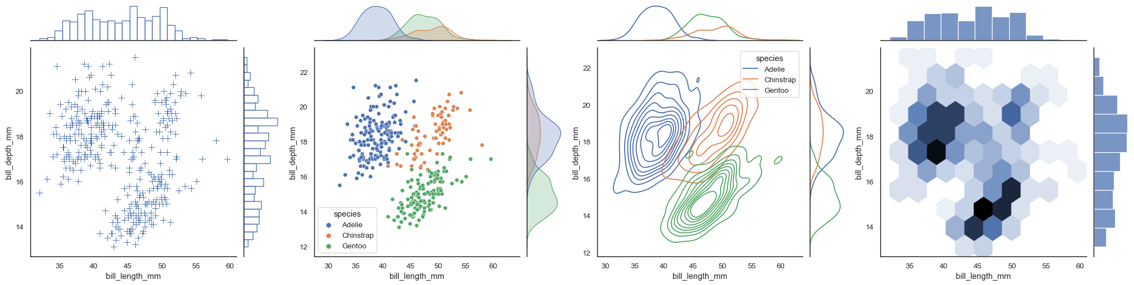
**2. seaborn + matplotlib을 이용한 jointplot 보완**

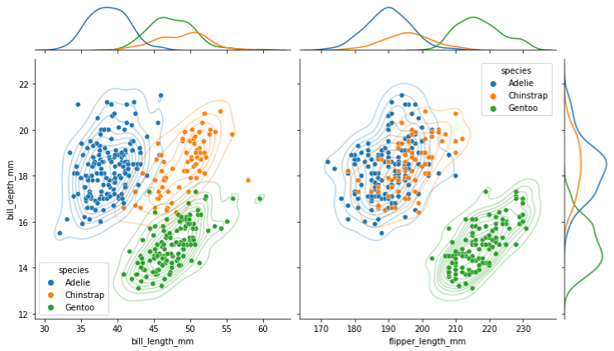
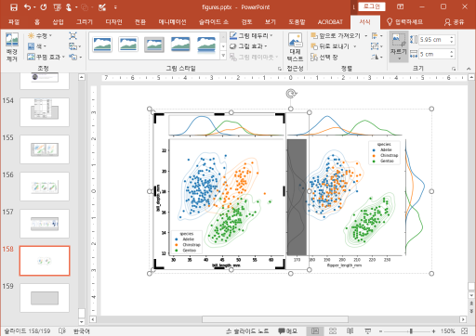
* seaborn을 matplotlib과 섞어쓰는 방법입니다.
* 4부 중 두 번째 시간입니다.
* seaborn jointplot의 단점을 보완합니다.

**2.1. seaborn jointplot**

[*seaborn jointplot*](https://seaborn.pydata.org/generated/seaborn.jointplot.html)

* seaborn의 jointplot은 매력적인 기능입니다.
  + 두 변수 **각각의 분포**와 **2차원 분포**를 함께 보여줍니다.
  + 각각의 분포는 **histogram과 kdeplot**으로 표현할 수 있고
  + 2차원 분포는 **scatterplot, regression, kdeplot, hexbin**등으로 표현할 수 있습니다.
* 만약 matplotlib에서 일일이 만들어야 한다면 눈물이 앞을 가릴 일입니다.
* seaborn에서는 **단 한 줄**로 이런 그림들이 탄생합니다.  
    
  [](https://jehyunlee.github.io/2020/10/03/Python-DS-35-seaborn_matplotlib2/35_s2m2_14.png)jointplot 예제

**2.2. multiple jointplot**

* 그러나 seaborn 만으로는 여러 jointplot이 합쳐진 그림을 **표현할 수 없습니다.**
* y축을 공유시키는 것 만으로도 훨씬 기능이 강력해질텐데요.  
    
  [](https://jehyunlee.github.io/2020/10/03/Python-DS-35-seaborn_matplotlib2/35_s2m2_15.png)multiple jointplot
* 현실적으로 가장 빠른 방법은 **파워포인트**입니다.
* 같은 y축 범위로 그림을 여러 개 그린 후 붙이면 됩니다.  
    
  [](https://jehyunlee.github.io/2020/10/03/Python-DS-35-seaborn_matplotlib2/35_s2m2_16.png)파워포인트 작업
* 그러나 수작업이 동반되고, 오류 가능성이 커집니다.
* seaborn과 matplotlib의 힘을 합해서 그려봅시다.

**2.3. matplotlib + seaborn**

* [지난 글](https://jehyunlee.github.io/2020/09/30/Python-DS-34-seaborn_matplotlib/)에서 **seaborn과 matplotlib이 섞일 수 있음**을 보였습니다.
* **matplotlib으로 틀을 만들고 여기에 seaborn을 삽입**합니다.

|  |  |
| --- | --- |
| 1 2 3 4 5 | import matplotlib.pyplot as plt import seaborn as sns  penguins = sns.load\_dataset("penguins") # 펭귄 데이터셋으로 시작합니다. penguins.head() |

* 가로축에 두 개의 데이터 : bill\_length\_mm, flipper\_length\_mm,
* 세로축에 한 개의 데이터 : bill\_depth\_mm를 놓아보겠습니다.

