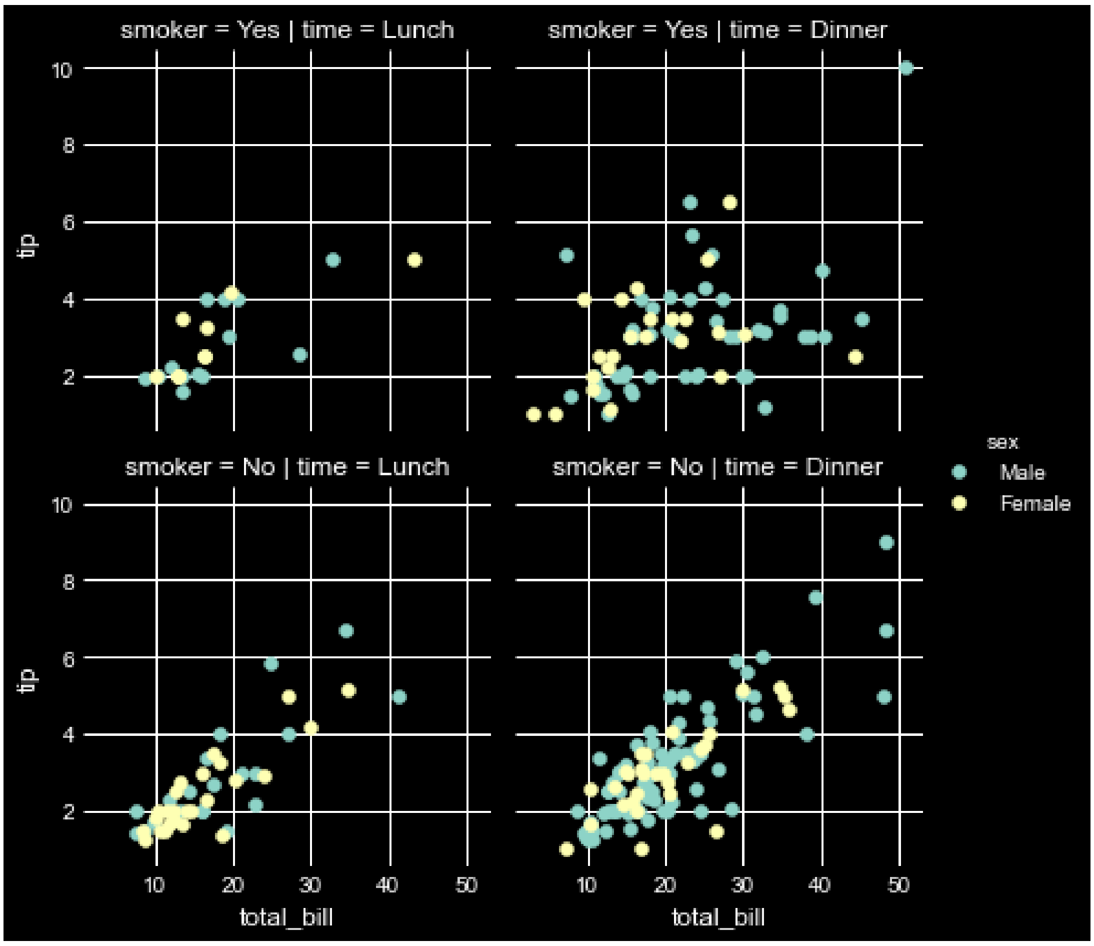
A Comprehensive Guide to Data Visualization with Python for Complete Beginners – More on Seaborn (Grids and Customization)

helloml.org/a-comprehensive-guide-to-data-visualization-with-python-for-complete-beginners-more-on-seaborn-grids-and-customization/

By Ajit Singh October 1, 2021

This article is a continuation of the ‘Comprehensive guide to data visualization with Python for complete beginners’ series, particularly the one in which we covered using Seaborn for data visualization. Today, we will expand upon that article and cover more on Seaborn, particularly using grids and then looking at customizing plots based on our requirements rather than using Seaborn’s built-in default styles. Let’s begin!



Example of a **FacetGrid**

Grids

Grids are general types of plots. A literal grid allows us to structure our data into rows and columns. In our case, we will map various plot types to rows and columns of a grid. This helps us to create similar plots which are separated by features.

Installing dependencies

We will require Matplotlib and Seaborn. Assuming you're working on a Jupyter notebook (Anaconda installation of Python), installing Matplotlib and Seaborn on your PC is very simple. All you need to do is type in the following command in the Anaconda command prompt:

```
conda install matplotlib seaborn
```

Or if you are using Python without anaconda, you can also install the required libraries using pip.

```
pip install matplotlib seaborn
```

We will be working with two of Seaborn's built-in datasets, **penguins,** and **tips.**

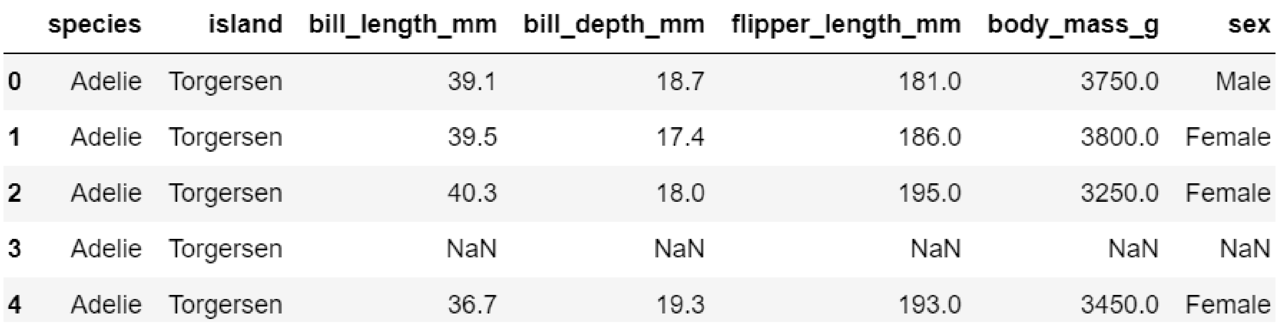
Setting Up

We'll import the required libraries and load the datasets:

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
# Allows us to show plots automatically within the Jupyter notebooks without
typing in plt.show()
```

First up, we'll be working with the penguins' dataset. Likewise, it is important to check its head to understand the data.

