A Guide to Matplotlib Subfigures for Creating Complex Multi-Panel Figures

Subfigures — a powerful tool for beautiful multi-panel figures

Tim Rose

.

Published in

Towards Data Science

.

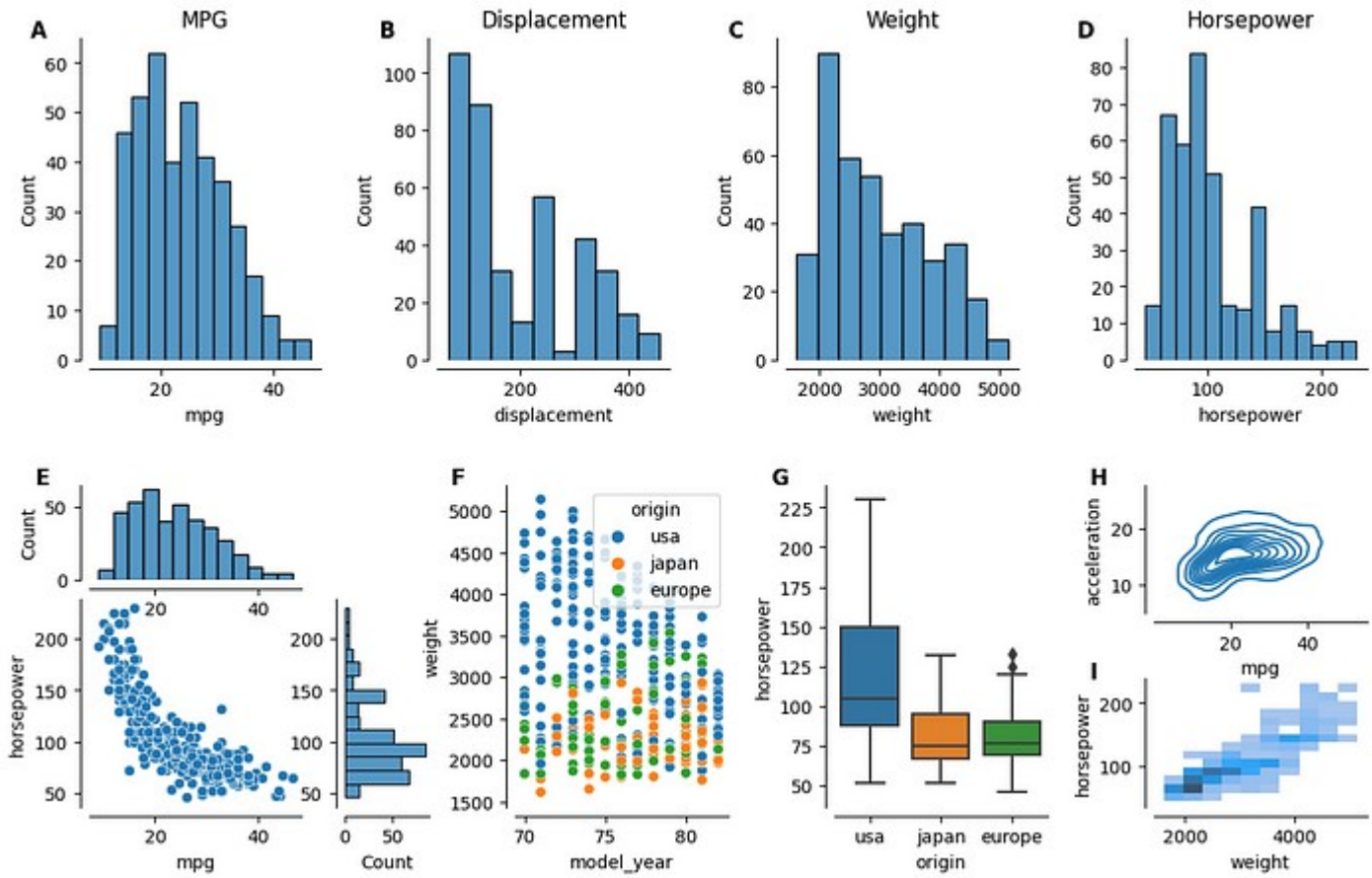
8 min read

.

2 days ago

Motivation

Complex (scientific) figures often consist of multiple plots with different sizes or annotations. If you work with the matplotlib/seaborn ecosystem, there are many ways to create complex figures, e.g. using gridspec.
However, this can get challenging very fast, especially if you want to integrate multi-axes plots from seaborn such as jointplot or pairgrid into your figure because they don't have the option to provide axes as input parameters. But there is another way to assemble figures in matplotlib instead of just working with subplots: **Subfigures**. A powerful framework to create multi-panel figures like this one:



In this article, I will give an introduction to subfigures and their capabilities. We will combine subfigures with subplots and gridspecs to recreate this figure.
To follow this article, you should have a basic understanding of matplotlib subplots and gridspec (if not, you can check out the linked tutorials).

Matplotlib subfigures

First, we import matplotlib, seaborn and load some example data, which we will use to fill the plots with content:

```
import matplotlib.pyplot as plt
import seaborn as sns
data = sns.load_dataset('mpg')
```

Let’s start with the concept of subfigures in matplotlib. To create subfigures, we first need to create a figure:

