Seaborn

최희윤 강사

Seaborn이란

- Seaborn은 Matplotlib을 기반으로 다양한 색상 테마와 통계용 차트 등의 기능을 추가한 시각화 패키지이다.
- 기본적인 시각화 기능은 Matplotlib 패키지에 의존하며 통계 기능은 Statsmodels 패키지에 의존한다.

테마를 활용해서 스타일 적용하기

set_theme()는 global 범위의 테마를 적용해서 그래프를 쉽게 꾸밀 수 있다. style과 palette 키워드 인수를 활용해서 그 값을 설정.

- style: darkgrid, whitegrid, dark, white, ticks, …
- palette: pastel, husl, Spectral, flare, …

```
1  import seaborn as sns
2  sns.set_theme(style="whitegrid")
```

(참고) set_context()의 인수로 paper, notebook, talk, poster 중 하나를 넣으면 해당하는 설정을 세팅할 수 있다.

카운트 플롯

- countplot()을 사용하면 각 **카테고리** 값마다의 데이터가 얼마나 있는지 표시할 수 있다.
- 카운트 플롯은 **카테고리별** 히스토그램이라고 볼 수 있다. API와 옵션은 바 차트와 동일하다.

seaborn.countplot

```
seaborn.countplot(data=None, *, x=None, y=None, hue=None, order=None,
hue_order=None, orient=None, color=None, palette=None, saturation=0.75,
width=0.8, dodge=True, ax=None, **kwargs)
```

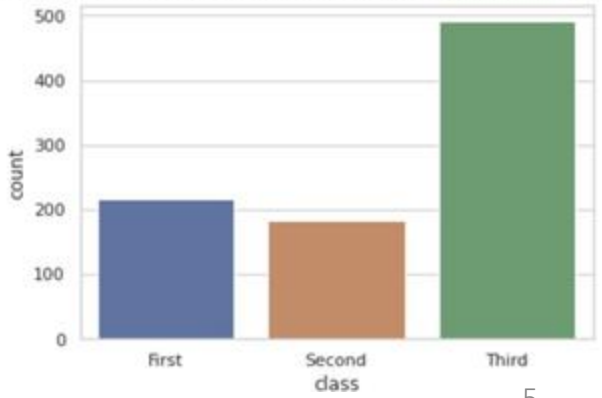
Show the counts of observations in each categorical bin using bars.

카운트 플롯

- 타이타닉호의 데이터셋을 활용해서 카운트 플롯을 확인해보자.
- class는 선실 등급을 나타내는 column.
- 바 차트와 동일한 결과를 볼 수 있다.
- 각 유니크한 값들이 몇개씩 있는지 시각적으로 확인할 수 있다.

```
1  # Show the number of datapoints with each value of a categorical variable
2  df = sns.load_dataset("titanic")
3  sns.countplot(x=df["class"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c558539a0>
```



카운트 플롯

- x 키워드 인수에 Series를 전달하고 있다.
- 해당 Series에 있는 값을 유니크하게 가져와서 카운트한 결과를 가져오는 것을 확인할 수 있다.

```
1  # Show the number of datapoints with each value of a categorical variable
2  df = sns.load_dataset("titanic")
3  sns.countplot(x=df["class"])
```

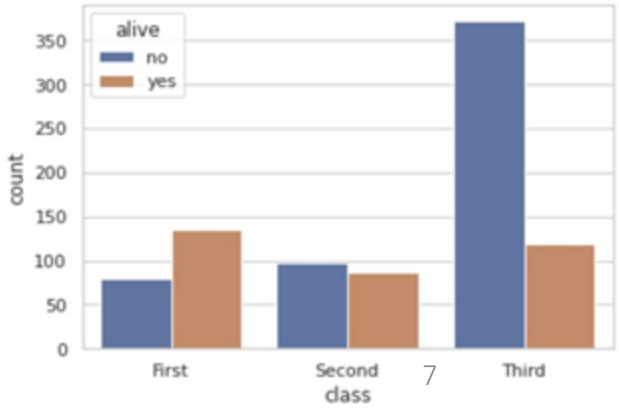
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c558539a0>
```

카운트 플롯

- hue 키워드 인수를 활용하여 분류할 두 번째 변수를 설정할 수 있다.
- 여기서는 alive를 column을 지정하였고 선실별 생존 여부를 가시적으로 확인할 수 있게 돕고 있다.

```
1  # Group by a second variable
2  sns.countplot(data=df, x="class", hue="alive")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c55803160>
```

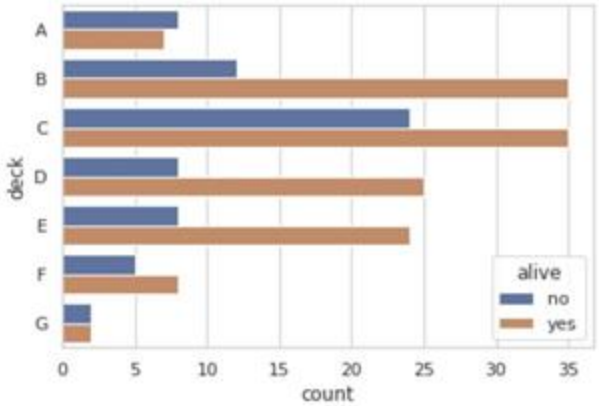


카운트 플롯

- 플롯을 수평하게도 작성할 수 있다.
- y 키워드 인수에 data로 넘긴 DataFrame의 column name 값을 전달하면 된다.

```
1  # Plot horizontally to make more space for category labels
2  sns.countplot(data=df, y="deck", hue="alive")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c5532b520>
```



히스토그램 – histplot()

- 히스토그램: 연속형 데이터 분포를 시각화
  - 연속형 변수를 일정 구간(bin)으로 나누어 각 빈에 속하는 데이터의 개수 계산
- histplot()은 1차원 혹은 2차원 데이터의 분포를 히스토그램으로 보여준다.

seaborn.histplot

```
seaborn.histplot(data=None, *, x=None, y=None, hue=None, weights=None,
stat='count', bins='auto', binwidth=None, binrange=None, discrete=None,
cumulative=False, common_bins=True, common_norm=True, multiple='layer',
element='bars', fill=True, shrink=1, kde=False, kde_kws=None,
line_kws=None, thresh=0, pthresh=None, pmax=None, cbar=False, cbar_ax=None,
cbar_kws=None, palette=None, hue_order=None, hue_norm=None, color=None,
log_scale=None, legend=True, ax=None, **kwargs)
```

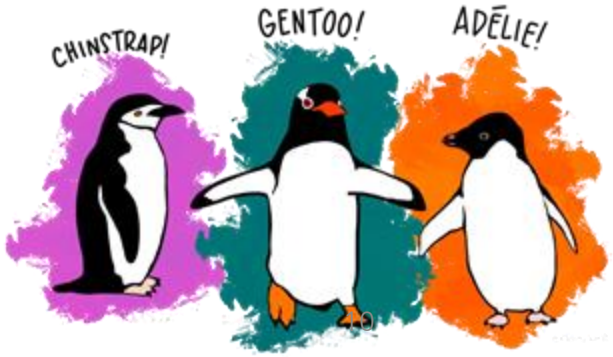
Plot univariate or bivariate histograms to show distributions of datasets.

히스토그램 – penguins dataset

- 'Adelie', 'Chinstrap', 'Gentoo' 총 3 분류의 펭귄에 대한 데이터셋입니다.

```
1  penguins = sns.load_dataset("penguins")
2  penguins.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female



source: @allison_horst
https://github.com/allisonhorst/penguins

히스토그램 – penguins dataset

- species : 펭귄의 종(Chinstrap, Adélie, Gentoo)
- bill_length_mm : bill length (mm)
- bill_depth_mm : bil depth (mm)
- flipper_length_mm : flipper length (mm)
- body_mass_g : 체중(g)
- island : 서식지 섬 (Dream, Torgersen, or Biscoe)
- sex : 펭귄 성별

```
penguins.info()
"""
species: penguin species (Chinstrap, Adélie, or Gentoo)
culmen_length_mm: culmen length (mm)
culmen_depth_mm: culmen depth (mm)
flipper_length_mm: flipper length (mm)
body_mass_g: body mass (g)
island: island name (Dream, Torgersen, or Biscoe)
        in the Palmer Archipelago (Antarctica)
sex: penguin sex
"""
```

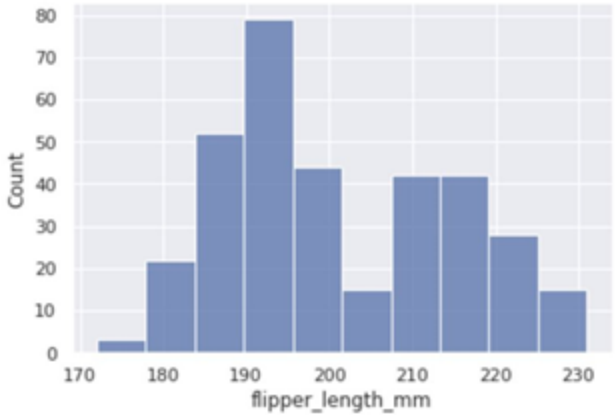
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

히스토그램 – histplot() x축 기준

- flipper 길이에 대한 분포를 x축 기준으로 히스토그램을 나타내는 예제.
- data로 DataFrame을 전달하고 x 키워드 인수로 columns label 값을 전달하고 있다.
- 히스토그램을 보면 좌측 y축은 개수를 표현하고 있다.

```
1  # Assign a variable to x to plot a univariate distribution along the x axis
2  sns.histplot(data=penguins, x="flipper_length_mm")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f173463a040>
```

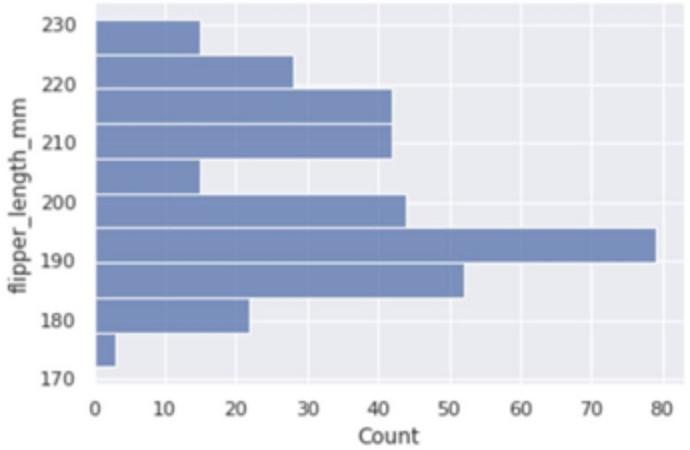


히스토그램 – histplot() y축 기준

- y축을 기준으로 히스토그램을 표현하고 있다.
- x 키워드 인수를 y 키워드 인수로 변경하여 표현했다.

```
1  # Flip the plot by assigning the data variable to the y axis
2  sns.histplot(data=penguins, y="flipper_length_mm")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f17345c3910>
```

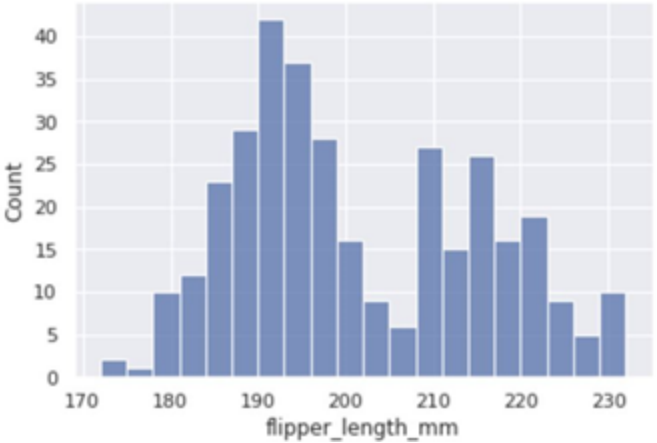


히스토그램 – histplot() bin 너비 지정하기

- binwidth 키워드 인수를 사용하면 bin의 너비를 지정할 수 있다.
- 현재는 3으로 값을 줘서 3의 범위만큼 히스토그램이 표현되는 것을 확인할 수 있다.

```
1  # Check how well the histogram represents the data by specifying a different bin width
2  sns.histplot(data=penguins, x="flipper_length_mm", binwidth=3)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1734576670>
```

