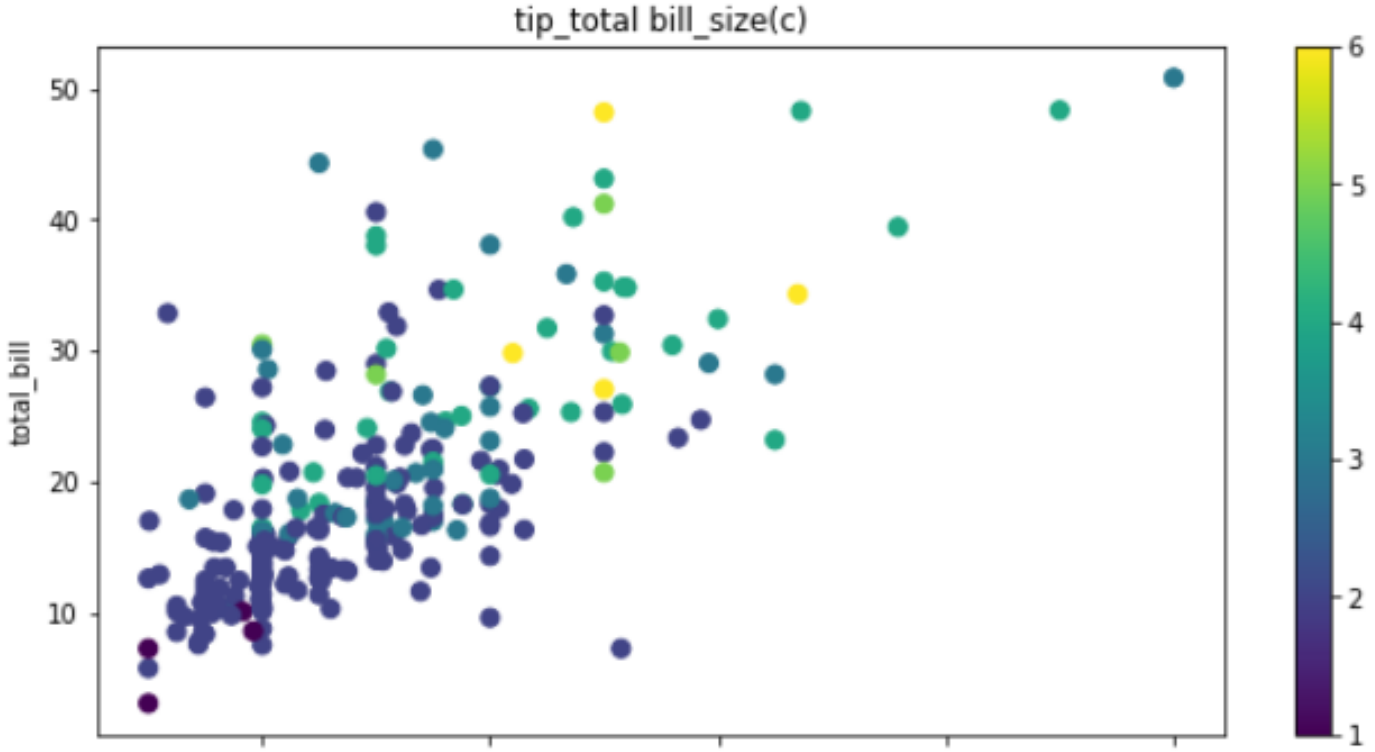
- 점의 크기에 변화를 준 모양이 비눗방울 같다하여 버블차트로도 부른다
- 데이터의 흐름 파악하는 시스템 분석에 주로 사용
- 구조적 순차적 또는 절차상의 관계 보여줌

* 번외 – 값에 따라 색으로 표현하기

```
# 값에 따라 다른 색상으로 표현
tip.plot(kind='scatter', x='tip', y='total_bill', figsize=(10, 5),
         c=tip['size'],cmap='viridis', s = 50)
plt.show()
```

colormap을 의미

값에 따라 색깔 다르게
표시 가능

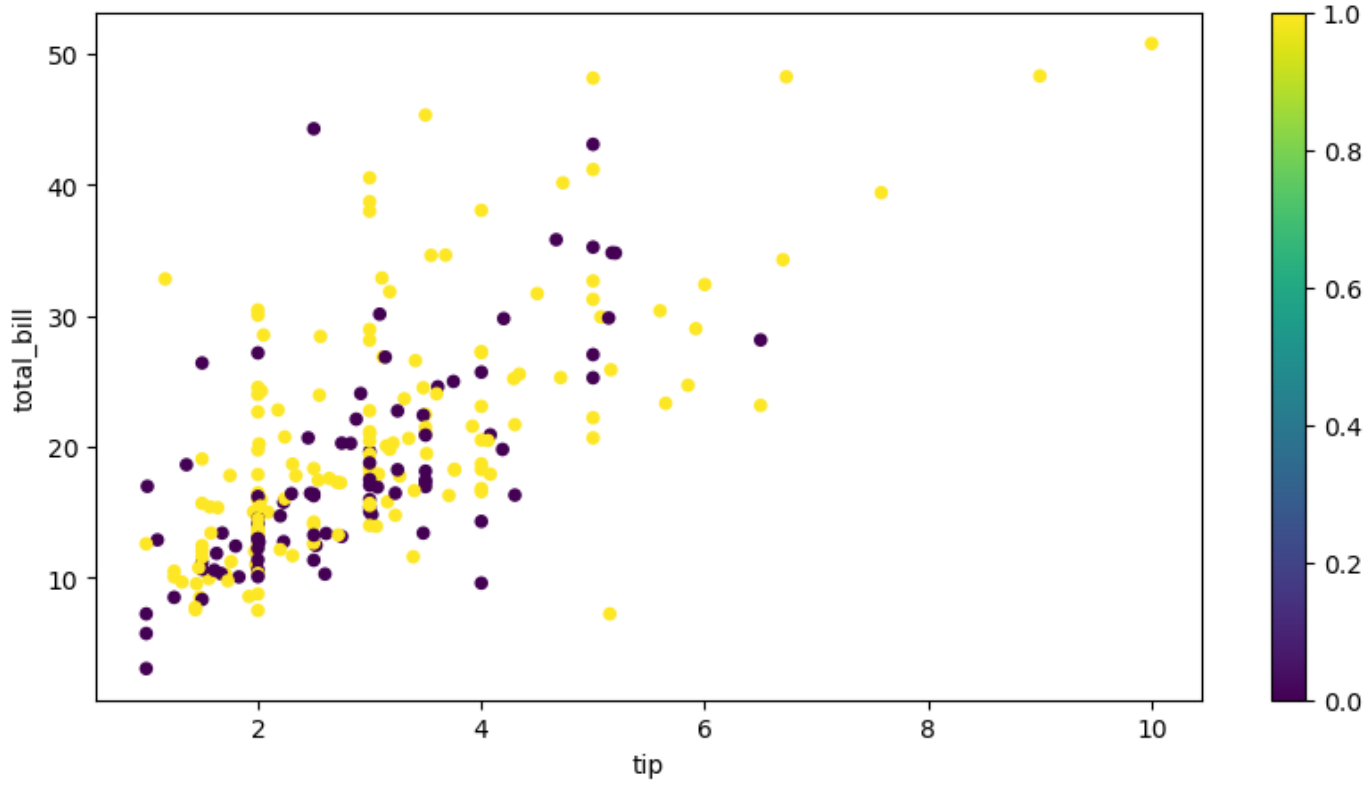


Tip의 변수 중 범주형 변수인 `sex'를 색깔별로 표현하고 싶다면 어떻게 해야할까?

* 번외 – 값에 따라 색으로 표현하기

```
# 범주를 숫자로 바꿔주는 전처리 필요
def recode_sex(sex):
    if sex=='Female':
        return 0
    else:
        return 1
sex_color = tip['sex'].apply(recode_sex)
```

```
tip.plot(kind='scatter', x='tip', y='total_bill', figsize=(10, 5),
         c=sex_color, colormap='viridis')

plt.show()
```



- 복잡한 코드는 아니지만 별도의 전처리가 필요하다
- 다소 불필요하고 비효율적인 방식

* 번외 – 값에 따라 색으로 표현하기

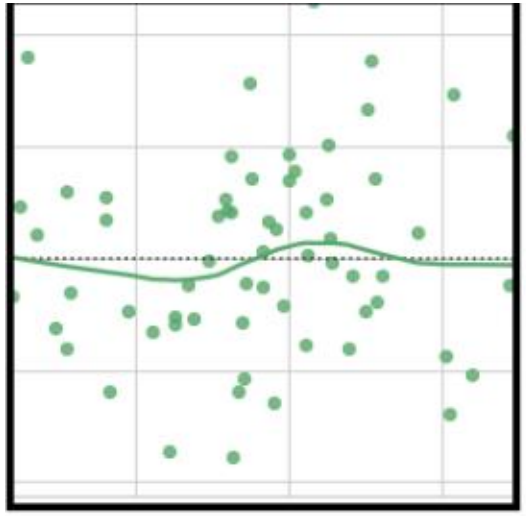
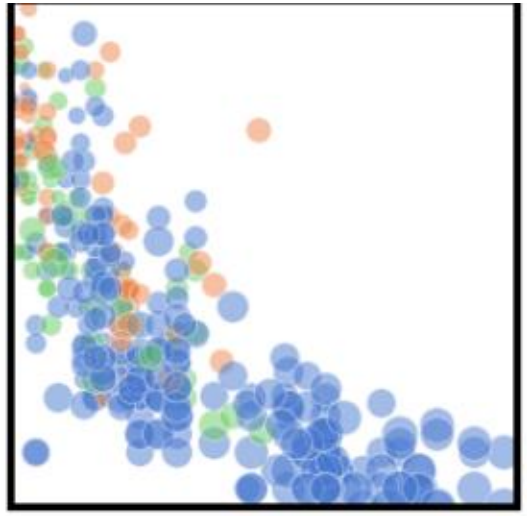
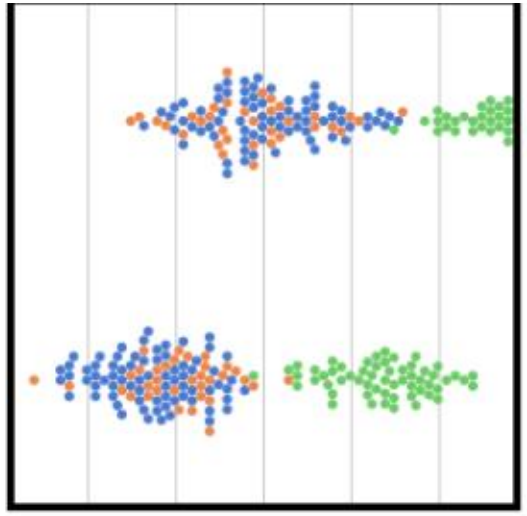
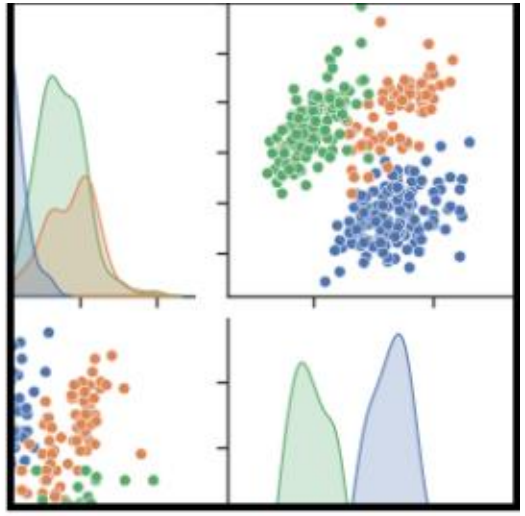
```
sns.set_style('darkgrid')

sns.lmplot(x='tip', y='total_bill',data=tip,hue='sex',aspect = 1.5,fit_reg=False)
plt.show()
```

- seaborn 라이브러리의 lmplot() 함수 사용
- 앞의 예시와는 다르게 간단히 코드 작성이 끝남
- seaborn 라이브러리는 뒤에서 자세히 학습

04. Seaborn 라이브러리

* seaborn은 무엇인가?