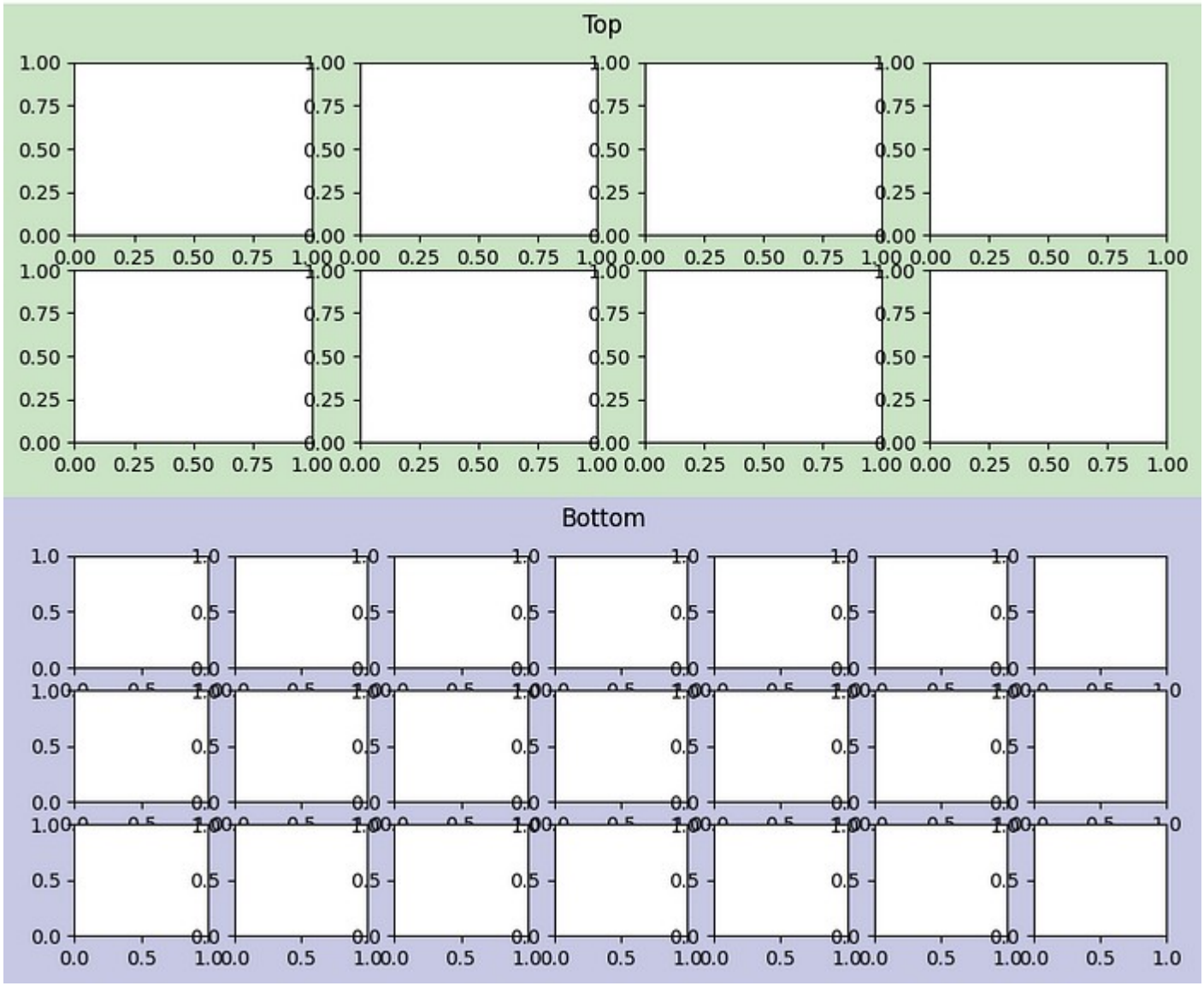
```
fig = plt.figure(figsize=(10, 7))
```

From this point, we can define subfigures similarly to subplots. It is possible to create a grid of subfigures by providing the number of rows (*2*) and columns (*1*). We additionally color the figure backgrounds to highlight them:

```
(topfig, bottomfig) = fig.subfigures(2, 1)
topfig.set_facecolor('#cbe4c6ff')
topfig.suptitle('Top')
bottomfig.set_facecolor('#c6c8e4ff')
bottomfig.suptitle('Bottom')
```

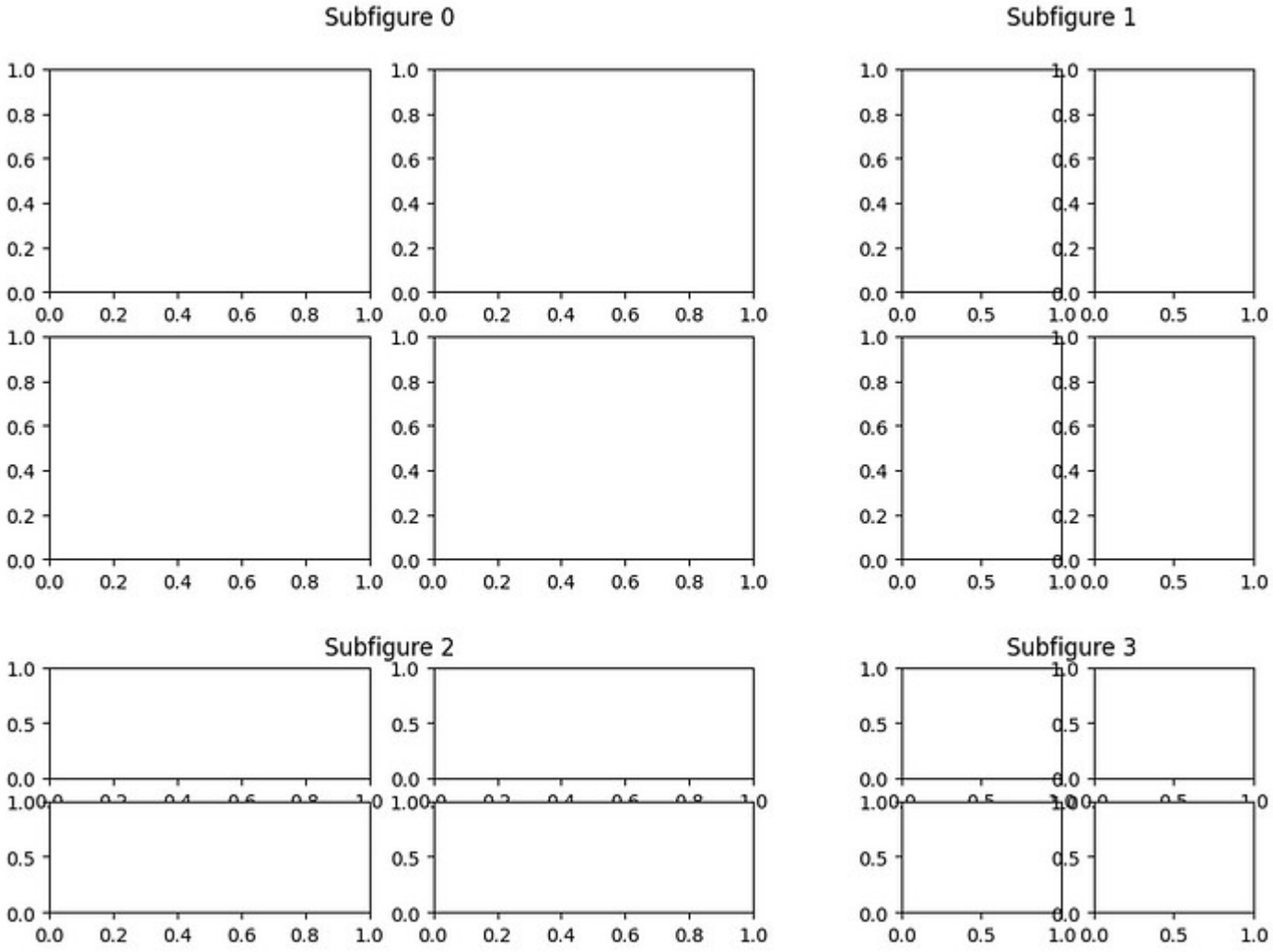
Just figures without any plots (axes) will not be shown, therefore we need to define subplots for each subfigure. Here we can already see one great feature of subfigures, for each subfigure we can define different layouts of subplots:

```
top_axs = topfig.subplots(2, 4)
bottom_axs = bottomfig.subplots(3, 7)
plt.show()
```