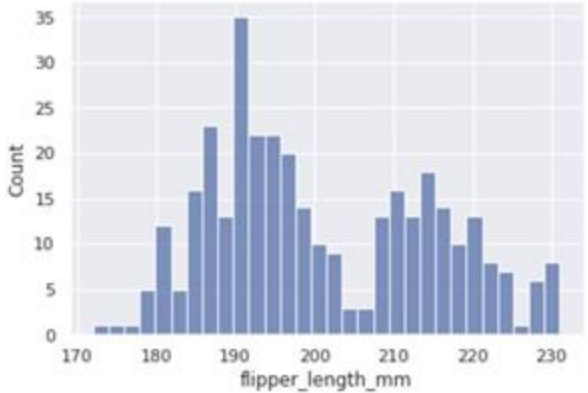


히스토그램 – histplot() bin 너비 지정하기

- bins 키워드 인수를 사용해서 bin의 개수를 지정할 수도 있다.
- 아래에서는 30개의 bin을 만들고 이를 히스토그램으로 출력하고 있다.

```
1  # You can also define the total number of bins to use
2  sns.histplot(data=penguins, x="flipper_length_mm", bins=30)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f173448ef70>
```

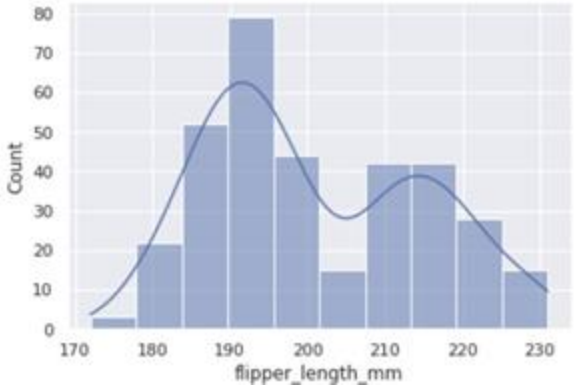


히스토그램 – histplot() kde를 동시에 표시하기

- kde는 kernel density의 약자로 커널이라는 함수를 겹치는 방법으로 히스토그램보다 부드러운 형태의 분포 곡선을 보여줍니다.
- 아래 코드에서는 kde 키워드 인수에 True 값을 전달하여 히스토그램과 kde 그래프를 동시에 출력하고 있다.

```
1  # Add a kernel density estimate to smooth the histogram,
2  # providing complementary information about the shape of the distribution
3  sns.histplot(data=penguins, x="flipper_length_mm", kde=True)
```

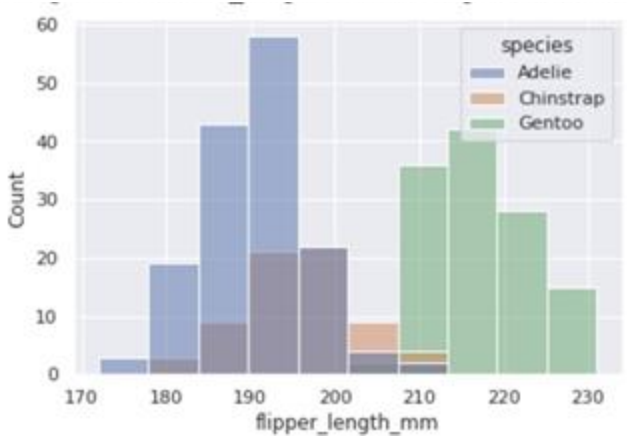
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1734cb7f70>
```



히스토그램 – histplot() hue 키워드 인수로 데이터 분리하기

- hue 키워드 인수에 분류 기준이 될 column label을 전달해서 여러 개의 히스토그램을 합친 것과 같은 결과의 그래프를 얻을 수 있다.
- 펭귄의 종류마다 분포를 나눴기 때문에 이제야 분포가 더 명확하게 보이는 것을 확인할 수 있다.

```
2  sns.histplot(data=penguins, x="flipper_length_mm", hue="species")
```

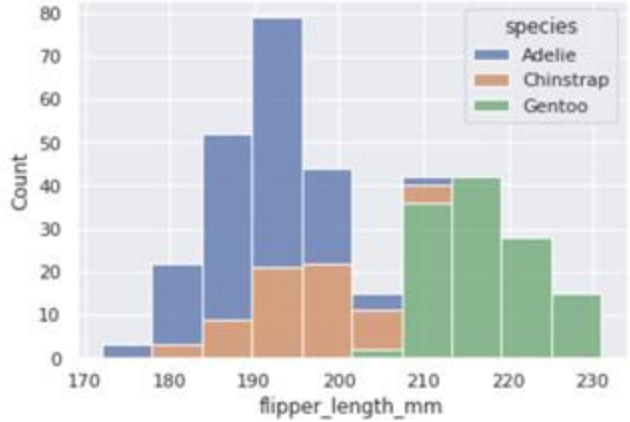


히스토그램 – histplot() 다양한 표현법 stack

- 앞선 예제는 각 히스토그램의 layer가 겹쳐서 표현됐다면, 이를 누적하듯 표현하는 방법도 있다.
- multiple 키워드 인수에 "stack" 값을 전달하면 된다.
- 펭귄의 종류별 누적 히스토그램을 아래와 같이 확인할 수 있다.

```
1  # can also "stack" them
2  sns.histplot(data=penguins, x="flipper_length_mm", hue="species", multiple="stack")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f17343ce580>
```

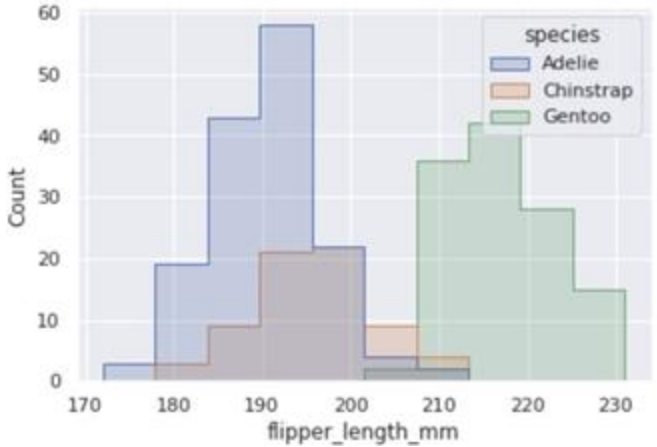


히스토그램 – histplot() 다양한 표현법 step

- 오버래핑된 결과가 보기 힘들 때의 해결 방법 중 하나
- elements 키워드 인수에 'step' 이란 값을 전달하여 아래의 그래프처럼 표현할 수 있다.

```
1  # Overlapping bars can be hard to visually resolve.
2  # A different approach would be to draw a step function
3  sns.histplot(penguins, x="flipper_length_mm", hue="species", element="step")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f173437bd30>
```

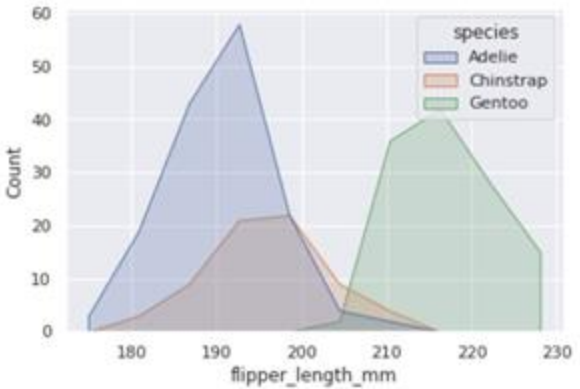


히스토그램 – histplot() 다양한 표현법 poly

- 다각형 형태로도 표현 가능하다.
- element 키워드 인수에 'poly' 값을 전달한다.
- 이는 전체적인 모양을 살피는데 더 유용하다.

```
1  # You can move even farther away from bars by drawing a polygon
2  # with vertices in the center of each bin.
3  # This may make it easier to see the shape of the distribution,
4  # but use with caution: it will be less obvious to your audience that they are looking at a histogram
5  sns.histplot(penguins, x="flipper_length_mm", hue="species", element="poly")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f173426be20>
```

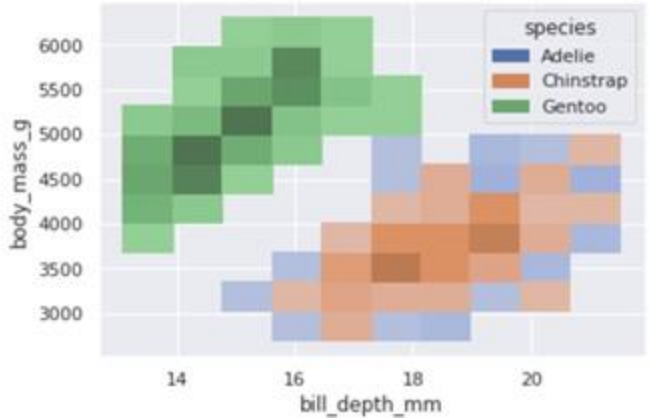


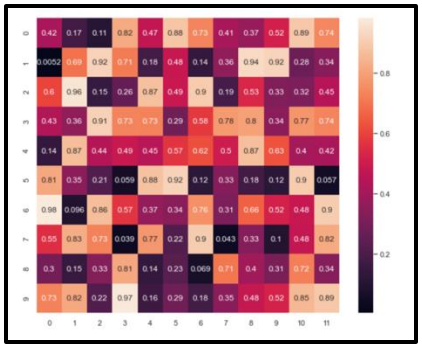
히스토그램 – histplot()

- x축과 y축 모두 column을 할당하게 되면 히트맵(heatmap) 형태의 히스토그램을 표현한다.

```
1  # When both x and y are assigned,
2  # a bivariate histogram is computed and shown as a heatmap
3  sns.histplot(penguins, x="bill_depth_mm", y="body_mass_g", hue="species")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1733f97400>
```





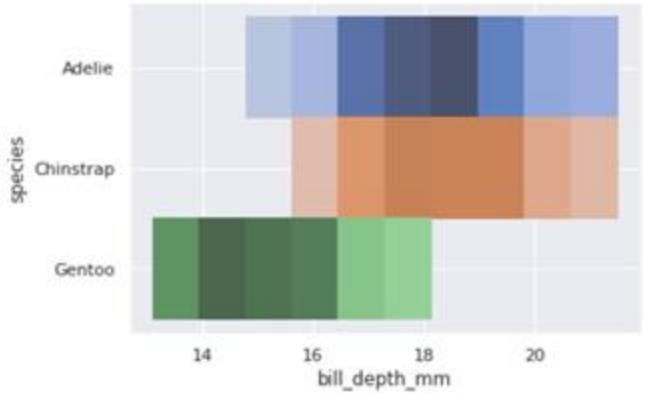
히트맵 예시 – 데이터의 상관관계 파악 위함

히스토그램 – histplot()

- x축과 y축 모두 column을 할당할 때 그 중 하나의 값이 **이산된 값(구분 가능한 개별적인 값)**이라면 아래와 같이 데이터를 시각화하여 표현하면 훨씬 이해가 쉽게 표현할 수 있다.