**2.3.1. 축공간 배열 : gridspec**

[*matplotlib.pyplot.subplots*](https://matplotlib.org/3.3.2/api/_as_gen/matplotlib.pyplot.subplots.html)[*matplotlib.gridspec.GridSpec*](https://matplotlib.org/api/_as_gen/matplotlib.gridspec.GridSpec.html#matplotlib.gridspec.GridSpec)

* 저는 matplotlib을 객체지향 방식으로 사용할 때 이렇게 합니다.
  + fig, ax = plt.subplots() 명령으로 figure와 axes를 만드는데
  + 가로세로 여러 axes를 만들 때는 ncols, nrows 인자를 사용합니다.
  + 그러나 이렇게 하면 **모든 축공간의 크기가 같아집니다.**
* jointplot은 축공간의 크기가 일정하지 않습니다.
* **gridspec**을 이용해서 비대칭 축공간을 만듭니다.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | fig = plt.figure(figsize=(12,7))  widths = [4, 4, 1] heights = [1, 4]  ### 1. gridspec preparation spec = fig.add\_gridspec(ncols=3, nrows=2, width\_ratios=widths, height\_ratios=heights)  ### 2. setting axes axs = {} for i in range(len(heights)\*len(widths)):  axs[i] = fig.add\_subplot(spec[i//len(widths), i%len(widths)])  axs[i].text(0.5, 0.5, f"axs[{i}]", fontdict={"horizontalalignment":"center", "color":"gray"}) |

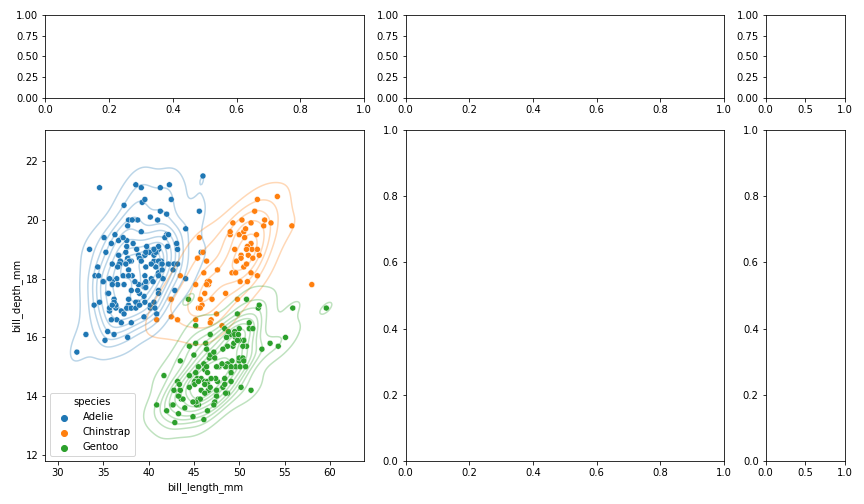
* [](https://jehyunlee.github.io/2020/10/03/Python-DS-35-seaborn_matplotlib2/35_s2m2_3.png)
* 축공간은 원래 2차원 배열입니다.
* 하지만 이를 별로 좋아하지 않아서, dictionary를 사용해 1차원으로 취급합니다.
* 개인적인 취향입니다. 꼭 따르지 않아도 좋습니다.

**2.3.2. 첫 번째 jointplot : scatterplot + kdeplot**

[*seaborn.scatterplot*](https://seaborn.pydata.org/generated/seaborn.scatterplot.html)[*seaborn.kdeplot*](https://seaborn.pydata.org/generated/seaborn.kdeplot.html)

* 좌측 하단에 첫 번째 2차원 분포도를 그립니다.
* x축 변수는 bill\_length\_nm, y축 변수는 bill\_depth\_mm로 지정합니다.
* seaborn 명령의 ax= 인자를 사용해 축공간을 지정합니다.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 | fig = plt.figure(figsize=(12,7))  widths = [4, 4, 1] heights = [1, 4]  # 1. gridspec preparation spec = fig.add\_gridspec(ncols=3, nrows=2, width\_ratios=widths, height\_ratios=heights)  # 2. setting axes axs = {} for i in range(len(heights)\*len(widths)):  axs[i] = fig.add\_subplot(spec[i//len(widths), i%len(widths)])  # 3. bill\_length\_mm vs bill\_depth\_mm # 3.1. kdeplot sns.kdeplot("bill\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", alpha=0.3, ax=axs[3]) # 3.2. scatterplot sns.scatterplot("bill\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", ax=axs[3])  fig.tight\_layout() |

