By shreeyansh, source: gfg

It is summarized that if Matplotlib “tries to make easy things easy and hard things possible”, Seaborn "tries to make a well-defined set of hard things easy too.”

Seaborn helps resolve the two major problems faced by Matplotlib; the problems are −

- Default Matplotlib parameters
- Working with data frames

Important Features of Seaborn

Seaborn is built on top of Python’s core visualization library Matplotlib. It is meant to serve as a complement, and not a replacement. However, Seaborn comes with some very important features. The features help in −

- Built in themes for styling matplotlib graphics
- Visualizing univariate and bivariate data
- Fitting in and visualizing linear regression models
- Plotting statistical time series data
- Seaborn works well with NumPy and Pandas data structures
- It comes with built in themes for styling Matplotlib graphics
- In most cases, you will still use Matplotlib for simple plotting. The knowledge of Matplotlib is recommended to tweak Seaborn’s default plots.

There are three main plot kinds; in addition to histograms and kernel density estimates (KDEs, **lines** that occur in the plot), you can also draw empirical cumulative distribution functions (ECDFs). *`displot` was formerly known as `distplot`. `displot` has no argument `hist`, instead we use the kwarg `kind`. We will be using `distplot`*

In [1]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from numpy import random as rnd
```

Some pre loaded datasets in seaborn module

In [2]:
```
sns.get_dataset_names()
```

Out[2]:
```
['anagrams',
 'anscombe',
 'attention',
```

```
'brain_networks',
'car_crashes',
'diamonds',
'dots',
'exercise',
'flights',
'fmri',
'gammas',
'geyser',
'iris',
'mpg',
'penguins',
'planets',
'tips',
'titanic']
```

We will use 'iris' dataset for most part of our visualization.

In [3]:
```
data = sns.load_dataset("iris")
```

Visualizing the first few rows of dataset

In [4]:
```
data.head()
```

Out[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Some metrics for each feature

In [5]:
```
data.describe()
```

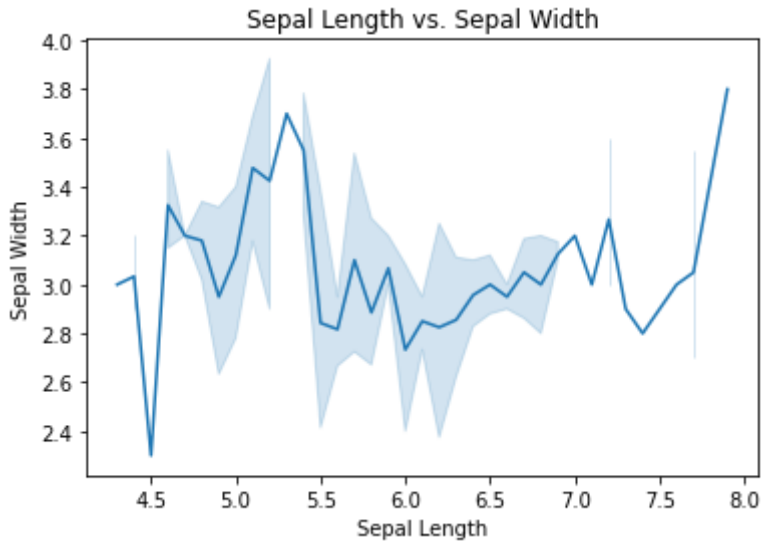
Out[5]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

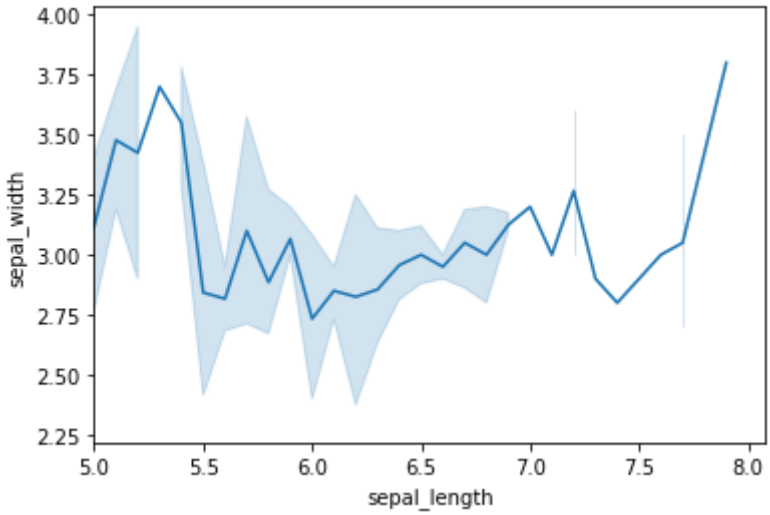
Using Seaborn w/ Matplotlib

Using both Matplotlib and Seaborn together is a very simple process. We just have to invoke the Seaborn Plotting function as normal, and then we can use Matplotlib's customization function. The light blue shade indicates the confidence level around that point if it has higher confidence the shaded line will be thicker.

In [6]:
```
sns.lineplot(x = "sepal_length", y = "sepal_width", data = data)
plt.title("Sepal Length vs. Sepal Width")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.show()
```



In [7]:
```
sns.lineplot(x = "sepal_length", y = "sepal_width", data = data)

#Setting limit for x-axis
plt.xlim(5)
plt.show()
```



Customizing Seaborn Plots

Seaborn comes with some customized themes and a high-level interface for customizing the looks of the graphs. Consider the above example where the default of the Seaborn is used. It still looks nice and pretty but we can customize the graph according to our own needs. So let's see the styling of plots in detail.

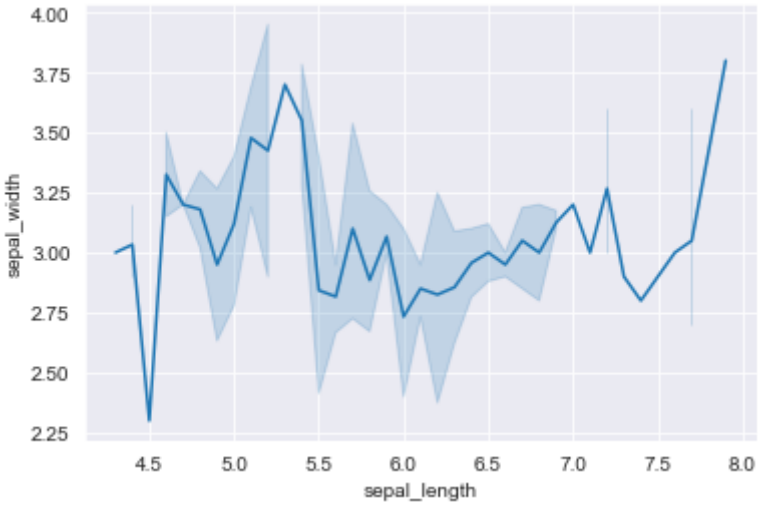
1. Changing Figure Aesthetic

`set_style()` method is used to set the aesthetic of the plot. It means it affects things like the color of the axes, whether the grid is active or not, or other aesthetic elements. There are five themes available in Seaborn.

- darkgrid
- whitegrid
- dark
- white
- ticks

Syntax: `set_style(style=None, rc=None)`

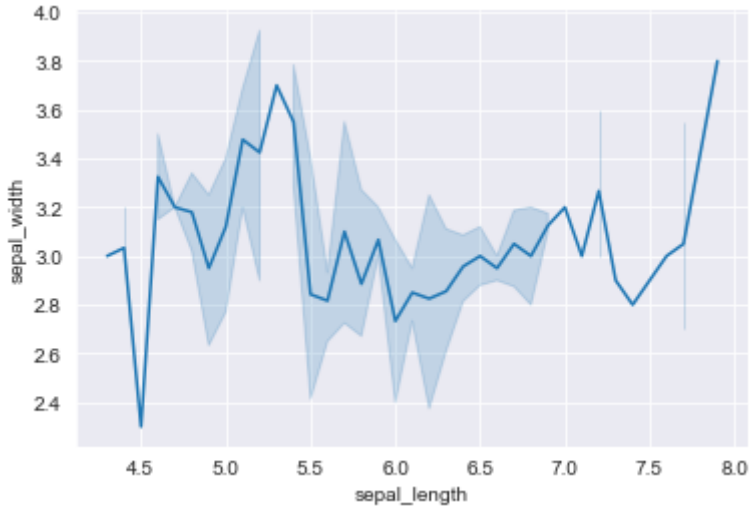
In [8]:
```
sns.set_style("darkgrid")
sns.lineplot(x = "sepal_length", y = "sepal_width", data = data)
plt.show()
```



2. Removal of Spines

Spines are the lines noting the data boundaries and connecting the axis tick marks. It can be removed using the `despine()` method.

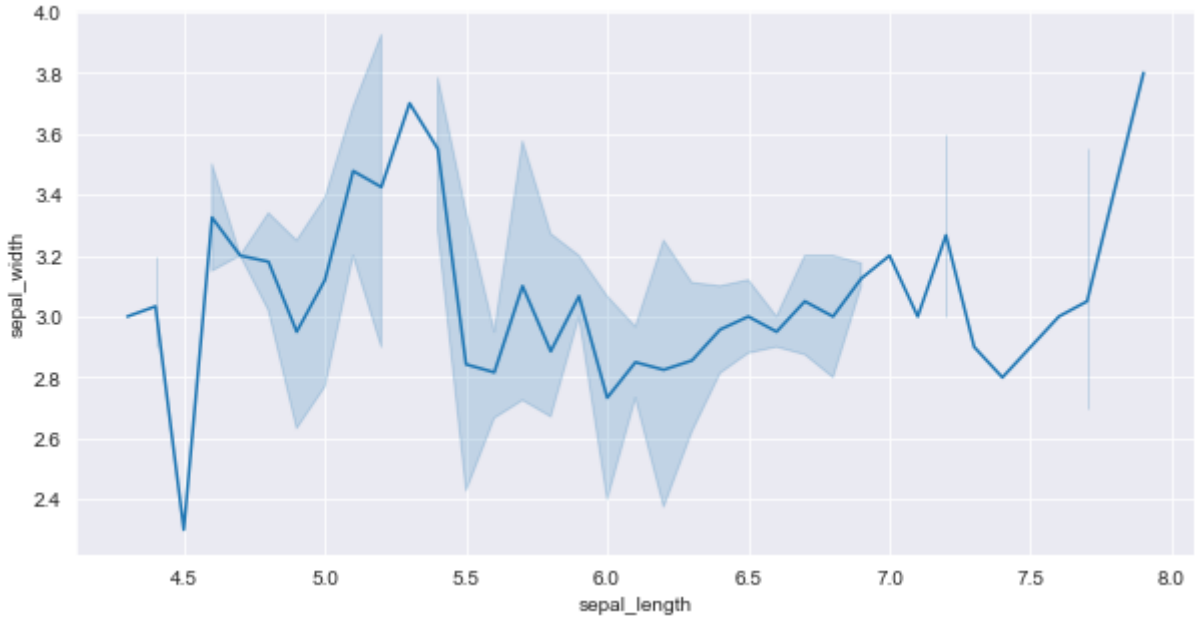
In [9]:
```
sns.lineplot(x = "sepal_length", y = "sepal_width", data = data)
sns.despine()
plt.show()
```



3. Changing Figure Size

The figure size can be changed using the `figure()` method of Matplotlib. `figure()` method creates a new figure of the specified size passed in the `figsize` parameter.

In [10]:
```
plt.figure(figsize = (10,5))