```
# Multiple color maps can make sense when one of the variables is discrete
sns.histplot(
    penguins, x="bill_depth_mm", y="species", hue="species", legend=False
)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1733eced60>
```



히스토그램 – displot()

- displot()도 마찬가지로 1차원 혹은 2차원 데이터의 분포를 히스토그램으로 보여준다.
- 다만 rug와 kde에 대해 동시에 표현이 가능해서 표현 범위가 더 넓고 이에 따라 많이 사용됩니다.

seaborn.displot

```
seaborn.displot(data=None, *, x=None, y=None, hue=None, row=None,
col=None, weights=None, kind='hist', rug=False, rug_kws=None,
log_scale=None, legend=True, palette=None, hue_order=None, hue_norm=None,
color=None, col_wrap=None, row_order=None, col_order=None, height=5,
aspect=1, facet_kws=None, **kwargs)
```

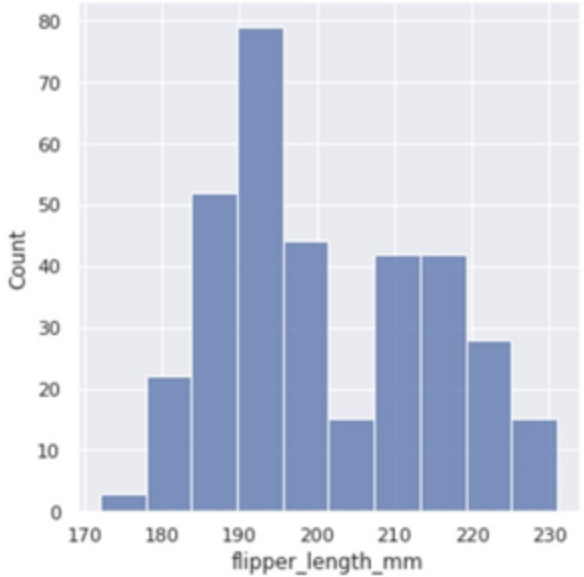
Figure-level interface for drawing distribution plots onto a FacetGrid.

히스토그램 – displot()

- 기본적인 형태의 히스토그램을 그리고 있다.
- histplot()과 동일한 인수와 그래프 모양을 보여준다.

```
1  # The default plot kind is a histogram
2  sns.displot(data=penguins, x="flipper_length_mm")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f17341da160>
```

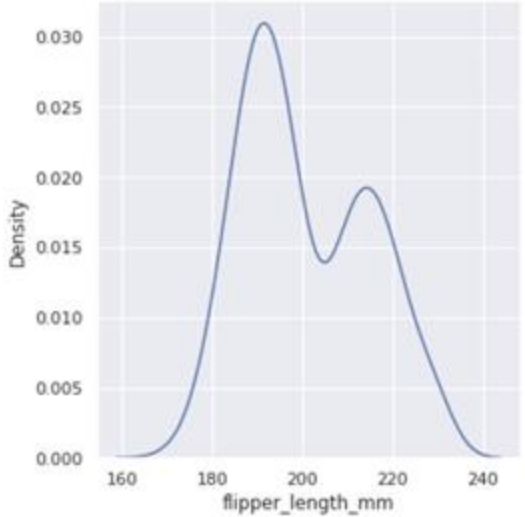


히스토그램 – displot() kde로 표현하기

- kind 키워드 인수를 활용해서 기본적으로 표현되는 히스토그램의 종류를 kde로 변경했다.

```
1  # Use the kind parameter to select a different representation
2  sns.displot(data=penguins, x="flipper_length_mm", kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f1733f783d0>
```

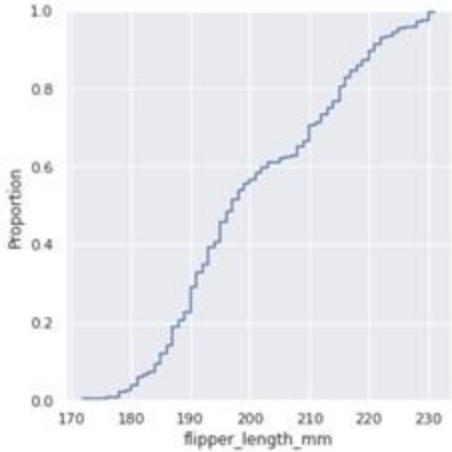


히스토그램 – displot() ecdf로 표현하기

- 이번에는 kind 키워드 인수에 ecdf(empirical cumulative distribution functions)를 활용해서 누적 분포 그래프로 표현하고 있다.

```
# There are three main plot kinds;
# in addition to histograms and kernel density estimates (KDEs),
# you can also draw empirical cumulative distribution functions (ECDFs):
sns.displot(data=penguins, x="flipper_length_mm", kind="ecdf")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f17344d2b80>
```

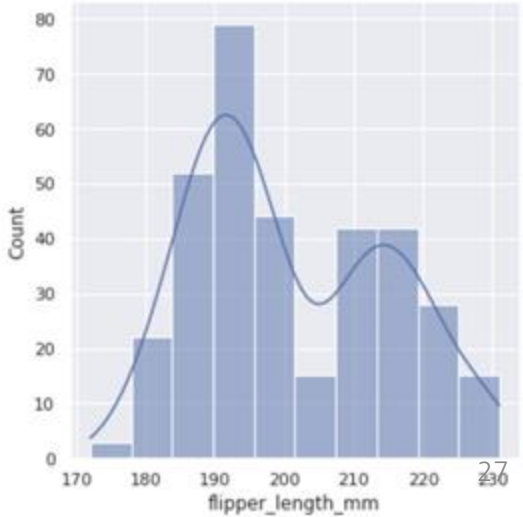


히스토그램 – displot() 히스토그램과 kde 동시 표현

- kde 키워드 인수 값을 True로 전달하여 히스토그램과 kde를 동시에 표현하고 있다.
- 앞에서의 kind=kde와 비교해서 차이점을 생각해보세요.

```
1  # While in histogram mode, it is also possible to add a KDE curve
2  sns.displot(data=penguins, x="flipper_length_mm", kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f1734275f40>
```

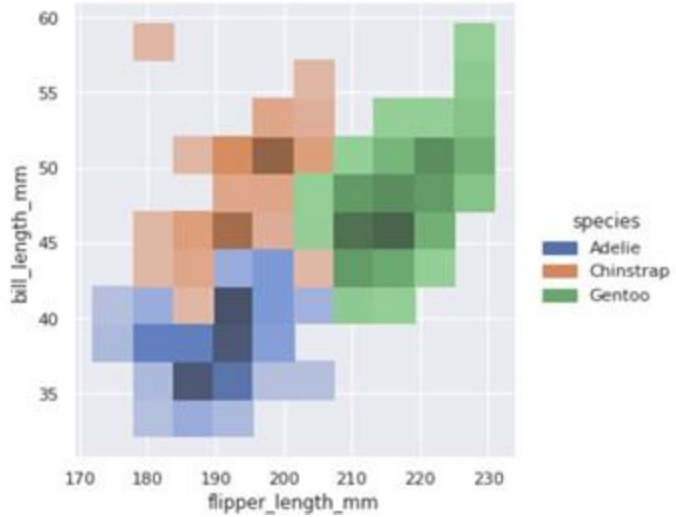


히스토그램 – displot() x, y모두 값 할당하기

- histplot()과 동일하게 x와 y에 각각 값을 할당하여 히트맵 형태로 그래프를 출력할 수 있다.

```
1  # To draw a bivariate plot, assign both x and y
2  sns.displot(data=penguins, x="flipper_length_mm", y="bill_length_mm", hue="species")
```

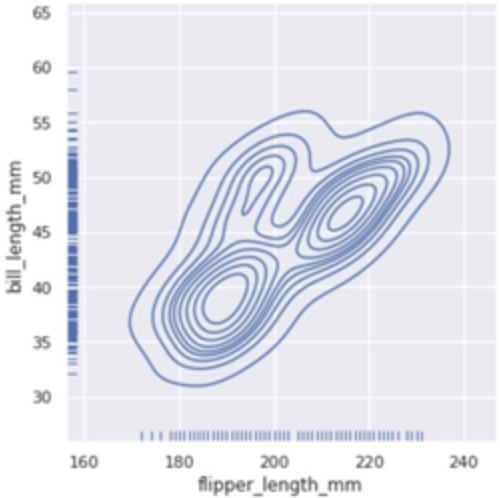
```
<seaborn.axisgrid.FacetGrid at 0x7f1733d8bca0>
```



히스토그램 – displot() 2개 값을 kde로 표현하기

- x, y축 모두 할당한 앞의 히스토그램을 kde로도 표현 가능하다.
- 뿐만 아니라 rug=True 키워드 인수를 전달해서 rug에 대한 표현도 동시에 할 수 있습니다.

```
# Currently, bivariate plots are available only for histograms and KDEs
# show individual observations with a marginal "rug"
g = sns.displot(data=penguins, x="flipper_length_mm", y="bill_length_mm", kind="kde", rug=True)
```

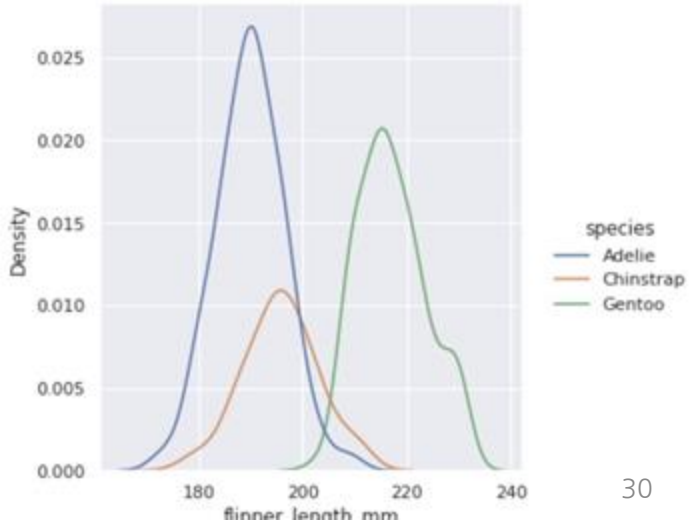


히스토그램 – displot() hue로 subset 나누기

- hue 키워드 인수를 통해 subset을 구분하고 이를 각각의 그래프로 표현할 수 있다.
- 아래는 hue 값으로 species(펭귄의 종)에 대해 설정해서 각각 species별 그래프.

```
1  # Each kind of plot can be drawn separately for subsets of data using hue mapping
2  sns.displot(data=penguins, x="flipper_length_mm", hue="species", kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f173410cb80>
```

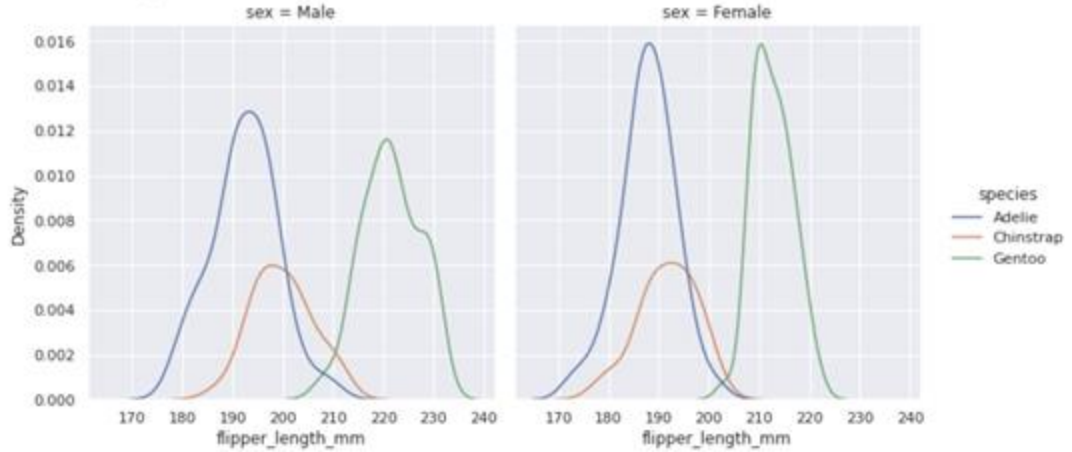


히스토그램 – displot() col로 그래프 나누기

- displot()은 더 나아가 col 키워드 인수를 활용하여 subset을 한번 더 분류할 수 있다.
- col에 sex column label을 설정하여 subset을 나눔.