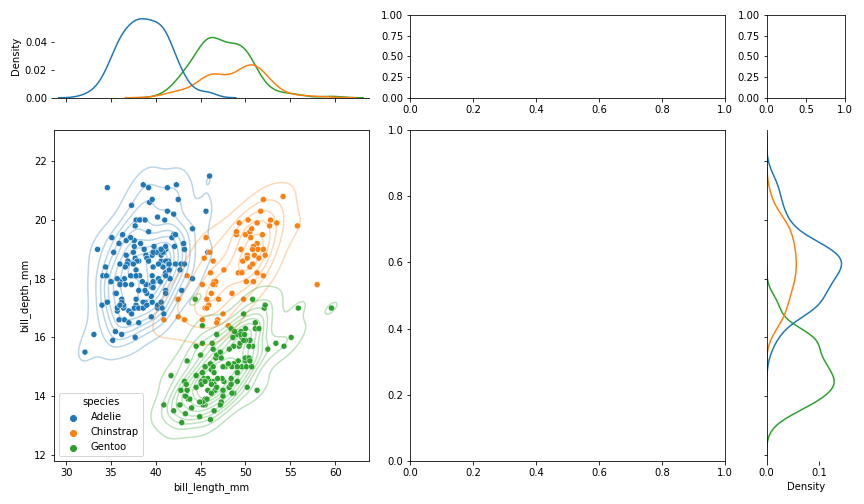
* [](https://jehyunlee.github.io/2020/10/03/Python-DS-35-seaborn_matplotlib2/35_s2m2_4.png)

**2.3.3. 첫 번째 jointplot : 1차원 kdeplot**

* 첫 번째 jointplot의 상단과 우측에 1차원 분포도를 그립니다.
* 두 가지를 주의해야 합니다.

1. 중심이 되는 jointplot과 **가로세로 범위를 일치**시켜야 합니다.
2. 1차원 분포도의 **눈금과 수치는 필요없으니 제거**합니다.

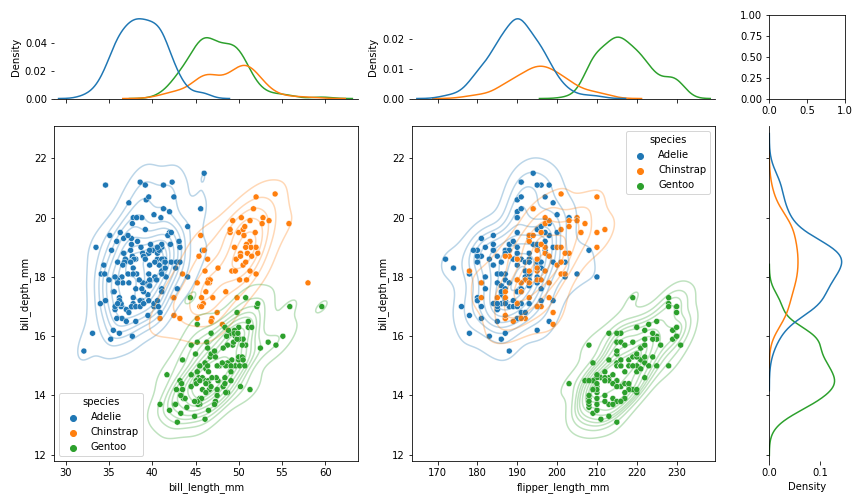
|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 | fig = plt.figure(figsize=(12,7))  widths = [4, 4, 1] heights = [1, 4]  ### 1. gridspec preparation spec = fig.add\_gridspec(ncols=3, nrows=2, width\_ratios=widths, height\_ratios=heights)  ### 2. setting axes axs = {} for i in range(len(heights)\*len(widths)):  axs[i] = fig.add\_subplot(spec[i//len(widths), i%len(widths)])  ### 3. bill\_length\_mm vs bill\_depth\_mm # 3.1. kdeplot sns.kdeplot("bill\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", alpha=0.3, ax=axs[3])  # 3.2. scatterplot sns.scatterplot("bill\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", ax=axs[3])  # 3.3. histogram (bill\_length\_mm) sns.kdeplot("bill\_length\_mm", data=penguins, hue="species", ax=axs[0], legend=False) axs[0].set\_xlim(axs[3].get\_xlim()) axs[0].set\_xlabel('') axs[0].set\_xticklabels([]) axs[0].spines["left"].set\_visible(False) axs[0].spines["top"].set\_visible(False) axs[0].spines["right"].set\_visible(False)  # 3.3. histogram (bill\_depth\_mm) sns.kdeplot(y="bill\_depth\_mm", data=penguins, hue="species", ax=axs[5], legend=False) axs[5].set\_ylim(axs[3].get\_ylim()) axs[5].set\_ylabel('') axs[5].set\_yticklabels([]) axs[5].spines["bottom"].set\_visible(False) axs[5].spines["top"].set\_visible(False) axs[5].spines["right"].set\_visible(False)  fig.tight\_layout() |

1. [](https://jehyunlee.github.io/2020/10/03/Python-DS-35-seaborn_matplotlib2/35_s2m2_6.png)

**2.3.4. 두 번째 jointplot**

* 같은 요령으로 두 번째 분포도 도시합니다.
* 첫 번째 jointplot에서 x 변수만 바꿔주면 됩니다.