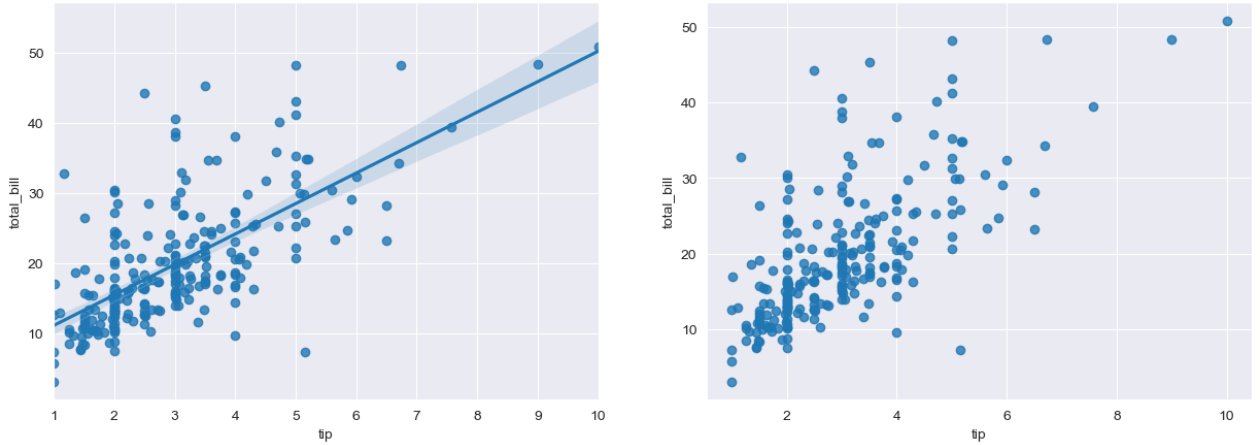
- Matplotlib의 기능과 스타일을 확장한 파이썬 시각화 도구의 고급버전
- 단순한 인터페이스 덕분에 사용하기 간편

04. Seaborn 라이브러리 – 회귀선이 있는 산점도

* sns.regplot(x, y, data=None, ax=None, fit_reg=False, ci=95, color=None, marker=None …)

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

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

# 그래프 그리기 - 선형회귀선 표시(fit_reg=True)
sns.regplot(x='tip',           #x축 변수
            y='total_bill',    #y축 변수
            data=tip,          #데이터
            ax=ax1)            #axe 객체 - 1번째 그래프

# 그래프 그리기 - 선형회귀선 표시(fit_reg=True)
sns.regplot(x='tip',           #x축 변수
            y='total_bill',    #y축 변수
            data=tip,          #데이터
            ax=ax2,            #axe 객체 - 1번째 그래프
            fit_reg=False)     #회귀선 미표시

plt.show()
```

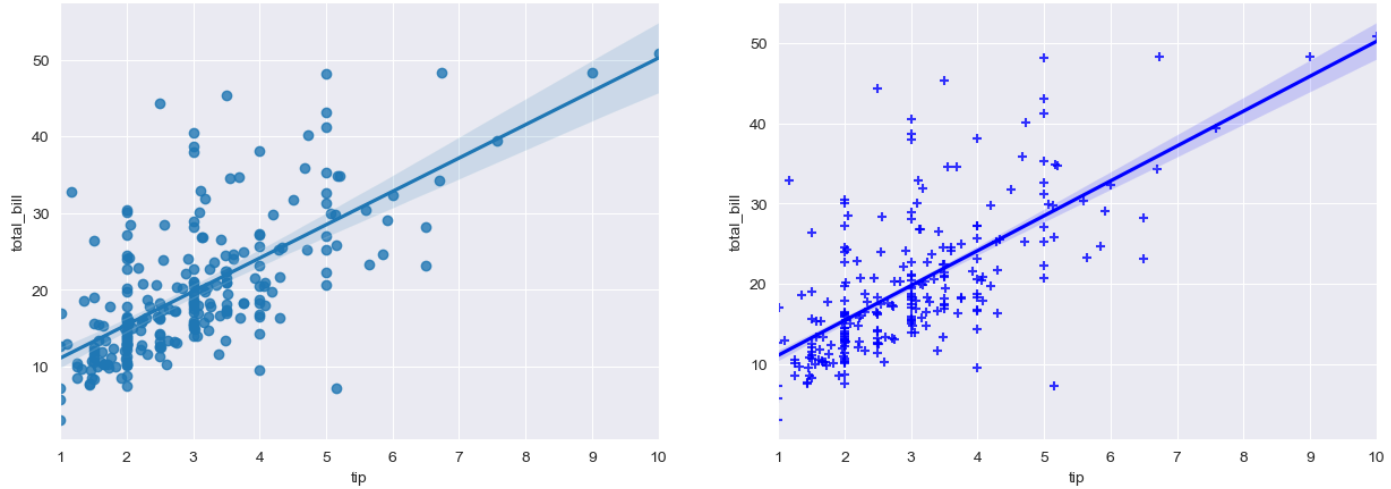


- 산점도 및 선형회귀분석에 의한 회귀선 함께 보여준다
- x, y : x축, y축에 들어갈 변수명 문자열로 지정(data 지정 안한 경우 데이터 프레임 or 시리즈로 지정)
- data = 데이터프레임 : 데이터 지정
- ax = axe객체 선택(pyplot과 달리 내부에서 선택 가능)
- fit_reg=False : 회귀선 제거한 산점도 그릴 수 있다

04. Seaborn 라이브러리 – 회귀선이 있는 산점도

* sns.regplot(x, y, data=None, ax=None, fit_reg=False, ci=95, color=None, marker=None …)

```
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

# 그래프 그리기 - 신뢰구간 95% 선형회귀선
sns.regplot(x='tip', y='total_bill',
            data=tip, ax=ax1,
            ci = 95)                    # 회귀선의 95% 신뢰구간

# 그래프 그리기 - 신뢰구간 70% 선형회귀선
sns.regplot(x='tip', y='total_bill',
            data=tip, ax=ax2,
            ci = 70,                    # 회귀선의 70% 신뢰구간
            color='blue',marker='+')    # 색,점 변경

plt.show()
```



- ci=95(default) : 신뢰구간 설정
  → 신뢰구간70%로 설정하면 그림의 신뢰구간 폭이 감소

- pyplot 그래프와 비슷하게 color, marker 옵션 통해 색, 점의 모양 변경 가능

04. Seaborn 라이브러리

* 라이브러리 및 데이터 준비

```
# 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())
```

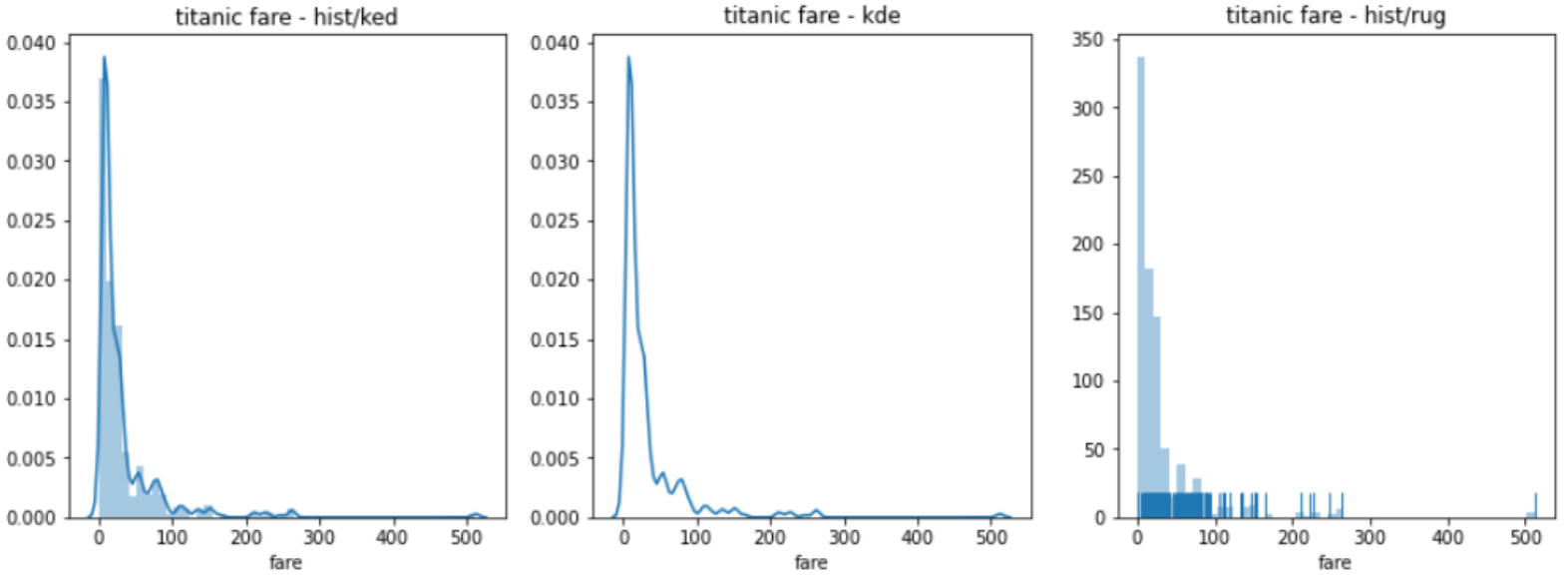
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.6+ KB
None
```

04. Seaborn 라이브러리 - 히스토그램/커널밀도그래프

* sns.distplot(a, bins=None, hist=True, kde=True, rug=False,…)
  - 단변수 데이터의 분포 확인 → 히스토그램과 커널밀도추정 그래프를 출력

```
# 그래프 객체 생성 (figure에 3개의 서브 플롯을 생성)
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(1, 3, 1)
ax2 = fig.add_subplot(1, 3, 2)
ax3 = fig.add_subplot(1, 3, 3)

# 기본값
sns.distplot(titanic['fare'], ax=ax1)
# hist=False
sns.distplot(titanic['fare'], hist=False, ax=ax2)
# kde=False, rug=True
sns.distplot(titanic['fare'],
             kde=False, rug=True, ax=ax3)

# 차트 제목 표시
ax1.set_title('titanic fare - hist/ked')
ax2.set_title('titanic fare - kde')
ax3.set_title('titanic fare - hist/rug')

plt.show()
```



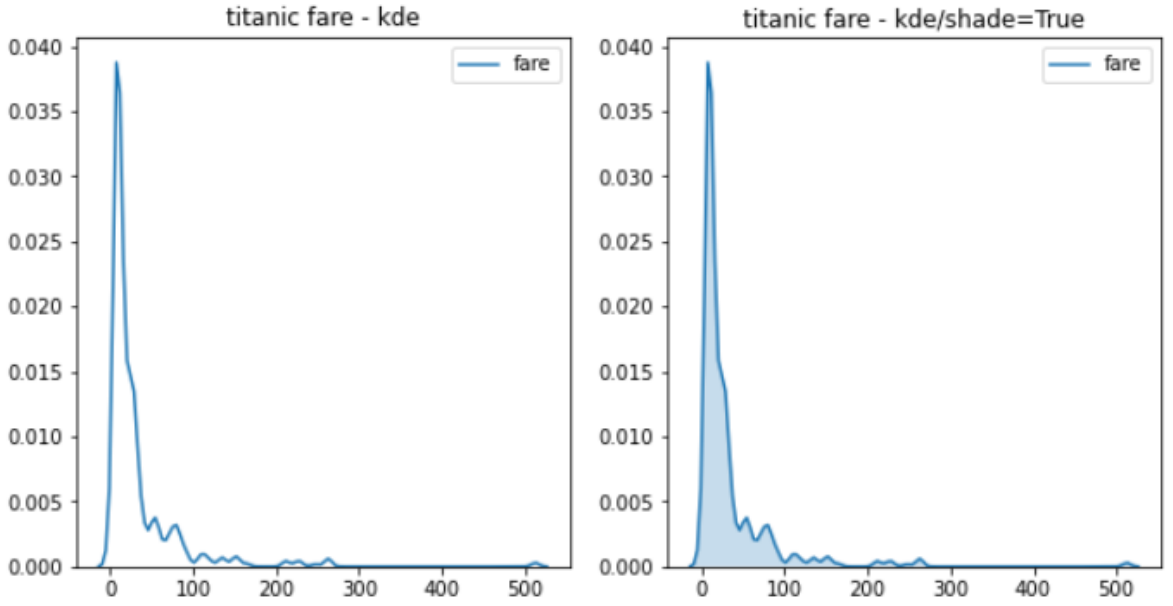
- a : 시리즈 or 데이터프레임 형태로 변수 지정
- hist = False : 히스토그램 제거
- kde = False : 커널밀도그래프 제거
- rug = True : 데이터의 실제 위치를 x축 위에 표시

04. Seaborn 라이브러리 - 히스토그램/커널밀도그래프

* sns.kdeplot(x, shade=False, …)

- 커널 밀도 추정 그래프
  - 데이터를 반영하는 확률밀도함수를 추정한 그래프
  - x축과 그래프선 사이의 면적은 1
  - 히스토그램을 smoothing한 그래프라고 볼 수도 있음

```
fig = plt.figure(figsize=(10, 5))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

# 기본값
sns.kdeplot(titanic['fare'], ax=ax1)
# shade=True
sns.kdeplot(titanic['fare'],shade =True,ax=ax2)

# 차트 제목 표시
ax1.set_title('titanic fare - kde')
ax2.set_title('titanic fare - kde/shade=True')

plt.show()
```



04. Seaborn 라이브러리 - 히트맵(heatmap)

* Heatmap(히트맵)이란?