sns.lineplot(x = "sepal_length", y = "sepal_width", data = data)
sns.set_style("ticks")
plt.show()
```

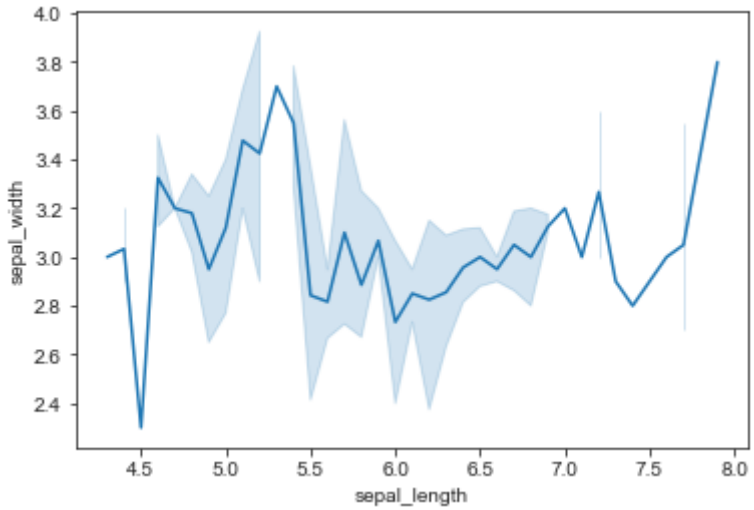


4. Scaling the Plots

It can be done using the `set_context()` method. It allows us to override default parameters. This affects things like the size of the labels, lines, and other elements of the plot, but not the overall style. The base context is " `notebook` ", and the other contexts are " `paper` ", " `talk` ", and " `poster` ". `font_scale` sets the font size.

In [11]:
```
sns.lineplot(x = "sepal_length", y = "sepal_width", data = data)
sns.set_style("white")
```

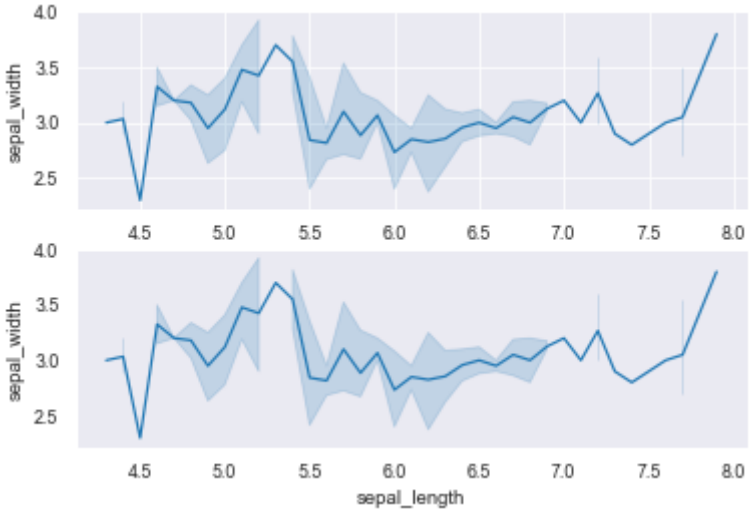
```
sns.set_context("paper")
plt.show()
```



5. Setting Temporary Style

`axes_style()` method is used to set the style temporarily. It is used along with the `with` statement.

In [12]:
```
with sns.axes_style('darkgrid'):
    plt.subplot(2,1,1)
    sns.lineplot(x = "sepal_length", y = "sepal_width", data = data)


with sns.axes_style('dark'):
    plt.subplot(2,1,2)
    sns.lineplot(x = "sepal_length", y = "sepal_width", data = data)

plt.show()
```



Color Palette

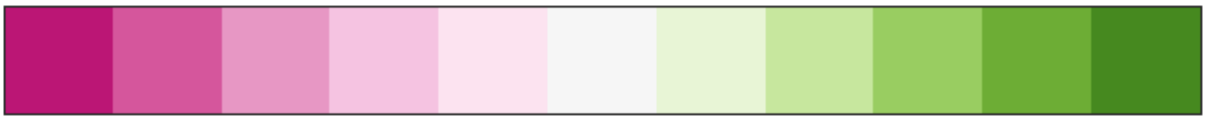
Colormaps are used to visualize plots effectively and easily. One might use different sorts of colormaps for different kinds of plots. `color_palette()` method is used to give colors to the plot. Another function `palplot()` is used to deal with the color palettes and plots the color palette as a horizontal array.

In [13]:
```
sns.palplot(sns.color_palette())
plt.show()
```

1. Diverging Color Palette

This type of color palette uses two different colors where each color depicts different points ranging from a common point in either direction. Consider a range of -10 to 10 so the value from -10 to 0 takes one color and values from 0 to 10 take another.

In [14]:
```
sns.palplot(sns.color_palette('PiYG',11))
plt.show()
```



2. Sequential Color Palette

A sequential palette is used where the distribution ranges from a lower value to a higher value. To do this add the character ' `s` ' to the color passed in the color palette.

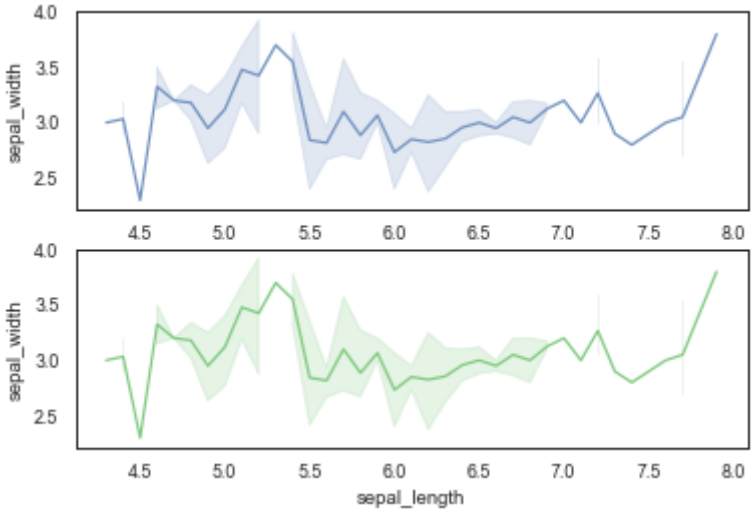
In [15]:
```
sns.palplot(sns.color_palette('Greens',11))
plt.show()
```

3. Setting Default Color Palette

`set_palette()` method is used to set the default color palette for all the plots. The arguments for both `color_palette()` and `set_palette()` is same. `set_palette()` changes the default matplotlib parameters.

In [16]:
```
sns.set_palette('vlag')
plt.subplot(2,1,1)
sns.lineplot(x = "sepal_length", y = "sepal_width", data = data)

sns.set_palette('Accent')
plt.subplot(2,1,2)
sns.lineplot(x = "sepal_length", y = "sepal_width", data = data)
```