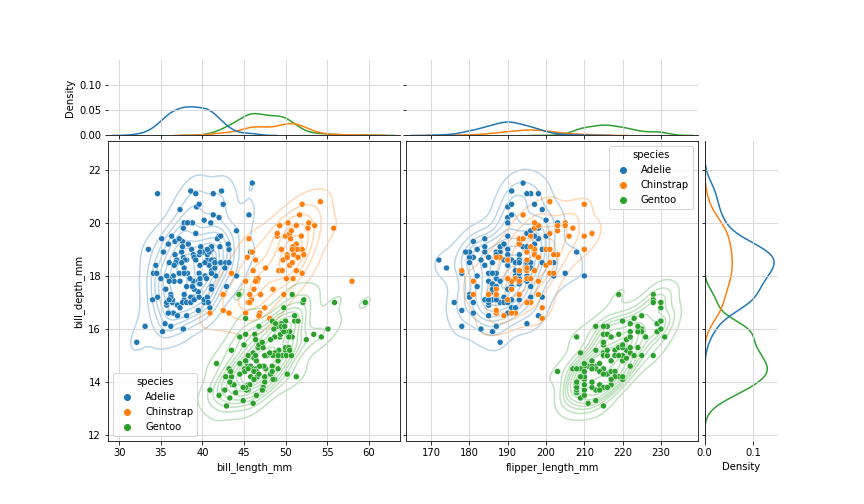
|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 | fig = plt.figure(figsize=(12,7))  widths = [4, 4, 1] heights = [1, 4]  ### 1. gridspec preparation spec = fig.add\_gridspec(ncols=3, nrows=2, width\_ratios=widths, height\_ratios=heights)  ### 2. setting axes axs = {} for i in range(len(heights)\*len(widths)):  axs[i] = fig.add\_subplot(spec[i//len(widths), i%len(widths)])  ### 3. bill\_length\_mm vs bill\_depth\_mm # 3.1. kdeplot sns.kdeplot("bill\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", alpha=0.3, ax=axs[3])  # 3.2. scatterplot sns.scatterplot("bill\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", ax=axs[3])  # 3.3. histogram (bill\_length\_mm) sns.kdeplot("bill\_length\_mm", data=penguins, hue="species", ax=axs[0], legend=False) axs[0].set\_xlim(axs[3].get\_xlim()) axs[0].set\_xlabel('') axs[0].set\_xticklabels([]) axs[0].spines["left"].set\_visible(False) axs[0].spines["top"].set\_visible(False) axs[0].spines["right"].set\_visible(False)  # 3.3. histogram (bill\_depth\_mm) sns.kdeplot(y="bill\_depth\_mm", data=penguins, hue="species", ax=axs[5], legend=False) axs[5].set\_ylim(axs[3].get\_ylim()) axs[5].set\_ylabel('') axs[5].set\_yticklabels([]) axs[5].spines["bottom"].set\_visible(False) axs[5].spines["top"].set\_visible(False) axs[5].spines["right"].set\_visible(False)  ### 4. flipper\_length\_mm vs bill\_depth\_mm # 3.1. kdeplot sns.kdeplot("flipper\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", alpha=0.3, ax=axs[4])  # 3.2. scatterplot sns.scatterplot("flipper\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", ax=axs[4])  # 3.3. histogram (bill\_length\_mm) sns.kdeplot("flipper\_length\_mm", data=penguins, hue="species", ax=axs[1], legend=False) axs[1].set\_xlim(axs[4].get\_xlim()) axs[1].set\_xlabel('') axs[1].set\_xticklabels([]) axs[1].spines["left"].set\_visible(False) axs[1].spines["top"].set\_visible(False) axs[1].spines["right"].set\_visible(False)  fig.tight\_layout() |

* [](https://jehyunlee.github.io/2020/10/03/Python-DS-35-seaborn_matplotlib2/35_s2m2_7.png)