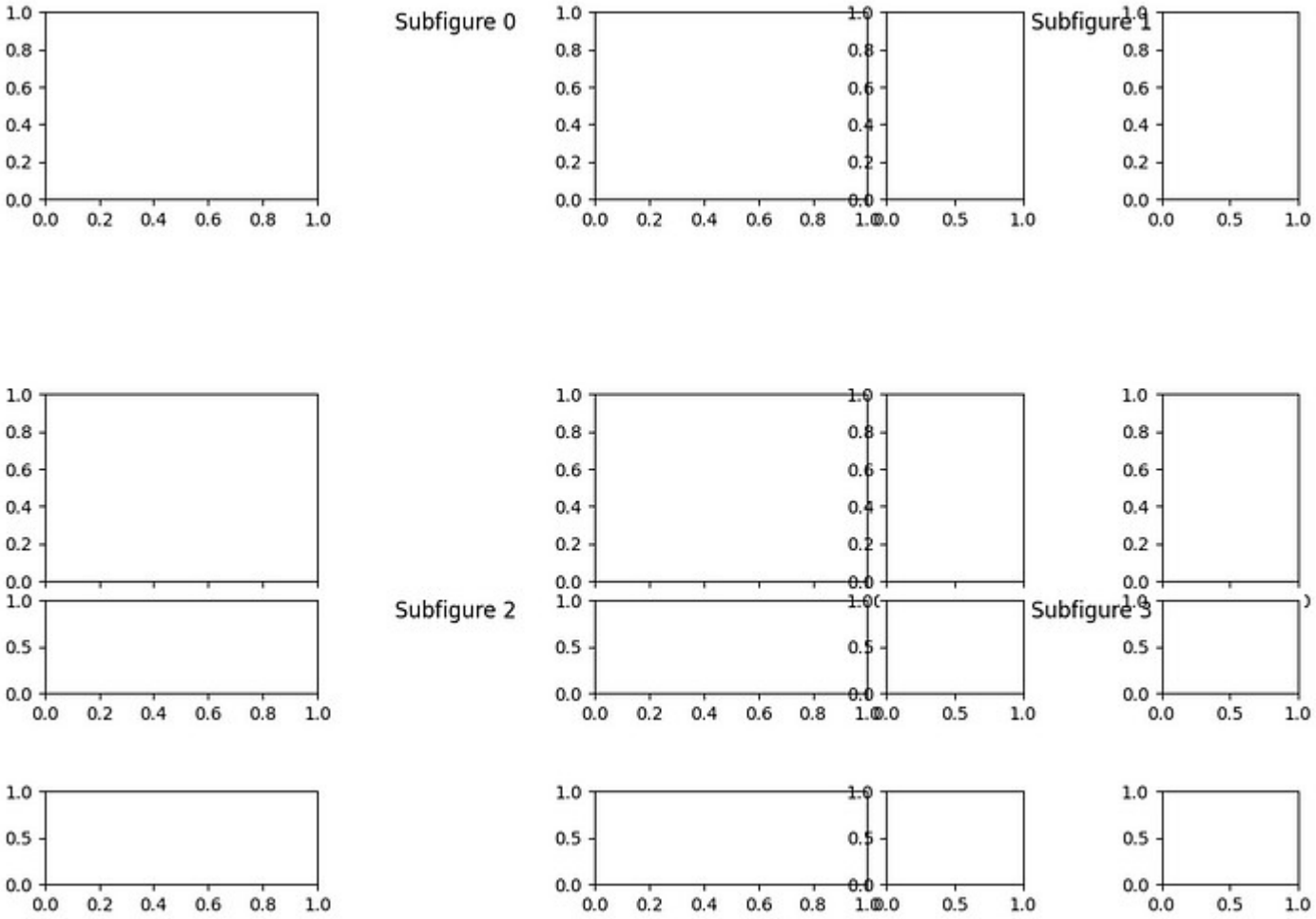
We now have two separate figures that we can set up differently but place together in one final figure. Of course, we can also play with the size ratios of subfigures:

```
figure = plt.figure(figsize=(10, 7))
figs = figure.subfigures(2, 2, height_ratios=(2,1), width_ratios=(2,1))
figs = figs.flatten()
for i, fig in enumerate(figs):
 fig.suptitle(f'Subfigure {i}')
 axs = fig.subplots(2, 2)
plt.show()
```