```
peng = sns.load_dataset('penguins')
peng.head()
```

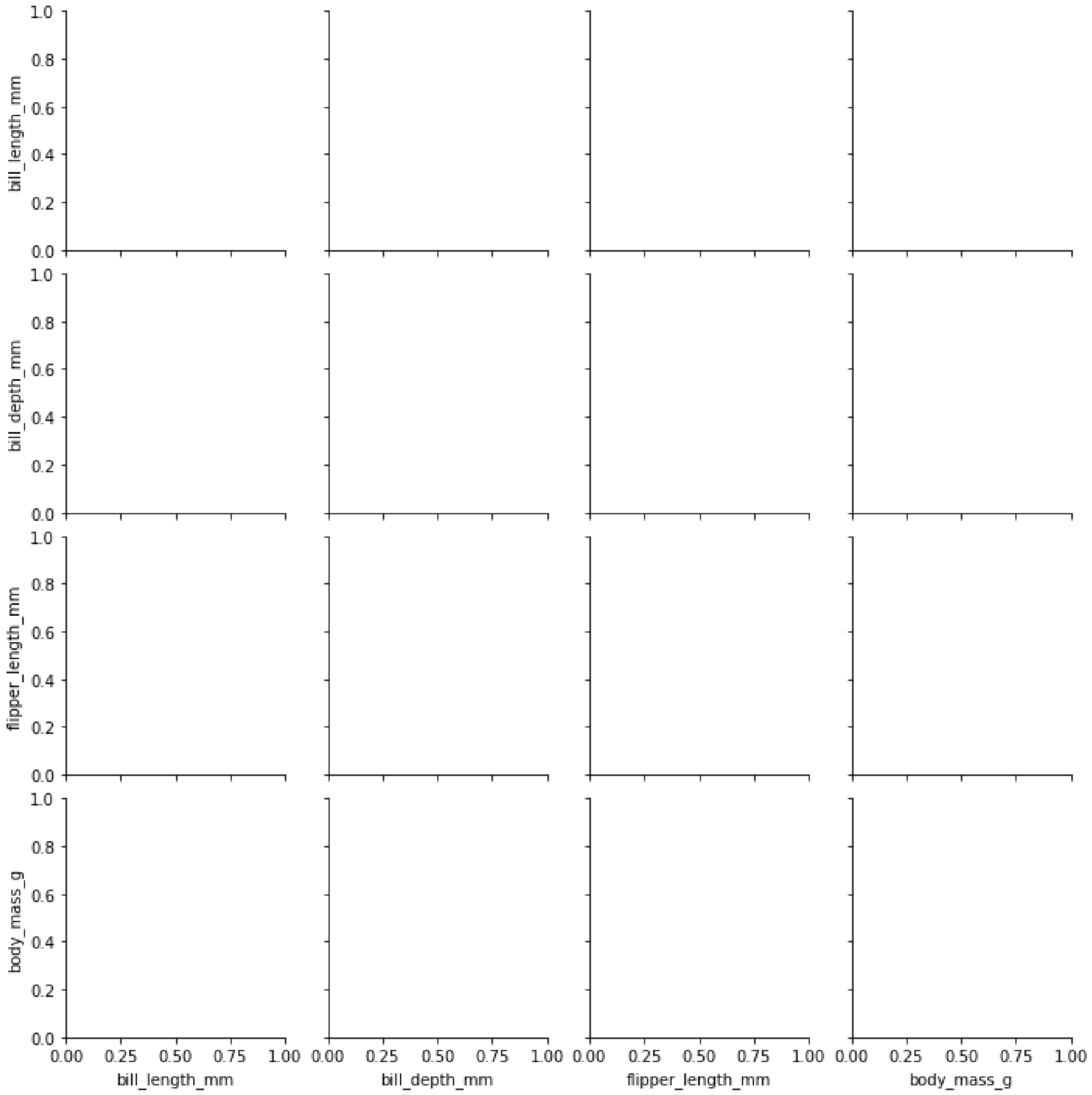


	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0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

PairGrid

PairGrid provides us with a subplot grid for plotting pairwise relationships in a dataset. The following line of code will generate the pairgrid (Just the grid, no plots) for the penguins dataset.

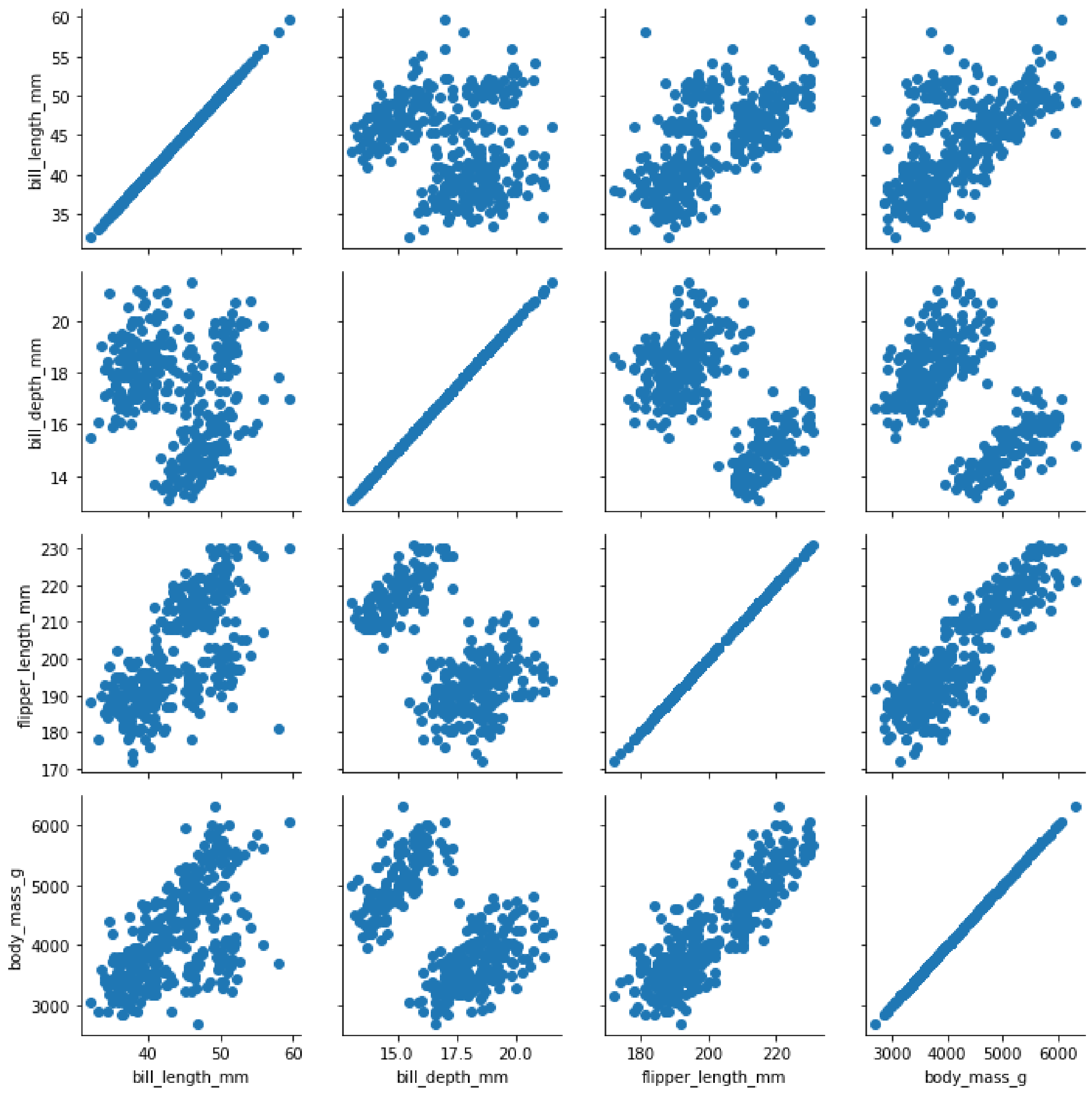
```
sns.PairGrid(peng)
```



Output

Then, we will save this generated pairgrid into a variable ‘**grid**‘ and then map the plots (scatter plot) onto the grid.

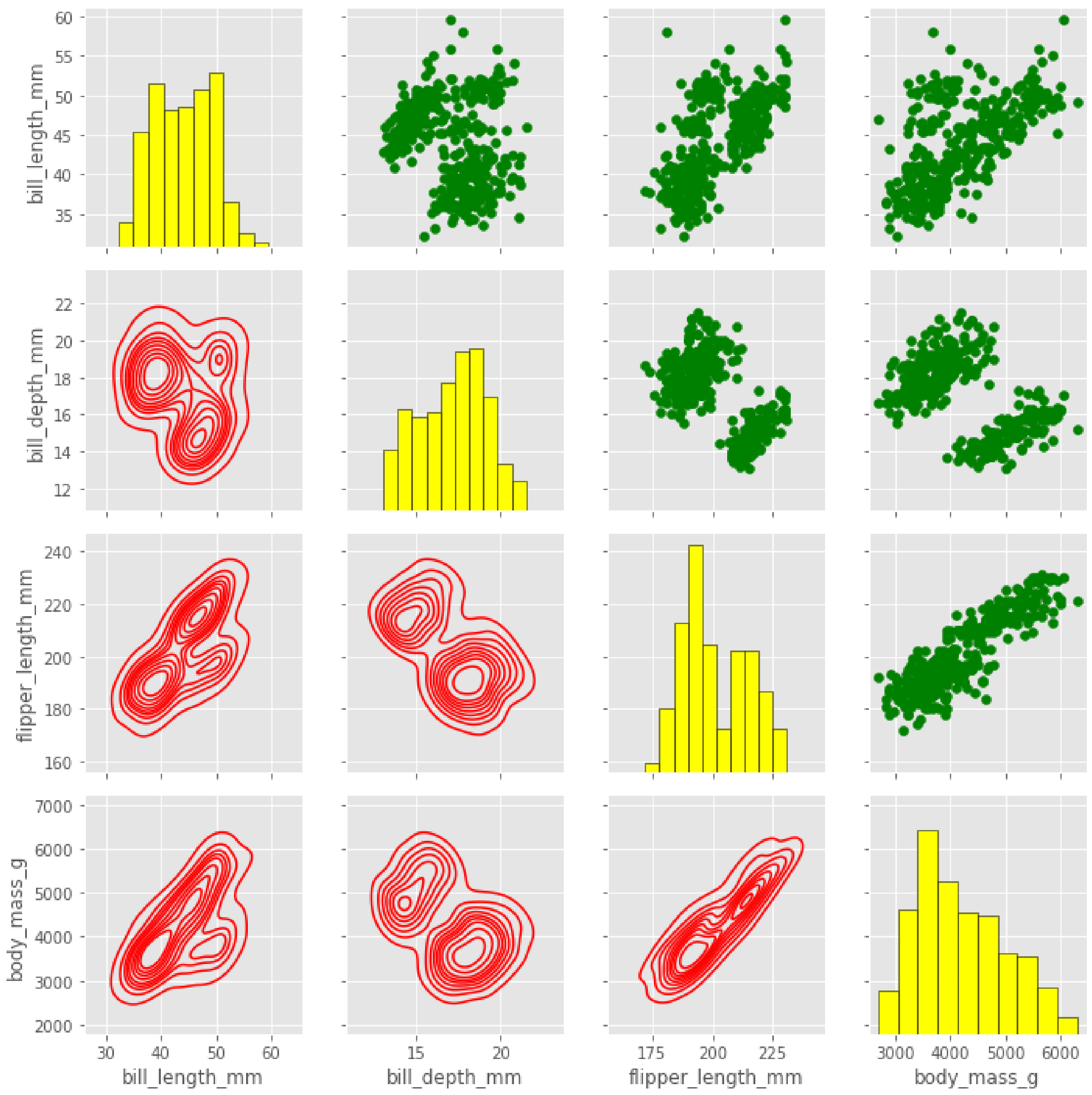
```
grid = sns.PairGrid(peng)
grid.map(plt.scatter)
```



Output

We can choose what kind of plot we want to graph at which position on the grid. We can even customize the plots and set styles based on our needs as shown in the example below.