- 두 개의 범주형 변수 비교, 한눈에 파악할 때 주로 사용
- x, y축에 각각의 변수 놓고 매트릭스 형태로 분류 후 값에 따라 색상으로 표현
- x축, y축, 그래프 내용에 어떤 변수를 쓸 것인지가 중요

* 테이블표 만들기(pivot_table() 이용)

```
# 피벗테이블로 범주형 변수를 각각 행, 열로 재구분하여 정리
table = titanic.pivot_table(index=['sex'], columns=['class'], aggfunc='size')
print(table)
```

```
class   First  Second  Third
sex
female     94      76    144
male      122     108    347
```

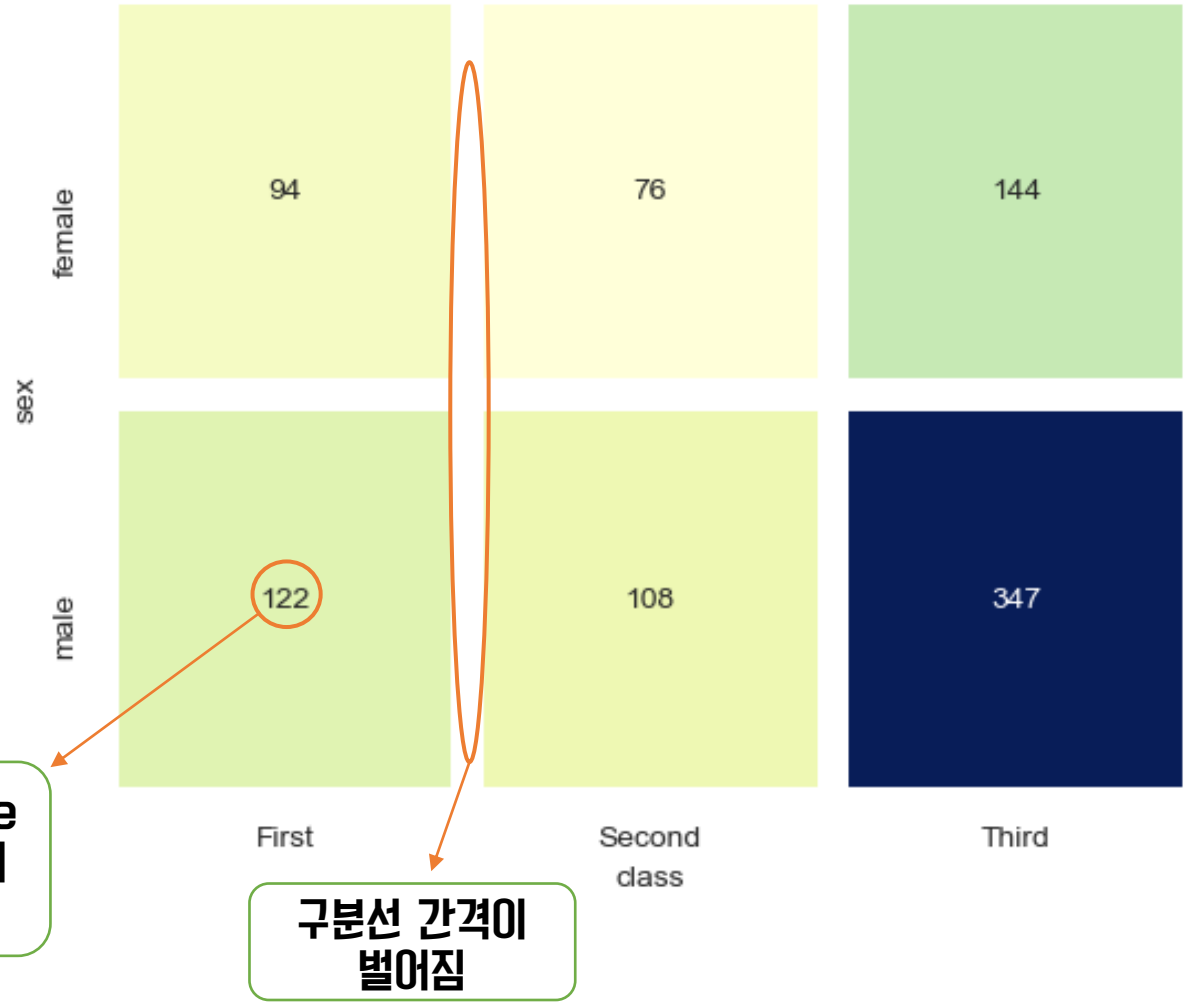
- index, columns 옵션에 변수명 지정
- aggfunc=`size' : 데이터 값의 크기를 기준으로 집계

04. Seaborn 라이브러리 - 히트맵(heatmap)

* sns.heatmap(data, annot=None, cmap=None, linewidth=0, cbar=True…)

```
# 히트맵 그리기
sns.heatmap(table,                  # 데이터프레임
            annot=True, fmt='d',    # 데이터 값 표시 여부, 정수형 포맷
            cmap='YlGnBu',          # 컬러 맵
            linewidth=10,           # 구분 선
            cbar=False)             # 컬러 바 표시 여부

plt.show()
```

- data : 앞서 만든 table표
- annot=True : 데이터값 표시
- cmap : 원하는 컬러맵 지정
- linewidth : 구분선 사이의 간격
- cbar = False : 컬러바 제거



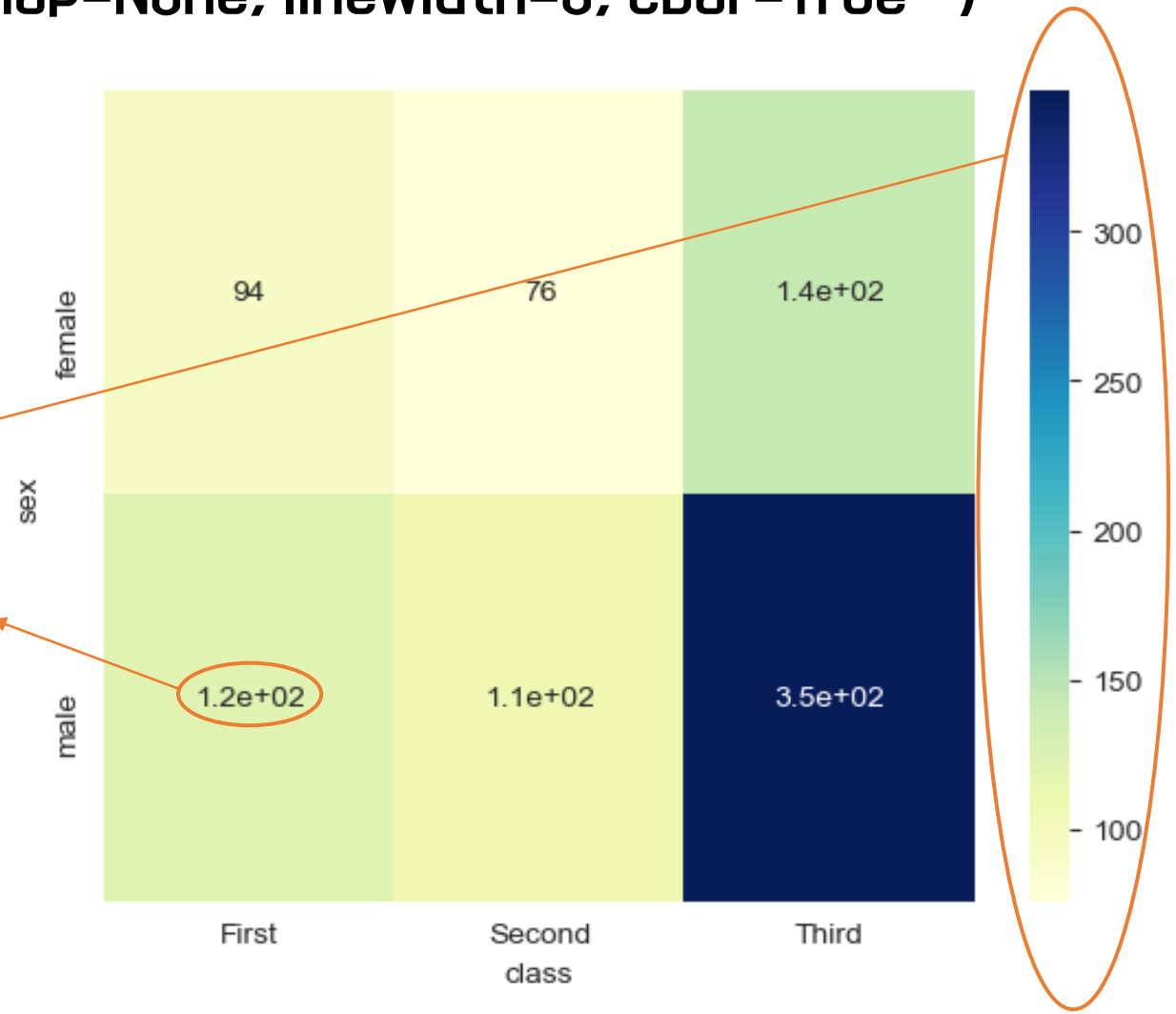
04. Seaborn 라이브러리 - 히트맵(heatmap)

* sns.heatmap(data, annot=None, cmap=None, linewidth=0, cbar=True…)

```
# 히트맵 그리기
sns.heatmap(table,                  # 데이터프레임
            annot=True,             # 데이터 값 표시, 포멧지정x
            cmap='YlGnBu',          # 컬러 맵
            linewidth=0,            # 구분 선 간격x
            cbar=True)              # 컬러 바 생성

plt.show()
```

컬러바 생성

포멧 지정 안하니까 지수 표기법으로 숫자 표시됨



04. Seaborn 라이브러리 - 히트맵(heatmap)

* 번외 - 히트맵 활용(전출입 데이터를 활용)

```
col_years = list(map(str, range(1980, 2000)))
df_5 = df_seoul.loc[['충청남도','경상북도', '강원도', '전라남도'], col_years]
df_5_T = df_5.T
df_5_T.index.names = ['year']
```

앞서 살펴본 전출입 데이터 중 변화가 가장 두드러지는 80~99년도의 데이터

```
print(df_5_T.dtypes)
print('₩n')
df_5_T = df_5_T.astype('int64')
print(df_5_T.dtypes)
print('₩n')
```

```
전입지
충청남도    object
경상북도    object
강원도     object
전라남도    object
dtype: object

전입지
충청남도    int64
경상북도    int64
강원도     int64
전라남도    int64
dtype: object
```

데이터 타입이 문자로 되어 있는 걸 확인 → 정수형으로 변경

```
print(df_5_T.columns)
print('₩n')
df_5_T.columns = df_5_T.columns.astype('category')
print(df_5_T.columns)
print('₩n')
```