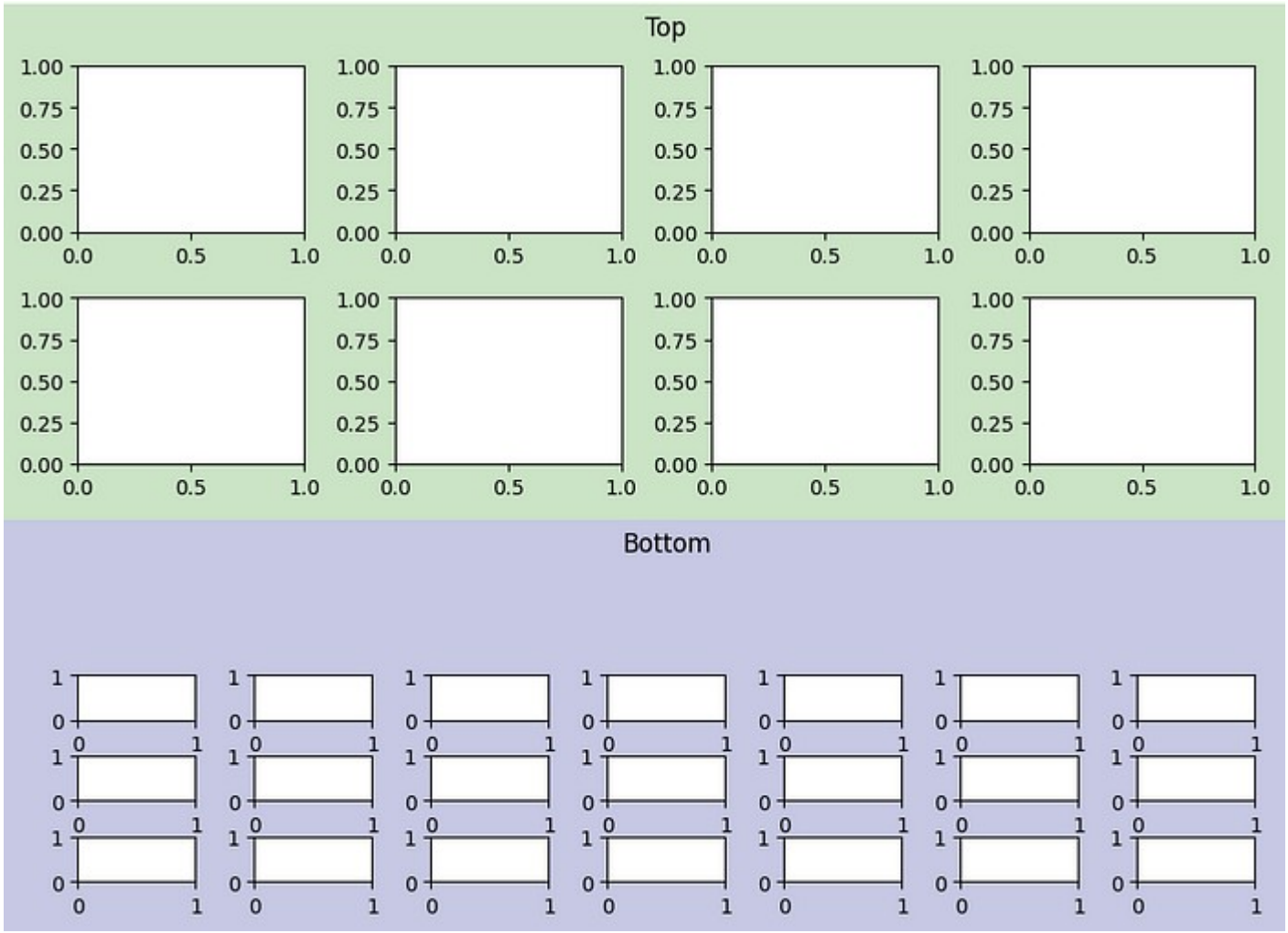
However, there is one drawback of subfigures. To eliminate overlapping labels or elements outside the figure, `plt.tight_layout()` is a good way of squeezing everything nicely into the figure. However, this is not supported for subfigures. Here you can see what happens if you try to use it:

```
figure = plt.figure(figsize=(10, 7))
figs = figure.subfigures(2, 2, height_ratios=(2,1), width_ratios=(2,1))
figs = figs.flatten()
for i, fig in enumerate(figs):
 fig.suptitle(f'Subfigure {i}')
 axs = fig.subplots(2, 2)
plt.tight_layout()
plt.show()
```



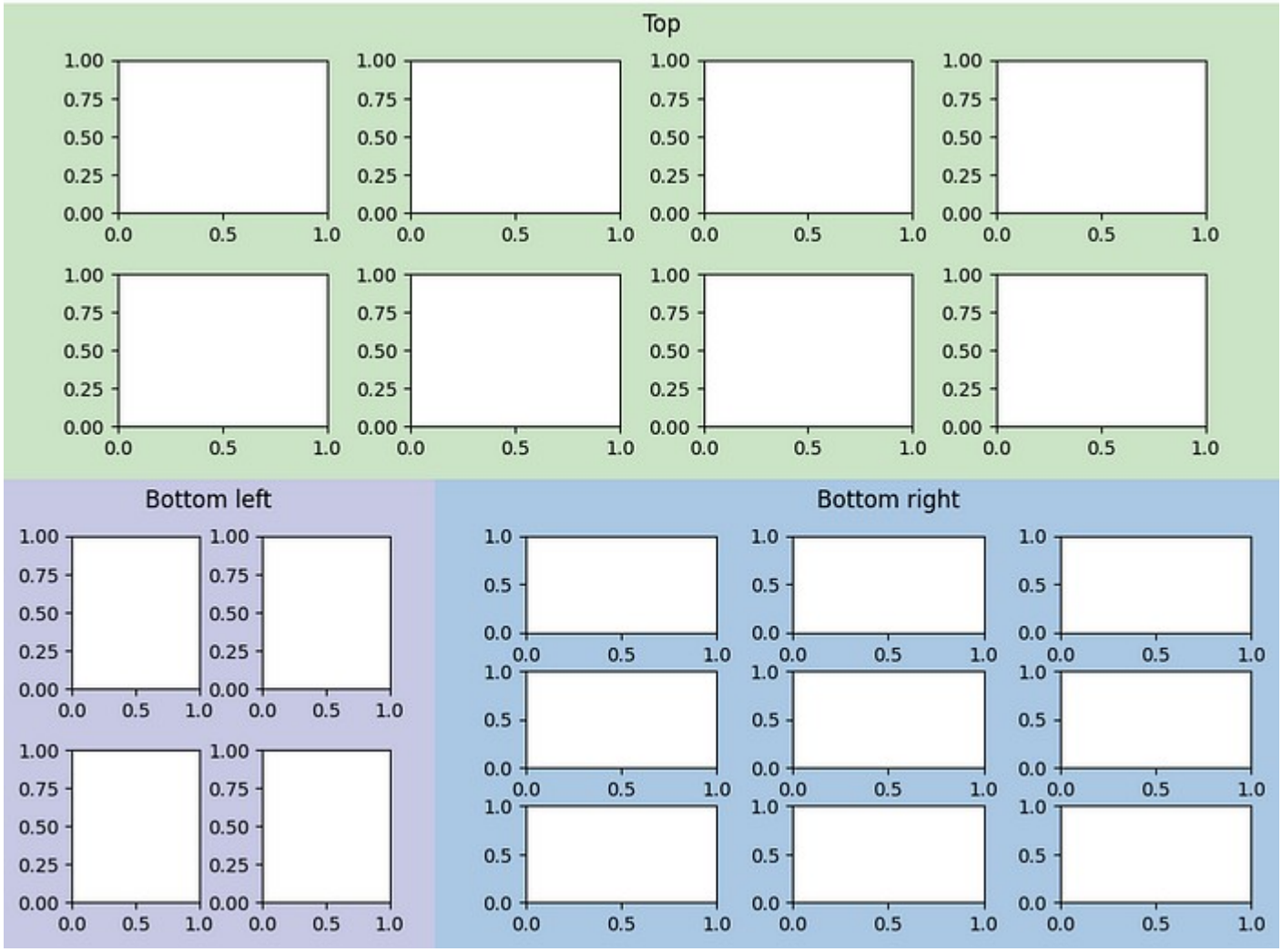
Not really what we intended… To insert spacing between plots and remove any overlaps, we need to use the `subplots_adjust` function, which allows us to insert (or remove) more space between subplots and the borders:

```
fig = plt.figure(figsize=(10, 7))
(topfig, bottomfig) = fig.subfigures(2, 1)
topfig.set_facecolor('#cbe4c6ff')
topfig.suptitle('Top')
bottomfig.set_facecolor('#c6c8e4ff')
bottomfig.suptitle('Bottom')
top_axs = topfig.subplots(2, 4)
bottom_axs = bottomfig.subplots(3, 7)
# Adding more space between plots and reducing the space to the sides
topfig.subplots_adjust(left=.1, right=.9, wspace=.5, hspace=.5)
# We can also squeeze subplots to the bottom
bottomfig.subplots_adjust(wspace=.5, hspace=.8, top=.7, bottom=.3)
plt.show()
```