**2.3.5. 부대효과 조정 : spines, grids, 간격**

* 데이터는 모두 올라갔으니 부대 효과를 조정합니다.
* 불필요한 요소(ex. spines)는 제거하고, 애매한 요소(ex. 위치)는 grid로 명확히 합니다.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 | fig = plt.figure(figsize=(12,7))  widths = [4, 4, 1] heights = [1, 4]  ### 1. gridspec preparation spec = fig.add\_gridspec(ncols=3, nrows=2, width\_ratios=widths, height\_ratios=heights,   wspace=0.03, hspace=0.03) # setting spaces  ### 2. setting axes axs = {} for i in range(len(heights)\*len(widths)):  axs[i] = fig.add\_subplot(spec[i//len(widths), i%len(widths)])  ### 3. bill\_length\_mm vs bill\_depth\_mm # 3.1. kdeplot sns.kdeplot("bill\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", alpha=0.3, ax=axs[3], zorder=1)  # 3.2. scatterplot sns.scatterplot("bill\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", ax=axs[3], zorder=2)  # 3.3. histogram (bill\_length\_mm) sns.kdeplot("bill\_length\_mm", data=penguins, hue="species", ax=axs[0], legend=False, zorder=1) axs[0].set\_xlim(axs[3].get\_xlim()) axs[0].set\_xlabel('') axs[0].set\_xticklabels([]) axs[0].spines["left"].set\_visible(False) axs[0].spines["top"].set\_visible(False) axs[0].spines["right"].set\_visible(False)  # 3.3. histogram (bill\_depth\_mm) sns.kdeplot(y="bill\_depth\_mm", data=penguins, hue="species", ax=axs[5], legend=False, zorder=1) axs[5].set\_ylim(axs[3].get\_ylim()) axs[5].set\_ylabel('') axs[5].set\_yticklabels([]) axs[5].spines["bottom"].set\_visible(False) axs[5].spines["top"].set\_visible(False) axs[5].spines["right"].set\_visible(False)  ### 4. flipper\_length\_mm vs bill\_depth\_mm # 4.1. kdeplot sns.kdeplot("flipper\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", alpha=0.3, ax=axs[4], zorder=1)  # 4.2. scatterplot sns.scatterplot("flipper\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", ax=axs[4], zorder=2)  # 4.3. histogram (flipper\_length\_mm) sns.kdeplot("flipper\_length\_mm", data=penguins, hue="species", ax=axs[1], legend=False, zorder=1) axs[1].set\_xlim(axs[4].get\_xlim()) axs[1].set\_xlabel('') axs[1].set\_xticklabels([]) axs[1].spines["left"].set\_visible(False) axs[1].spines["top"].set\_visible(False) axs[1].spines["right"].set\_visible(False)  ### 5. unnecessary elements removal # 5.1. upper-right axes axs[2].axis("off")  # 5.2. margin kdeplot scale unification hist\_range\_max = max(axs[0].get\_ylim()[-1], axs[1].get\_ylim()[-1], axs[5].get\_xlim()[-1]) for i in range(len(widths)-1):  axs[i].set\_ylim(0, hist\_range\_max) axs[5].set\_xlim(0, hist\_range\_max)  # 5.3. redundent labels and titles removal axs[1].set\_yticklabels([]) axs[1].set\_ylabel('') axs[4].set\_yticklabels([]) axs[4].set\_ylabel('')  # 5.4. grids for i in range(len(heights)\*len(widths)):  axs[i].grid("on", color="lightgray", zorder=0)  fig.tight\_layout() |

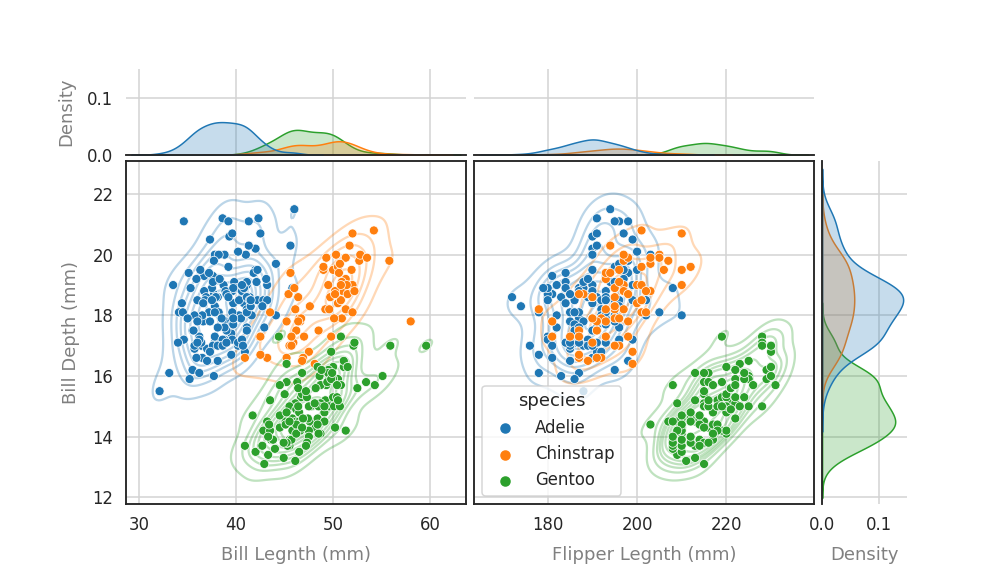
* [](https://jehyunlee.github.io/2020/10/03/Python-DS-35-seaborn_matplotlib2/35_s2m2_9.png)
* 1차원 분포의 스케일도 맞춰주었습니다.
  + 그러나 꼭 필요한 작업인지는 의구심이 듭니다.
  + **‘밀도’**라는 정의에 맞게 넓이를 1로 만드는 과정이 포함되어 있는데, 이로 인해 **x축 스케일이 크면 높이가 낮아지는 경향**이 있기 때문입니다.
  + 데이터의 범위에 집중하고 싶다면 스케일을 맞추지 않는 것이 나을지도 모릅니다.