```
plt.show()
```



Multiple Plots

Multiple plots in Seaborn can also be created using the Matplotlib as well as Seaborn also provides some functions for the same. Seaborn mainly provides the `FacetGrid()` method and `PairGrid()` method.

In [17]:
```
data.head()
```

Out[17]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [18]:
```
data.tail()
```

Out[18]:

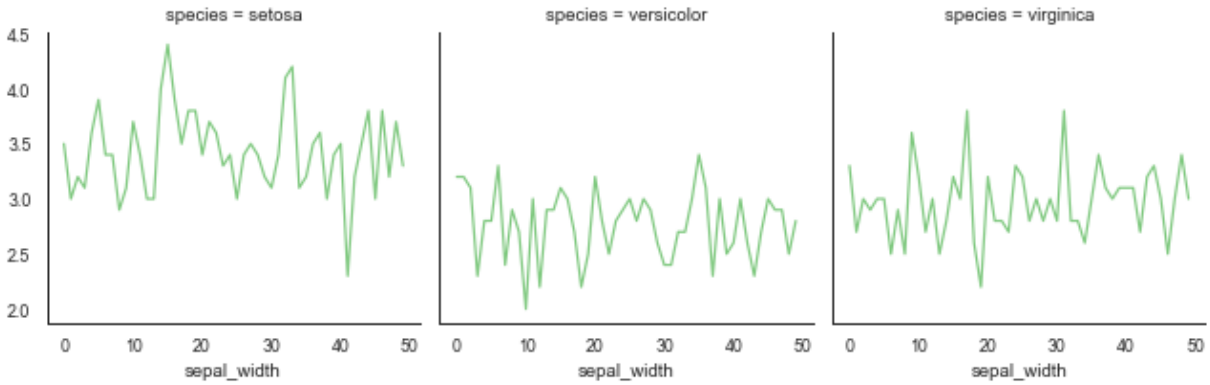
	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

1. `FacetGrid( )`

- FacetGrid class helps in **visualizing distribution of one variable as well as the relationship between multiple variables separately within subsets of your dataset using multiple panels.**
- A FacetGrid can be drawn with up to three dimensions - **row, col, and hue**. The first two have obvious correspondence with the resulting array of axes; think of the hue variable as a third dimension along a depth axis, where different levels are plotted with different colors.
- FacetGrid object takes a dataframe as input and the names of the variables that will form the row, column, or hue dimensions of the grid. **The variables should be categorical and the data at each level of the variable will be used for a facet along that axis.**

In [19]:
```
#to compare all the different species with their sepal width
plot = sns.FacetGrid(data, col = "species")
plot.map(plt.plot, "sepal_width")
```

Out[19]:
```
<seaborn.axisgrid.FacetGrid at 0x272b4ae8cd0>
```

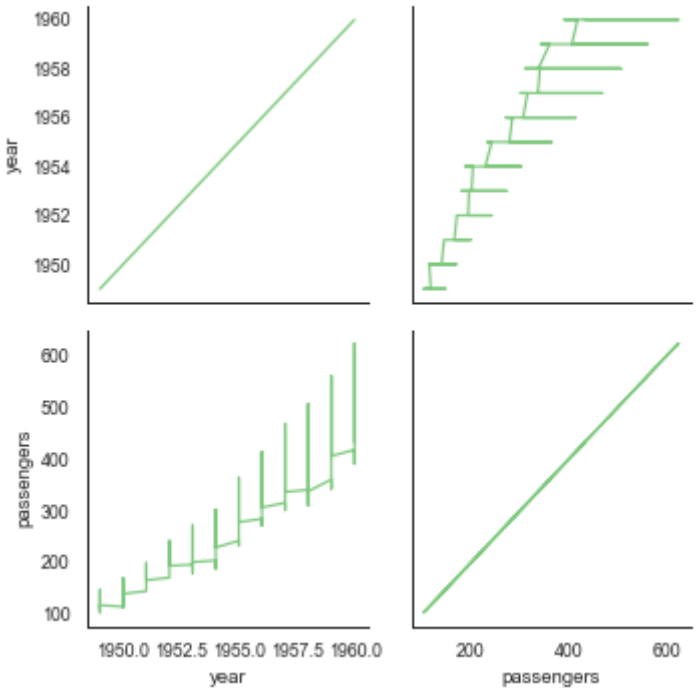


2. `PairGrid( )`

- Subplot grid for plotting pairwise relationships in a dataset.
- This class maps each variable in a dataset onto a column and row in a grid of multiple axes. Different axes-level plotting functions can be used to draw bivariate plots in the upper and lower triangles, and the marginal distribution of each variable can be shown on the diagonal.
- It can also represent an additional level of conventionalization with the hue parameter, which plots different subsets of data in different colors. This uses color to resolve elements on a third dimension, but only draws subsets on top of each other and will not tailor the hue parameter for the specific visualization the way that axes-level functions that accept hue will.

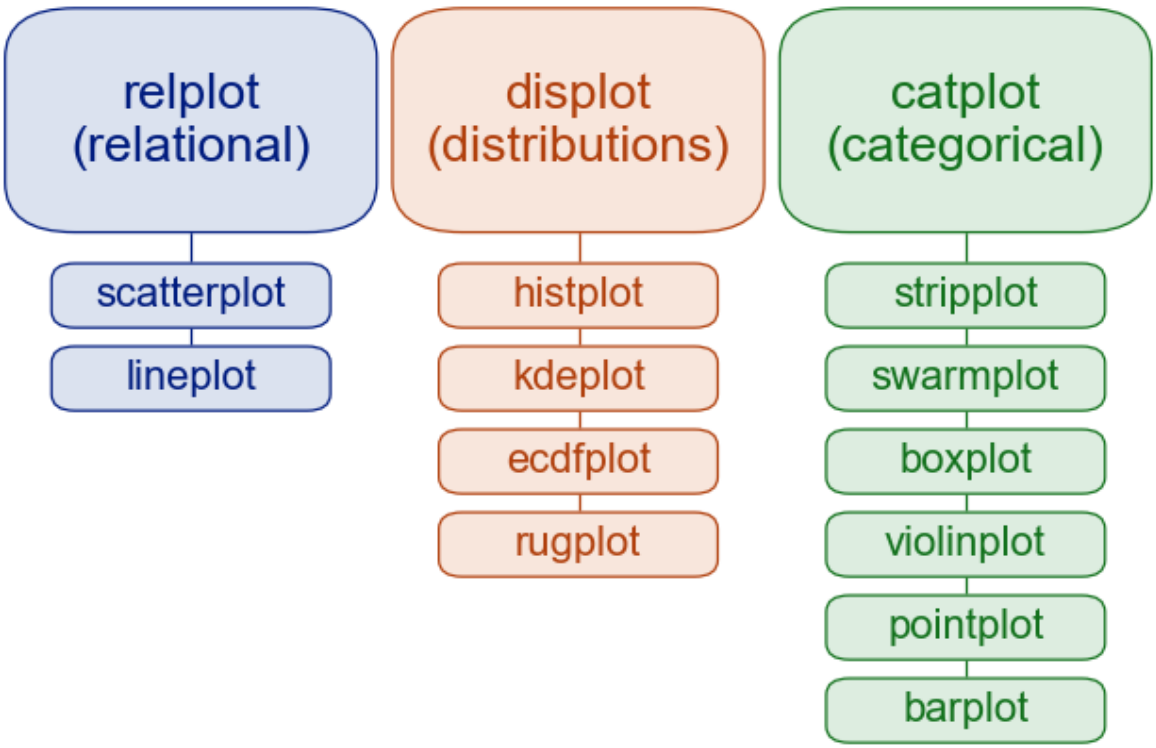
In [20]:
```
data = sns.load_dataset("flights")