Another great aspect of subfigures is that they can be nested, meaning we can divide every subfigure into more subfigures:

```
fig = plt.figure(figsize=(10, 7))
(topfig, bottomfig) = fig.subfigures(2, 1)
topfig.set_facecolor('#cbe4c6ff')
topfig.suptitle('Top')
top_axs = topfig.subplots(2, 4)
(bottomleft, bottomright) = bottomfig.subfigures(1, 2, width_ratios=(1,2))
bottomleft.set_facecolor('#c6c8e4ff')
bottomleft.suptitle('Bottom left')
bottom_axs = bottomleft.subplots(2, 2)
bottomright.set_facecolor('#aac8e4ff')
bottomright.suptitle('Bottom right')
bottom_axs = bottomright.subplots(3, 3)
# Spacing between subplots
topfig.subplots_adjust(left=.1, right=.9, wspace=.4, hspace=.4)
bottomleft.subplots_adjust(left=.2, right=.9, wspace=.5, hspace=.4)
bottomright.subplots_adjust(left=.1, right=.9, wspace=.4, hspace=.4)
plt.show()
```

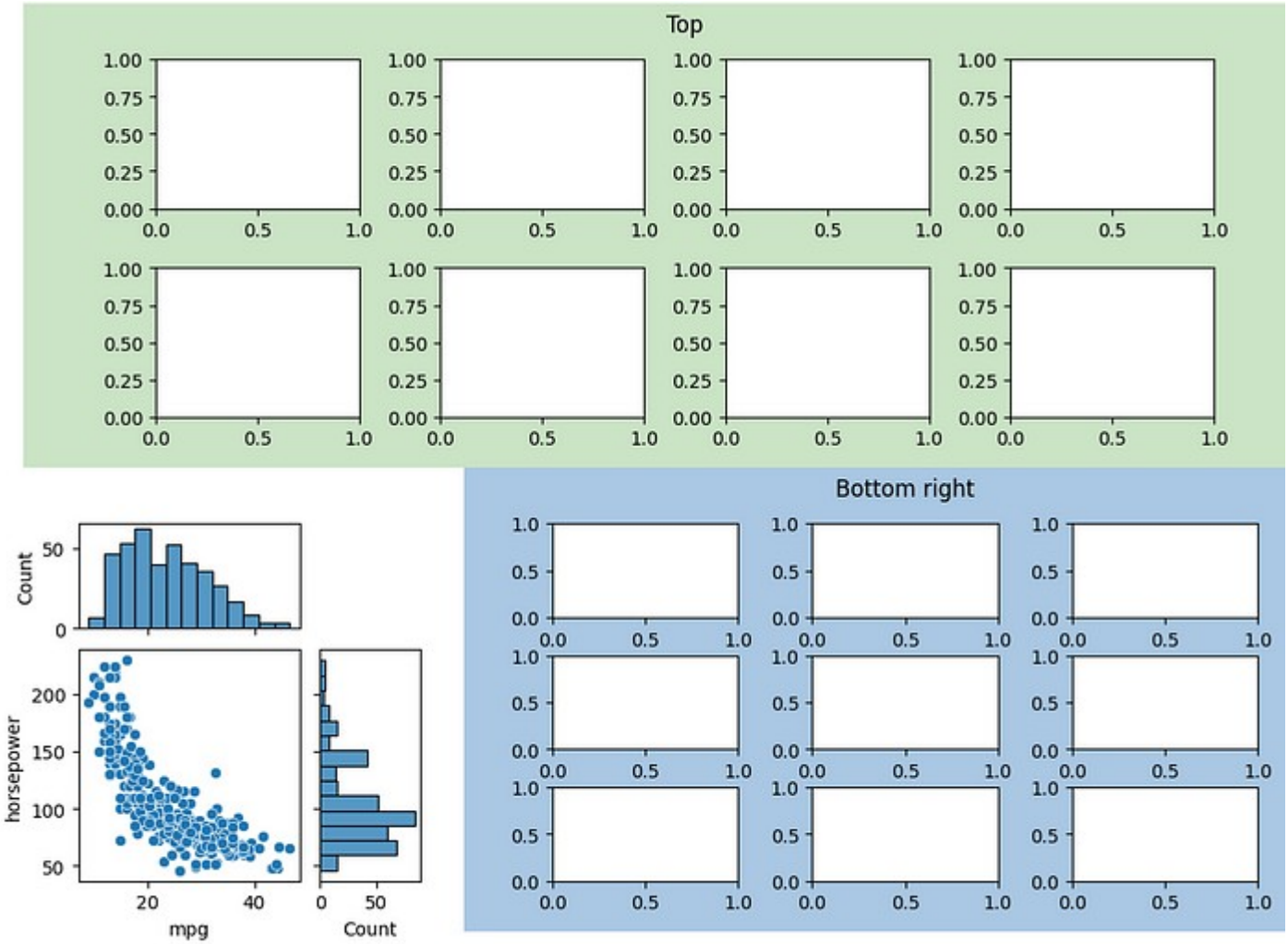


Let's insert a jointplot into this figure. Unfortunately, this is not straightforward, since the seaborn function does not have the possibility to provide a figure object as an input. But if we look into the source code of the function, we can see that this plot consists of three subplots with shared x and y axis that are defined through a gridspec.

This means we can easily plot it inside a subfigure:

```
fig = plt.figure(figsize=(10, 7))
(topfig, bottomfig) = fig.subfigures(2, 1)
topfig.set_facecolor('#cbe4c6ff')
topfig.suptitle('Top')
top_axs = topfig.subplots(2, 4)
# We are using the bottom left subfigure for the jointplot
(bottomleft, bottomright) = bottomfig.subfigures(1, 2, width_ratios=(1,2))
# This parameter defines the size ratio between the main plot and the margin
plots
ratio=2
# Defining a gridspec where the subplots are places
gs = plt.GridSpec(ratio + 1, ratio + 1)
# The main scatterplot
ax_joint  = bottomleft.add_subplot(gs[1:, :-1])