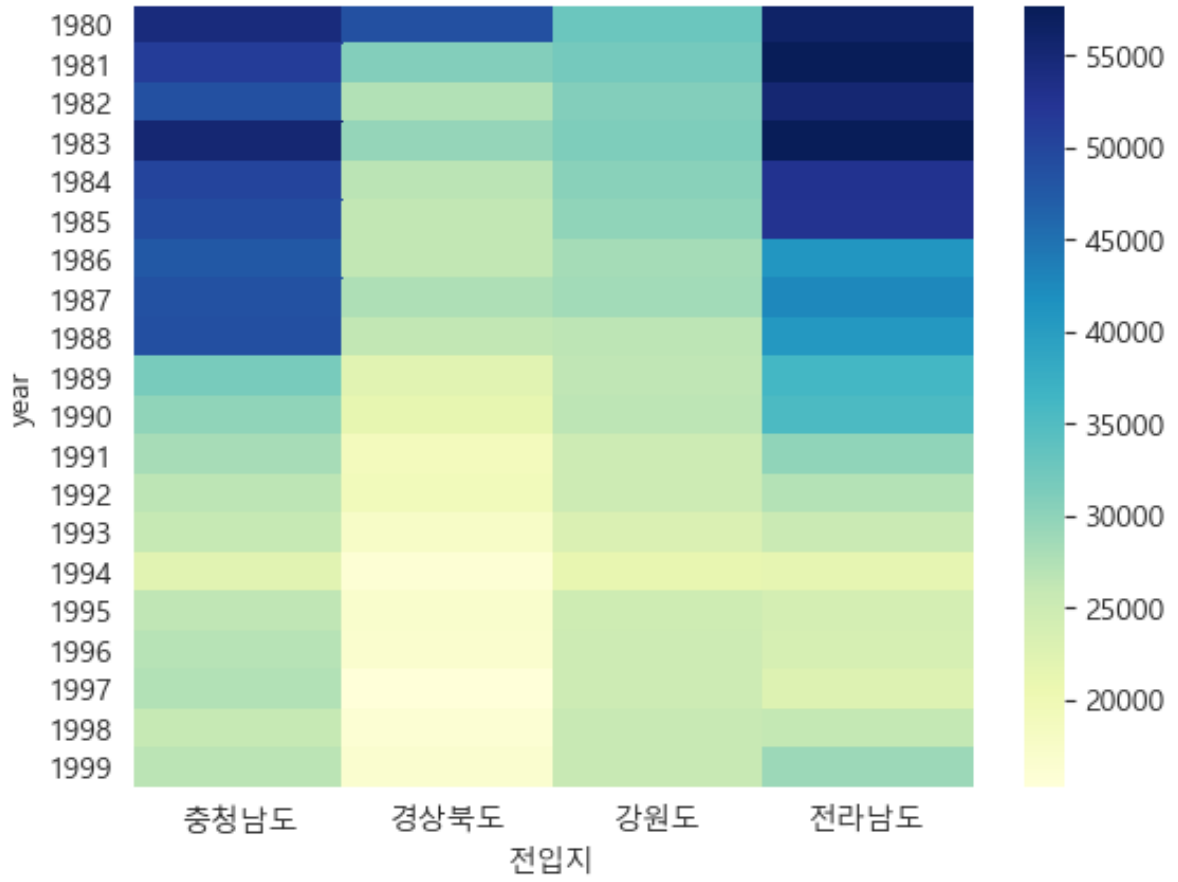
```
Index(['충청남도', '경상북도', '강원도', '전라남도'], dtype='object', name='전입지')


CategoricalIndex(['충청남도', '경상북도', '강원도', '전라남도'], categories=['강원도', '경상북도', '전라남도', '충청남도'], ordered=Fal
se, name='전입지', dtype='category')
```

columns의 type 또한 category로 변경 필요하다

04. Seaborn 라이브러리 - 히트맵(heatmap)

* 번외 - 히트맵 활용(전출입 데이터를 활용)

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

# 히트맵 그리기
sns.heatmap(df_5_T,                 # 데이터프레임
            cmap='YlGnBu',          # 컬러 맵
            linewidth=0,            # 구분 선
            cbar=True)              # 컬러 바 표시 여부
plt.show()
```



- 충청남도의 경우 88년,89년도 사이 급격히 하락하는 것을 확인 할 수 있음
- 강원도와 경상북도는 전입 인원이 약 30000에서 서서히 감소
- 전라남도 역시 전입입구가 빠르게 줄었지만 충청남도에 비해 비교적 완만한 경사를 보일 것으로 짐작 가능

04. Seaborn 라이브러리 – 히트맵(heatmap)

* 번외 – 히트맵 활용(상관계수 그래프)

```
correlation = titanic.corr()
correlation
```

- 상관계수 계산 위해 pandas의 corr() 함수 사용
- category, object 타입은 자동으로 제외해줌

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
survived	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307	-0.557080	-0.203367
pclass	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500	0.094035	0.135207
age	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067	0.280328	0.198270
sibsp	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651	-0.253586	-0.584471
parch	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225	-0.349943	-0.583398
fare	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000	-0.182024	-0.271832
adult_male	-0.557080	0.094035	0.280328	-0.253586	-0.349943	-0.182024	1.000000	0.404744
alone	-0.203367	0.135207	0.198270	-0.584471	-0.583398	-0.271832	0.404744	1.000000

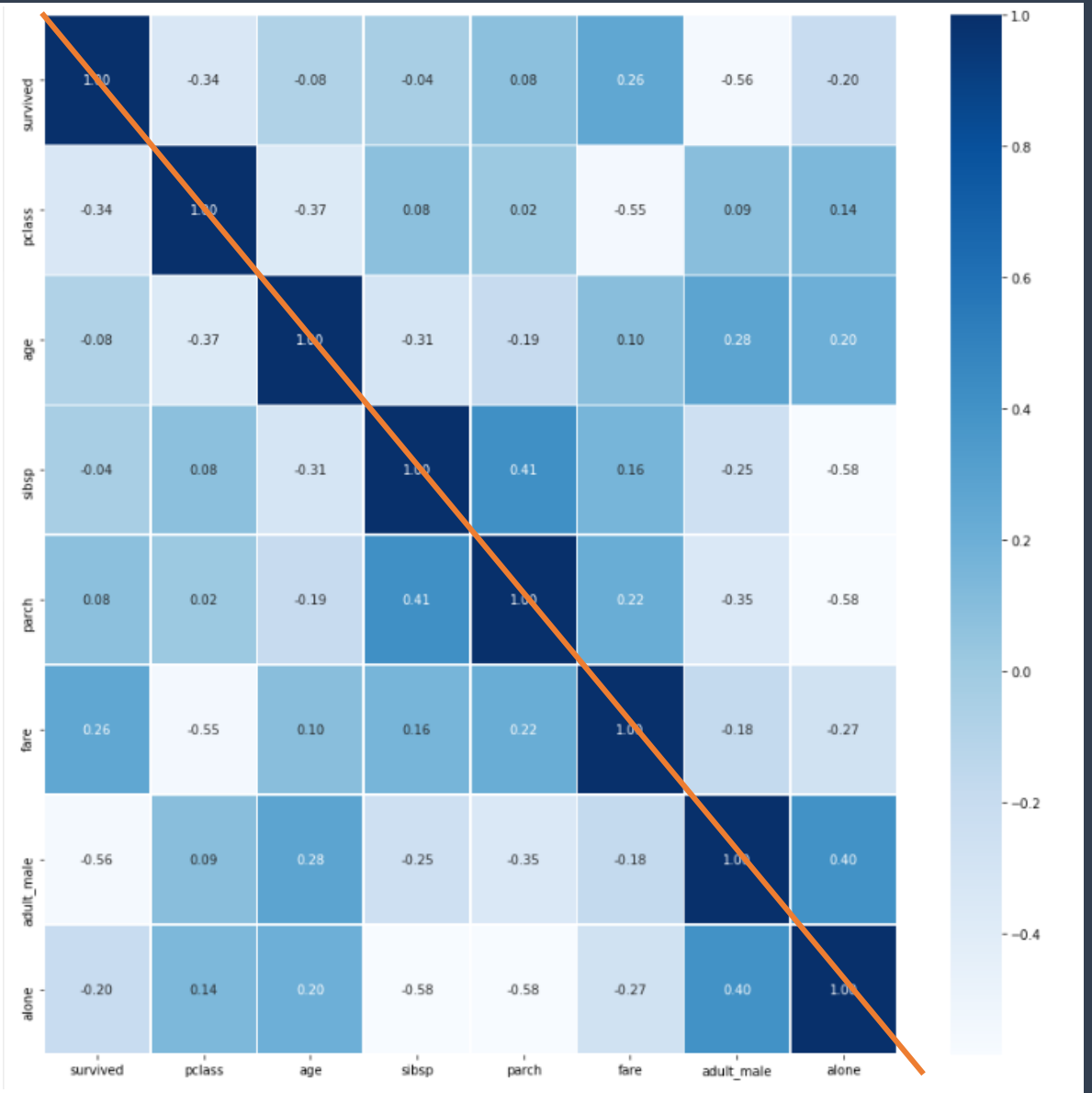
- 수치형(주로 연속형) 변수 간의 상관 관계를 파악
- 상관계수는 인과관계를 설명해주지 못함
- 상관계수가 0 이라는 것이 두 변수가 독립임을 의미하지 않음

04. Seaborn 라이브러리 – 히트맵(heatmap)

* 번외 – 히트맵 활용(상관계수 그래프)

```
plt.figure(figsize=(15,15))
sns.heatmap(data = correlation, annot=True,
            fmt = '.2f', linewidths=.5, cmap='Blues')
plt.show()
```

- 변수들 간의 상관관계를 그래프를 통해 한눈에 파악
- 대각선 열은 항상 1, 대각선을 기준으로 중복되는 값들이 표시됨



04. Seaborn 라이브러리 – 히트맵(heatmap)

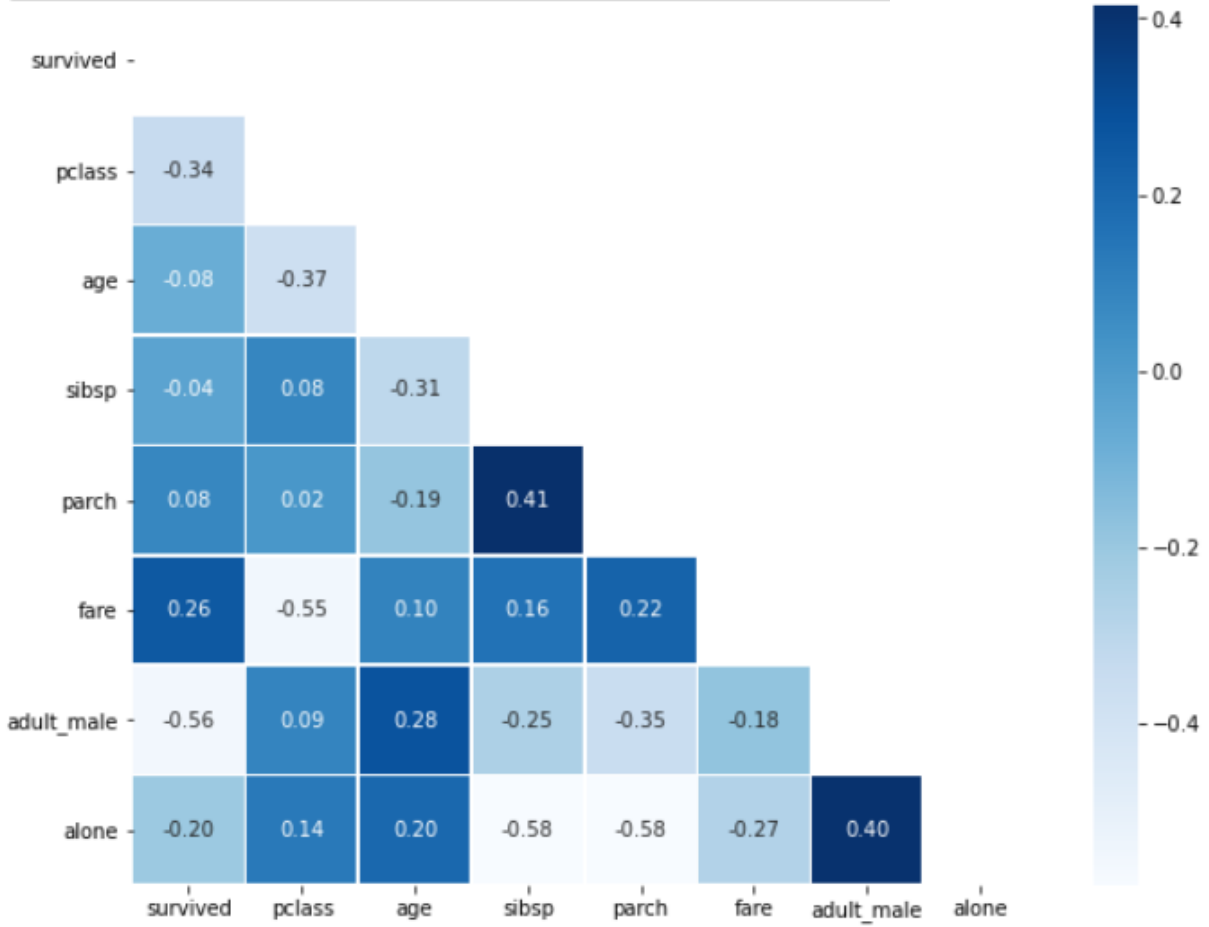
* 번외 – 히트맵 활용(상관계수 그래프)