```
1  # The figure is constructed using a FacetGrid,
2  # meaning that you can also show subsets on distinct subplots, or "facets":
3  sns.displot(data=penguins, x="flipper_length_mm", hue="species", col="sex", kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f1733cb9ee0>
```

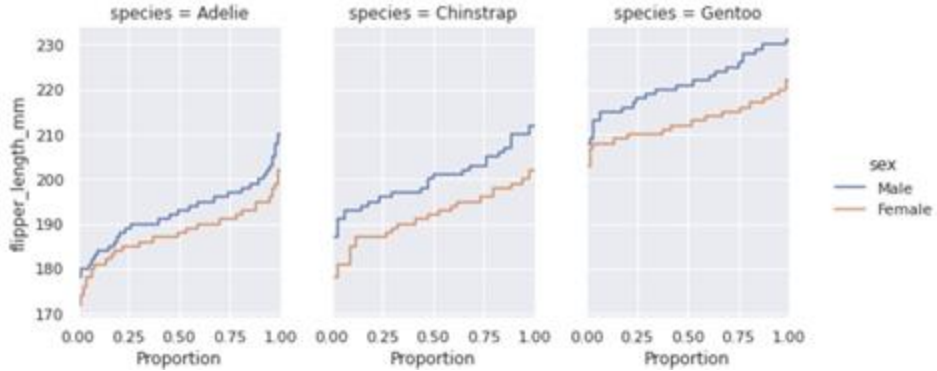


히스토그램 – displot() 그래프로 크기 설정

- height와 aspect를 활용해서 그래프 크기를 제어할 수 있습니다.
- height의 단위는 inches이며 aspect는 height와 aspect 값을 곱해서 얻음. 높이 대비 너비 비율을 지정한다고 이해하면 된다.

```
# Because the figure is drawn with a FacetGrid, you control its size and shape with the height and aspect parameters
sns.displot(
    data=penguins, y="flipper_length_mm", hue="sex", col="species",
    kind="ecdf", height=4, aspect=.7,
)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f1733ebaeb0>
```

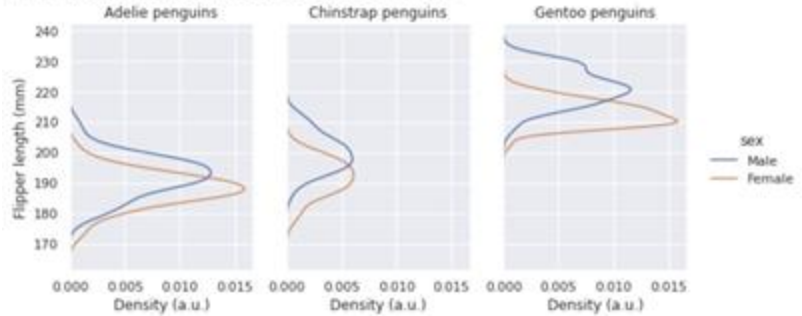


히스토그램 – displot() col로 그래프 나누기

- 각 axis label과 title도 지정해줄 수 있습니다.
- axis label은 set_axis_labels(xlabel, ylabel) 메서드를, title은 set_titles()을 활용하는데, formatting keys인 {col_var}와 {col_name}을 조합해서 포맷팅할 수 있습니다.

```
# The function returns the FacetGrid object with the plot,
# and you can use the methods on this object to customize it further
g = sns.displot(
    data=penguins, y="flipper_length_mm", hue="sex", col="species",
    kind="kde", height=4, aspect=.7,
)
g.set_axis_labels("Density (a.u.)", "Flipper length (mm)")
g.set_titles("{col_name} penguins")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f1731ef8b20>
```



barplot()

- 바 플롯은 카테고리마다의 평균을 각 사각형의 길이로 편차를 에러바(error bar)로 표현.
- 첫번째 인수로 data를 나머지를 키워드 인수로 전달한다.

seaborn.barplot

```
seaborn.barplot(data=None, *, x=None, y=None, hue=None, order=None,
hue_order=None, estimator='mean', errorbar=('ci', 95), n_boot=1000,
units=None, seed=None, orient=None, color=None, palette=None,
saturation=0.75, width=0.8, errcolor='.26', errwidth=None, capsize=None,
dodge=True, ci='deprecated', ax=None, **kwargs)
```

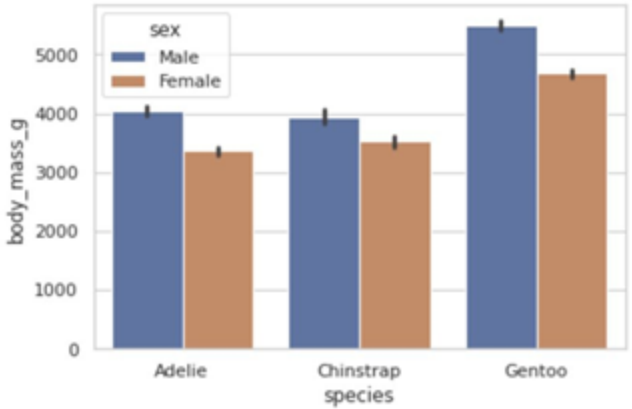
Show point estimates and errors as rectangular bars.

barplot()

- x와 hue를 카테고리화해서 body_mass_g 값의 평균과 신뢰구간(confidence intervals)을 구해서 바 플롯으로 표현하고 있다.

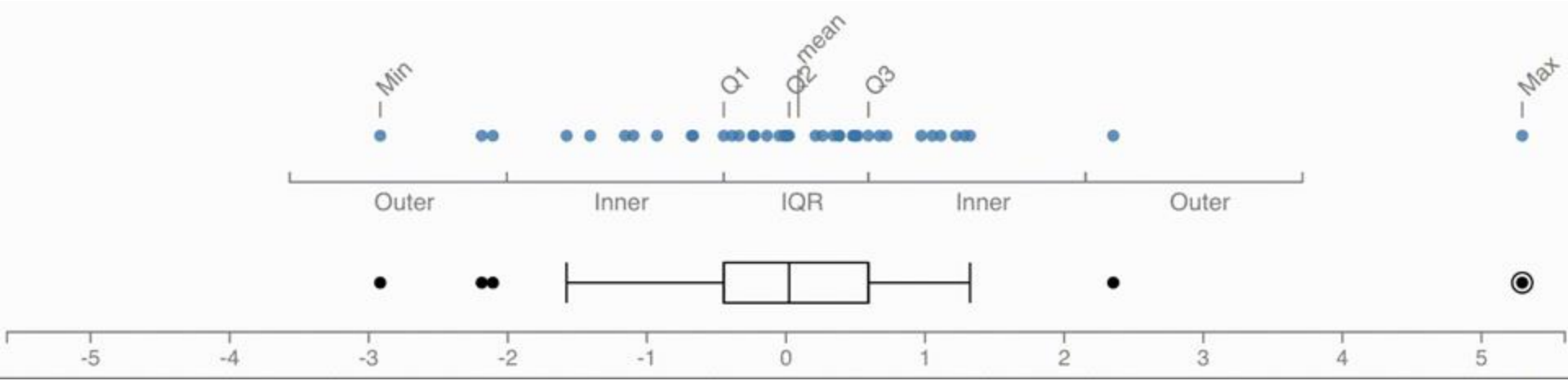
```
# Group by a categorical varaible and plot aggregated values, with confidence intervals
# Add a second layer of grouping
df = sns.load_dataset("penguins")
sns.barplot(data=df, x="species", y="body_mass_g", hue="sex")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c4765fcd0>
```



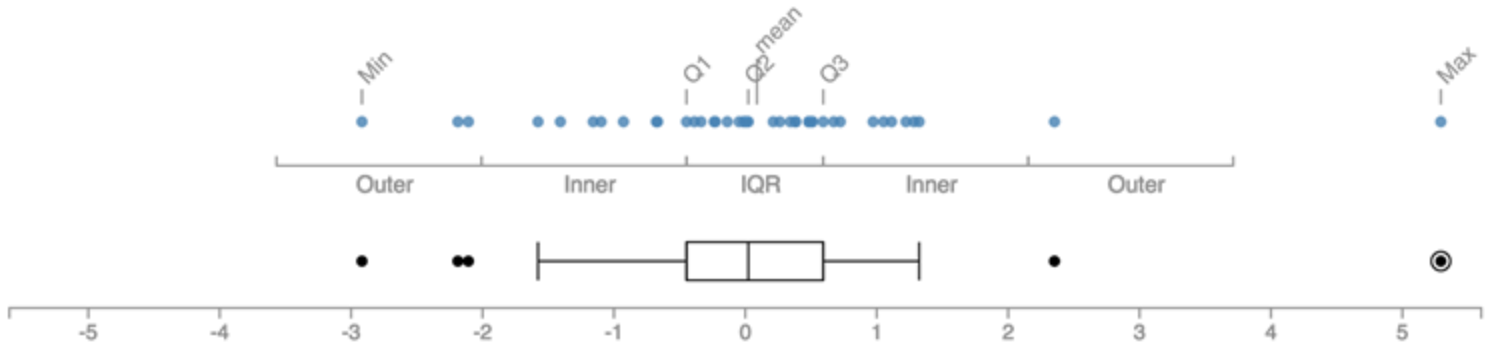
barplot()

- 박스-휘스커 플롯(Box-Whisker Plot) 혹은 간단히 박스 플롯이라 부른다.
- 박스 플롯은 데이터의 분포와 이상치를 동시에 보여주면서 서로 다른 데이터군을 쉽게 비교할 수 있다. 또한 이상치에 강건하기 때문에 안정적이다.



barplot()

- 박스는 실수 값 분포에서 1사분위수(Q1)와 3사분위수(Q3)를 뜻한다.
- 이 3사분위수와 1사분수의 차이(Q3 - Q1)를 IQR(interquartile range)라고 한다. 박스 내부의 선은 중앙값을 나타낸다.
- 박스 외부의 세로선은 **1사분위 수보다 1.5 x IQR 만큼 낮은 값**과 **3사분위 수보다 1.5 x IQR 만큼 높은 값**의 구간을 기준으로 그 구간의 내부에 있는 가장 큰 데이터와 가장 작은 데이터를 잇는 선분. 그 바깥의 점은 아웃라이어(outlier) 라고 부르는데 일일히 점으로 표시.

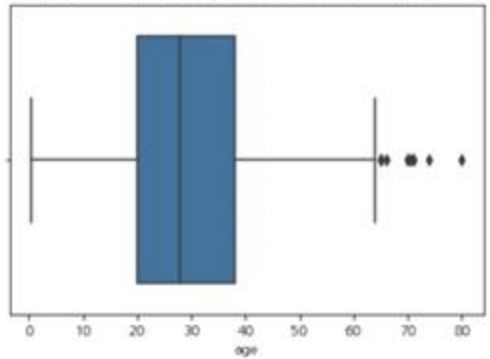


barplot()

- 아래 예제에서는 boxplot() 메서드에 x 값으로 Series만을 전달하고 있다.
- 이 결과로 하나의 box plot을 수평하게 그렸다.
  - ◯ 타이타닉호에 탑승한 사람들의 나이(age)에 대한 4분위를 그래프로 표현.

```
1  # Draw a single horizontal boxplot,
2  # assigning the data directly to the coordinate variable
3
4  df = sns.load_dataset("titanic")
5  sns.boxplot(x=df["age"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7e99fdf0>
```



barplot()

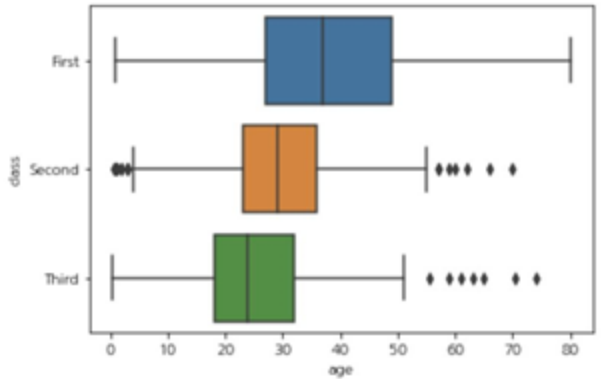
- 타이타닉호의 데이터셋 중 class column의 Dtype은 카테고리입니다.
- 이 카테고리화된 column을 하나의 분류 값으로 추가하기 위해서 y 키워드 인수로 설정.
  - y축을 기준으로 카테고리별 box plot을 unique한 value의 개수만큼 나눠서 얻을 수 있음.