plot = sns.PairGrid(data)
plot.map(plt.plot)
```

Out[20]:
```
<seaborn.axisgrid.PairGrid at 0x272b4954d60>
```



Creating Different Types of Plots

Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions. Most of your interactions with seaborn will happen through a set of plotting functions. The organization looks a bit like this:



There's also Regression Plot which is divide into:

- `lmplot( )`
- `regplot( )`
- Matrixplot
- Heatmap
- `Clustermap( )`

1. Relational Plots

Relational plots are used for visualizing the statistical relationship between the data points. Visualization is necessary because it allows us to see trends and patterns in the data. The process of understanding how the variables in the dataset relate each other and their relationships is termed as Statistical analysis. We will be using a default dataset named 'tips'. This dataset gives information about people who had food at some restaurant and whether they left tip for waiters or not, their gender and whether they do smoke or not, and more.

In [21]:
```
data = sns.load_dataset("tips")
```

In [22]:
```
data.head()
```

Out[22]:

	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

To draw the relational plots seaborn provides three functions. These are:

- `relplot()`
- `scatterplot()`
- `lineplot()`

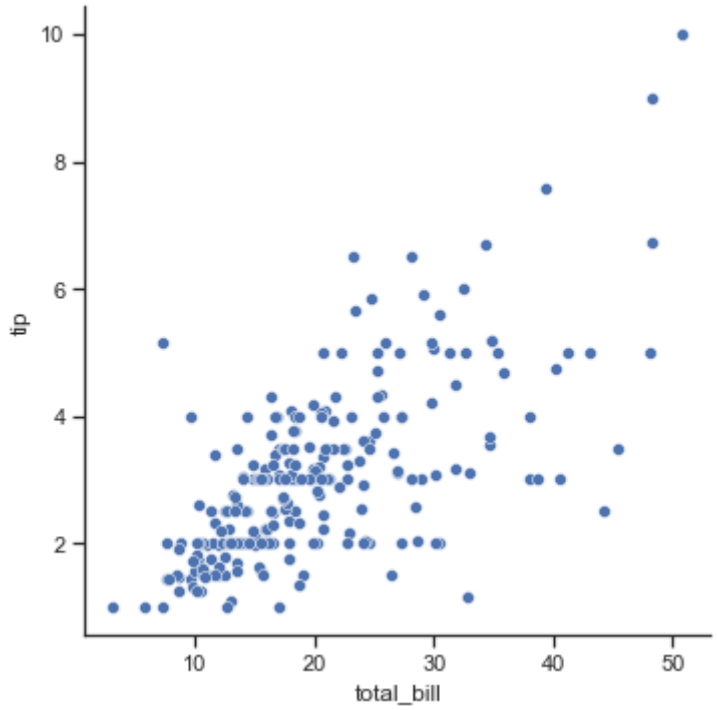
1.1 `sns.relplot( )`

This function provides us the access to some other different axes-level functions which shows the relationships between two variables with semantic mappings of subsets. Syntax:
`sns.relplot(x=None, y=None, data=None, **kwargs)`

In [23]:
```
sns.set(style = 'ticks')
tips = sns.load_dataset("tips")
```

In [24]:
```
sns.relplot(x = "total_bill", y = "tip", data = tips)
```

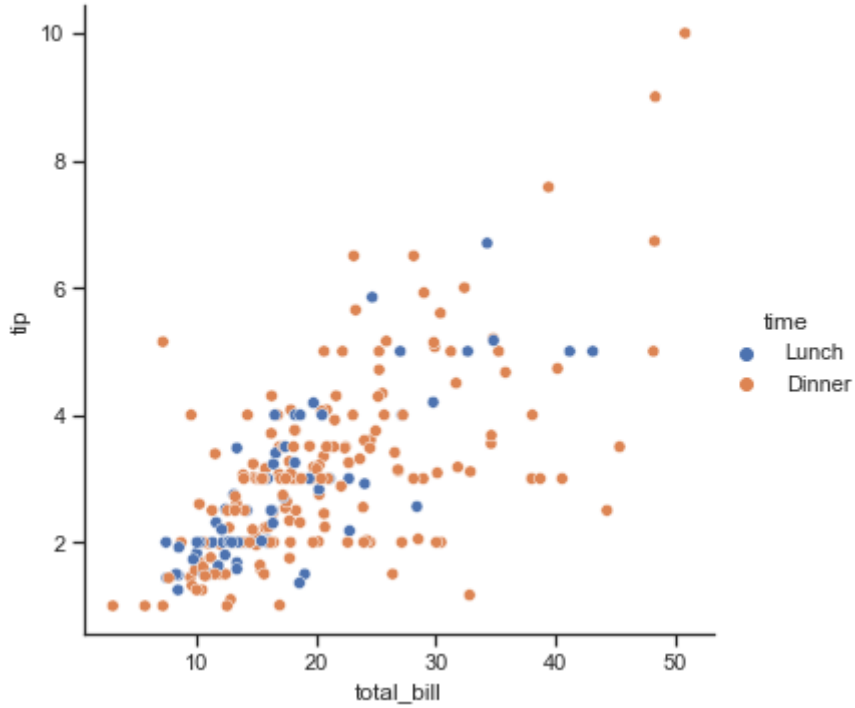
Out[24]:
```
<seaborn.axisgrid.FacetGrid at 0x272b4941eb0>
```



Grouping on basis of time category

In [25]:
```
sns.relplot(x = "total_bill", y = "tip", hue = "time", data = tips)
```

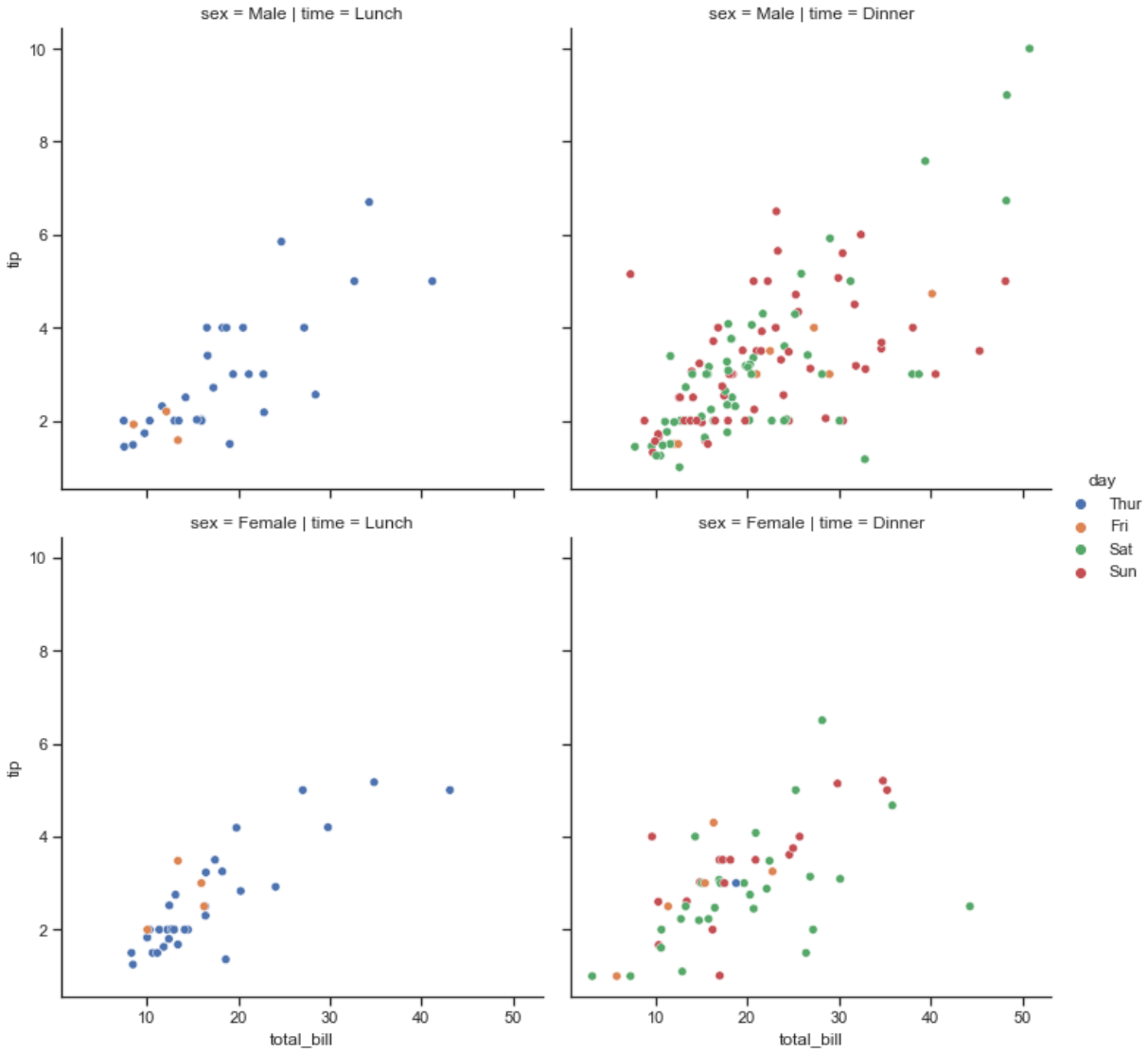
Out[25]: <seaborn.axisgrid.FacetGrid at 0x272b492ee20>



Using time and sex for determining the facet of the grid.

In [26]:
```
sns.relplot(x = "total_bill", y = "tip", hue = "day", row = "sex", col = "time", dat
```

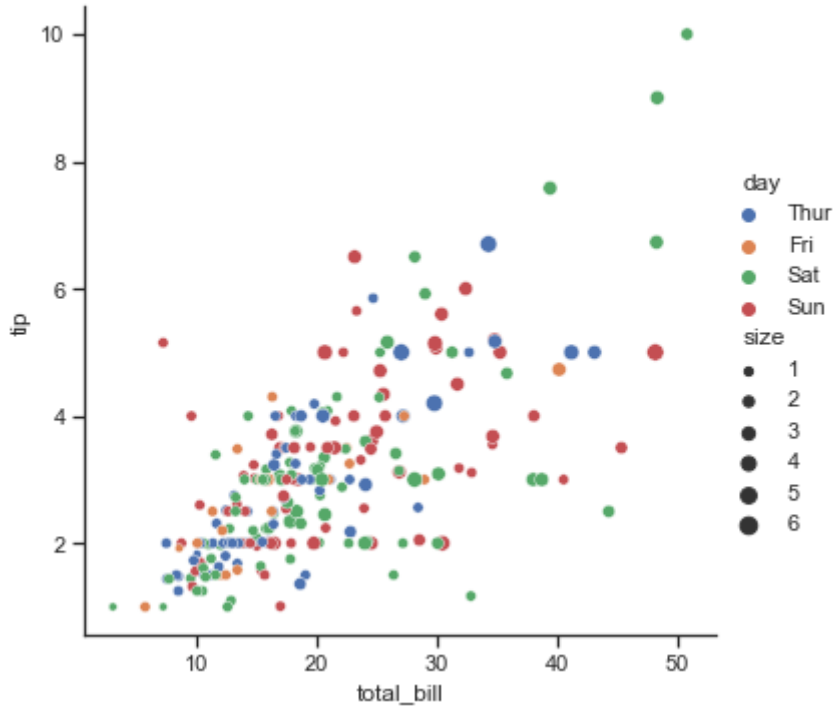