**2.3.6. 마무리 작업**

[*seaborn tutorial: controlling figure aesthetics*](https://seaborn.pydata.org/tutorial/aesthetics.html)

* 데이터를 표현하는 작업은 사실상 완료되었습니다.
* 폰트의 크기, 색상 등 가독성을 높입니다.
* 이 때도 seaborn의 set\_style()과 set\_context()를 사용하면 편리합니다.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 | sns.set\_style("white") sns.set\_context("talk")  fig = plt.figure(figsize=(14,8))  widths = [4, 4, 1] heights = [1, 4]  ### 1. gridspec preparation spec = fig.add\_gridspec(ncols=3, nrows=2, width\_ratios=widths, height\_ratios=heights,   wspace=0.03, hspace=0.03) # setting spaces  ### 2. setting axes axs = {} for i in range(len(heights)\*len(widths)):  axs[i] = fig.add\_subplot(spec[i//len(widths), i%len(widths)])  ### 3. bill\_length\_mm vs bill\_depth\_mm # 3.1. kdeplot sns.kdeplot("bill\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", alpha=0.3, ax=axs[3], zorder=1, legend=False)  # 3.2. scatterplot sns.scatterplot("bill\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", ax=axs[3], zorder=2, legend=False)  # 3.3. histogram (bill\_length\_mm) sns.kdeplot("bill\_length\_mm", data=penguins, hue="species", ax=axs[0], legend=False, zorder=1, fill=True) axs[0].set\_xlim(axs[3].get\_xlim()) axs[0].set\_xlabel('') axs[0].set\_xticklabels([]) axs[0].spines["left"].set\_visible(False) axs[0].spines["top"].set\_visible(False) axs[0].spines["right"].set\_visible(False)  # 3.3. histogram (bill\_depth\_mm) sns.kdeplot(y="bill\_depth\_mm", data=penguins, hue="species", ax=axs[5], legend=False, zorder=1, fill=True) axs[5].set\_ylim(axs[3].get\_ylim()) axs[5].set\_ylabel('') axs[5].set\_yticklabels([]) axs[5].spines["bottom"].set\_visible(False) axs[5].spines["top"].set\_visible(False) axs[5].spines["right"].set\_visible(False)  ### 4. flipper\_length\_mm vs bill\_depth\_mm # 4.1. kdeplot sns.kdeplot("flipper\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", alpha=0.3, ax=axs[4], zorder=1)  # 4.2. scatterplot sns.scatterplot("flipper\_length\_mm", "bill\_depth\_mm", data=penguins, hue="species", ax=axs[4], zorder=2)  # 4.3. histogram (flipper\_length\_mm) sns.kdeplot("flipper\_length\_mm", data=penguins, hue="species", ax=axs[1], legend=False, zorder=1, fill=True) axs[1].set\_xlim(axs[4].get\_xlim()) axs[1].set\_xlabel('') axs[1].set\_xticklabels([]) axs[1].spines["left"].set\_visible(False) axs[1].spines["top"].set\_visible(False) axs[1].spines["right"].set\_visible(False)  ### 5. unnecessary elements removal # 5.1. upper-right axes axs[2].axis("off")  # 5.2. margin kdeplot scale unification hist\_range\_max = max(axs[0].get\_ylim()[-1], axs[1].get\_ylim()[-1], axs[5].get\_xlim()[-1]) for i in range(len(widths)-1):  axs[i].set\_ylim(0, hist\_range\_max) axs[5].set\_xlim(0, hist\_range\_max)  # 5.3. redundent labels and titles removal axs[1].set\_yticklabels([]) axs[1].set\_ylabel('') axs[4].set\_yticklabels([]) axs[4].set\_ylabel('')  # 5.4. grids for i in range(len(heights)\*len(widths)):  axs[i].grid("on", color="lightgray", zorder=0)   # 5.5. labels font\_label = {"color":"gray"} axs[3].set\_xlabel("Bill Legnth (mm)", fontdict=font\_label, labelpad=12) axs[3].set\_ylabel("Bill Depth (mm)", fontdict=font\_label, labelpad=12) axs[4].set\_xlabel("Flipper Legnth (mm)", fontdict=font\_label, labelpad=12) axs[0].set\_ylabel("Density", fontdict=font\_label, labelpad=12) axs[5].set\_xlabel("Density", fontdict=font\_label, labelpad=12)   fig.tight\_layout() |

* [](https://jehyunlee.github.io/2020/10/03/Python-DS-35-seaborn_matplotlib2/35_s2m2_10.png)
* jointplot의 한계인 **2중 jointplot이 구현되었습니다.**