```
1  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
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: 80.7+ KB
```

```
1  # Group by a categorical variable,
2  # referencing columns in a dataframe:
3  sns.boxplot(data=df, x="age", y="class")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7e4dd160>
```

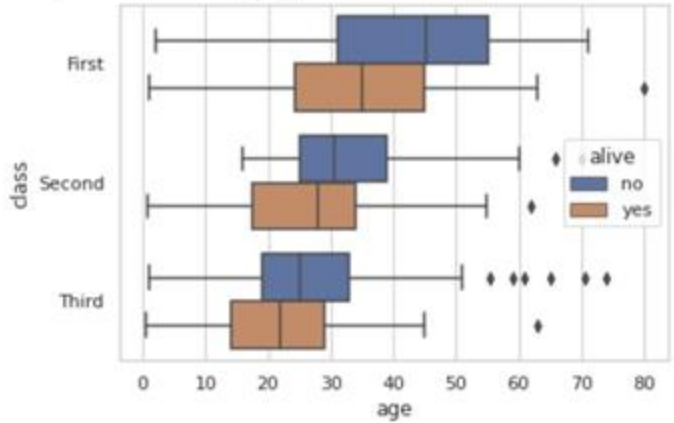


barplot()

- x에 할당한 값에 대한 분포를 구하는데, y와 hue에 전달된 column을 기준으로 그룹화한 결과를 박스플롯으로 그리고 있다.

```
1  # Group by a categorical variable, referencing columns in a dataframe
2  # Draw a vertical boxplot with nested grouping by two variables
3  sns.boxplot(data=titanic, x="age", y="class", hue="alive")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c47640dc0>
```

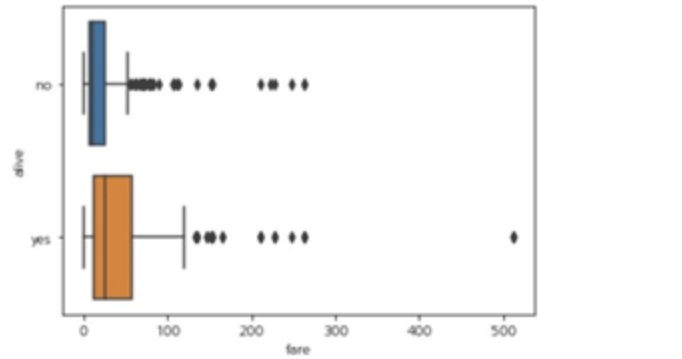


barplot()

- box plot을 그렸는데 여러개로 출력된 그래프의 순서가 맘에 들지 않는 경우가 있을 수 있다.
- 그때는 order 키워드 인수를 사용하면 된다. order 키워드 인수로 전달할 값을 리스트의 형태로 작성하면 되는데, 원하는 차례로 ticklabel을 적으면 된다.

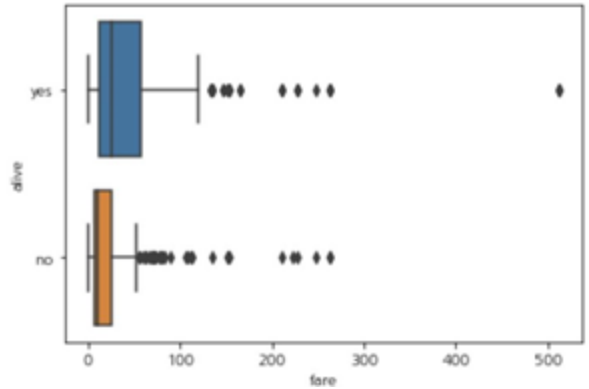
```
1  sns.boxplot(data=df, x="fare", y="alive")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7d8caac0>
```




```
1  # Control the order of the boxes
2  sns.boxplot(data=df, x="fare", y="alive", order=["yes", "no"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7d965b80>
```

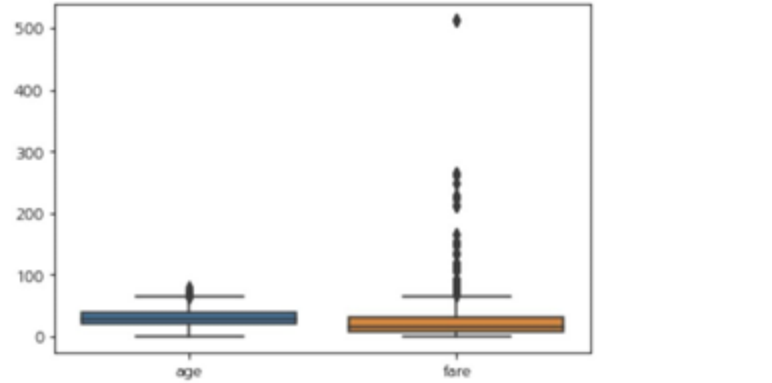


barplot()

- 수치 값을 갖는 column 여럿을 동시에 하나의 box plot에 표현할 수 있다. data 키워드 인수로 DataFrame을 전달할 때 그래프로 표현하고 싶은 수치 값을 갖는 column들을 인덱싱해서 DataFrame 형태로 전달하면 된다.
- box plot의 방향을 바꾸고 싶으면 orient 키워드 인수에 h(수평), v(수직) 값을 줘서 설정하면 수평 수직을 변경할 수 있다.

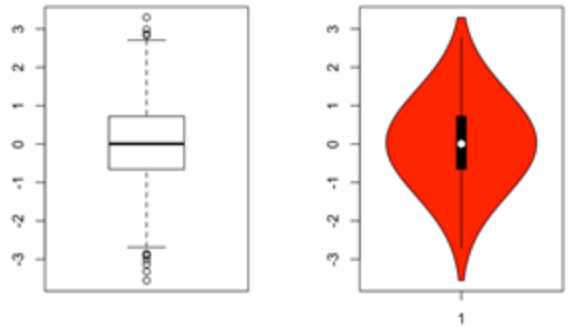
```
1  # Draw a box for multiple numeric columns
2  sns.boxplot(data=df[["age", "fare"]], orient="v")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7d5ee9d0>
```



violinplot()

- box plot과 kde(커널 밀도 함수) 둘을 콤비네이션한 그래프이다. 그런데 마치 그 생김새가 바이올린 같다하여 violin plot이라고 불린다. 세로 방향으로 kde의 모양을 그리는데 kde가 좌우 대칭되게 그려진다.
- 데이터의 분포와 범위를 한눈에 보기 쉽게 나타내는 형식으로 박스 플롯보다 실제에 가까운 분포를 확인할 수 있다.



seaborn.violinplot

```
seaborn.violinplot(data=None, *, x=None, y=None, hue=None, order=None,
hue_order=None, bw='scott', cut=2, scale='area', scale_hue=True,
gridsize=100, width=0.8, inner='box', split=False, dodge=True, orient=None,
linewidth=None, color=None, palette=None, saturation=0.75, ax=None,
**kwargs)
```

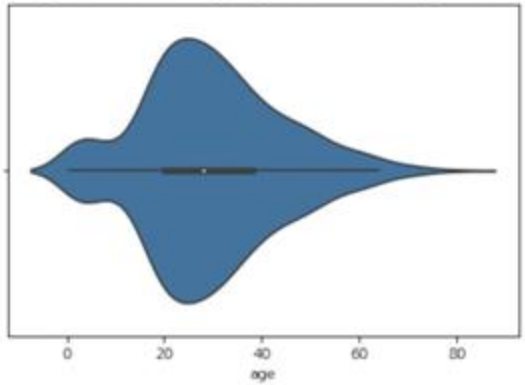
Draw a combination of boxplot and kernel density estimate.

violinplot()

- violinplot() 메서드에 x 키워드 인수만 값을 Series로 할당하여 violin plot을 그림.
  - Series 값으로 그래프를 그리면 기본적으로 수평 방향의 그래프를 그림.

```
1  # Draw a single horizontal boxplot,
2  # assigning the data directly to the coordinate variable
3  df = sns.load_dataset("titanic")
4  sns.violinplot(x=df["age"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7d5511c0>
```

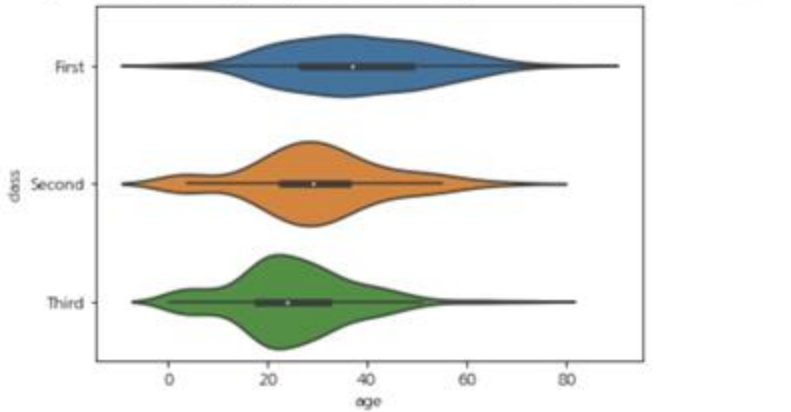


violinplot()

- 카테고리 Dtype을 갖는 class column을 활용하여 y축의 값으로 설정.
- 그렇게 하면 각 카테고리마다 갖는 데이터 분포를 y축을 기준으로 하여 분리해서 그릴 수 있다.

```
1  # Group by a categorical variable,
2  # referencing columns in a dataframe
3  sns.violinplot(data=df, x="age", y="class")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7cf66940>
```

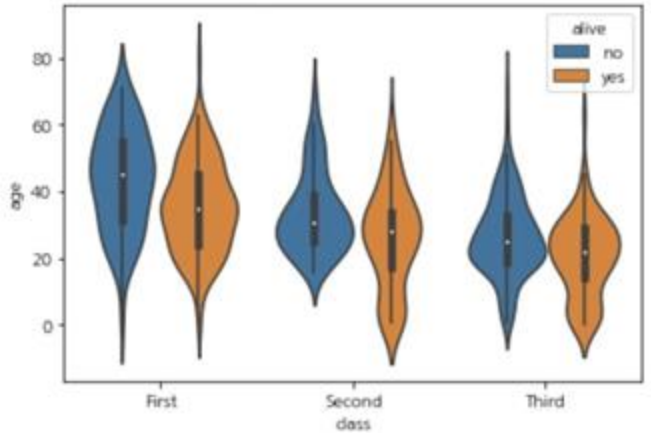


violinplot()

- x 키워드 인수에 카테고리 자료형을 설정하고 있다. 이에 따라 violin plot의 방향이 vertical 하게 변경된 것을 확인할 수 있다.

```
1  # Draw vertical violins, grouped by two variables
2  sns.violinplot(data=df, x="class", y="age", hue="alive")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7cec7be0>
```

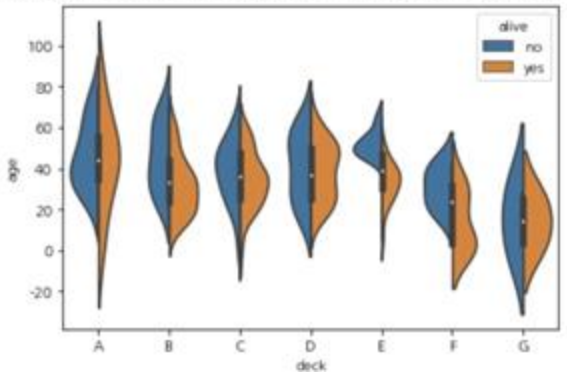


violinplot()

- violin의 그래프가 hue에 설정한 alive column의 값에 따라 좌측, 우측 각각 분포를 그려내고 있다.
- split 키워드 인수에 True 값을 전달하는 조건으로 hue에 설정된 값의 level이 단 두개여야 한다. 즉 현재 hue에 설정된 alive가 갖는 값이 'yes'와 'no' 단 두개이기 때문에 split 키워드 인수에 True 설정을 할 수 있는 것입니다.

```
1  # Draw split violins to take up less space
2  sns.violinplot(data=df, x="deck", y="age", hue="alive", split=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7cd78820>
```



violinplot()

- 만약 hue의 level이 정확히 2개가 아니라면 어떤 결과가 나올까?
  - ValueError를 반환합니다. split 키워드 인수의 값을 True로 하고 싶다면 반드시 hue에 오는 column의 값이 단 2가지만으로 분류되어야 한다.