```
import numpy as np
mask = np.zeros_like(correlation, dtype=np.bool)
print(mask)
print('₩n')
mask[np.triu_indices_from(mask)] = True
print(mask)
```

```
[[False False False False False False False False]
 [False False False False False False False False]
 [False False False False False False False False]
 [False False False False False False False False]
 [False False False False False False False False]
 [False False False False False False False False]
 [False False False False False False False False]
 [False False False False False False False False]]


[[ True  True  True  True  True  True  True  True]
 [False  True  True  True  True  True  True  True]
 [False False  True  True  True  True  True  True]
 [False False False  True  True  True  True  True]
 [False False False False  True  True  True  True]
 [False False False False False  True  True  True]
 [False False False False False False  True  True]
 [False False False False False False False  True]]
```

```
plt.figure(figsize=(10,8))
sns.heatmap(data = correlation, annot = True,
            fmt = '.2f',linewidths=.5,cmap = 'Blues',
            mask=mask)
plt.show()
```

mask 옵션 통해 표시할 부분 선택 가능



04. Seaborn 라이브러리 – 범주형 데이터의 산점도

* 범주형 또는 이산형 데이터의 산점도

- stripplot() 함수와 swarmplot() 함수 사용
- stripplot() : 범주형 변수의 특성상 포인트들이 겹쳐져서 표현됨
- swarmplot() : 데이터의 분산 고려, 포인트들이 중복되지 않음

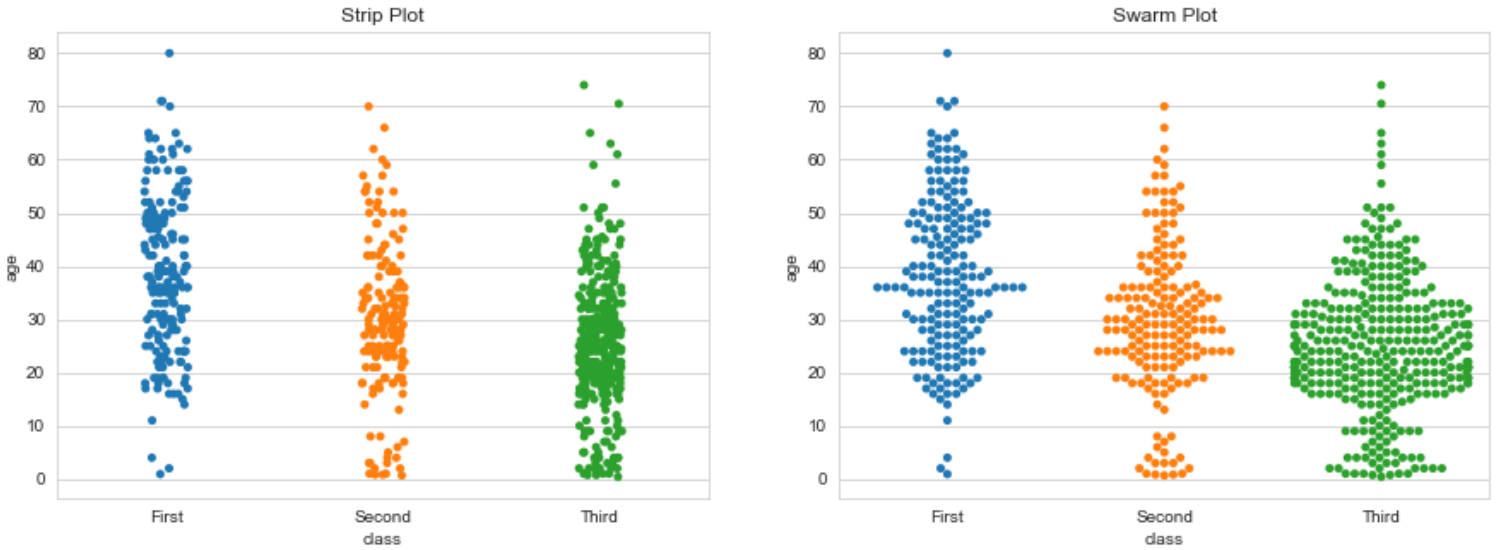
```
# 스타일 테마 설정
sns.set_style('whitegrid')

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

- 스타일 테마와 객체 생성(64p,65p 코드 앞에 넣고 같은 창에서 실행해주세요)

04. Seaborn 라이브러리 – 범주형 데이터의 산점도

* sns.stripplot(x, y, data, ax=None, hue=None, palette=None, jitter=True…)
* sns.swarmplot(x, y, data, ax=None, hue=None, palette=None, …)

```
# 이산형 변수의 분포 - 데이터 분산 미고려
sns.stripplot(x="class",      #x축 변수
              y="age",        #y축 변수
              data=titanic,   #데이터셋 - 데이터프레임
              ax=ax1)         #axe 객체 - 1번째 그래프

# 이산형 변수의 분포 - 데이터 분산 고려 (중복 X)
sns.swarmplot(x="class",      #x축 변수
              y="age",        #y축 변수
              data=titanic,   #데이터셋 - 데이터프레임
              ax=ax2)         #axe 객체 - 2번째 그래프

# 차트 제목 표시
ax1.set_title('Strip Plot')
ax2.set_title('Swarm Plot')
plt.show()
```



- strip plot은 점이 겹쳐져 있는 반면 swarm plot은 점이 옆으로 퍼져 있음
  → 분산 파악에 용이
- jitter 옵션을 제외하고는 옵션 동일
- x, y : x,y축 들어갈 변수명
- data : 데이터프레임

04. Seaborn 라이브러리 – 범주형 데이터의 산점도

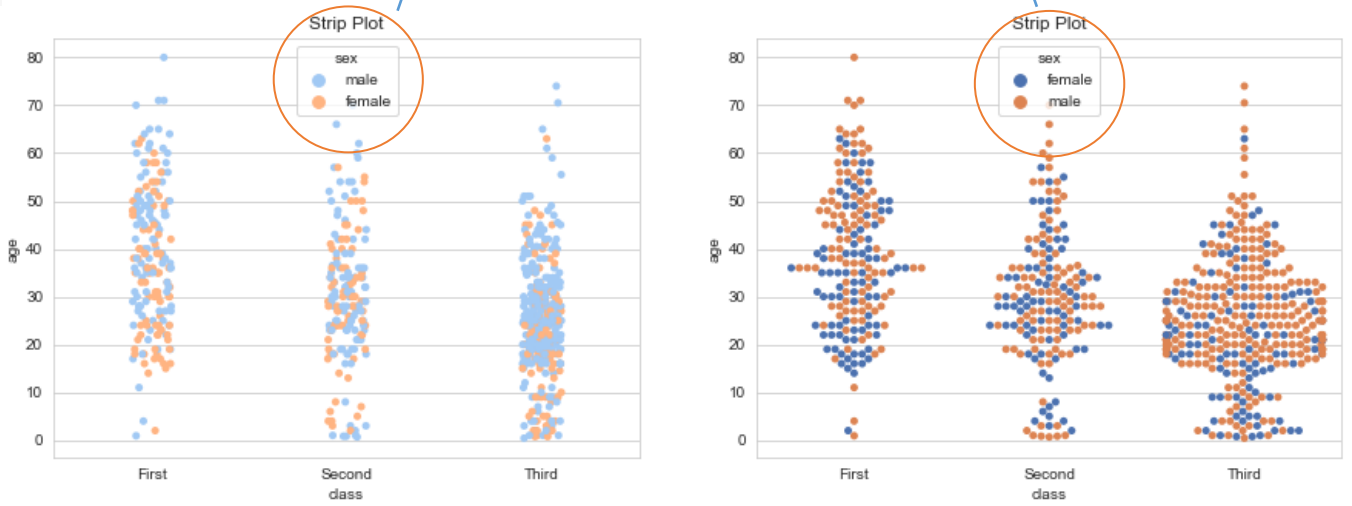
* sns.stripplot(x, y, data, ax=None, hue=None, palette=None,…)
* sns.swarmplot(x, y, data, ax=None, hue=None, palette=None, …)

```
# 이산형 변수의 분포 - 데이터 분산 미고려
sns.stripplot(x="class",      #x축 변수
              y="age",        #y축 변수
              data=titanic,   #데이터셋 - 데이터프레임
              ax=ax1,         #axe 객체 - 1번째 그래프
              hue='sex', palette='pastel')

# 이산형 변수의 분포 - 데이터 분산 고려 (중복 X)
sns.swarmplot(x="class",      #x축 변수
              y="age",        #y축 변수
              data=titanic,   #데이터셋 - 데이터프레임
              ax=ax2,         #axe 객체 - 2번째 그래프
              hue='sex', palette='deep',hue_order=['female','male'])

# 차트 제목 표시
ax1.set_title('Strip Plot')
ax2.set_title('Swarm Plot')
plt.show()
```

- hue : 3번째 변수명 입력(범주형)
  → 입력한 변수의 범주를 색으로 구분
- palette : hue에 입력된 변수를 구분하는 색 설정
- hue_order : 범례 표시순서

범례 표시 순서가 female, male 순으로 바뀜

`sex`의 두가지 범주 male,female을 파스텔 색으로 구분



04. Seaborn 라이브러리 - 막대그래프

* sns.barplot(x, y, data, ax=None, hue=None, palette=None, dodge=True…)
  - seaborn내의 막대 그래프를 그리는 함수
  - 데이터의 크고 작음, 양에 대한 비교가 쉬움
  - 이변량 데이터 뿐만 아니라 다변량 데이터 표현 또한 가능

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

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

# 스타일 테마 설정
sns.set_style('whitegrid')

# 그래프 객체 생성 (figure에 3개의 서브 플롯을 생성)
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(1, 3, 1)
ax2 = fig.add_subplot(1, 3, 2)
ax3 = fig.add_subplot(1, 3, 3)
```

- 스타일 테마와 객체 생성(67p,68p 코드 앞에 넣고 같은 창에서 실행해주세요)

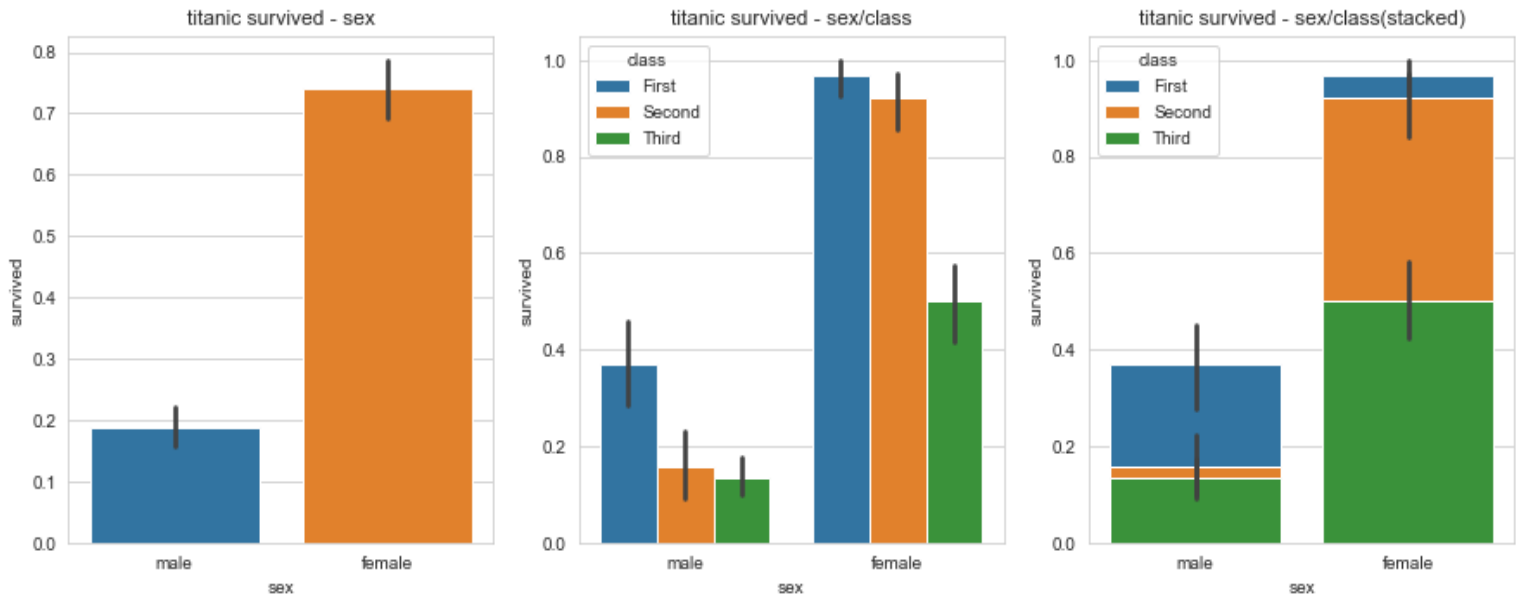
04. Seaborn 라이브러리 - 막대그래프

* sns.barplot(x, y, data, ax=None, hue=None, palette=None, dodge=True…)