```
plt.style.use('ggplot')
g = sns.PairGrid(peng)
g.map_diag(plt.hist, color='yellow',edgecolor='black')
g.map_upper(plt.scatter, color='green')
g.map_lower(sns.kdeplot, color='red')
```

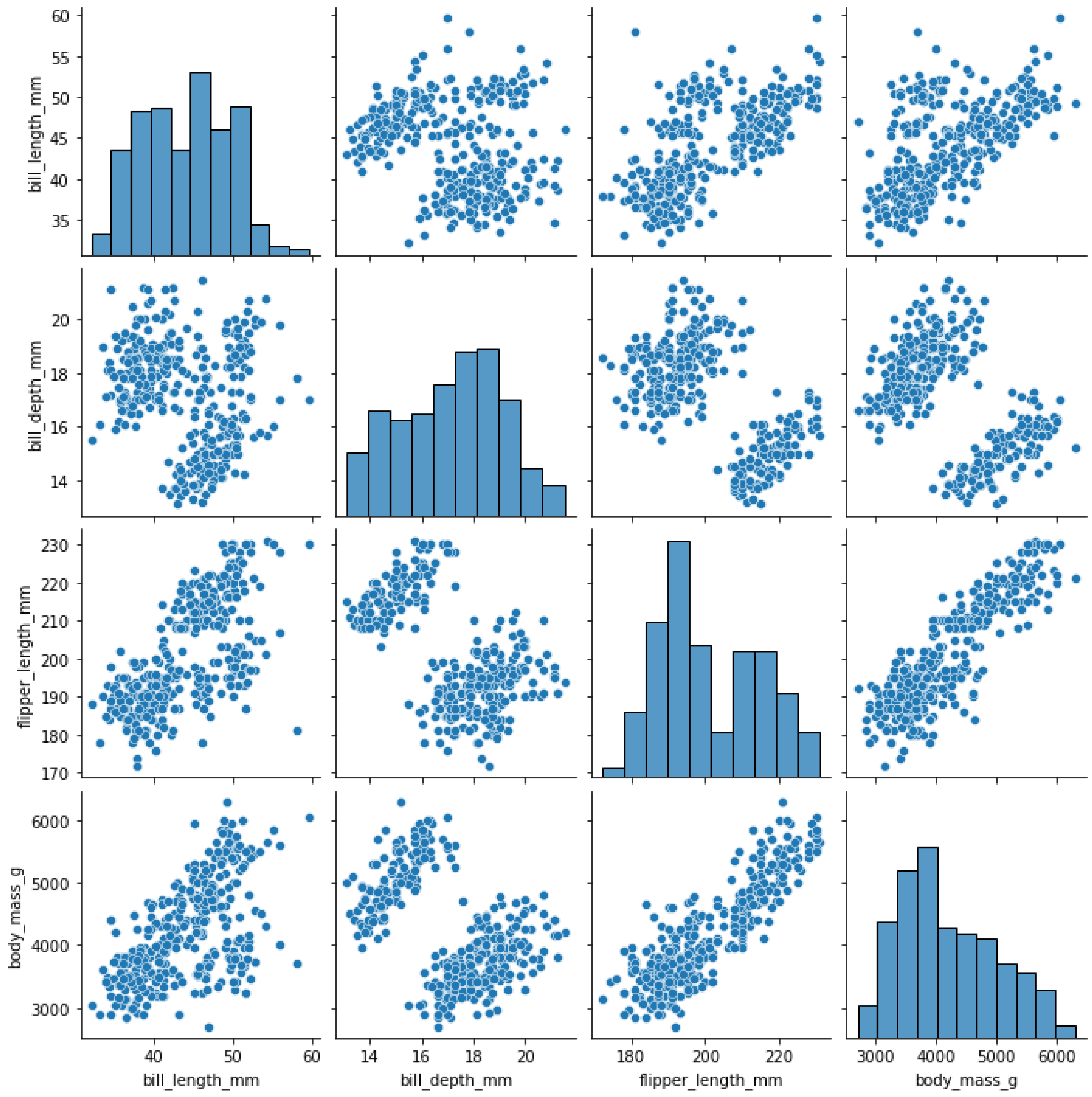


Output

Pairplot

Pairplot is the simpler version of PairGrid which we have already seen above. We will always use pairplot for plotting pairwise relationships between columns in our dataset; Pairgrids are rarely used.

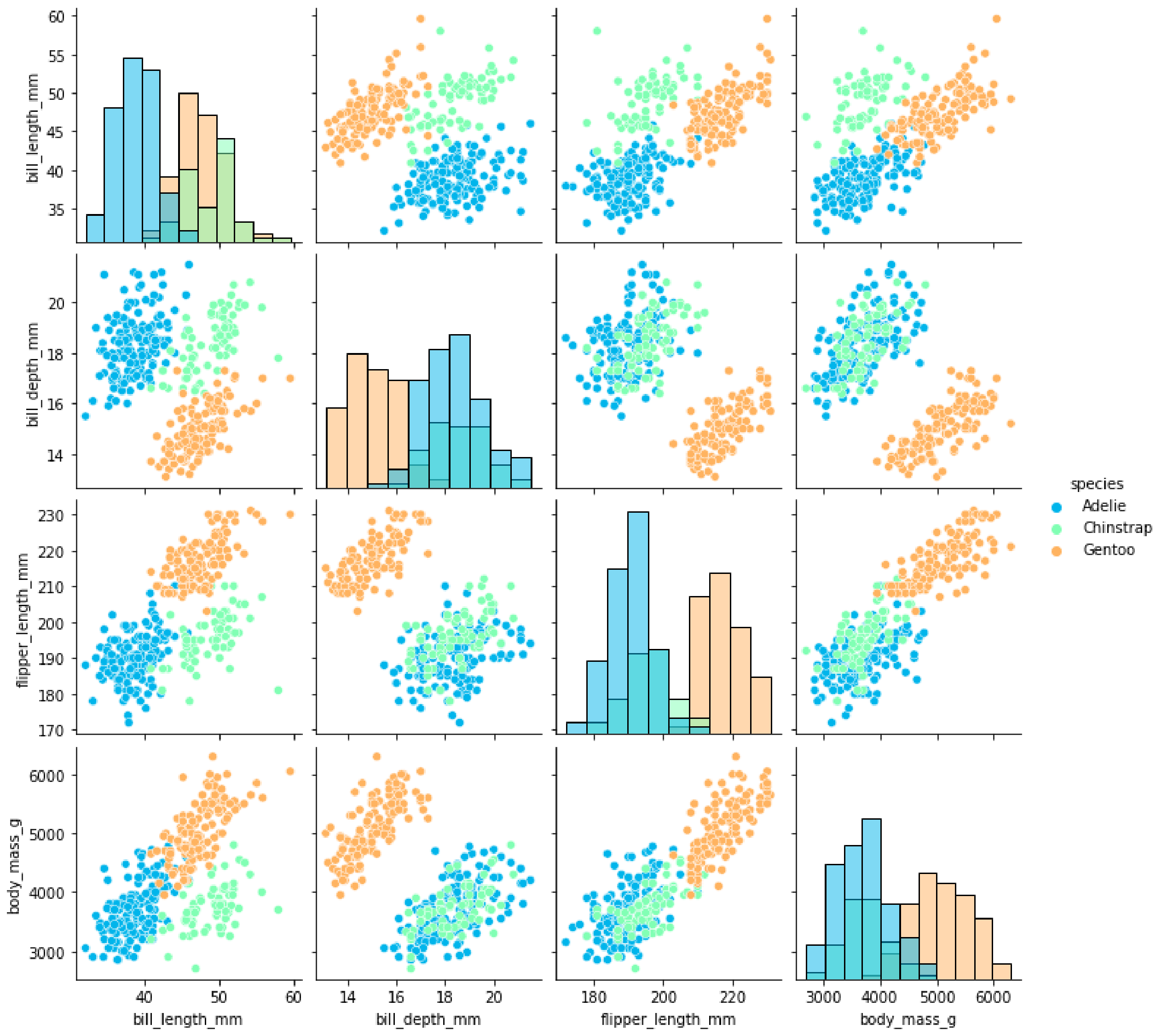
```
plt.style.use('default')
sns.pairplot(peng)
```



Output

We can also add in a **hue** element to differentiate between data better and set the **palette** for our plots as shown below.