Out[26]: <seaborn.axisgrid.FacetGrid at 0x272b4c9cf40>



Using size attribute, we can see data points having different size.

In [27]:
```
sns.relplot(x = "total_bill", y = "tip", hue = "day", size = "size", data = tips)
```

Out[27]:
```
<seaborn.axisgrid.FacetGrid at 0x272b4a38940>
```

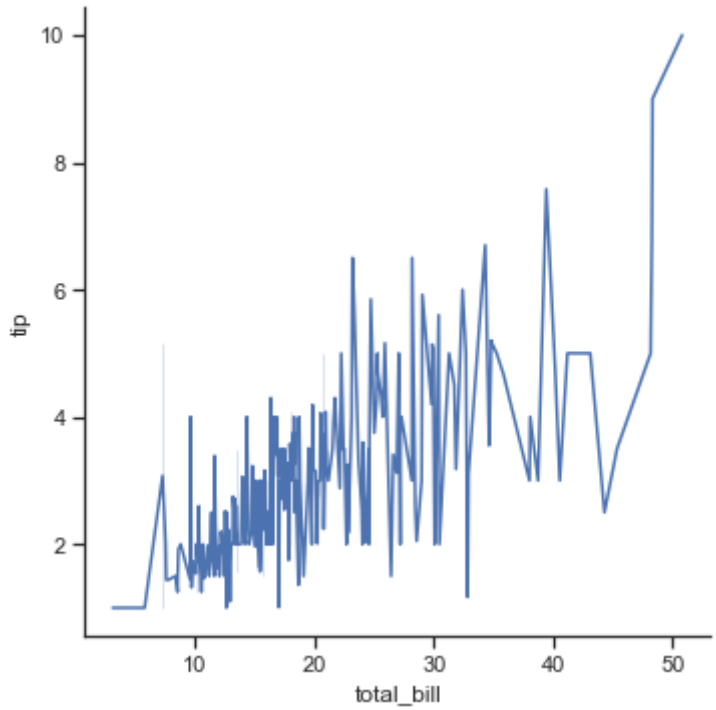


Getting lineplot in `relplot`

By default the `kind` argument is set to `scatter`. However, to get line plot we can pass its value as `line`

In [28]:
```
sns.relplot(x = "total_bill", y = "tip", kind = "line", data = tips)
```

Out[28]:
```
<seaborn.axisgrid.FacetGrid at 0x272b4fdfe50>
```



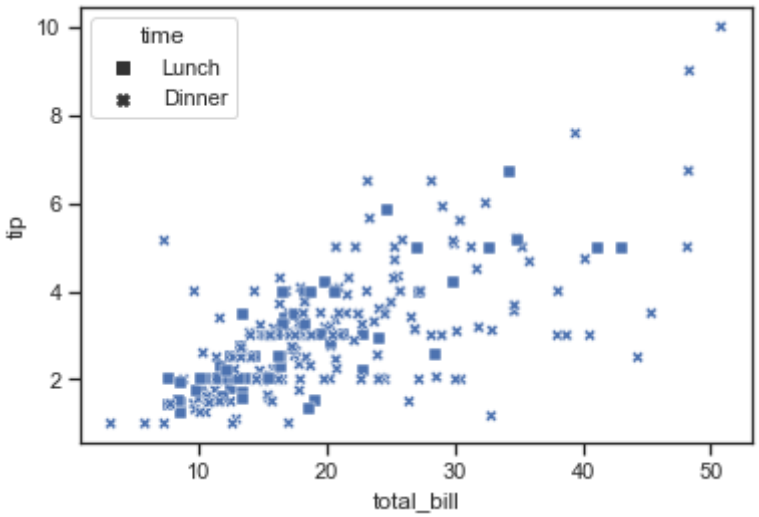
1.2 `sns.scatterplot( )`

The scatter plot is a mainstay of statistical visualization. It depicts the joint distribution of two variables using a cloud of points, where each point represents an observation in the dataset. This depiction allows the eye to infer a substantial amount of information about whether there is any

meaningful relationship between them. Syntax: `sns.scatterplot(x=None, y=None, data=None, **kwargs)`

In [29]:
```
markers = {"Lunch":"s", "Dinner":"X"}
sns.scatterplot(x = "total_bill", y = "tip", style = "time", markers = markers, data
```

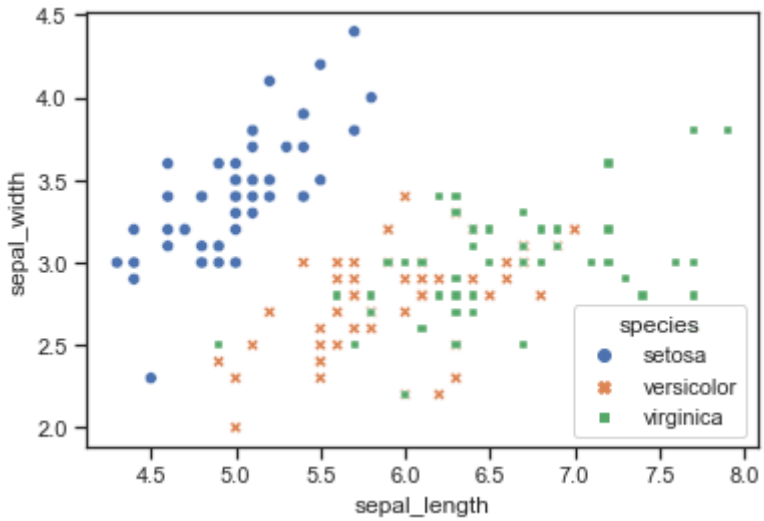
Out[29]: <AxesSubplot:xlabel='total_bill', ylabel='tip'>



We can also pass dataframe vectors as axes arguments' values instead of column names

In [30]:
```
data = sns.load_dataset("iris")
sns.scatterplot( x = data.sepal_length, y = data.sepal_width, hue = data.species, st
```

Out[30]: <AxesSubplot:xlabel='sepal_length', ylabel='sepal_width'>



1.3 `sns.lineplot( )`

Scatter plots are highly effective, but there is no universally optimal type of visualization. For certain datasets, you may want to consider changes as a function of time in one variable, or as a similarly continuous variable. In this case, drawing a line-plot is a better option.

Syntax: `seaborn.lineplot(x=None, y=None, data=None, **kwargs)`

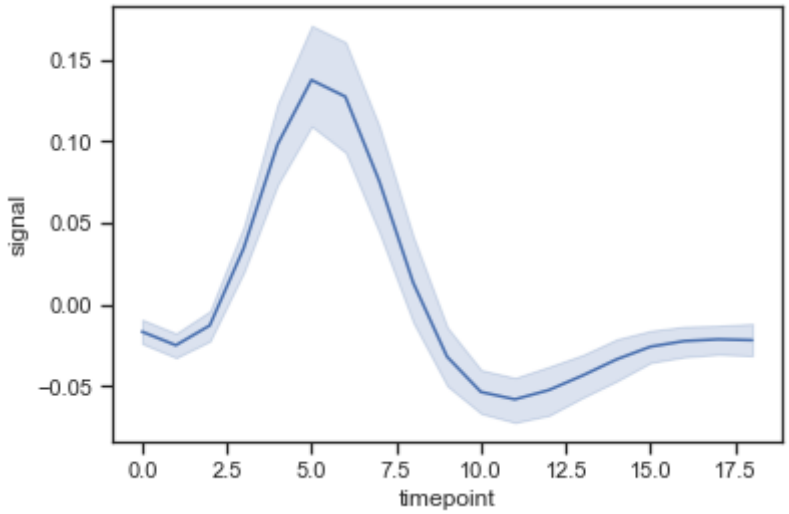
In [31]:
```
fmri = sns.load_dataset("fmri")
fmri.head()
```

Out[31]:

	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

In [32]:
```
sns.lineplot(x = fmri.timepoint, y = fmri.signal)
```

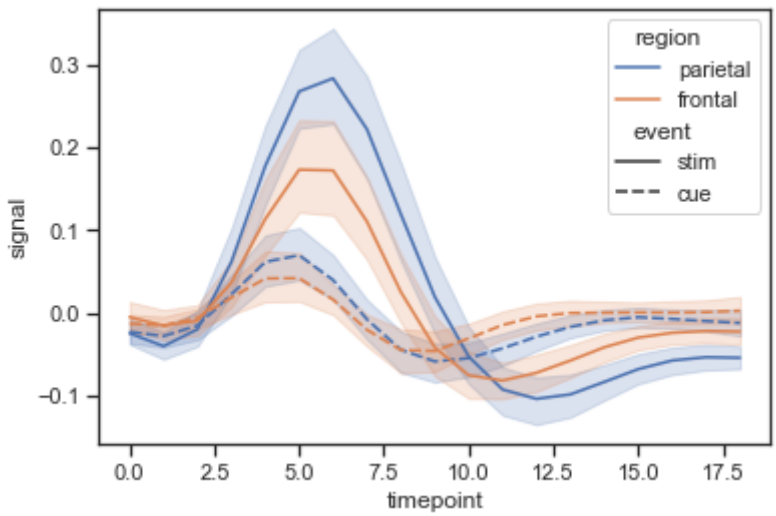
Out[32]:
```
<AxesSubplot:xlabel='timepoint', ylabel='signal'>