# The margin plots are sharing an axis with the main plot
ax_marg_x = bottomleft.add_subplot(gs[0, :-1], sharex=ax_joint)
ax_marg_y = bottomleft.add_subplot(gs[1:, -1], sharey=ax_joint)
# Axis labels and ticklabels for the margin plots are set to not visible
# Since they are shared with the main plot,
# removing them for the margin will also remove them from the main plot
```

```
plt.setp(ax_marg_x.get_xticklabels(), visible=False)
plt.setp(ax_marg_y.get_yticklabels(), visible=False)
plt.setp(ax_marg_x.get_xticklabels(minor=True), visible=False)
plt.setp(ax_marg_y.get_yticklabels(minor=True), visible=False)
# Filling the plots with data:
sns.scatterplot(data=data, y='horsepower', x='mpg', ax=ax_joint)
sns.histplot(data=data, y='horsepower', ax=ax_marg_y)
sns.histplot(data=data, x='mpg', ax=ax_marg_x)
bottomright.set_facecolor('#aac8e4ff')
bottomright.suptitle('Bottom right')
bottom_axs = bottomright.subplots(3, 3)
# Spacing between subplots
topfig.subplots_adjust(left=.1, right=.9, wspace=.4, hspace=.4)
bottomright.subplots_adjust(left=.1, right=.9, wspace=.4, hspace=.4)
plt.show()
```



You can play with the ratio parameter and see how the plot is changing.

Now, we have all the tools we need to create complex figures, by using subfigure, subplots and grids. For such figures, it is often crucial to annotate each plot with letters to explain them in the caption or reference them in a text. This is often done with other software such as Adobe Illustrator or Inkscape after the figure is created. But we can also do it directly in python, which will save us additional effort later.

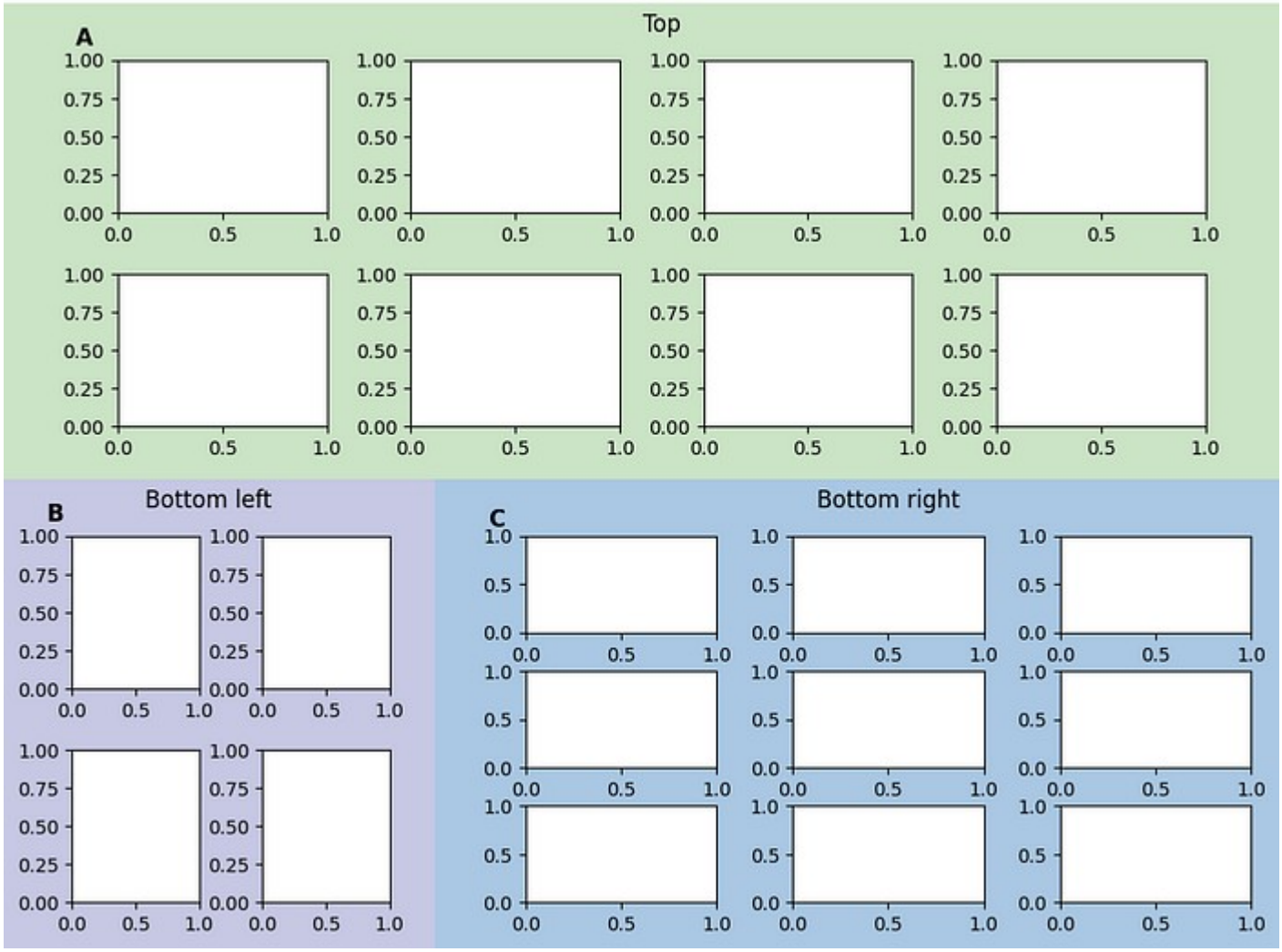
For this, we will define a function to make such annotations:

```
def letter_annotation(ax, xoffset, yoffset, letter):
```

```
 ax.text(xoffset, yoffset, letter, transform=ax.transAxes,
         size=12, weight='bold')
```