```
sns.pairplot(peng, diag_kind='hist', hue='species',palette='rainbow')
```



Output

For the next examples, we'll switch the dataset from **penguins** to **tips** and then check the head.

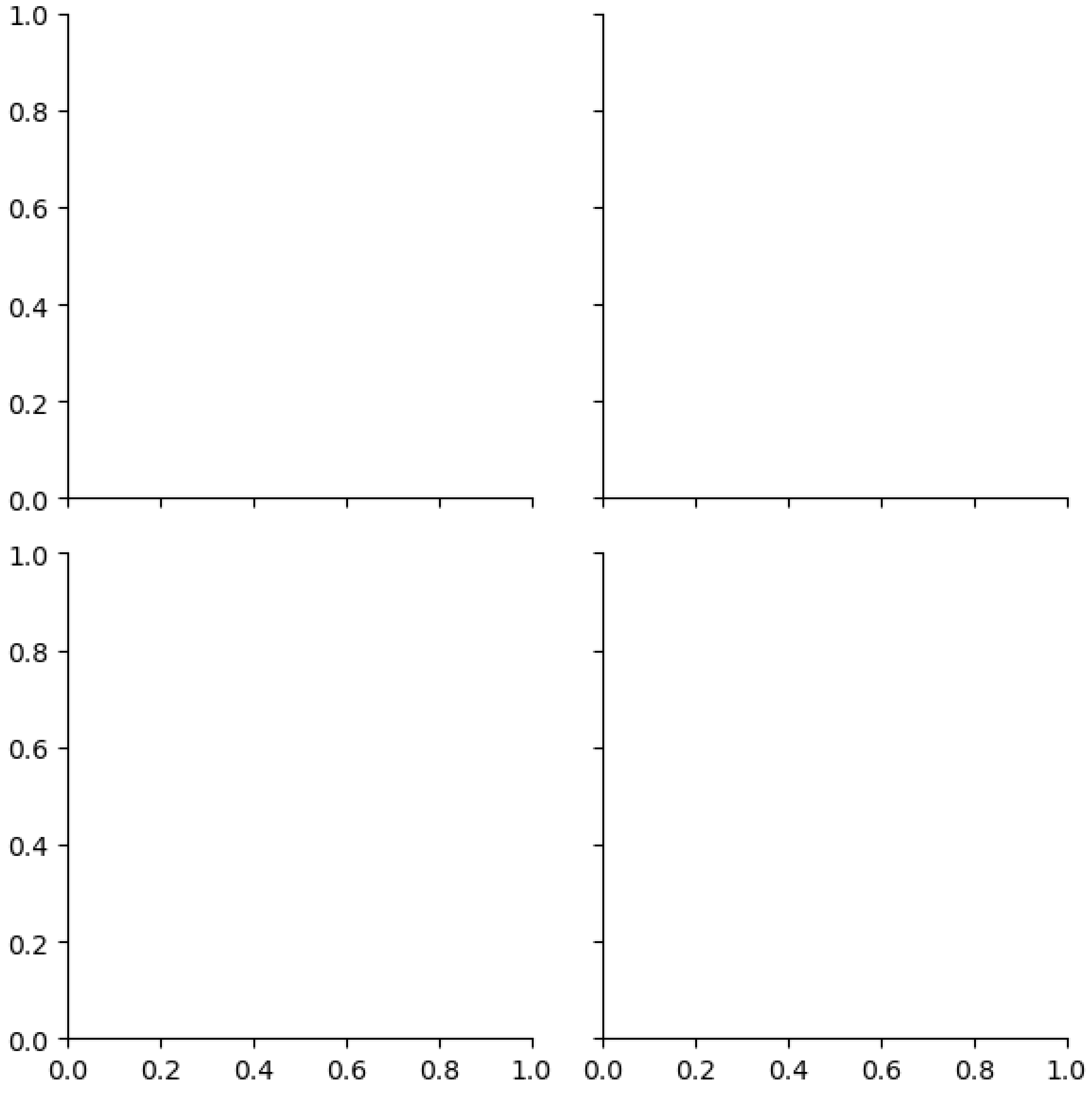
```
tips = sns.load_dataset('tips')
tips.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

FacetGrid

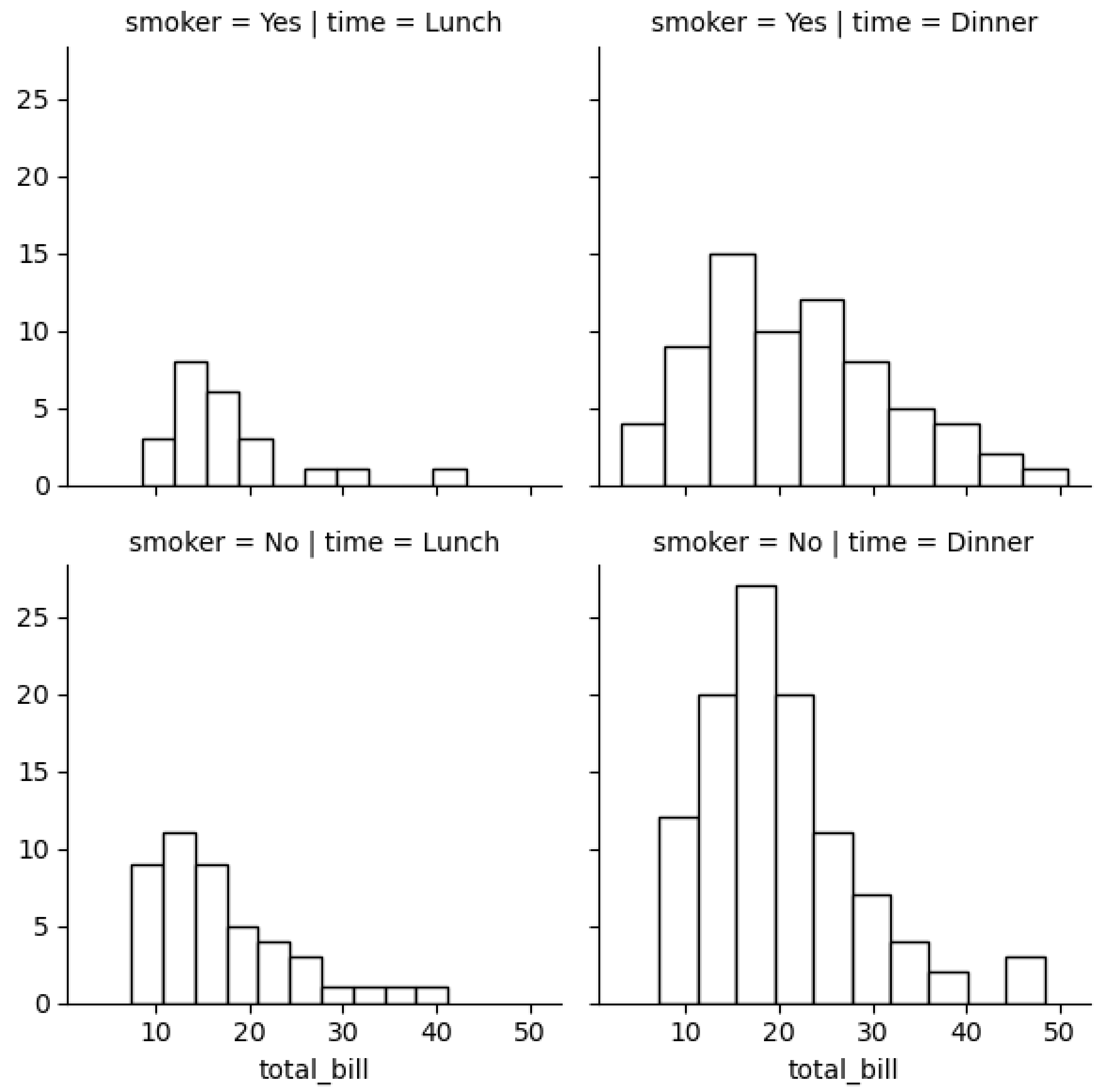
FacetGrid enables us to create grids of plots based on a feature/characteristic. Just like PairGrid, we first have to create an empty grid and then map our plots onto the grid.

```
g = sns.FacetGrid(tips, col="time", row="smoker")
```