```



Grouping data points on the basis of category, here as region and event.

In [33]:
```
sns.lineplot(x = fmri.timepoint, y = fmri.signal, hue = fmri.region, style = fmri.ev
```

Out[33]:
```
<AxesSubplot:xlabel='timepoint', ylabel='signal'>
```

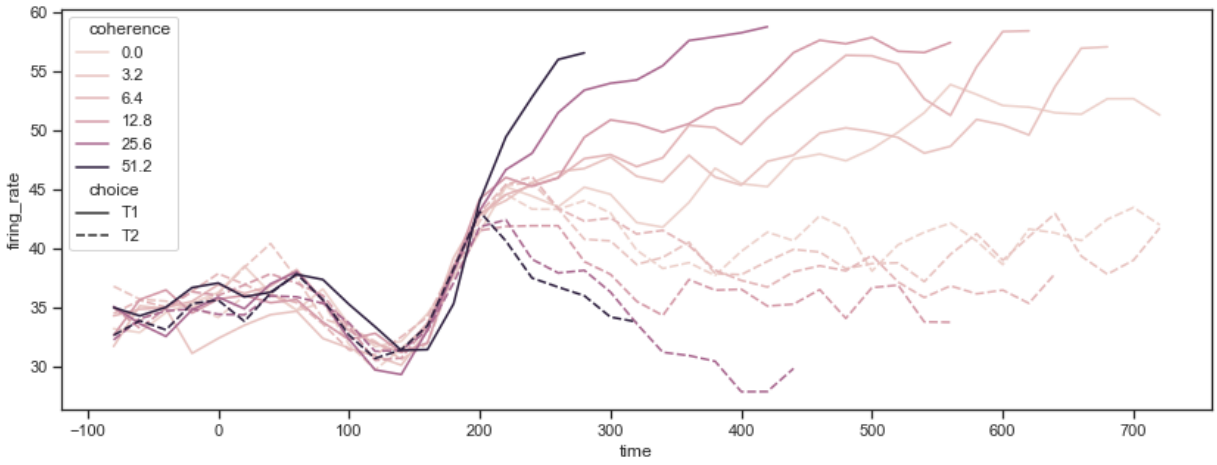


A complex plot visualizing “dots” dataset, to show the power of seaborn. Here, in this example, quantitative color mapping is used.

The 'tips' dataset contains information about people who probably had food at a restaurant and whether or not they left a tip for the waiters, their gender, whether they smoke and so on.

In [34]:
```
data = sns.load_dataset("dots").query("align == 'dots'")
plt.figure(figsize = (14,5))
sns.lineplot(x = data.time, y = data.firing_rate, hue = data.coherence, style = data
```

Out[34]:
```
<AxesSubplot:xlabel='time', ylabel='firing_rate'>
```



---

2. Categorical Plots

Categorical Plots are used where we have to visualize relationship between two numerical values. A more specialized approach can be used if one of the main variable is categorical which means such variables that take on a fixed and limited number of possible values. We will be using 'tips' dataset for our visualization.

In [35]:
```
df = sns.load_dataset('tips')
```

In [36]:
```
df.head()
```

Out[36]:

	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

2.1 Barplot

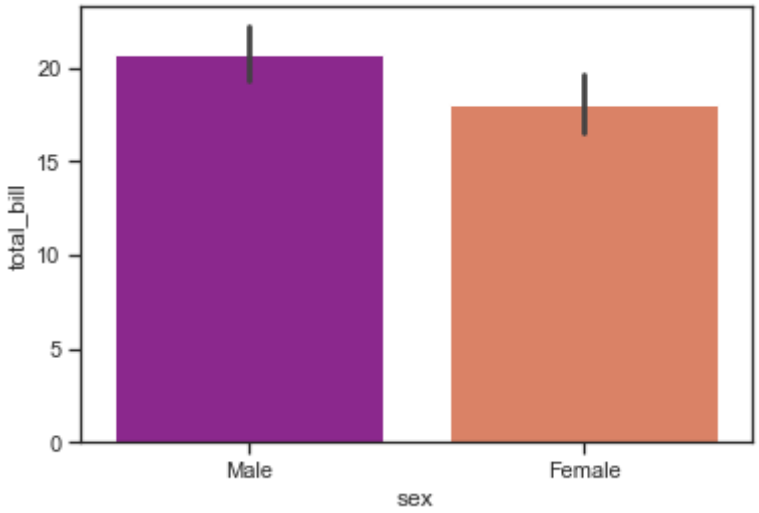
A barplot is basically used to aggregate the categorical data **according to some methods and by default its the mean**. It can also be understood as a visualization of the group by action. To use this plot we choose a **categorical column for the x axis** and a **numerical column for the y axis** and we see that it creates a plot taking a **mean per categorical column.**

Syntax: `barplot([x, y, hue, data, order, hue_order, …])`

- `palette` is used to set the color of the plot
- `estimator` is used as a statistical function for estimation within each categorical bin.

In [37]:
```
# plot the graph using the default estimator mean
sns.barplot(x = df.sex, y = df.total_bill, palette = 'plasma')
```

Out[37]:
```
<AxesSubplot:xlabel='sex', ylabel='total_bill'>
```

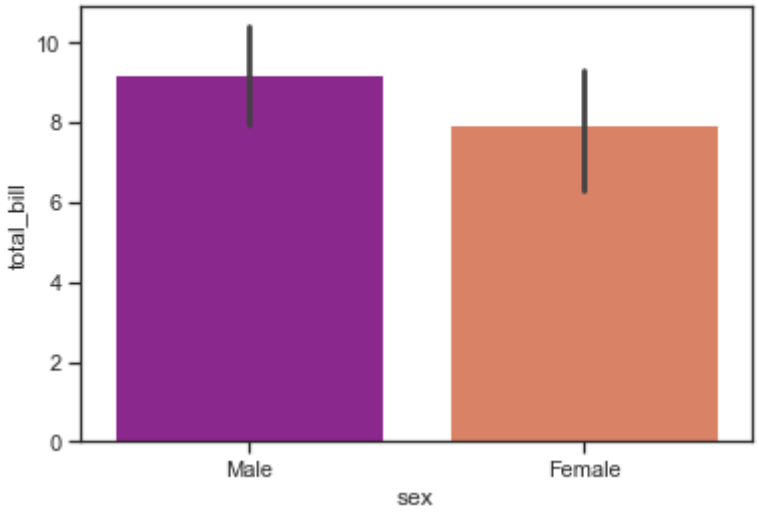