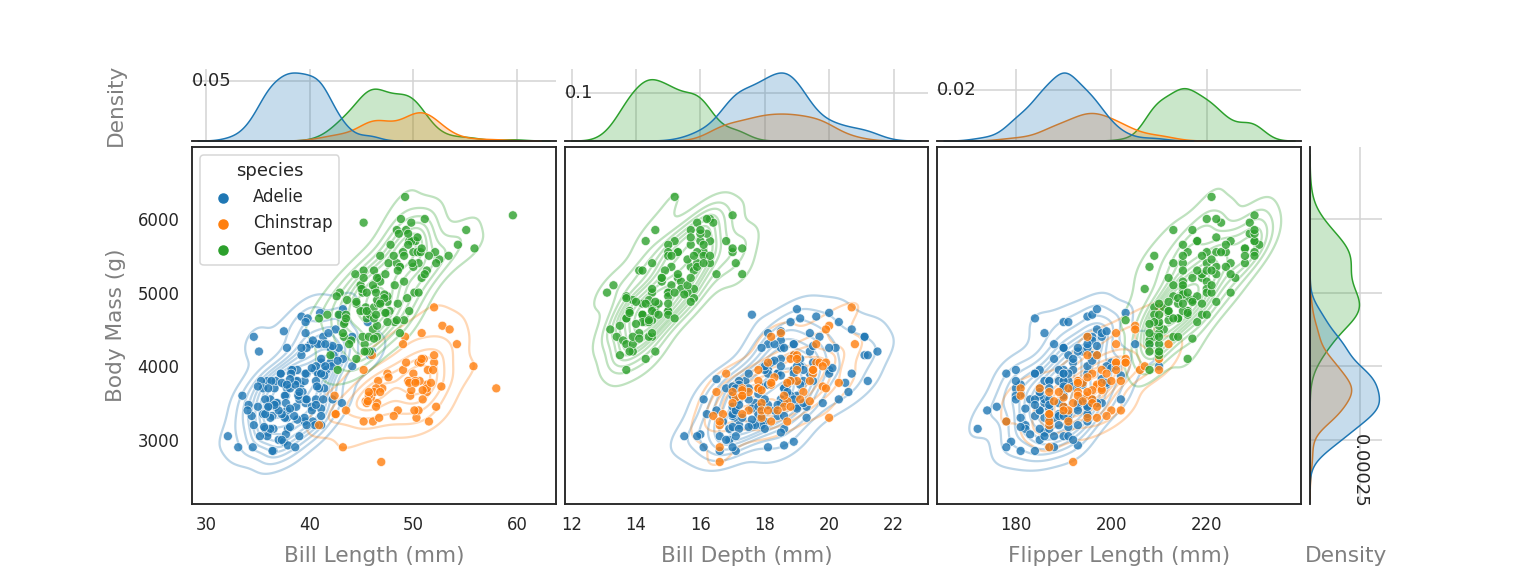
**2.3.7. generalize : 함수로 만들기**

* 하는 김에, 기능을 일반화합시다.
* 다중 jointplot 제작 기능을 함수로 만들고,
* x 변수의 수와 그림 크기, 1차원 분포 스케일 통일 등을 인자로 만듭니다.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 | def jointplots(xs, y, data, hue=None, height=6, ratio=5, space=0.2, xlabels=None, ylabel=None, margin\_norm=False):  """  -------------------  Input Parameters  -------------------  xs : (list or str) feature name(s) of data  y : (str) feature name of data  data : (pandas.DataFrame)  hue : (str) semantic variable that is mapped to determine the color of plot elements. Semantic variable that is mapped to determine the color of plot elements.    height : (float) size of the figure  ratio : (float) ratio of the joint axes height to marginal axes height.  space : (float) space between the joint and marginal axes    xlabels : (list or str) xlabels  ylabel : (str) ylabel  margin\_norm : (boolean) if True, kdeplots at marginal axes have same scale.  """  ### 1. input check  # input type  assert isinstance(xs, list) or isinstance(xs, str)  if isinstance(xs, list):  assert all([isinstance(x, str) for x in xs])  else:  xs = [xs]      if xlabels != None:  assert isinstance(xlabels, list) or isinstance(xlabels, str)  if isinstance(xlabels, list):  assert all([isinstance(xlabel, str) for xlabel in xlabels])  else:  xlabels = [xlabels]    if ylabel != None:  assert isinstance(ylabel, str)    if hue != None:  assert isinstance(hue, str)    # input data  assert all([x in data.columns for x in xs])  assert y in data.columns  if hue != None:  assert hue in data.columns      ### 2. figure  h\_margin = height / (ratio + 1)  h\_joint = height - h\_margin    if isinstance(xs, list):  n\_x = len(xs)  else:  n\_x = 1    widths = [h\_joint] \* n\_x + [h\_margin]  heights = [h\_margin, h\_joint]  ncols = len(widths)  nrows = len(heights)    fig = plt.figure(figsize=(sum(widths), sum(heights)))    ### 3. gridspec preparation  spec = fig.add\_gridspec(ncols=ncols, nrows=nrows,   width\_ratios = widths, height\_ratios = heights,  wspace=space, hspace=space  )    ### 4. setting axes  axs = {}  for i in range(ncols \* nrows):  axs[i] = fig.add\_subplot(spec[i//ncols, i%ncols])      ### 5. jointplots (scatterplot + kdeplot)  for i, x in enumerate(xs, ncols):  if i == ncols:  legend=True  else:  legend=False  sns.kdeplot(x=x, y=y, data=data, hue=hue, alpha=0.3, ax=axs[i], zorder=2, legend=False)  sns.scatterplot(x=x, y=y, data=data, hue=hue, alpha=0.8, ax=axs[i], zorder=3, legend=legend)      ### 6. kdeplots at marginal axes  axs[ncols-1].axis("off")    axes\_mx = list(range(ncols-1))  axes\_my = 2\*ncols - 1    for i, x in zip(axes\_mx, xs):  sns.kdeplot(x=x, data=data, hue=hue, fill=True, ax=axs[i], zorder=2, legend=False)  axs[i].set\_xlim(axs[i+ncols].get\_xlim())  axs[i].set\_xlabel("")  axs[i].set\_xticklabels([])  axs[i].spines["left"].set\_visible(False)  