```
# x축, y축에 변수 할당
sns.barplot(x='sex', y='survived', data=titanic, ax=ax1)

# x축, y축에 변수 할당하고 hue 옵션 추가
sns.barplot(x='sex', y='survived', hue='class', data=titanic, ax=ax2)

# x축, y축에 변수 할당하고 hue 옵션을 추가하여 누적 출력
sns.barplot(x='sex', y='survived', hue='class', dodge=False, data=titanic, ax=ax3)

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

plt.show()
```

- x, y, hue, palette에 대한 설명은 앞과 동일
- dodge=False : hue 옵션에 추가된 변수의 누적 그래프 출력(True는 분리된 그래프)

- 일반적으로 x는 범주형, y는 수치형 변수
- x 변수의 범주별로 y 변수의 평균을 표현
- 해당 그래프는 생존율로 해석 가능

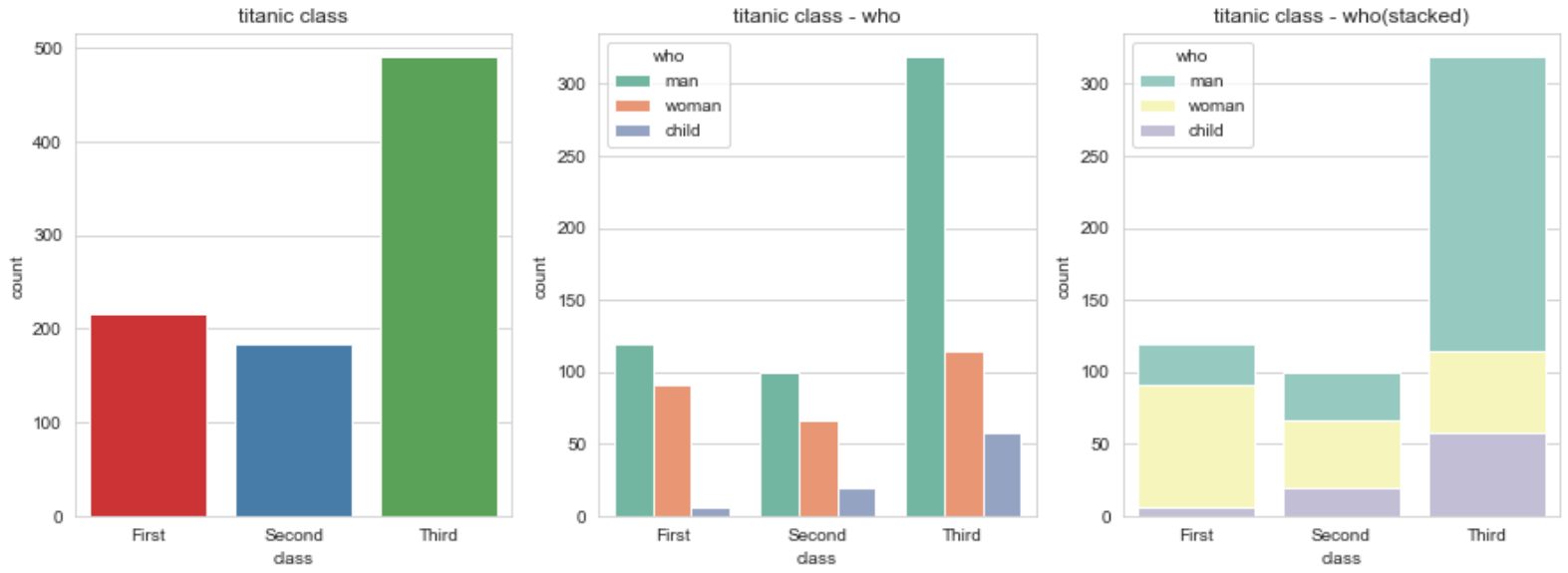


04. Seaborn 라이브러리 – 빈도그래프

* sns.countplot(x, y, data, ax=None, hue=None, palette=None, dodge=True…)

```
# 기본값
sns.countplot(x='class', palette='Set1', data=titanic, ax=ax1)

# hue 옵션에 'who' 추가
sns.countplot(x='class', hue='who', palette='Set2', data=titanic, ax=ax2)

# 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()
```

- 단변량 범주형 변수를 그릴 때 사용
- 막대그래프의 일부로 범주의 개수 파악에 용이
- 옵션은 barplot()과 유사



04. Seaborn 라이브러리

* 참고 – 팔레트의 종류(코드 안치셔도 됩니다)

Return one of the perceptually-uniform colormaps included in seaborn:

```
sns.color_palette("flare", as_cmap=True)
```

Return a customized cubehelix color palette:

```
sns.color_palette("ch:s=.25,rot=-.25", as_cmap=True)
```

Return a light-themed sequential colormap to a seed color:

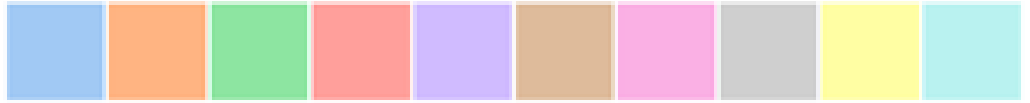
```
sns.color_palette("light:#5A9", as_cmap=True)
```

Calling with no arguments returns all colors from the current default color cycle:

```
sns.color_palette()
```

Other variants on the seaborn categorical color palette can be referenced by name:

```
sns.color_palette("pastel")
```



Return a specified number of evenly spaced hues in the "HUSL" system:

```
sns.color_palette("husl", 9)
```

Return all unique colors in a categorical Color Brewer palette:

```
sns.color_palette("Set2")
```

04. Seaborn 라이브러리 – 박스플롯/바이올린그래프

* sns.boxplot(x, y, data, hue=None, ax=None, palette=None…)
* sns.violinplot(x, y, data, hue=None, ax=None, palette=None…)

- 박스 플롯과 바이올린 플롯 그리는 함수
- 박스 플롯은 데이터의 분산 정확하게 알기 어려움
  → 이를 보완한 것이 바이올린 플롯
- 바이올린 플롯은 커널밀도 함수 그래프를 y축 방향으로 추가하여 그려줌
- 주로 x축은 범주형 변수, y축은 연속형 변수

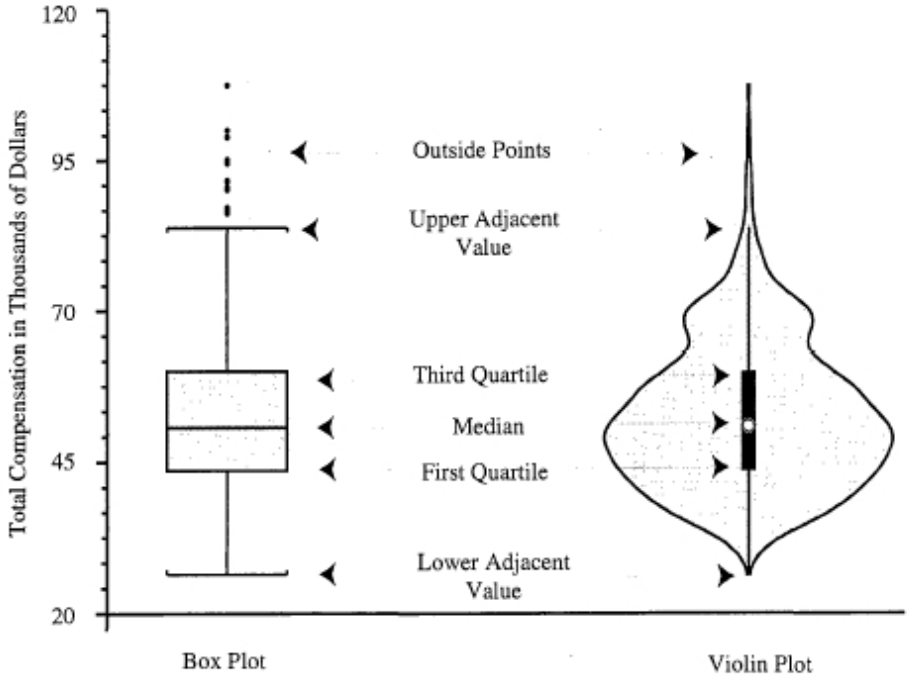


Figure 1. Common Components of Box Plot and Violin Plot. Total compensation for all academic ranks.

04. Seaborn 라이브러리 - 박스플롯/바이올린그래프

* sns.boxplot(x, y, data, hue=None, ax=None, palette=None…)
* sns.violinplot(x, y, data, hue=None, ax=None, palette=None, split=False…)

```
# 그래프 객체 생성 (figure에 4개의 서브 플롯을 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 그래프 - 기본값
sns.boxplot(x='alive', y='age', data=titanic, ax=ax1)