Seeing the above plot, one can concur that an average male has a bill of 20 while average female has less than 20.

Seeing the below plot, one can concur that a male has a std. deviation of 9 in total bill while female has a std.deviation less than 8 in total bill.

In [38]:
```
sns.barplot(x = df.sex, y = df.total_bill, palette = 'plasma', estimator = np.std)
```

Out[38]:
```
<AxesSubplot:xlabel='sex', ylabel='total_bill'>
```



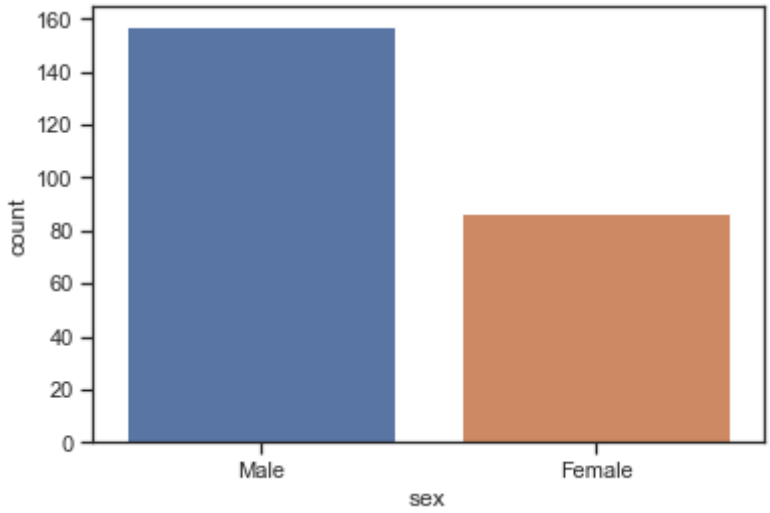
2.2 Countplot

A countplot basically counts the categories and returns a count of their occurrences. It is one of the most simple plots provided by the seaborn library.

Syntax: `countplot([x, y, hue, data, order, …])`

In [39]:
```
sns.countplot(x = df.sex)
```

Out[39]: `<AxesSubplot:xlabel='sex', ylabel='count'>`



Looking at the plot we can say that the number of males is more than the number of females in the dataset. As it only returns the count based off a categorical column, we need to specify only the x parameter.

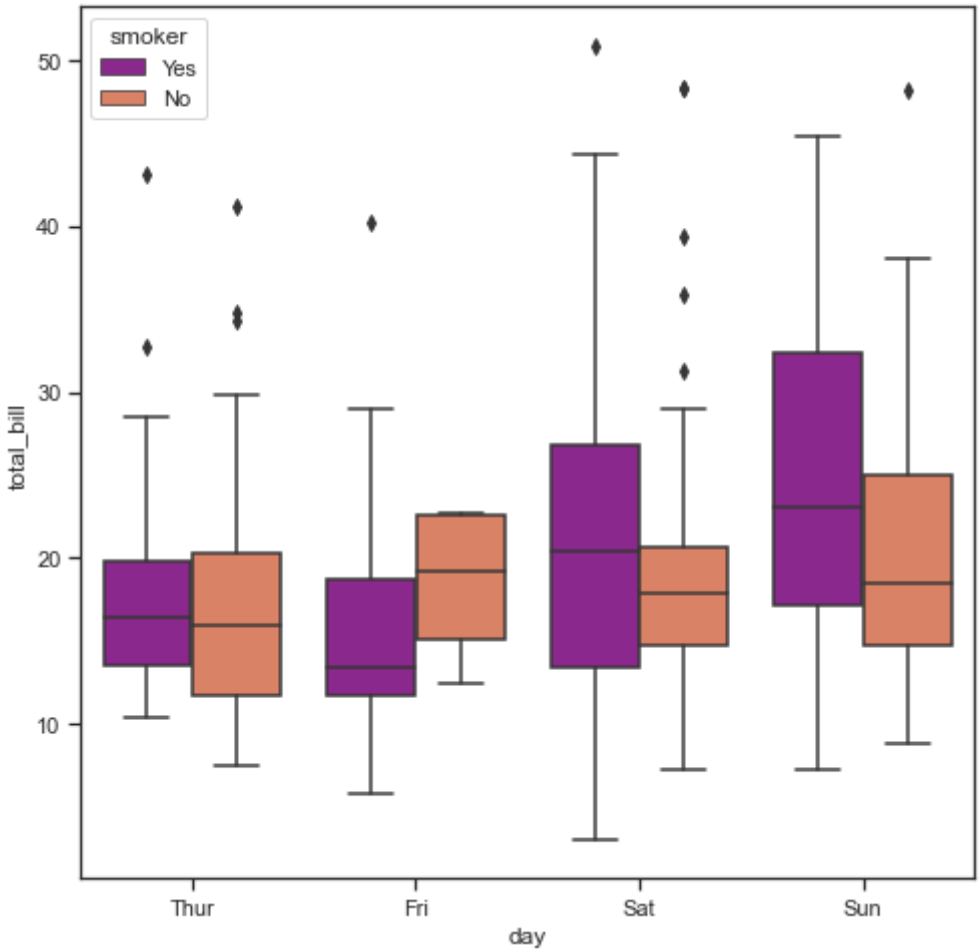
2.3 Boxplot

A boxplot is sometimes known as the box and whisker plot.It shows the distribution of the quantitative data that represents the comparisons between variables. **Boxplot shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution i.e. the *dots indicating the presence of outliers*.**

Syntax: `sns.boxplot([x, y, hue, data, order, hue_order, …])`

In [40]:
```
plt.figure(figsize = (8,8))
sns.boxplot(x = df.day, y = df.total_bill, hue = df.smoker, palette = 'plasma')
```

Out[40]: `<AxesSubplot:xlabel='day', ylabel='total_bill'>`

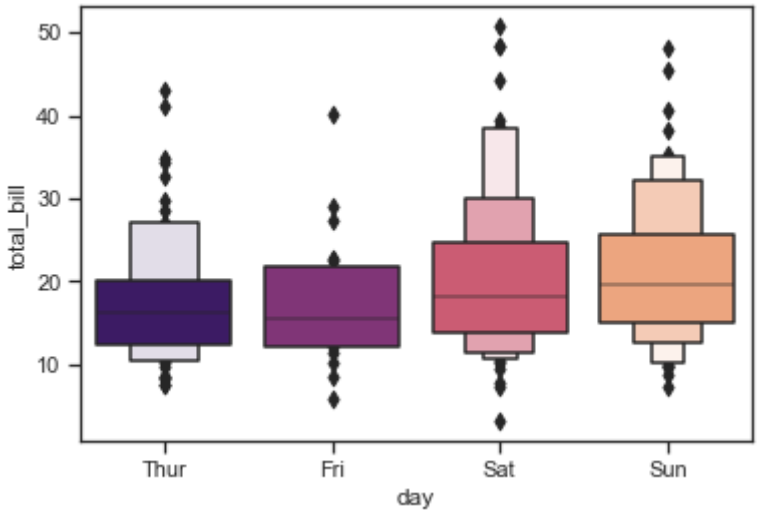


x takes the categorical column and y is a numerical column. Hence we can see the total bill spent each day. 'hue' parameter is used to further add a categorical separation. By looking at the plot we can say that the **people who do not smoke had a higher bill on Friday as compared to the people who smoked.**

`sns.boxenplot()` : Draw an enhanced box plot for larger datasets. This style of plot was originally named a “letter value” plot because it shows a large number of quantiles that are defined as “letter values”. It is similar to a box plot in plotting a nonparametric representation of a distribution in which all features correspond to actual observations. By plotting more quantiles, it provides more information about the shape of the distribution, particularly in the tails.