```
1  # Draw split violins to take up less space
2  sns.violinplot(data=df, x="deck", y="age", hue="class", split=True)
```

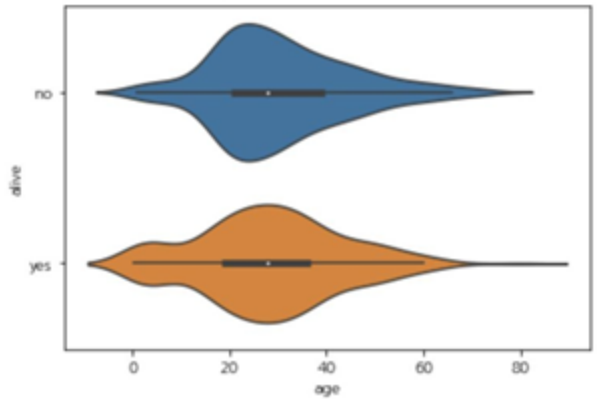
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-49-f9ee968b724b> in <module>
      1 # Draw split violins to take up less space
----> 2 sns.violinplot(data=df, x="deck", y="age", hue="class", split=True)

                                  2 frames
/usr/local/lib/python3.8/dist-packages/seaborn/categorical.py in __init__(self, x, y, hue, data, order, hue_order, bw, cut, scale, scale_hue, gridsize,
width, inner, split, dodge, orient, linewidth, color, palette, saturation)
    539         if split and self.hue_names is not None and len(self.hue_names) != 2:
    540             msg = "There must be exactly two hue levels to use `split`.'"
--> 541             raise ValueError(msg)
    542         self.split = split
    543

ValueError: There must be exactly two hue levels to use `split`.'
```

violinplot()

- violin plot을 표현할 때 범위를 제한할 수 있다. 타이타닉호 데이터셋에서 나이(age) 의 경우에 0 미만의 범위에 대해 그래프가 표현될 수 있다.
  - ○ 나이 데이터 특성상 음수는 있을 수 없기 때문에 이런 표현은 바람직하지 않다.

```
1  # Prevent the density from smoothing beyond the limits of the data
2  sns.violinplot(data=df, x="age", y="alive")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7cc1db20>
```

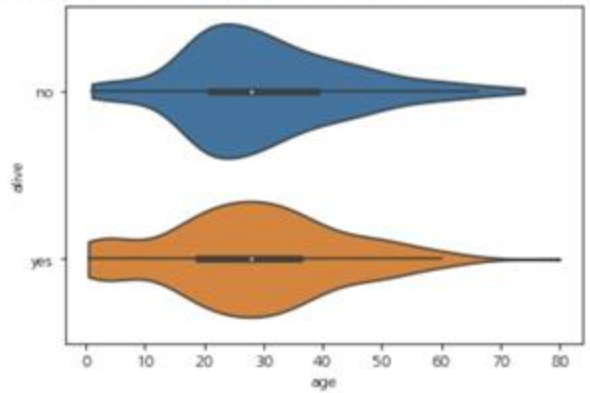


violinplot()

- 값의 표현 범위를 명확히 하기 위해서 cut이라는 키워드 인수를 사용할 수 있다.
- cut 키워드 인수에 0 값을 작성하면 가지고 있는 데이터의 범위를 초과해서 그래프가 표현되지 않는다. 0 이외의 실수 값을 입력하면 대역폭(bandwidth)의 크기가 됩니다.

```
1  # Prevent the density from smoothing beyond the limits of the data
2  sns.violinplot(data=df, x="age", y="alive", cut=0)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7cbfe880>
```

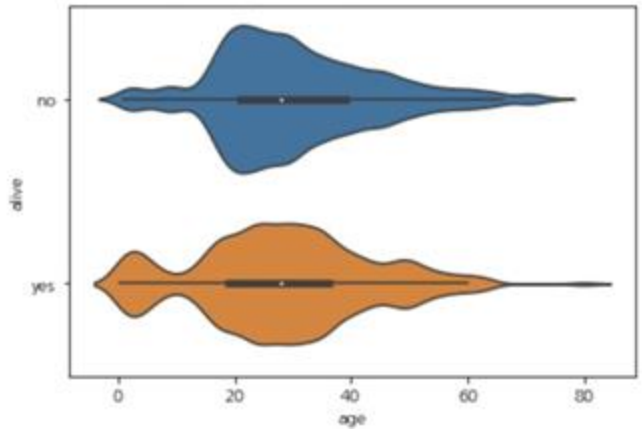


violinplot()

- kde의 그래프 모양을 부드럽게 처리하다보면 대역폭의 크기가 커지는데, 이를 줄이기 위해선 bw 키워드 인수에 소수 값을 갖는 비율을 설정해주면 됨.

```
1  # Use a narrower bandwidth to reduce the amount of smoothing
2  sns.violinplot(data=df, x="age", y="alive", bw=.15)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7ab02520>
```

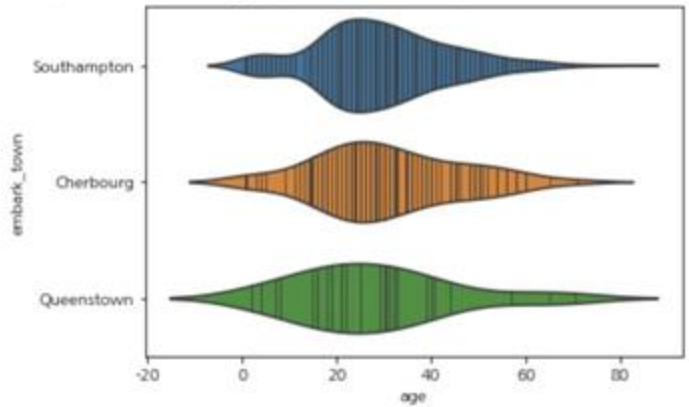


violinplot()

- violin plot 내부에 데이터의 분포를 직접적으로 표현할 수도 있다. inner 키워드 인수에 stick이란 값을 전달하면 됨.
- 아래의 예제를 보면 inner 키워드 인수에 stick이란 값을 전달하여 데이터의 분포를 그래프 안에 표현했음.

```
1  # Represent every observation inside the distribution
2  sns.violinplot(data=df, x="age", y="embark_town", inner="stick")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7aa49a30>
```

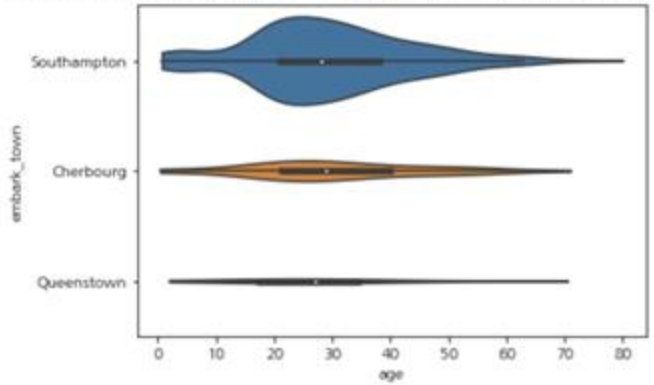


violinplot()

- scale에 따라 각 그래프의 크기가 달라진다. scale은 총 area, count, width의 값을 지정할 수 있으며 기본 값은 area이다.
- area는 violin의 영역 너비가 같게 그려진다. count는 실제 갖는 데이터 개수에 따라 그 violin의 크기가 달라진다. width는 violin이 동일한 폭을 갖게 된다.

```
1  # Use a different scaling rule for normalizing the density
2  sns.violinplot(data=df, x="age", y="embark_town", scale="count", cut=0)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fef7a00b100>
```



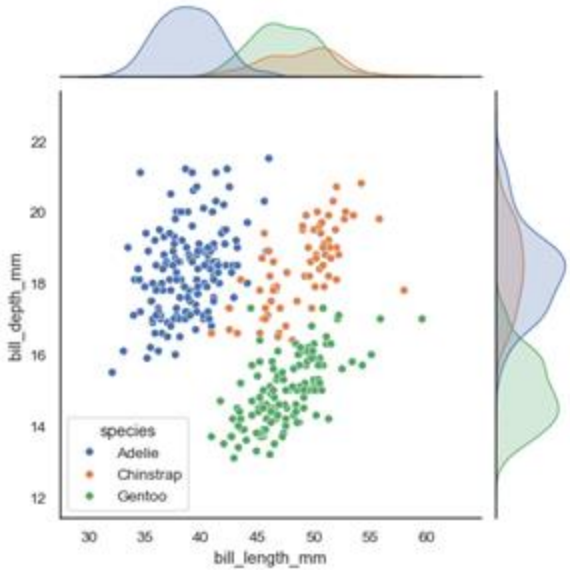
joinplot()

- 두 개의 변수의 분포를 나타낼 때 활용하면 좋은 플롯이다.
- histogram과 scatter plot을 동시에 사용해서 시각적 효과를 표현.

seaborn.jointplot

```
seaborn.jointplot(data=None, *, x=None, y=None, hue=None, kind='scatter',
height=6, ratio=5, space=0.2, dropna=False, xlim=None, ylim=None,
color=None, palette=None, hue_order=None, hue_norm=None,
marginal_ticks=False, joint_kws=None, marginal_kws=None, **kwargs)
```

Draw a plot of two variables with bivariate and univariate graphs.

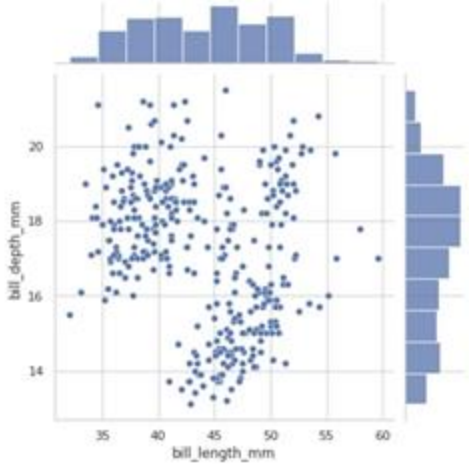


joinplot()

- jointplot() 메서드를 사용하면 작성할 수 있다.
- 이때 data 키워드 인수로 penguins의 DataFrame을 전달.
  - x에는 DataFrame의 column name인 bill_length_mm을 전달.
  - y에는 DataFrame의 column name인 bill_depth_mm을 전달.

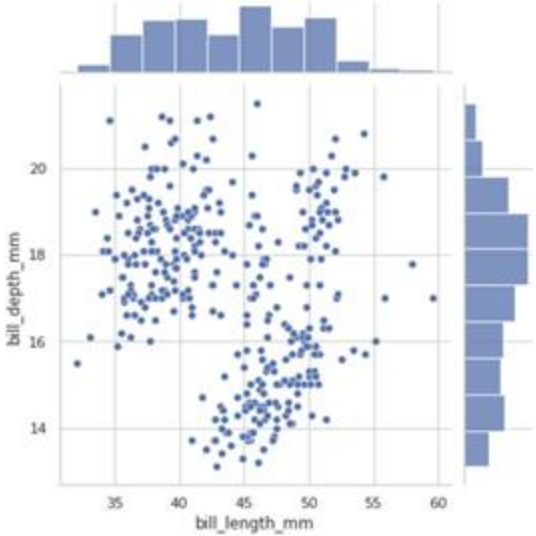
```
# In the simplest invocation,
# assign x and y to create a scatterplot (using scatterplot())
# with marginal histograms (using histplot())
penguins = sns.load_dataset("penguins")
sns.jointplot(data=penguins, x="bill_length_mm", y="bill_depth_mm")
```

```
<seaborn.axisgrid.JointGrid at 0x7f0c682f3070>
```



joinplot()

- histplot()과 scatterplot()을 동시에 사용해서 데이터의 분포를 더 자세하고 직관적으로 파악해볼 수 있다. 하지만 아래 차트로는 어떤 특징을 찾기 어렵습니다.