Output

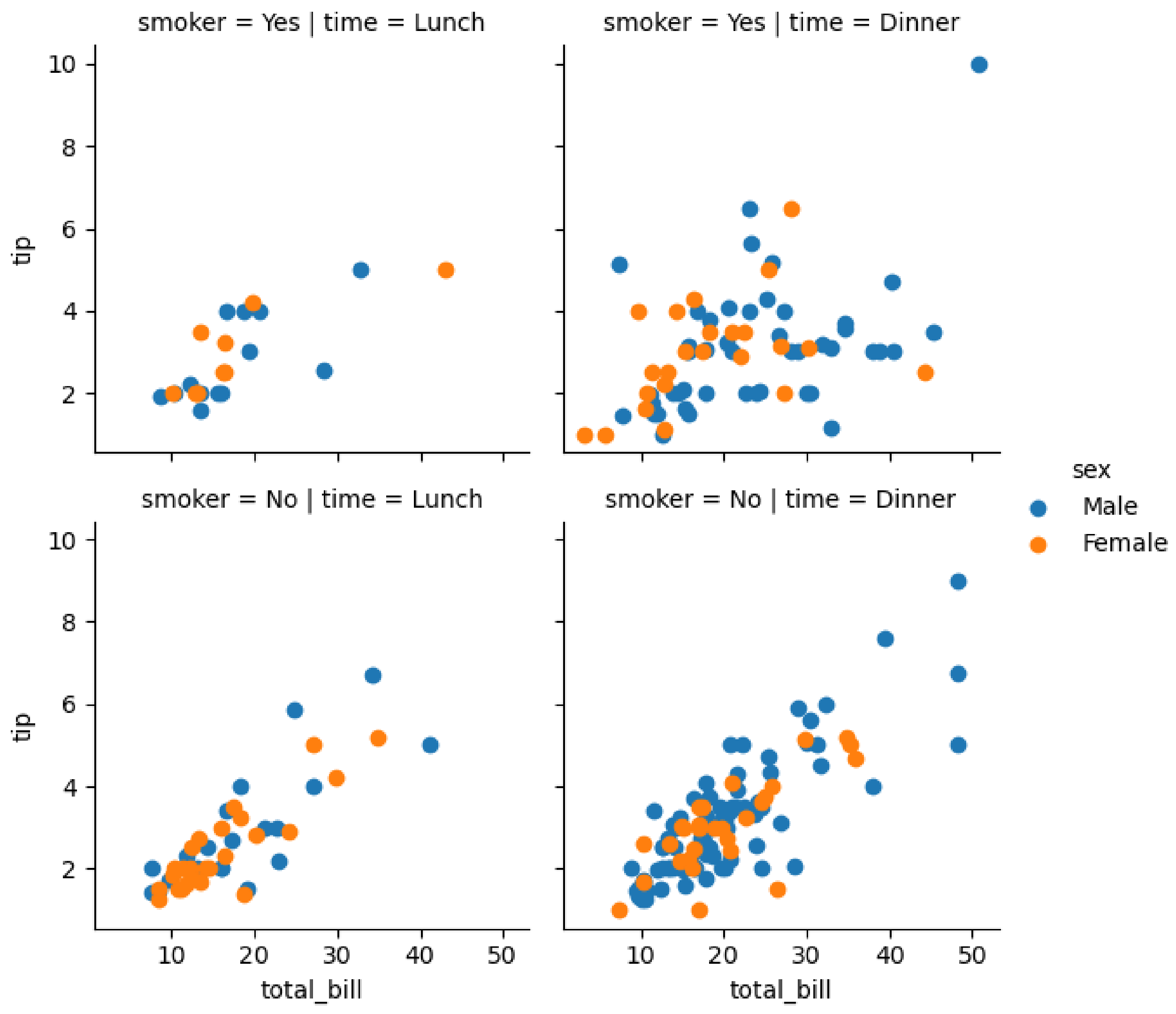
```
g = g.map(plt.hist, "total_bill", color='white', edgecolor='black')
```



Output

We can control the type of plots and also add in a **hue** element. We'll also make sure to add a **legend** to our plot using the below code.

```
g = sns.FacetGrid(tips, col="time",  row="smoker",hue='sex')
g = g.map(plt.scatter, "total_bill", "tip").add_legend()
```

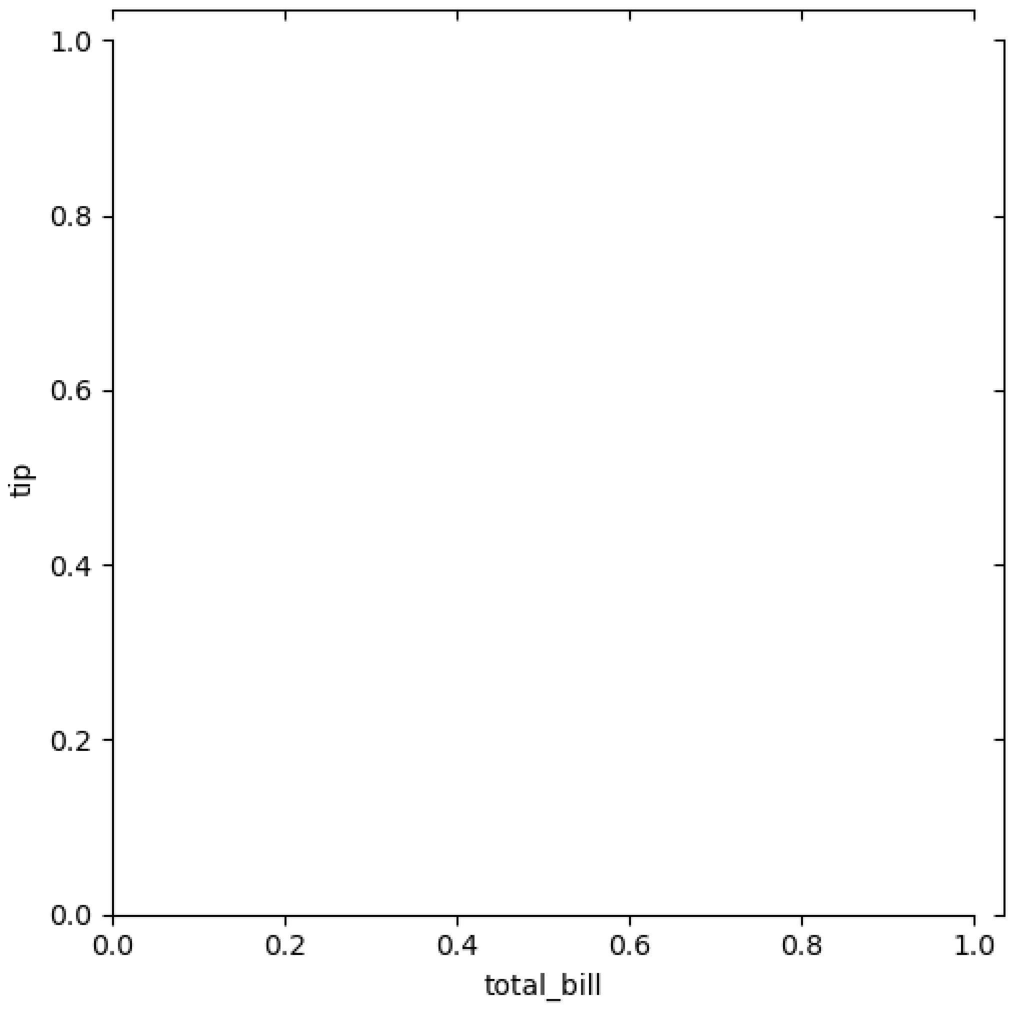


Output

JointGrid

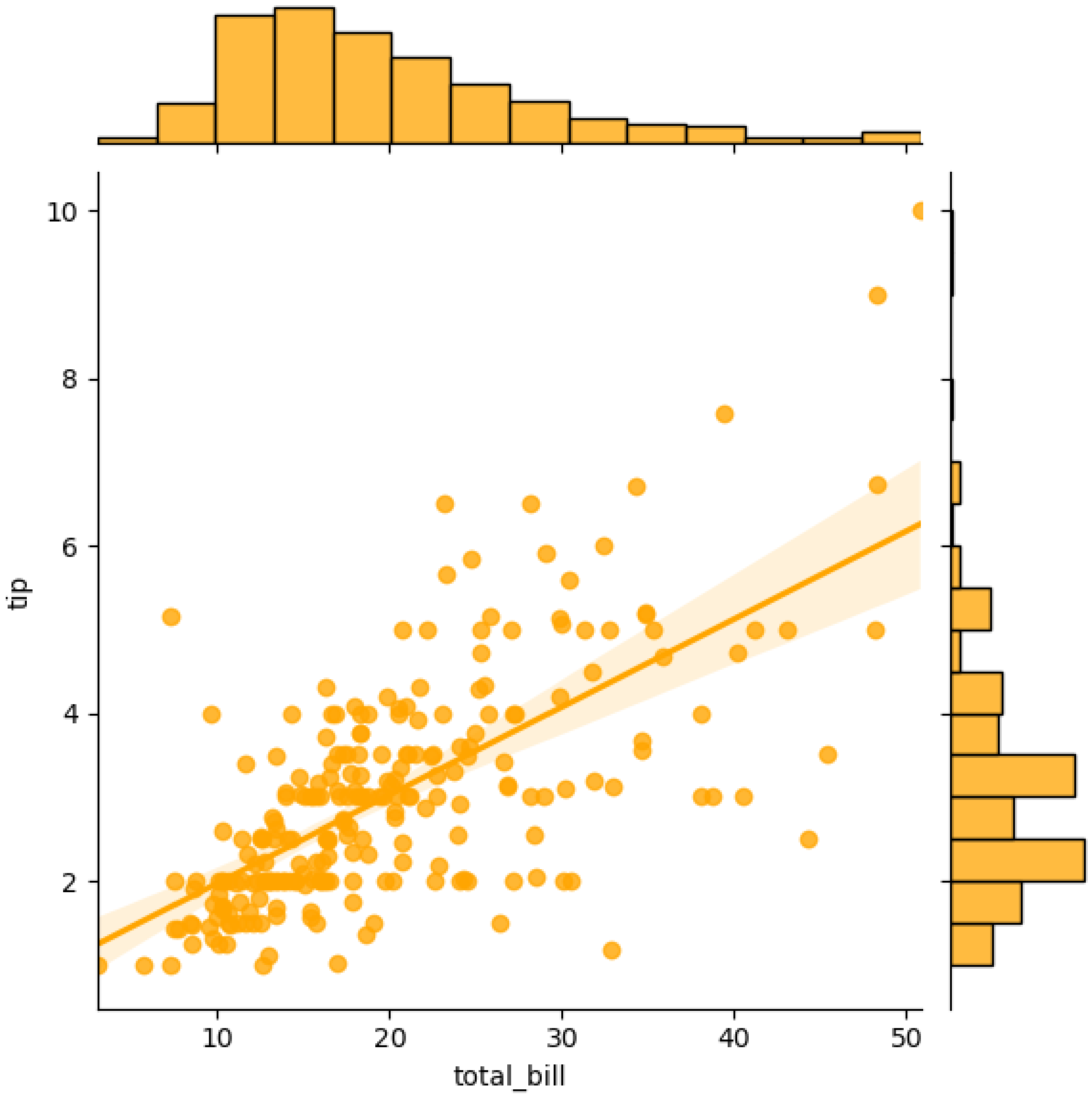
JointGrid is the general version for jointplot() type grids, similar to PairGrid and pairplot(). Again first we'll initialize the grid and then map our plots onto the grid. In this case, we're plotting **regplot** and **histogram**.

```
g = sns.JointGrid(x="total_bill", y="tip", data=tips)
```