axs[i].spines["top"].set\_visible(False)  axs[i].spines["right"].set\_visible(False)    sns.kdeplot(y=y, data=data, hue=hue, fill=True, ax=axs[axes\_my], zorder=2, legend=False)  axs[axes\_my].set\_ylim(axs[ncols].get\_ylim())  axs[axes\_my].set\_ylabel("")  axs[axes\_my].set\_yticklabels([])  axs[axes\_my].spines["bottom"].set\_visible(False)  axs[axes\_my].spines["top"].set\_visible(False)  axs[axes\_my].spines["right"].set\_visible(False)    if margin\_norm == True:  hist\_range\_max = max([axs[m].get\_ylim()[-1] for m in axes\_mx] + [axs[axes\_my].get\_xlim()[-1]])  for i in axes\_mx:  axs[i].set\_ylim(0, hist\_range\_max)  axs[axes\_my].set\_xlim(0, hist\_range\_max)      ### 7. unnecessary elements removal  # 7.1. labels and ticklabels  axes\_j = list(range(ncols, 2\*ncols-1))  for i in axes\_j:  if i != ncols:  axs[i].set\_ylabel("")  axs[i].set\_yticklabels([])    # 7.2. marginal axes  for i in axes\_mx:  if i != 0:  axs[i].set\_ylabel("")    axs[i].grid("on", color="lightgray", zorder=0)  axs[i].set\_yticklabels([])    yticks = axs[i].get\_yticks()  ylim = axs[i].get\_ylim()  for ytick in yticks:  if 0 < ytick < ylim[-1]:  axs[i].text(axs[i].get\_xlim()[0], ytick, str(ytick),   fontdict={"verticalalignment":"center"})    axs[axes\_my].grid("on", color="lightgray", zorder=0)  axs[axes\_my].set\_xticklabels([])  axes\_my\_xticks = axs[axes\_my].get\_xticks()  axes\_my\_xlim = axs[axes\_my].get\_xlim()  for xtick in axes\_my\_xticks:  if 0 < xtick < axes\_my\_xlim[-1]:  axs[axes\_my].text(xtick, axs[axes\_my].get\_ylim()[0], str(xtick),   rotation=270, fontdict={"horizontalalignment":"center"})    # 7.3. labels  font\_label = {"color": "gray", "fontsize":"large"}  labelpad = 12  for i, x in zip(axes\_j, xlabels):  axs[i].set\_xlabel(x, fontdict=font\_label, labelpad=labelpad)  if i == ncols:  axs[i].set\_ylabel(ylabel, fontdict=font\_label, labelpad=labelpad)    axs[0].set\_ylabel("Density", fontdict=font\_label, labelpad=labelpad)  axs[2\*ncols-1].set\_xlabel("Density", fontdict=font\_label, labelpad=labelpad)    fig.align\_ylabels([axs[0], axs[ncols]])  fig.align\_xlabels([axs[x] for x in range(ncols, 2\*ncols)])  plt.tight\_layout()    return fig, axs |

* **X인자의 수가 바뀌어도** jointplot이 안정적으로 그려집니다.
* 1차원 분포도 **표현 방식을 바꾸어** 전보다 깔끔해졌습니다.

|  |  |
| --- | --- |
| 1 2 3 | jointplots(["bill\_length\_mm", "bill\_depth\_mm", "flipper\_length\_mm"], "body\_mass\_g", penguins, hue="species",   height=8, ratio=5, space=0.03,   xlabels=["Bill Length (mm)", "Bill Depth (mm)", "Flipper Length (mm)"], ylabel="Body Mass (g)") |

* [](https://jehyunlee.github.io/2020/10/03/Python-DS-35-seaborn_matplotlib2/35_s2m2_11.png)

**2.4. 결론**

* seaborn의 jointplot을 가져다 쓰는 데 그치지 않았습니다.
* **matplotlib의 객체지향 방식을 이용해 seaborn의 한계를 벗어날 수 있었습니다.**
* 본 예제에서는 scatterplot과 2차원 kdeplot만 결합했습니다.
* 그러나 이 외에도 seaborn과 matplotlib이 제공하는 거의 모든 기능을 결합할 수 있습니다.
* 매뉴얼의 한계에 얽매이지 말고 상상력을 동원해 보시면 어떨까요.