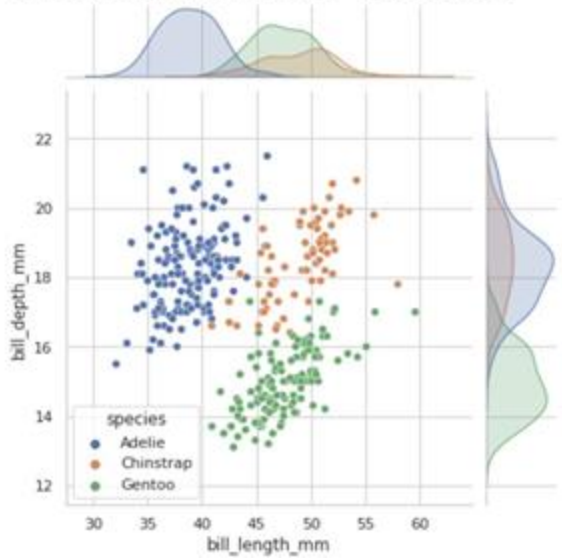


joinplot()

- **hue 키워드 인수를 활용**하여 species column에 대한 변수를 추가.
- 앞의 그래프와 달리 **scatter plot에 species별로 색상이 다르게 적용**되었고 **히스토그램으로 표현된 분포가 자동적으로 kde로 변경**된 것을 확인할 수 있다.
- 펭귄의 종에 따른 분포가 나뉘는 것을 **시각적으로 쉽게 확인**할 수 있게 되었다.

```
1  # Assigning a hue variable will add conditional colors to the scatterplot
2  # and draw separate density curves (using kdeplot()) on the marginal axes
3  sns.jointplot(data=penguins, x="bill_length_mm", y="bill_depth_mm", hue="species")
```

```
<seaborn.axisgrid.JointGrid at 0x7f0c552d0a00>
```

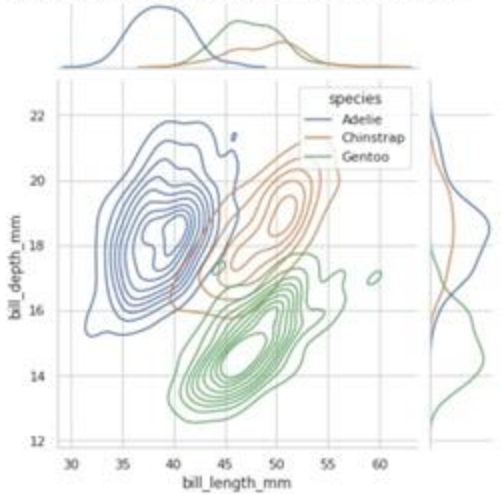


joinplot()

- kind 키워드 인수를 하나 추가하고 그 값으로 kde를 전달했다.
- 그 결과로 scatter 분포가 kde의 형태를 띄면서 값의 밀도가 더 눈에 잘 보이도록 변경된 것을 확인할 수 있다.

```
1  # Several different approaches to plotting are available through the kind parameter.
2  # Setting kind="kde" will draw both bivariate and univariate KDEs
3  sns.jointplot(data=penguins, x="bill_length_mm", y="bill_depth_mm", hue="species", kind="kde")
```

```
<seaborn.axisgrid.JointGrid at 0x7f0c4e4e1820>
```

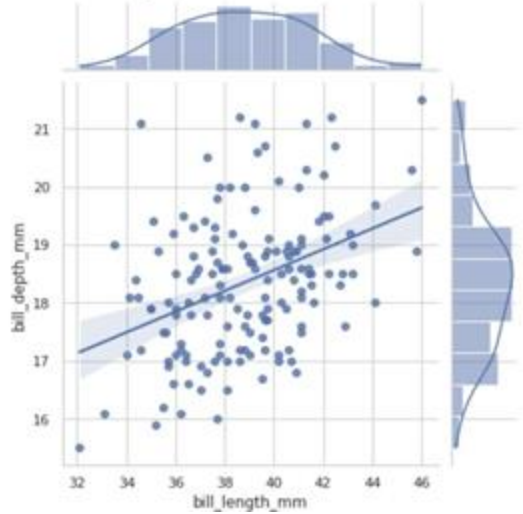


joinplot()

- kind 키워드 인수에 reg 값을 전달해서 선형 회귀에 대한 결과를 그래프로 바로 볼 수 있다.
- 우측 예제에서는 아델리 펭귄에 대한 데이터를 추린 후 아델리 펭귄의 부리에 대한 선형 결과임.
- hue와 kind='reg'는 동시에 사용될 수 없는 옵션임.

```
# Set kind="reg" to add a linear regression fit (using regplot())
# and univariate KDE curves
adelie_penguins = penguins[penguins["species"] == 'Adelie']
sns.jointplot(data=adelie_penguins, x="bill_length_mm", y="bill_depth_mm", kind="reg")
```

```
<seaborn.axisgrid.JointGrid at 0x7f0c4b9ba2b0>
```



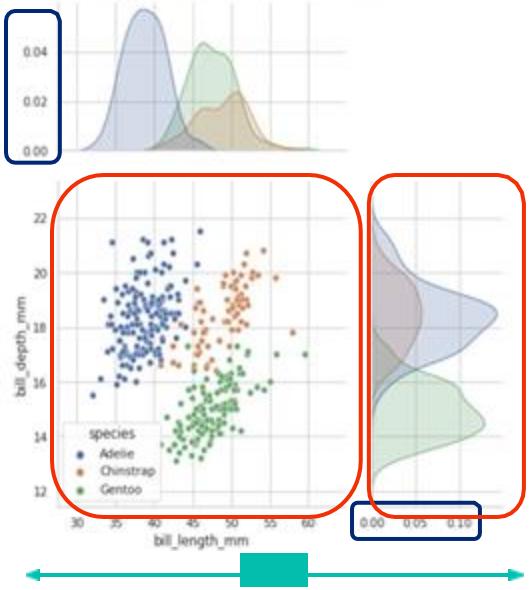
joinplot()

height, ratio, marginal_ticks

- height : 전체 그래프의 크기(height),
- ratio : main과 marginal 간의 비율(ratio),
- marginal_ticks : marginal histogram에 ticks을 표현할지 조절할 수 있다.

```
1  # Use JointGrid parameters to control the size and layout of the figure
2  sns.jointplot(data=penguins, x="bill_length_mm", y="bill_depth_mm", hue="species",
3                height=7, ratio=2, marginal_ticks=True)
```

```
<seaborn.axisgrid.JointGrid at 0x7f0c4b2405b0>
```



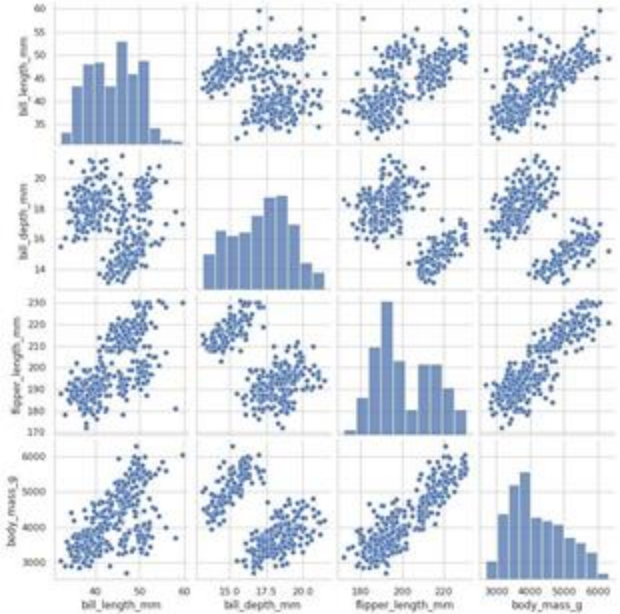
pairplot()

- 데이터셋 내의 각 column마다 pair로 경우에 수에 따라 묶고 그에 따른 결과를 일목요연하게 한번에 확인할 수 있는 플롯이다.

seaborn.pairplot

```
seaborn.pairplot(data, *, hue=None, hue_order=None, palette=None,
vars=None, x_vars=None, y_vars=None, kind='scatter', diag_kind='auto',
markers=None, height=2.5, aspect=1, corner=False, dropna=False,
plot_kws=None, diag_kws=None, grid_kws=None, size=None)
```

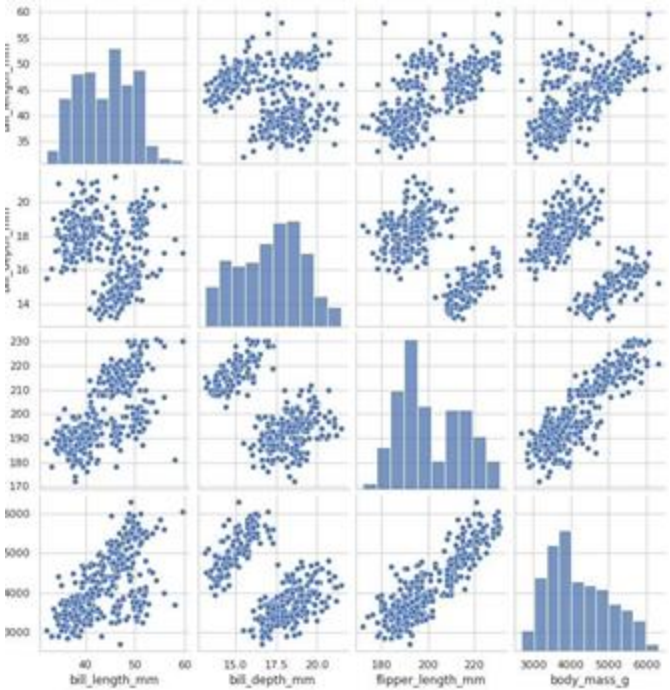
Plot pairwise relationships in a dataset.



pairplot()

- pairplot()으로 서로 다른 column 간에 비교할 때 가장 단순하게 적용하기 좋은 것이 scatterplot()이다.
- 같은 column이 겹치는 위치에는 histplot() 의 결과를 보여줘서 값의 밀도가 어떻게 되는 지 보여줌.

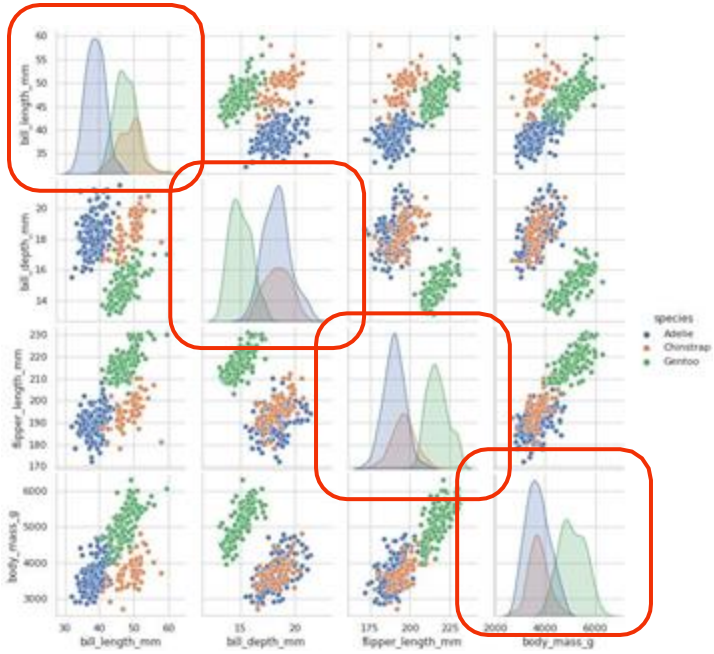
```
1  # The simplest invocation uses scatterplot()
2  # for each pairing of the variables and histplot()
3  # for the marginal plots along the diagonal
4  penguins = sns.load_dataset("penguins")
5  sns.pairplot(penguins)
```



pairplot()

- 의미 있는 관계로 엮기 위해서 hue 키워드 인수에 species를 전달하여 펭귄의 종마다 어떤 분포를 갖는지 분류를 하였음.
- joint plot 때와 같게도 자동적으로 marginal plot의 형태가 kde로 변경된 것을 확인할 수 있다.