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

# 바이올린 그래프 - 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()
```

alive 변수 범주 no, yes의 age 박스 플롯 그려줌

sex 변수의 범주 male, female 색으로 구분

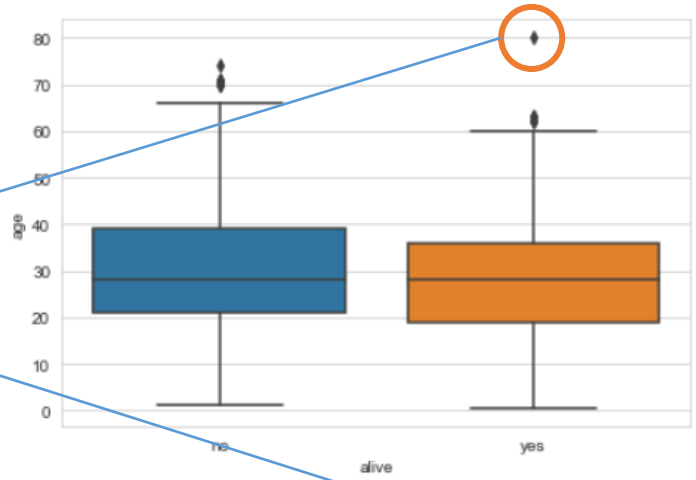
alive 변수 범주 no, yes의 age 바이올린 플롯 그려줌

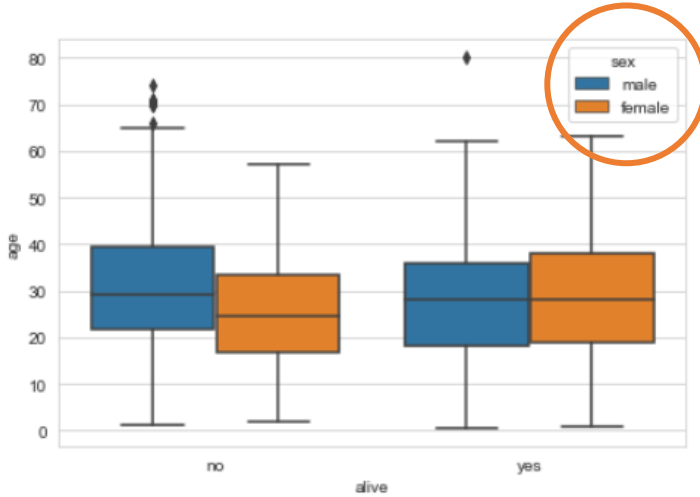
sex 변수의 범주 male, female 색으로 구분

04. Seaborn 라이브러리 – 박스플롯/바이올린그래프

* sns.boxplot(x, y, data, hue=None, ax=None, palette=None…)
* sns.violinplot(x, y, data, hue=None, ax=None, palette=None…)

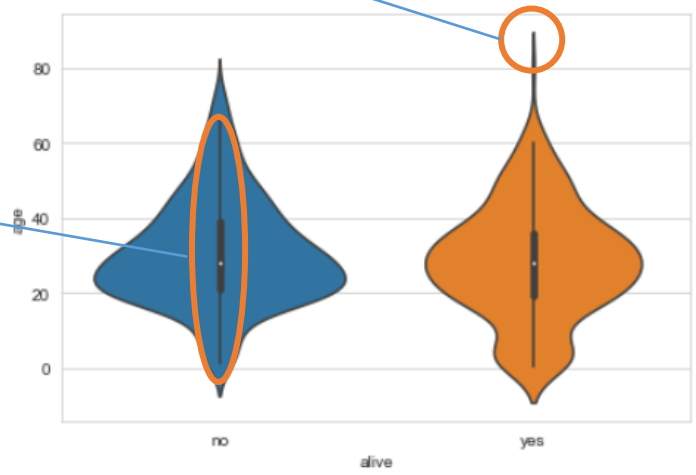
바이올린 플롯은 이상치 따로 표시 하지 않음

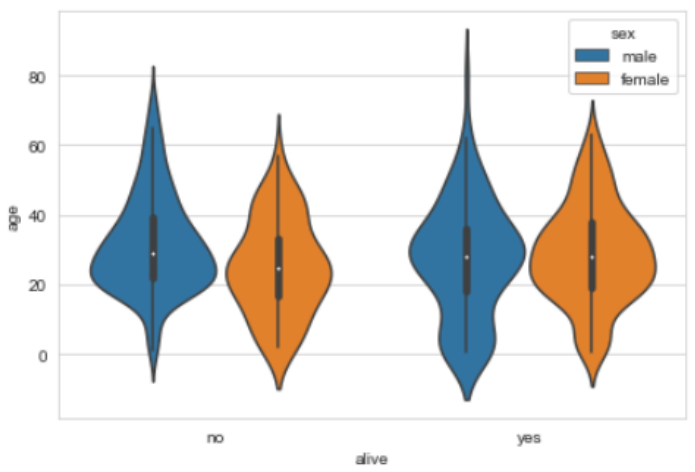




sex 변수의 범주 male, female 색으로 구분

이상치 제외한 박스 플롯이 가진 정보 모두 포함





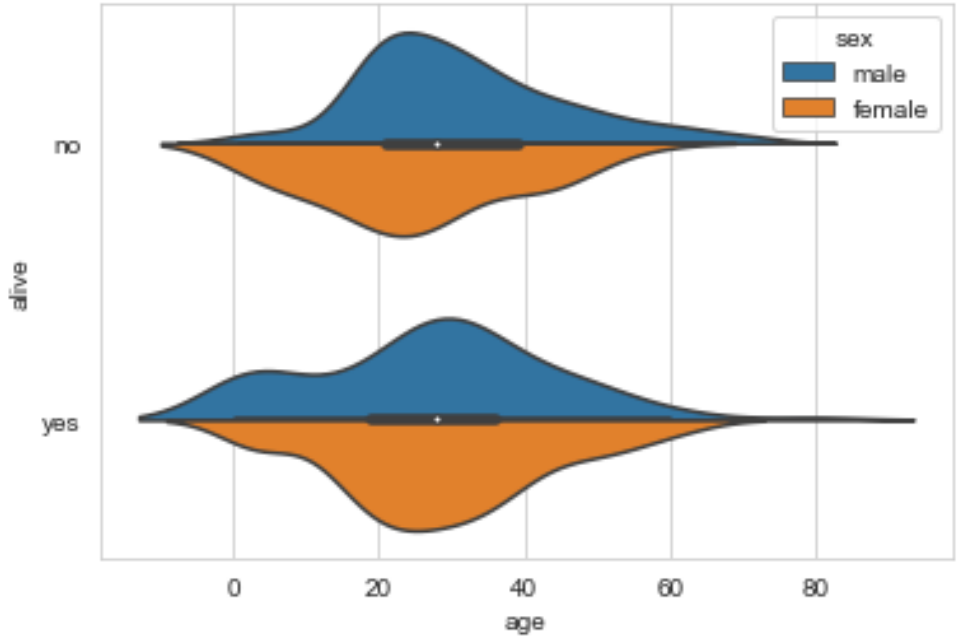
04. Seaborn 라이브러리 - 박스플롯/바이올린그래프

* sns.violinplot(x, y, data, hue=None, ax=None, palette=None, split=False…)

```
sns.violinplot(x='age', y='alive', hue='sex', data=titanic, split=True)

plt.show()
```

- x에 범주형 변수, y에 연속형 변수 입력
  → 가로로 누운 바이올린 플롯 생성

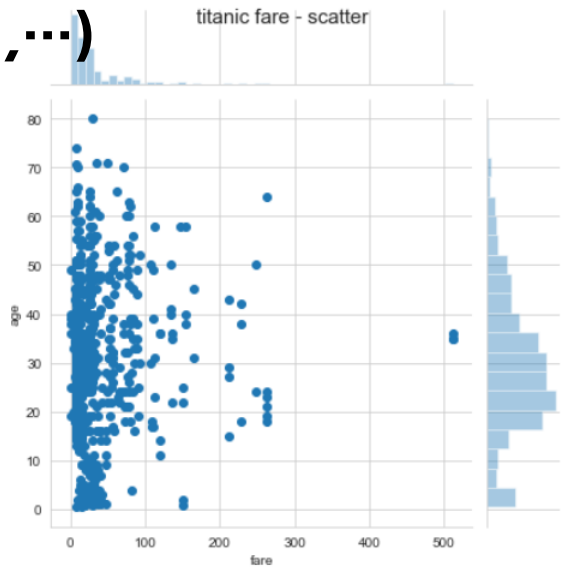
- split = True : male, female을 하나의 바이올린 플롯에서 구분

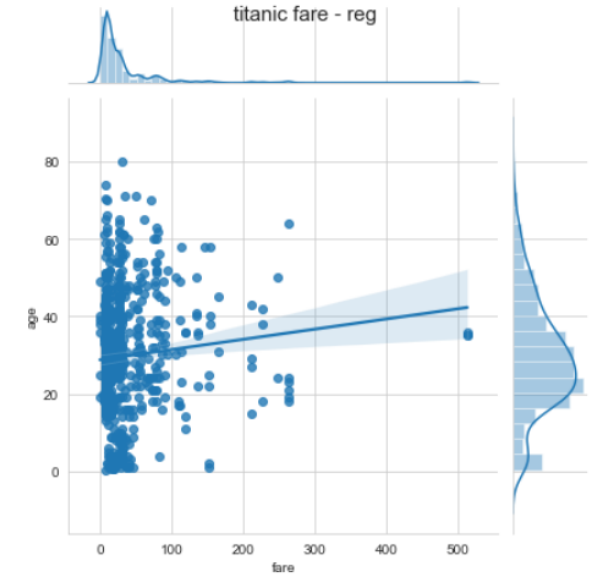


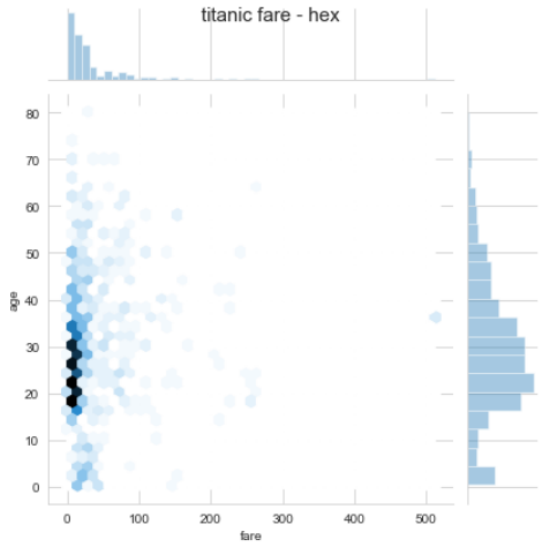
04. Seaborn 라이브러리 – 조인트그래프

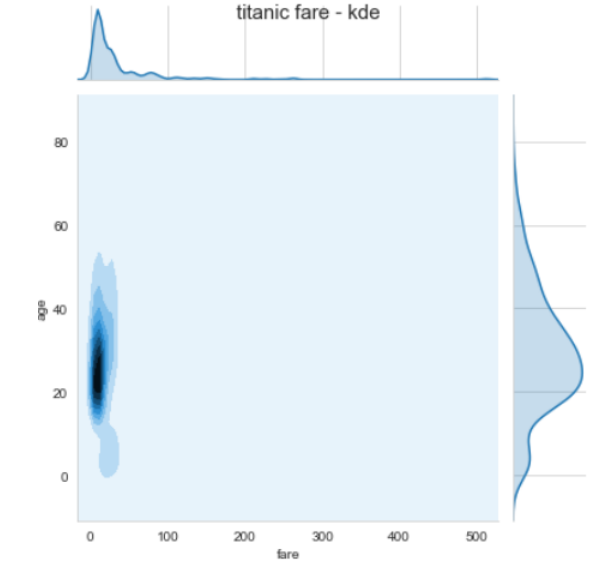
* sns.jointplot(x, y, data, kind='scatter',…)
  - 산점도를 기본으로 표시
  - x,y축 각 변수에 대한 히스토그램 함께 보여줌
  - kind 옵션 통해 다양한 그래프 그릴 수 있음

```
# 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()
```







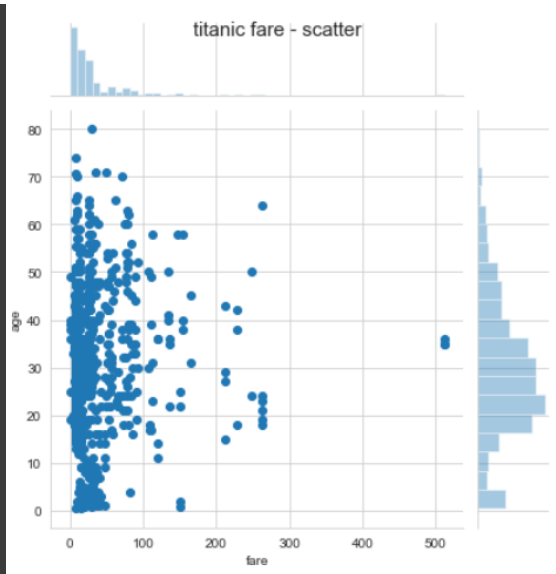


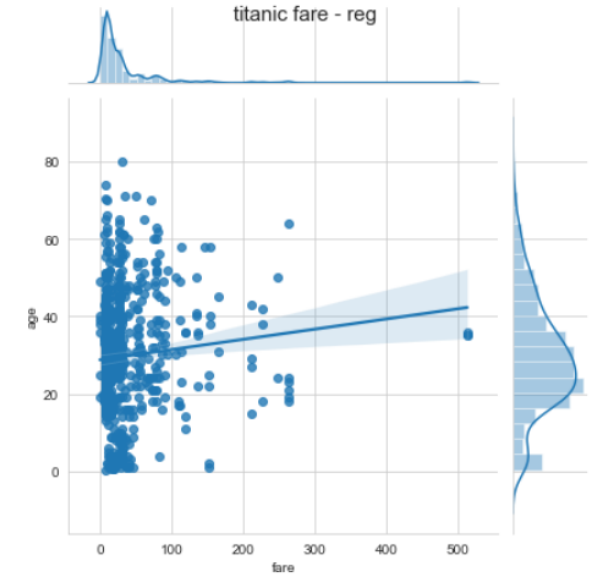
04. Seaborn 라이브러리 - 조인트그래프

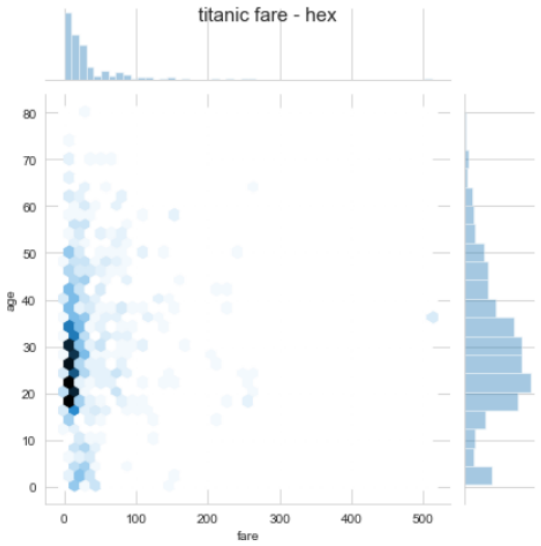
* sns.jointplot(x, y, data, kind='scatter',…)
  - 산점도를 기본으로 표시
  - x,y축 각 변수에 대한 히스토그램 함께 보여줌
  - kind 옵션 통해 다양한 그래프 그릴 수 있음

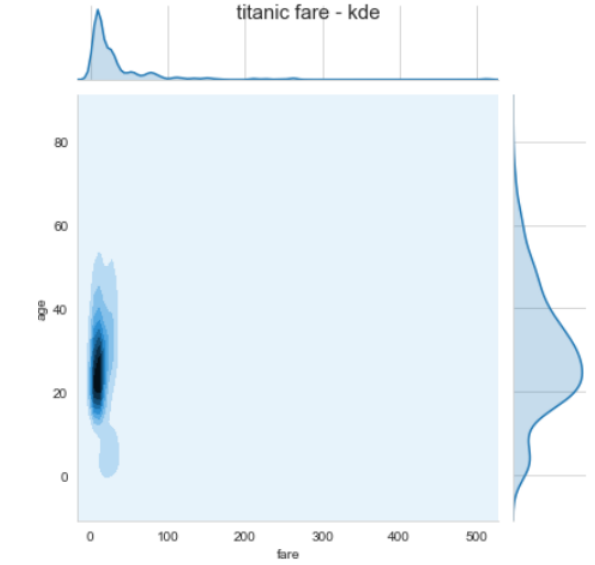
```
# 조인트 그래프 - 산점도(기본값)
j1 = sns.jointplot(x='fare', y='age', data=titanic)
# 조인트 그래프 - 회귀선
j2 = sns.jointplot(x='fare', y='age', kind='reg', data=titanic)
# 조인트 그래프 - 육각 그래프
j3 = sns.jointplot(x='fare', y='age', kind='hex', data=titanic)