The function takes an axes as an input, together with x and y coordinates, which will transformed into relative axes coordinates. We can use this to annotate some plots in our previously created figure:

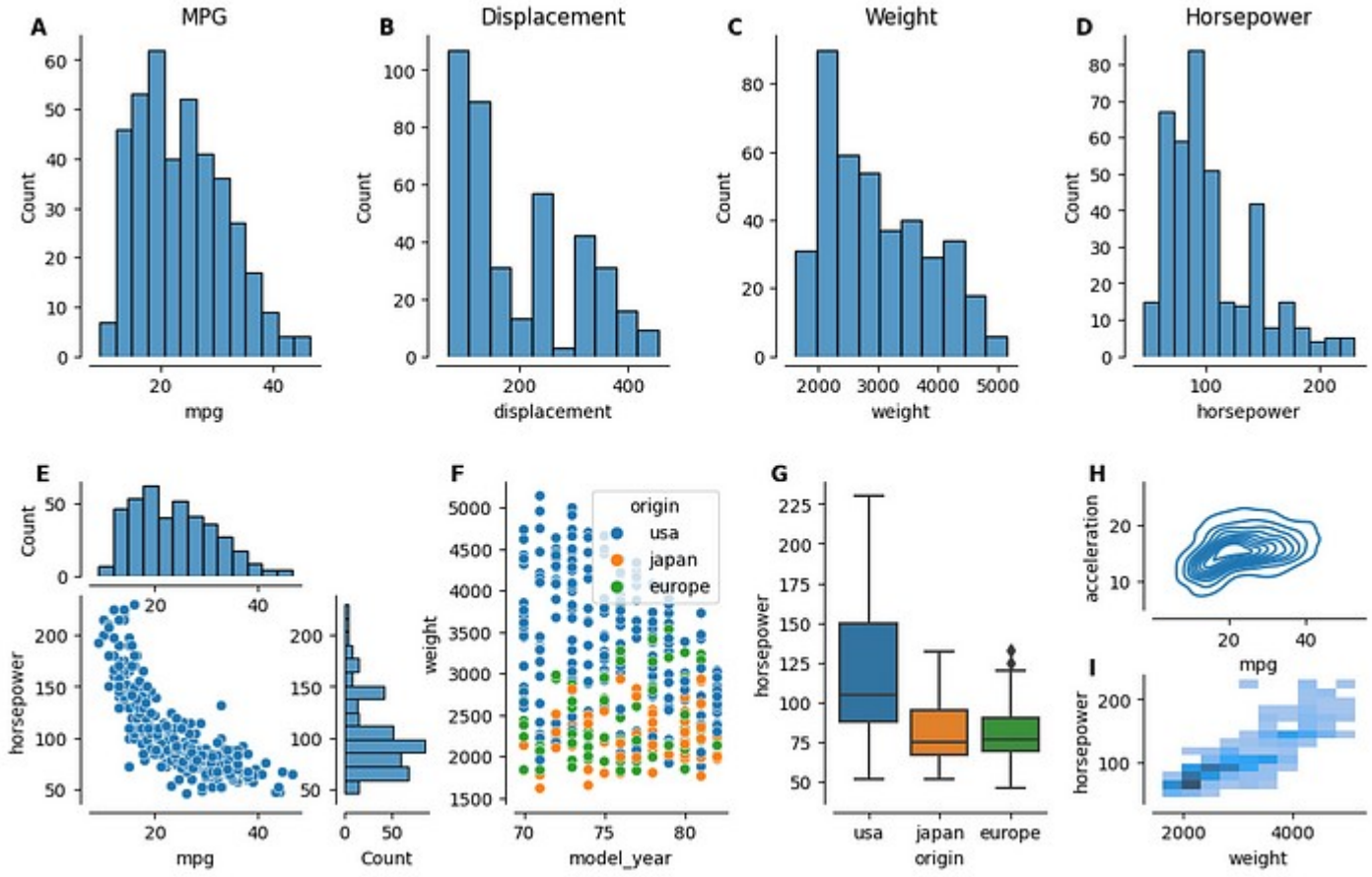
```
fig = plt.figure(figsize=(10, 7))
(topfig, bottomfig) = fig.subfigures(2, 1)
topfig.set_facecolor('#cbe4c6ff')
topfig.suptitle('Top')
top_axs = topfig.subplots(2, 4)
letter_annotation(top_axs[0][0], -.2, 1.1, 'A')
(bottomleft, bottomright) = bottomfig.subfigures(1, 2, width_ratios=(1,2))
bottomleft.set_facecolor('#c6c8e4ff')
bottomleft.suptitle('Bottom left')
bottoml_axs = bottomleft.subplots(2, 2)
letter_annotation(bottoml_axs[0][0], -.2, 1.1, 'B')
bottomright.set_facecolor('#aac8e4ff')
bottomright.suptitle('Bottom right')
bottomr_axs = bottomright.subplots(3, 3)
letter_annotation(bottomr_axs[0][0], -.2, 1.1, 'C')
# Spacing between subplots
topfig.subplots_adjust(left=.1, right=.9, wspace=.4, hspace=.4)
bottomleft.subplots_adjust(left=.2, right=.9, wspace=.5, hspace=.4)
bottomright.subplots_adjust(left=.1, right=.9, wspace=.4, hspace=.4)
plt.show()
```