Output

```
g = sns.JointGrid(x="total_bill", y="tip", data=tips)
g = g.plot(sns.regplot, sns.histplot, color='orange')
```



Output

That's it for grids! We have covered most use cases for grids and also covered the commonly used grid types. However, at many instances, we'll just use the easier plots which we had discussed earlier.

Seaborn Customizations – Style and Color

Till now, we have seen the different kinds of plots which we can graph using Seaborn. Till now, we have mostly relied on Seaborn's built-in default styles and color schemes for figures. Now, we'll have a look at how to tweak them according to our liking.

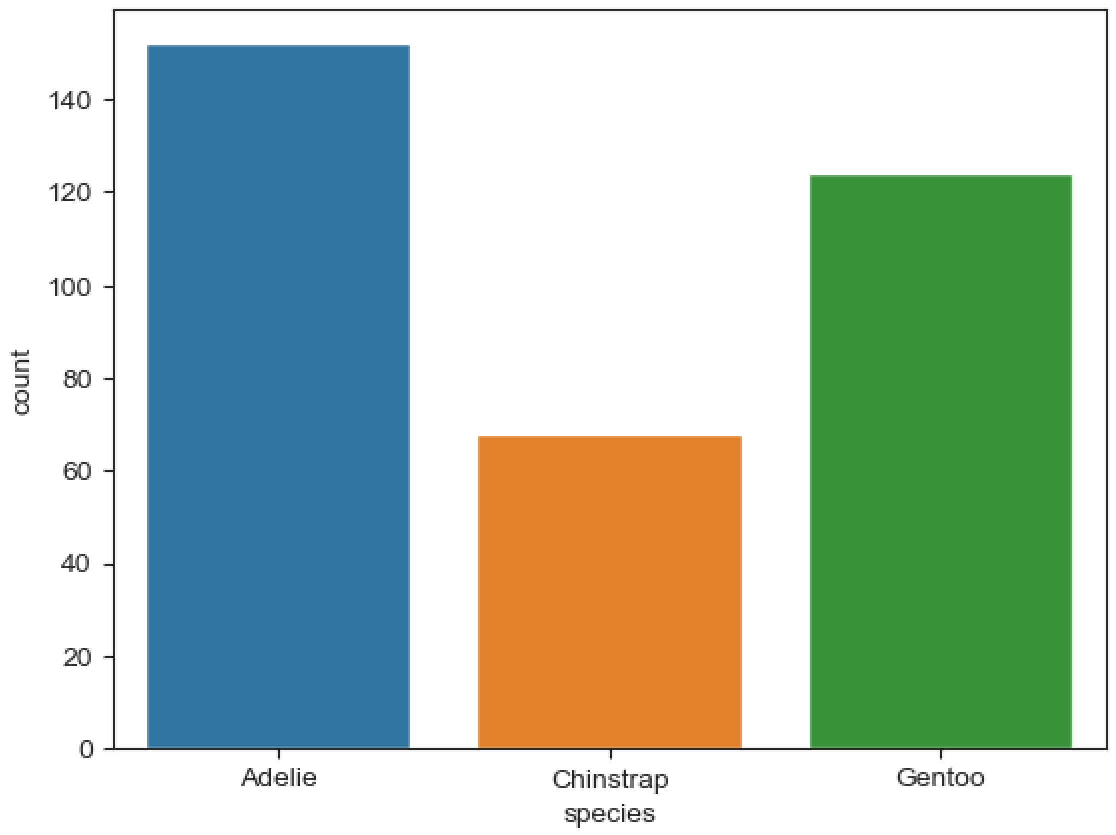
We'll work on the penguins dataset once again.

```
peng = sns.load_dataset('penguins')
peng.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

Let's begin by plotting out seaborn's default countplot for the '**species'** column. This will give us the count of all three species of penguins in our dataset.

```
sns.countplot(x='species',data=peng)
```

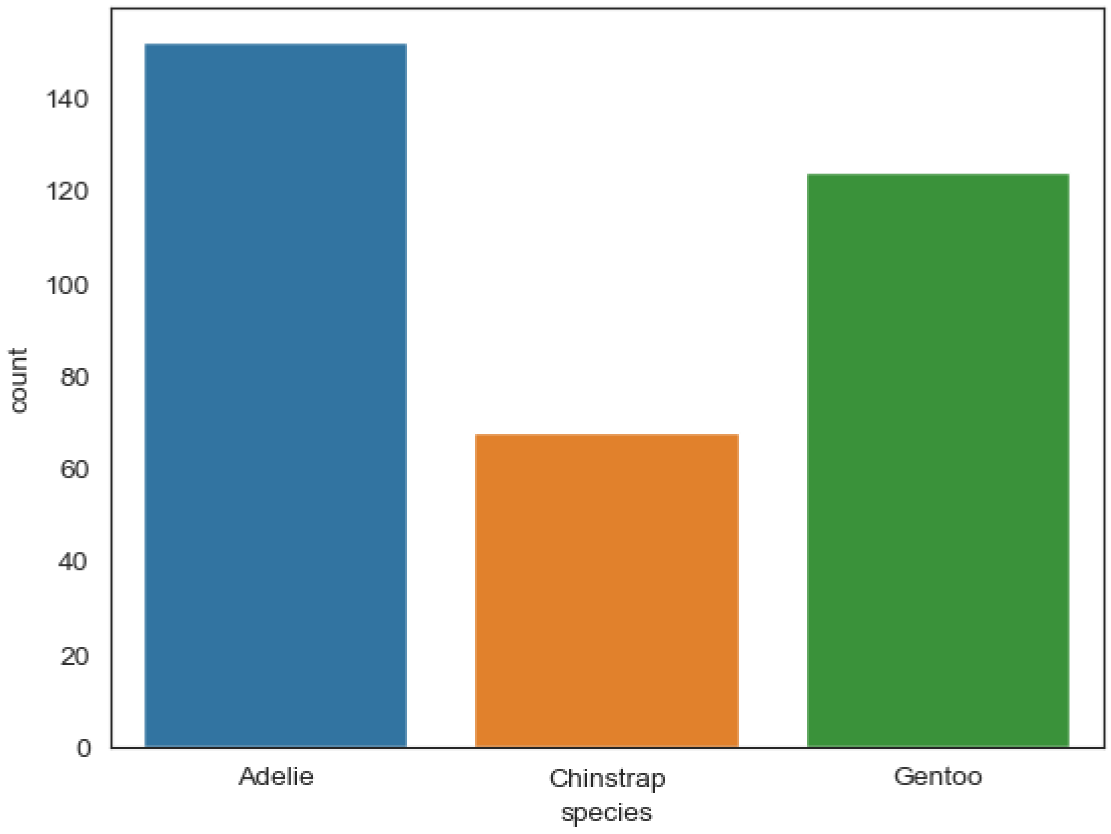


Output

Setting Styles

Now, we'll set particular styles:

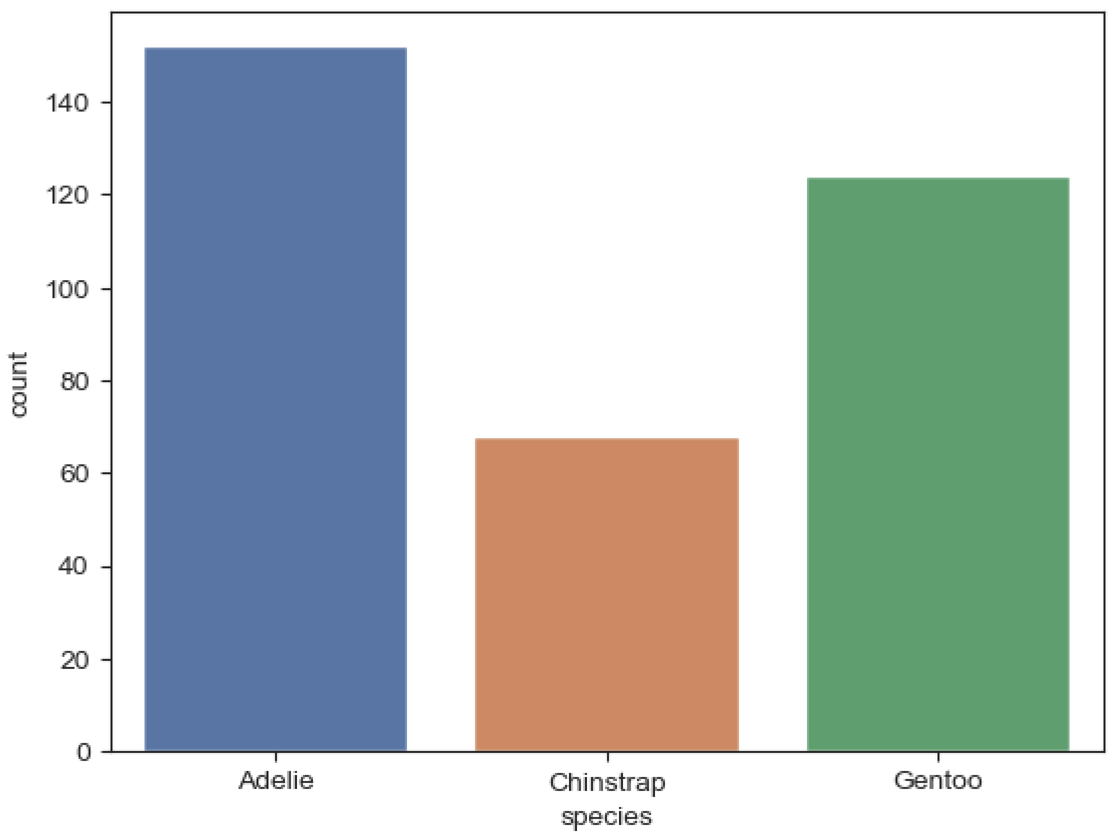
```
sns.set_style('white')
sns.countplot(x='species',data=peng)
```



Output

As we can see, the ‘**white**‘ style removes all the ticks from the figure. We can revert to ticks by setting the ‘**ticks**‘ style as shown below.