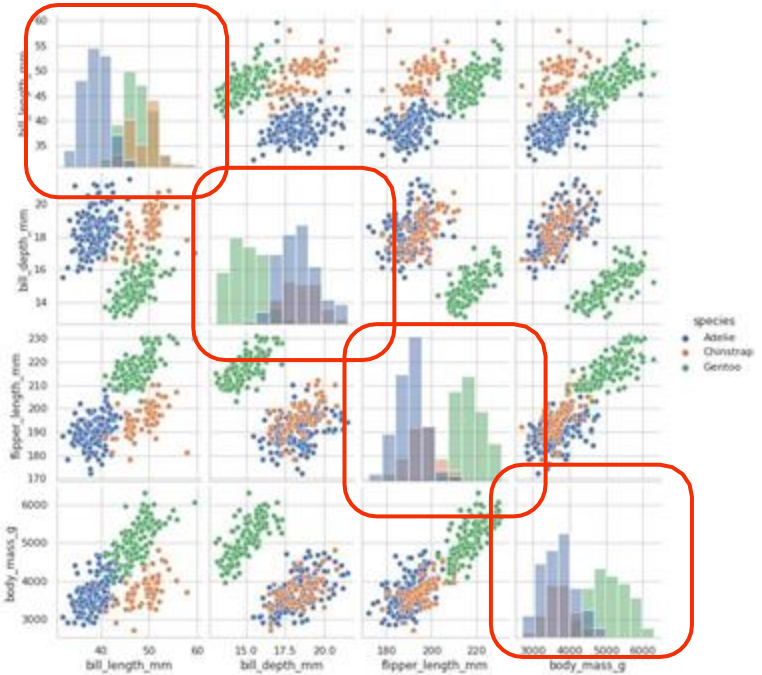
```
1  # Assigning a hue variable adds a semantic mapping
2  # and changes the default marginal plot
3  # to a layered kernel density estimate (KDE)
4  sns.pairplot(penguins, hue="species")
```



pairplot()

- marginal plot이 kde가 아닌 histogram으로 표현할 수도 있다. diag_kind 키워드 인수를 사용하면 된다.
- histogram을 의미하는 hist 문자열을 값으로 전달하면 된다.

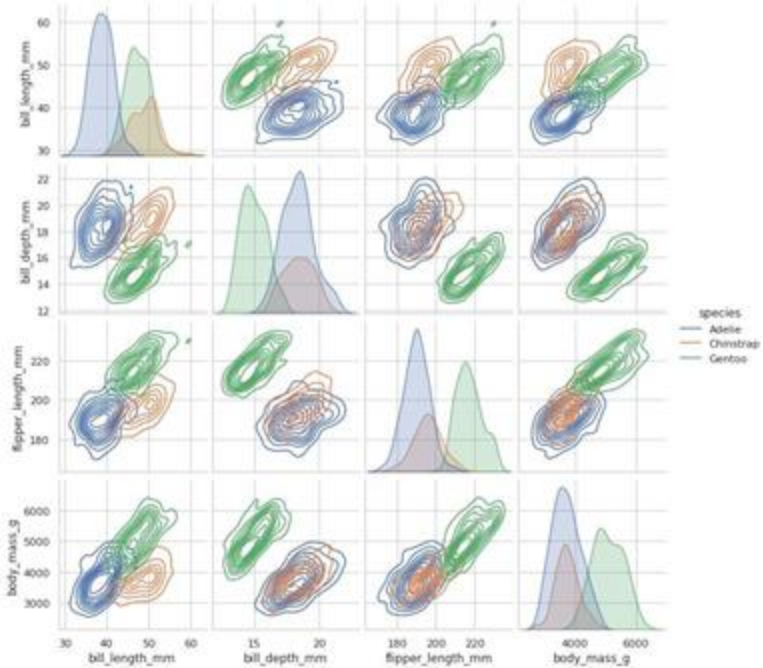
```
1  # It's possible to force marginal histograms
2  sns.pairplot(penguins, hue="species", diag_kind="hist")
```



pairplot()

- 다른 column과의 비교하는 영역의 플롯의 종류도 변경할 수 있다.
- 여기서는 kind라는 키워드 인수에 kde를 전달하여 서로 다른 column 간에 결과를 더 보기좋게 비교할 수 있게 하였음.

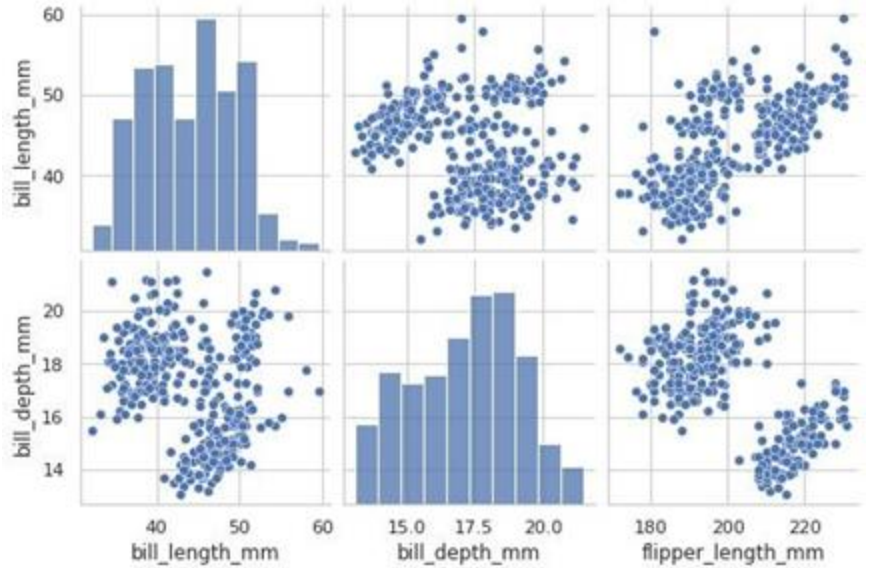
```
# The kind parameter determines both the diagonal
# and off-diagonal plotting style.
# Several options are available, including using kdeplot() to draw KDEs
sns.pairplot(penguins, hue="species", kind="kde")
```



pairplot()

- x_vars, y_vars 키워드 인수를 활용하여 원하는 column만 추려서 비교를 할 수도 있다.
- 꼭 정방형의 모양이 아니어도 괜찮음.

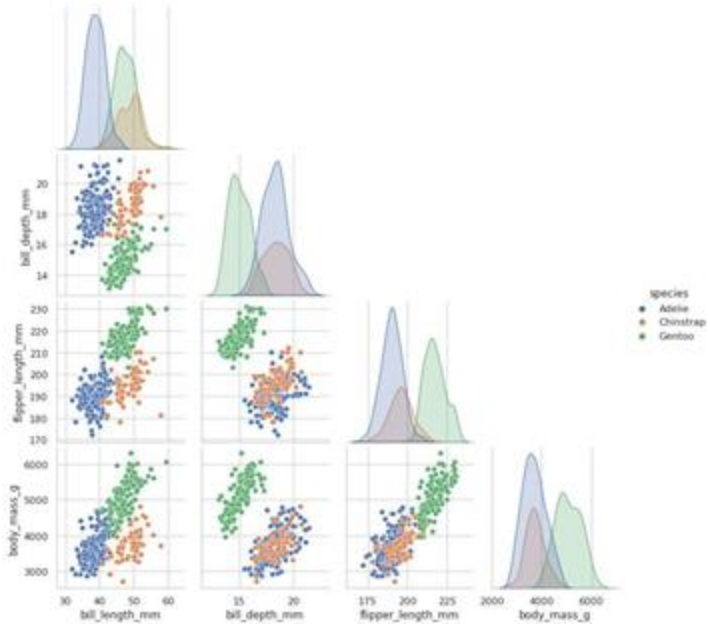
```
# Use vars or x_vars and y_vars to select the variables to plot
sns.pairplot(
    penguins,
    x_vars=["bill_length_mm", "bill_depth_mm", "flipper_length_mm"],
    y_vars=["bill_length_mm", "bill_depth_mm"],
)
```



pairplot()

- 중복을 제거하고 싶다면 corner라는 키워드 인수를 True로 설정하면 된다.
- 그럼 삼각형 모양의 중복 없는 결과를 얻을 수 있음.

```
1  # Set corner=True to plot only the lower triangle
2  sns.pairplot(penguins, hue="species", corner=True)
```



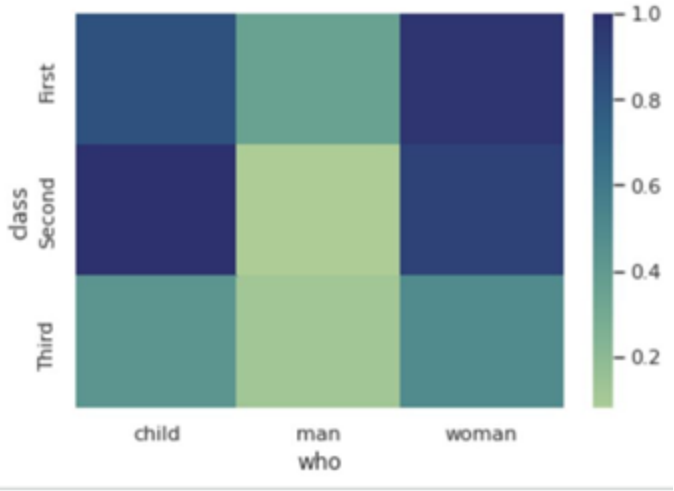
heatmap()

- 데이터셋 중 비교하고자 하는 column의 pair가 모두 카테고리 값이거나, 피벗 테이블의 결과를 가지고 heatmap()으로 표현하면 각 column 간의 상관 관계를 색상의 짙고 옅음으로 표현할 수 있다.

seaborn.heatmap

```
seaborn.heatmap(data, *, vmin=None, vmax=None, cmap=None, center=None,
robust=False, annot=None, fmt='.2g', annot_kws=None, linewidths=0,
linecolor='white', cbar=True, cbar_kws=None, cbar_ax=None, square=False,
xticklabels='auto', yticklabels='auto', mask=None, ax=None, **kwargs)
```

Plot rectangular data as a color-encoded matrix.

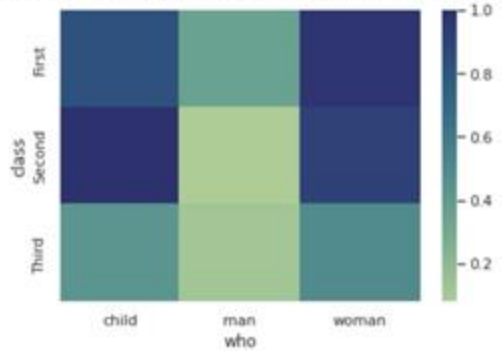


heatmap()

- 타이타닉호 데이터셋 중 선실 등급과 사람의 분류에 따라 생존율을 피벗 테이블의 결과로 만들었다.
- 그리고 이 피벗테이블을 heatmap()의 인수로 전달, 또 cmap을 활용하여 "crest" 테마를 적용. 짙은 푸른 빛을 띌수록 생존율이 높고 노란 빛에 가까울수록 생존율이 낮다.

```
1  # Pass a DataFrame to plot with indices as row/column labels
2  ttn_cls_who = pd.pivot_table(titanic, "survived", index="class", columns="who")
3  sns.heatmap(ttn_cls_who, cmap="crest")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c47b93e50>
```



heatmap()

- 같은 데이터에 이번에는 다른 키워드 인수들을 추가하여 봄.
- **annot** 키워드 인수에 True 값을 줘서 각 cell마다 value를 표기했고,
- 동시에 **fmt** 키워드 인수를 사용해서 cell마다 표기한 value의 포맷을 지정해줬음.
- 뿐만 아니라 **linewidth를** 명시하여 cell 간의 구분을 더 명확하게 지정.

```
1  # Use annot to represent the cell values with text
2  # Control the annotations with a formatting string
3  # Add lines between cells
4  sns.heatmap(ttn_cls_who, cmap="mako", annot=True, fmt=".2f", linewidth=.5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0c47b15310>
```