We can now create the plot shown at the beginning of the article. It consists of three subfigures. One top subfigure, spanning the first row, and two bottom subfigures. The left bottom subfigure will be used for the jointplot (as shown before) and for the right bottom subfigure we will define a gridspec for place 4 subplots of different sizes.

```
fig = plt.figure(figsize=(10, 7))
# Creating a subfigure for the first and second row
(row1fig, row2fig) = fig.subfigures(2, 1, height_ratios=[1, 1])
# Splitting the bottom row subfigure in two subfigures
(fig_row2left, fig_row2right) = row2fig.subfigures(1, 2, wspace=.08,
width_ratios = (1, 2))
# #####
# Row 1 plots
# #####
# Make 4 subplots for the first row subfigure
row1_axs = row1fig.subplots(1, 4)
row1fig.subplots_adjust(wspace=0.5, left=0, right=1, bottom=.16)
ax = row1_axs[0]
sns.histplot(data=data, x='mpg', ax=ax)
ax.set_title('MPG')
# Annotate plotots with letters
letter_annotation(ax, -.25, 1, 'A')
# Some styling for figures to make them look better
# and have a standardized look
sns.despine(offset=5, trim=False, ax=ax)
ax = row1_axs[1]
sns.histplot(data=data, x='displacement', ax=ax)
ax.set_title('Displacement')
letter_annotation(ax, -.25, 1, 'B')
sns.despine(offset=5, trim=False, ax=ax)
ax = row1_axs[2]
sns.histplot(data=data, x='weight', ax=ax)
ax.set_title('Weight')
letter_annotation(ax, -.25, 1, 'C')
sns.despine(offset=5, trim=False, ax=ax)
ax = row1_axs[3]
sns.histplot(data=data, x='horsepower', ax=ax)
ax.set_title('Horsepower')
letter_annotation(ax, -.25, 1, 'D')
sns.despine(offset=5, trim=False, ax=ax)
# #####
# Row 2 plots
# #####
# ##
# Seaborn jointplot
# ##
# Using code from the Seaborn JointGrid class
# size ratio between the main plots and the margin plots
ratio=2
# Defining a gridspec for inside the subfigure
gs = plt.GridSpec(ratio + 1, ratio + 1)
ax_joint  = fig_row2left.add_subplot(gs[1:, :-1])
# Share axis between the margin and main plots
ax_marg_x = fig_row2left.add_subplot(gs[0, :-1], sharex=ax_joint)
ax_marg_y = fig_row2left.add_subplot(gs[1:, -1], sharey=ax_joint)
# Remove Axis labels and ticklabels for the margin plots
plt.setp(ax_marg_x.get_xticklabels(), visible=False)
plt.setp(ax_marg_y.get_yticklabels(), visible=False)
plt.setp(ax_marg_x.get_xticklabels(minor=True), visible=False)
plt.setp(ax_marg_y.get_yticklabels(minor=True), visible=False)
sns.scatterplot(data=data, y='horsepower', x='mpg', ax=ax_joint)
sns.histplot(data=data, y='horsepower', ax=ax_marg_y)
```

```
sns.histplot(data=data, x='mpg', ax=ax_marg_x)
sns.despine(offset=5, trim=False, ax=ax_joint)
sns.despine(offset=5, trim=False, ax=ax_marg_y)
sns.despine(offset=5, trim=False, ax=ax_marg_x)
# Leaving some space to the right to remove overlaps
fig_row2left.subplots_adjust(left=0, right=.8)
letter_annotation(ax_marg_x, -.25, 1, 'E')
# ##
# Row 2 right plots
# ##
gs = plt.GridSpec(2, 3)
ax_left   = fig_row2right.add_subplot(gs[:, 0])
ax_middle = fig_row2right.add_subplot(gs[:, 1])
ax_up     = fig_row2right.add_subplot(gs[0, 2])
ax_down   = fig_row2right.add_subplot(gs[1, 2])
fig_row2right.subplots_adjust(left=0, right=1, hspace=.5)
ax = ax_left
sns.scatterplot(data=data, x='model_year', y='weight', hue='origin', ax=ax)
sns.despine(offset=5, trim=False, ax=ax)
letter_annotation(ax, -.3, 1, 'F')
ax = ax_middle
sns.boxplot(data=data, x='origin', y='horsepower', ax=ax)
sns.despine(offset=5, trim=False, ax=ax)
letter_annotation(ax, -.3, 1, 'G')
ax = ax_up
sns.kdeplot(data=data, x='mpg', y='acceleration', ax=ax)
sns.despine(offset=5, trim=False, ax=ax)
letter_annotation(ax, -.3, 1, 'H')
ax = ax_down
sns.histplot(data=data, x='weight', y='horsepower', ax=ax)
sns.despine(offset=5, trim=False, ax=ax)
letter_annotation(ax, -.3, 1, 'I')
plt.show()
```



Conclusion

Subfigures are a relatively new concept in matplotlib. They make it easy to assemble large figures with many plots. All the things shown in this article can also be achieved entirely using gridspec. However, this requires a large grid with many considerations for the sizes of each subplot. Subfigures are more plug-and-play and the same result can be achieved with less work.

For me, subfigures are a very convenient tool for creating scientific figures and I hope they can be helpful for you, too.

You can also find all the code from this article on GitHub: https://github.com/tdrose/blogpost-subfigures-code

Unless otherwise noted, all images were created by the author.

Python
Seaborn
Matplotlib
Tips And Tricks
Data Visualization

tds

Written by Tim Rose

28 Followers
·Writer for

Towards Data Science

Biological data scientist