```
sns.set_style('ticks')
sns.countplot(x='species',data=peng,palette='deep')
```



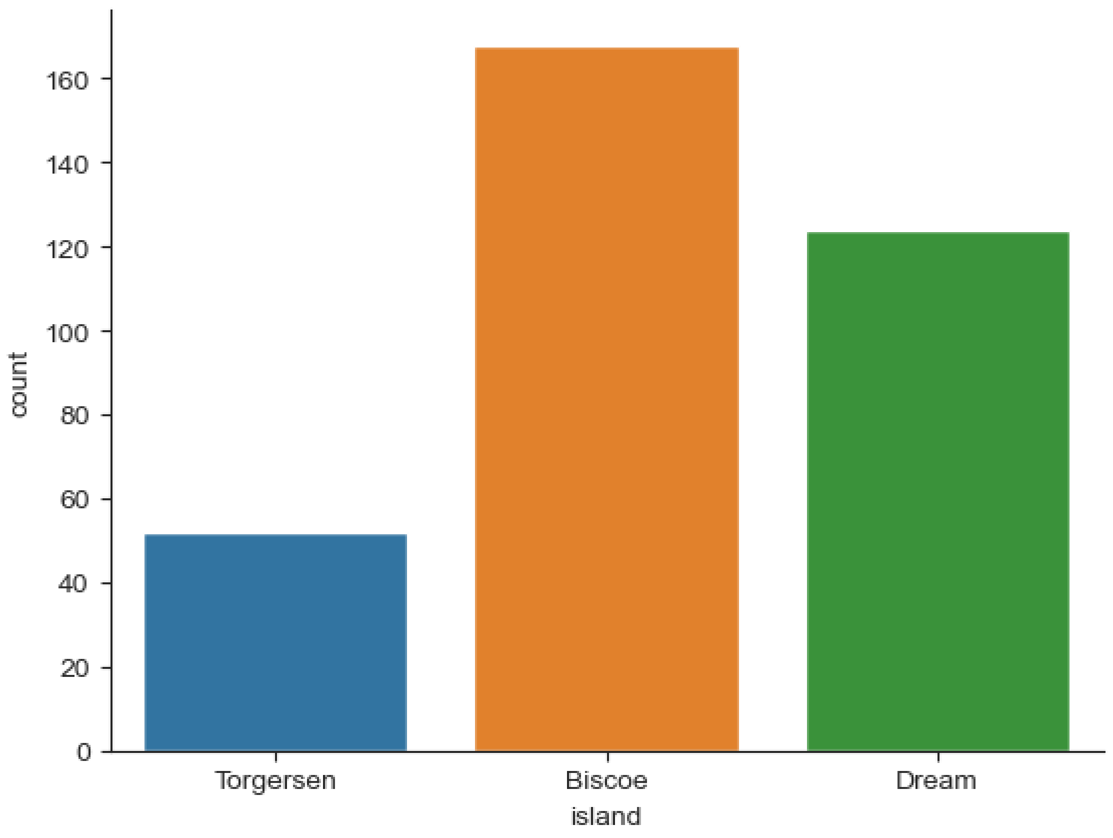
Output

We set the ticks back again and also changed the **palette** for the figure to ‘**deep**‘.

Removing Spine

Next, we’ll have a look at de-spining our figures. For this, we’ll use a countplot which will give us the population of penguins in all three islands present in our dataset.

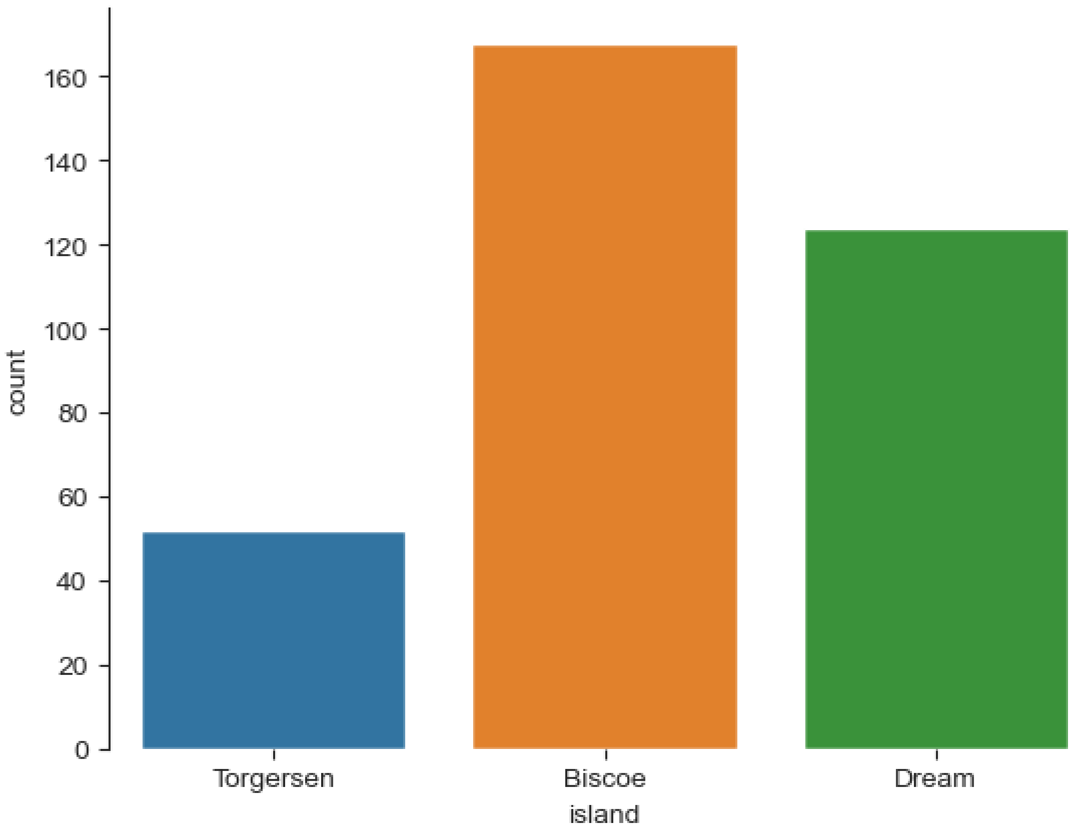
```
sns.countplot(x='island',data=peng)
sns.despine()
```



Output

We can also choose to further de-spine the left or the bottom axes.