# 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()
```

두 변수의 관계와 개별적인 분포를 한눈에 파악할 수 있다





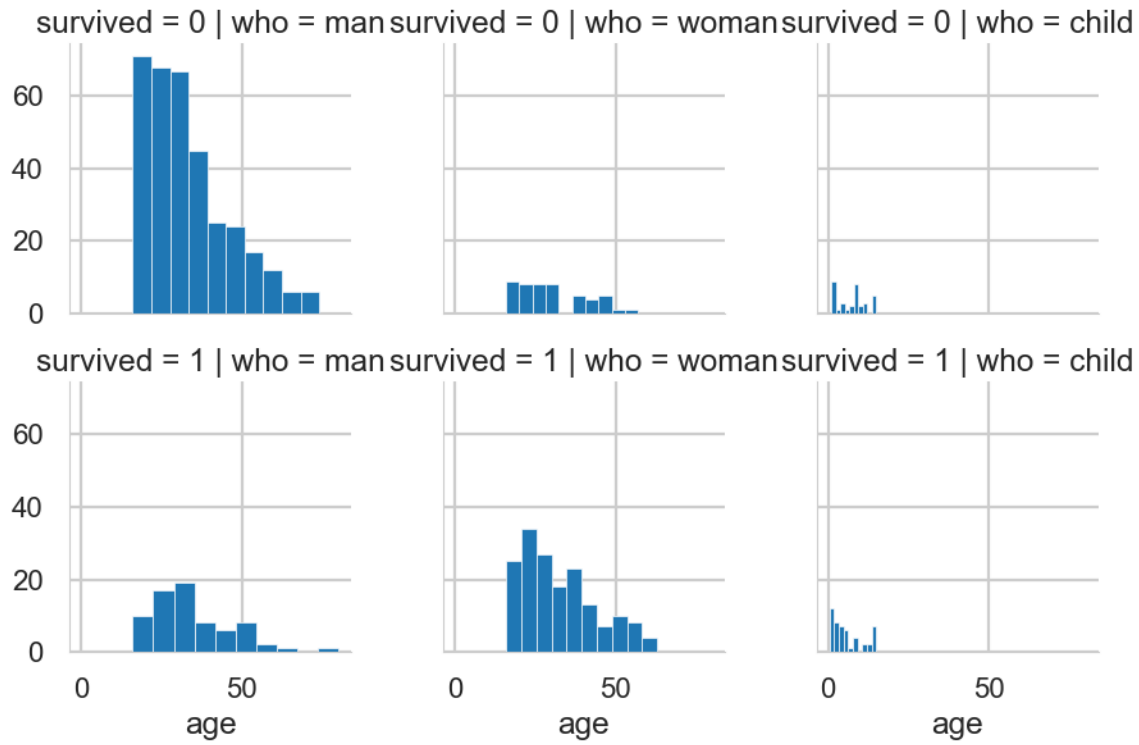




04. Seaborn 라이브러리 – 조건 적용하여 화면 그리드로 분할

```
# 조건에 따라 그리드 나누기
g = sns.FacetGrid(data=titanic, col='who', row='survived')

# 그래프 적용하기
g = g.map(plt.hist, 'age')
```

그리드객체.map(그래프 함수, 그래프 그릴 변수)



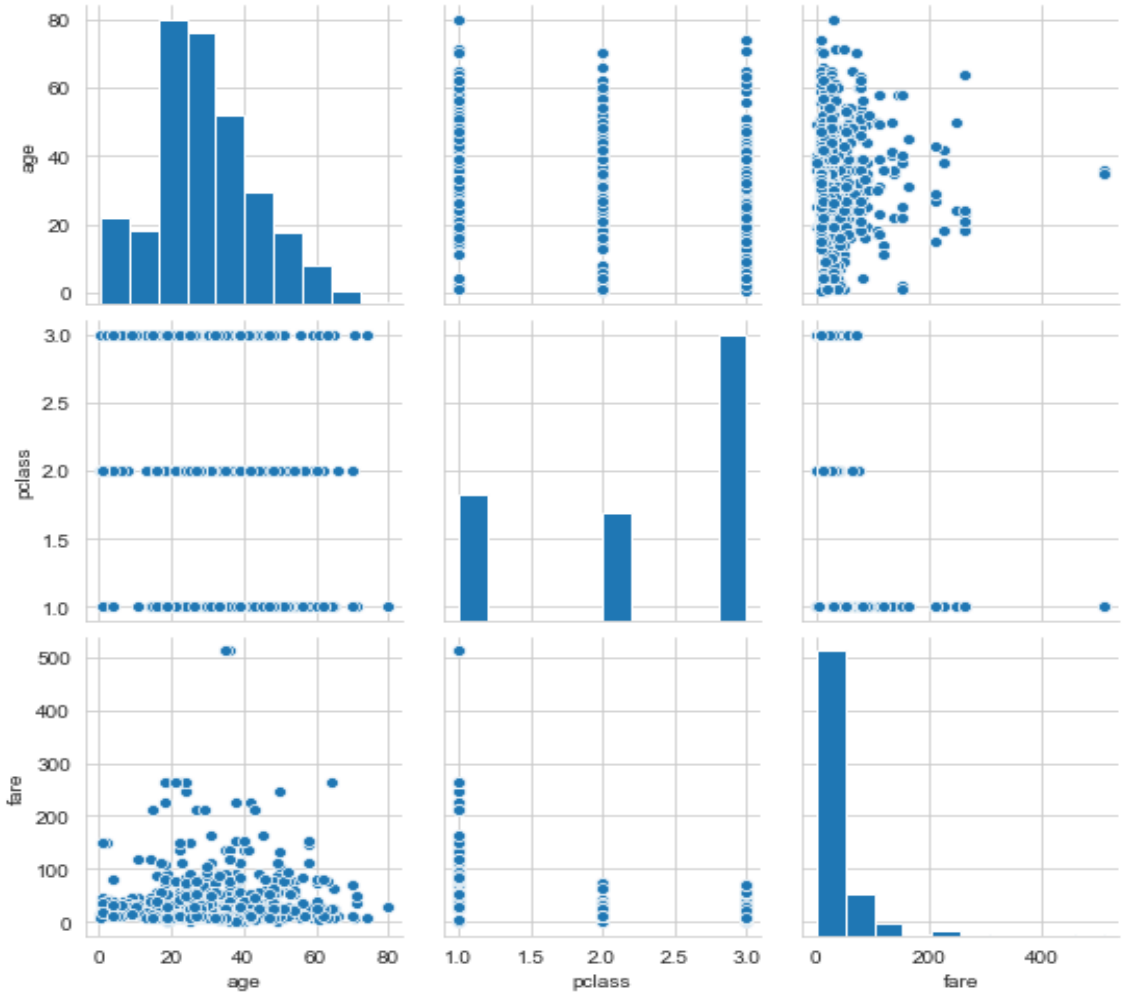
- FacetGrid() 함수 통해 카테고리 조합 수 만큼 그리드 생성
- map 메소드 이용하여 그래프종류와 변수 전달
- map 메소드의 그래프 종류와 변수 바꿔가며 그려보자
  → 변수는 여러 개 전달이 가능(산점도 등의 다변량 그래프도 그릴 수 있다)

04. Seaborn 라이브러리 – 이변수데이터의 분포

* sns.pairplot(data, diag_kind=`auto´, kind=`scatter´…)

```
# titanic 데이터셋 중에서 분석 데이터 선택하기
titanic_pair = titanic[['age','pclass', 'fare']]

# 조건에 따라 그리드 나누기
g = sns.pairplot(titanic_pair)
```

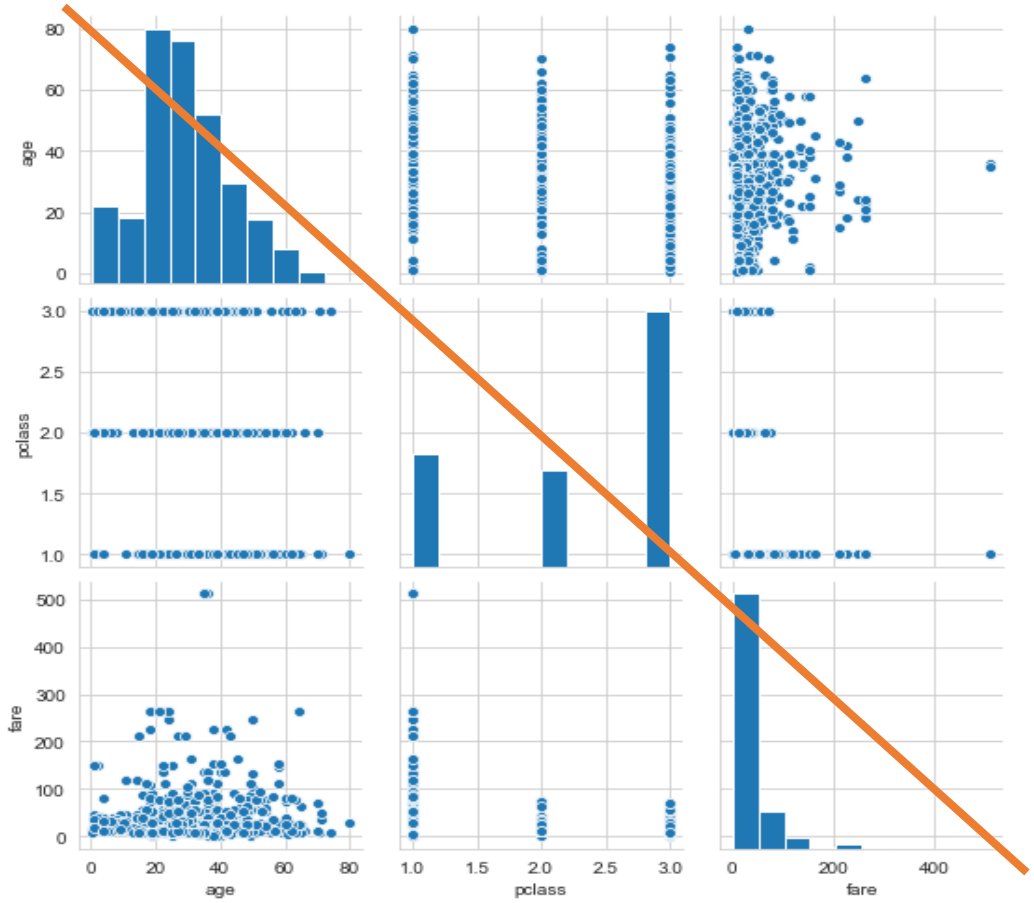
- 대각선 열에는 일변량 변수의 히스토그램, 나머지 그리드는 짝지어진 변수들의 산점도 그려줌
- data : 데이터프레임 입력(category type은 자동으로 제거)
- diag_kind : 대각선열 그래프 종류
- kind : 대각선열 제외한 그리드 그래프 종류



04. Seaborn 라이브러리 – 이변수데이터의 분포

* sns.pairplot(data, diag_kind='auto', kind='scatter'…)

  – 대각선 기준으로 중복된 정보가 표현된다는 단점



Q&A

감사합니다.