```
sns.countplot(x='island',data=peng)
sns.despine(bottom=True)
```



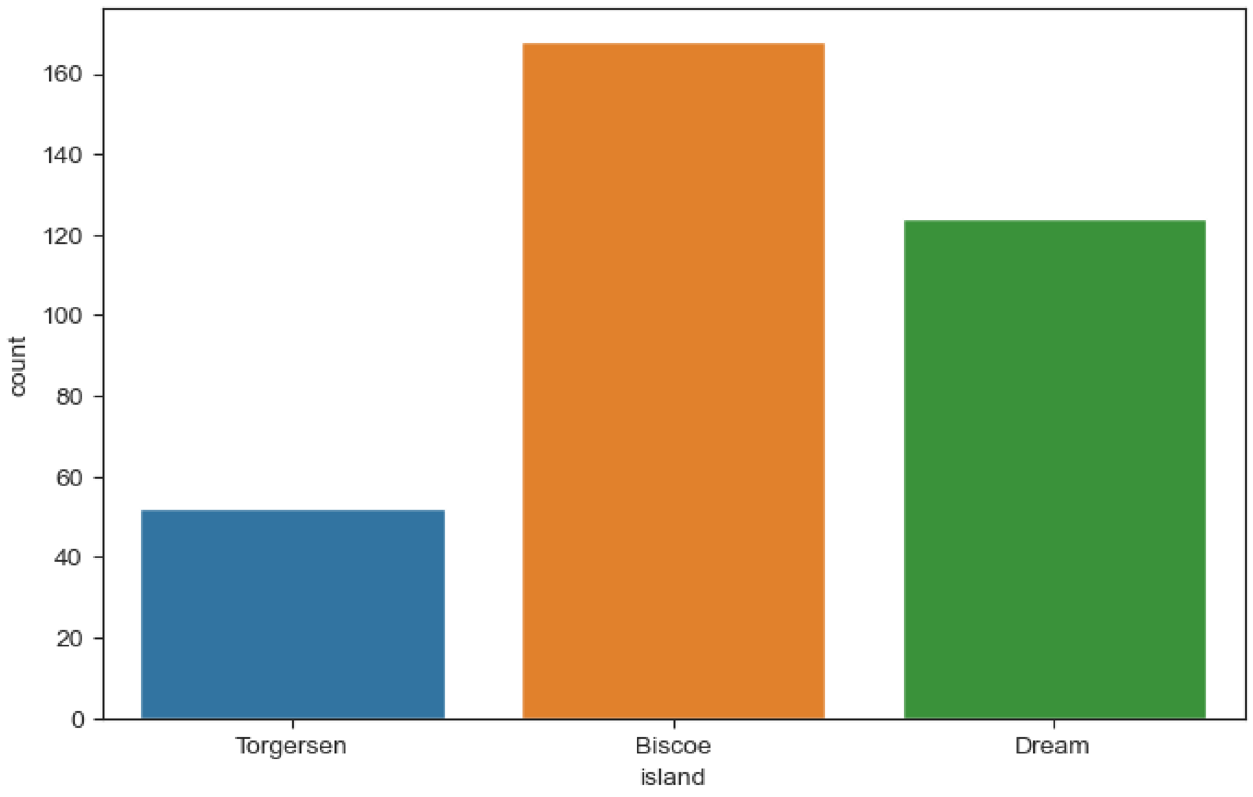
Output

Changing Figure size, aspect

To change the size of Seaborn plots, we mostly use Matplotlib's **plt.figure(figsize=(width height))**. We can also control the size and aspect ratio of seaborn grids by passing in the parameters **size** and **aspect**.

For a non-grid plot, we use **figsize** as shown below.

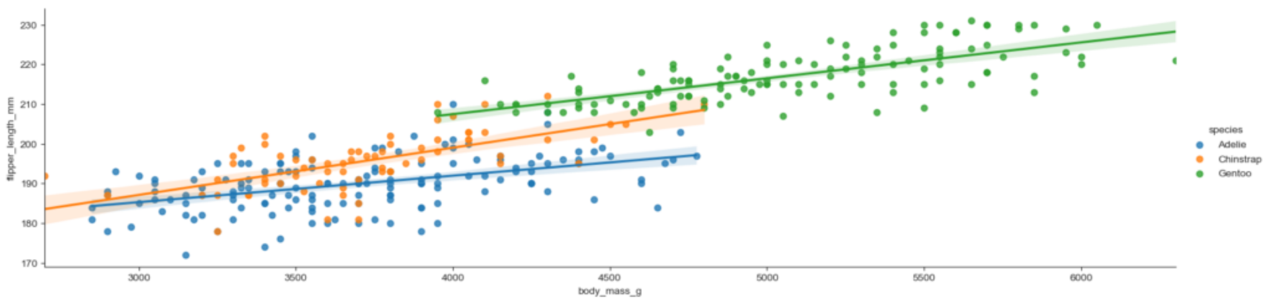
```
plt.figure(figsize=(8,5))
sns.countplot(x='island',data=peng)
```



Output

For a grid plot, we pass in the **height** and **aspect**.

```
sns.lmplot(x='body_mass_g',y='flipper_length_mm', hue='species', data=peng,
height=4,aspect=4)
```

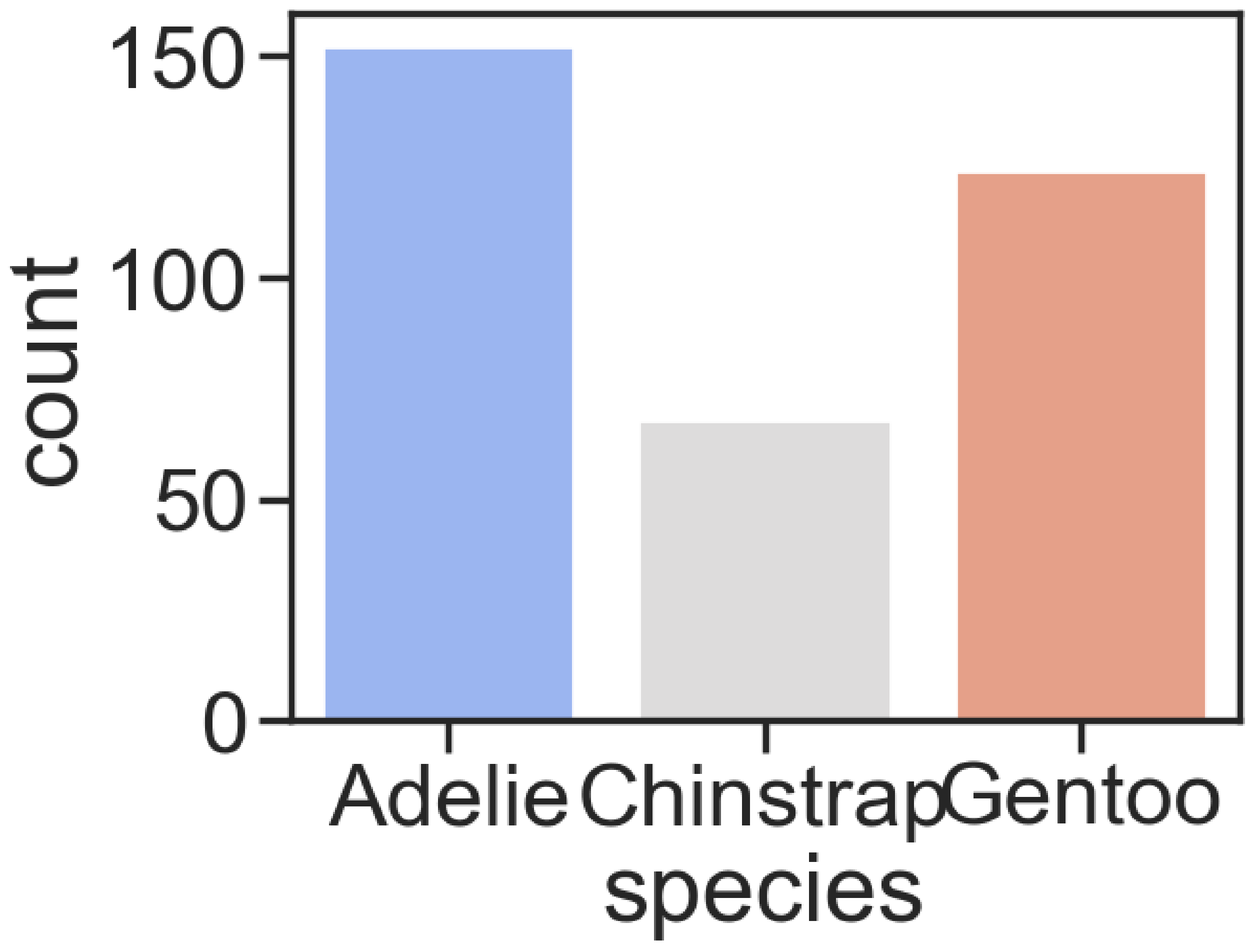


Output

Context, Scale

The **set_context()** method allows us to override default parameters set up on our own. We'll change the font scale to 1.5.

```
sns.set_context('poster',font_scale=1.5)
sns.countplot(x='species',data=peng,palette='coolwarm')
```



Output