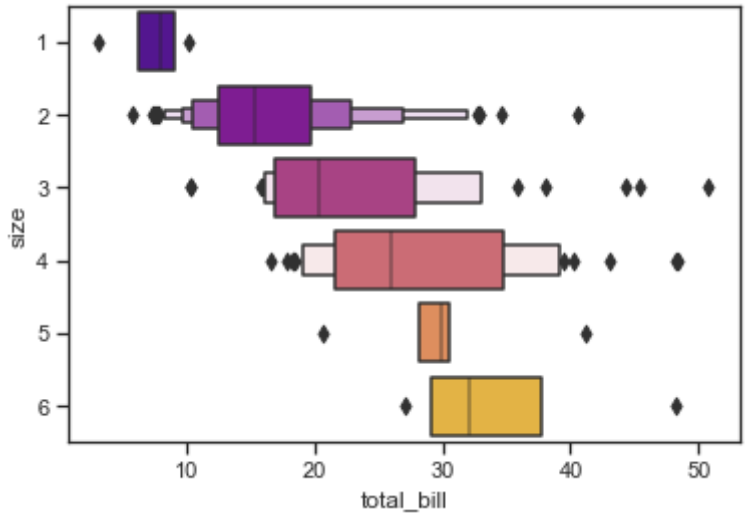
In [41]:
```
data = sns.load_dataset("tips")
sns.boxenplot(x = "day", y = "total_bill", data = data, palette = 'magma')
```

Out[41]:
```
<AxesSubplot:xlabel='day', ylabel='total_bill'>
```



In [42]:
```
sns.boxenplot(x = "total_bill", y = "size", data = data, palette = 'plasma', orient
```

Out[42]: <AxesSubplot:xlabel='total_bill', ylabel='size'>



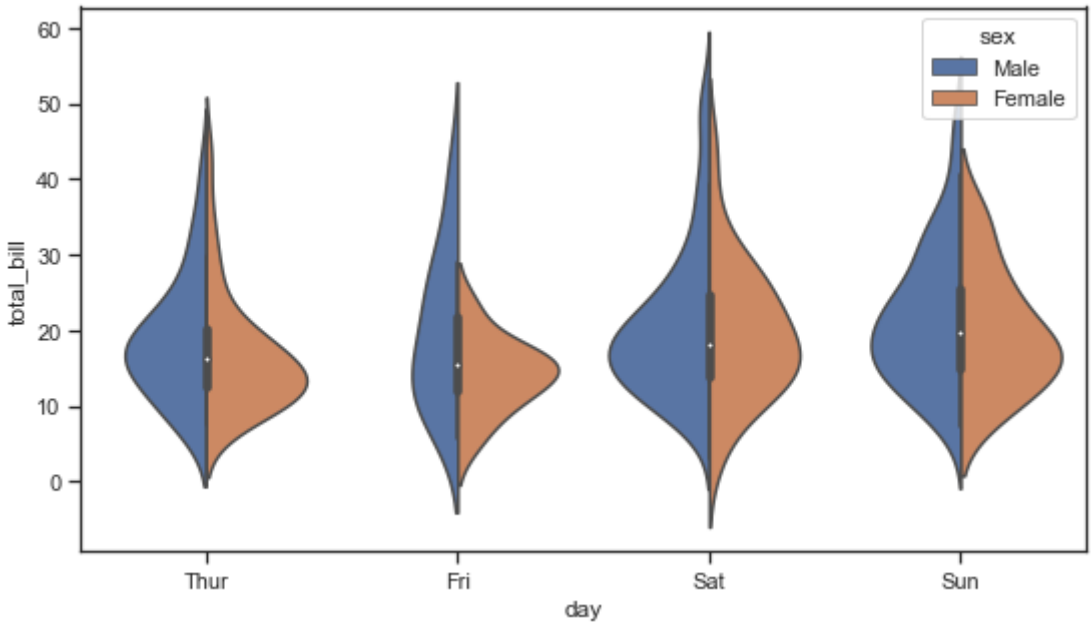
2.4 Violinplot

It is similar to the boxplot except that it provides a higher, more advanced visualization and uses the kernel density estimation to give a better description about the data distribution.

Syntax: `violinplot([x, y, hue, data, order, …])`

In [43]:
```
plt.figure(figsize = (9,5))
sns.violinplot(x = 'day', y ='total_bill', data = df, hue = 'sex', split = True)
```

Out[43]: <AxesSubplot:xlabel='day', ylabel='total_bill'>



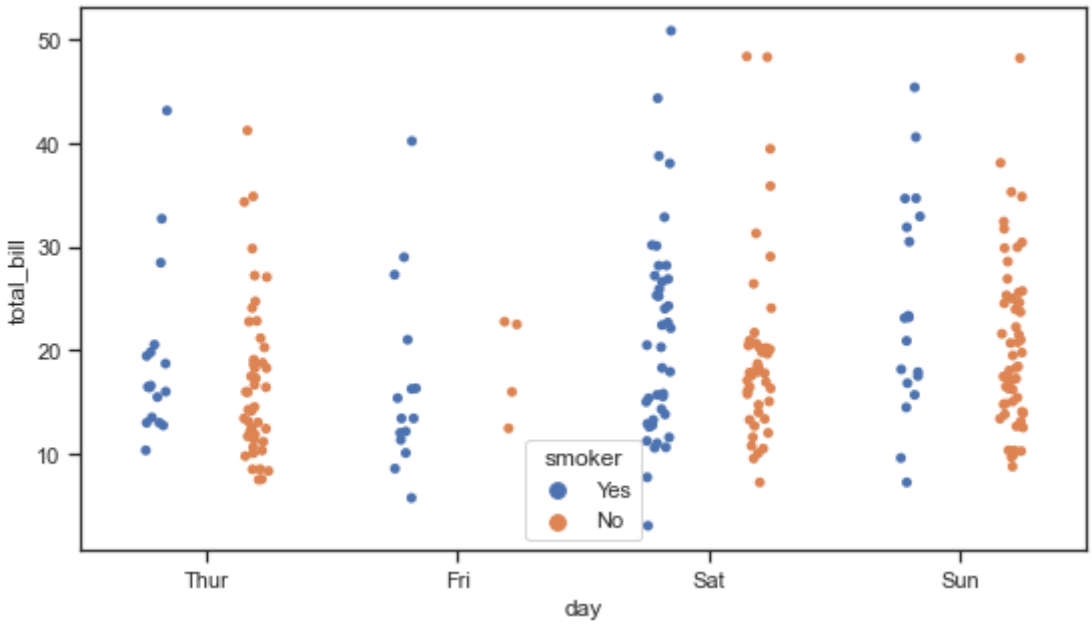
`hue` is used to separate the data further using the sex category. Setting `split = True` will draw half of a violin for each level. This can make it easier to directly compare the distributions.

2.5 Stripplot

It basically creates a scatter plot based on the category. *Syntax*: `stripplot([x, y, hue, data, order, …])`

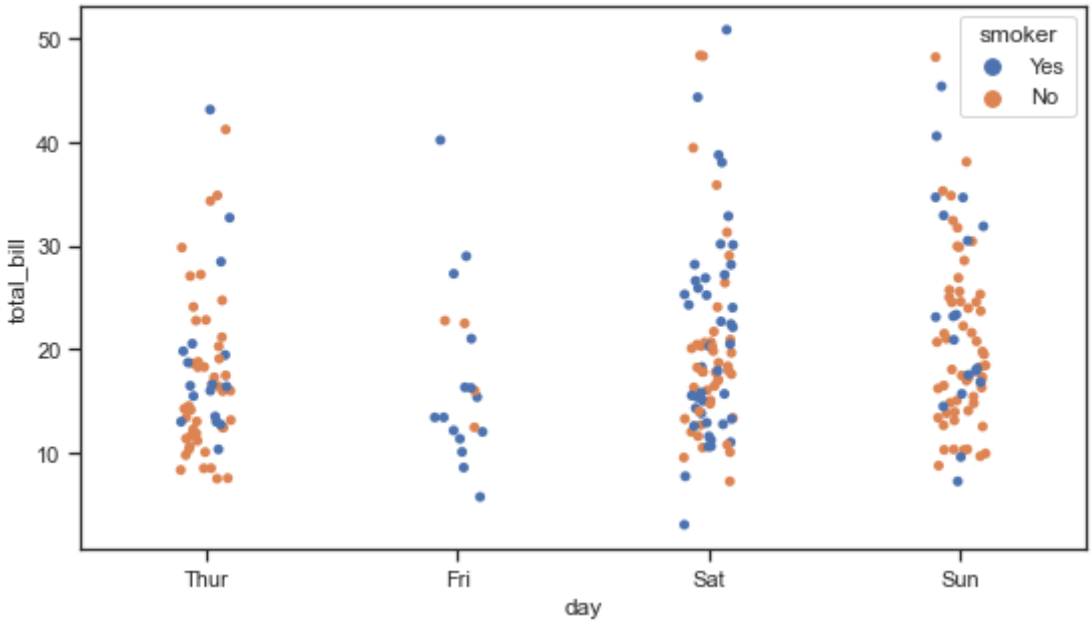
In [44]:
```
plt.figure(figsize = (9,5))
sns.stripplot(x ='day', y ='total_bill', data = df, jitter = True, hue ='smoker', do
```

Out[44]:
```
<AxesSubplot:xlabel='day', ylabel='total_bill'>
```



In [45]:
```
plt.figure(figsize = (9,5))
sns.stripplot(x ='day', y ='total_bill', data = df, jitter = True, hue ='smoker', do
```

Out[45]:
```
<AxesSubplot:xlabel='day', ylabel='total_bill'>
```



- One problem with strip plot is that you can't really tell which points are stacked on top of each other and hence we use the jitter parameter to add some random noise.
- `jitter` parameter is used to add an amount of jitter (only along the categorical axis) which can be useful when you have many points and they overlap, so that it is easier to see the distribution.
- `hue` is used to provide an addition categorical separation. Setting `split/dodge = True` is used to draw separate strip plots based on the category specified by the hue parameter.

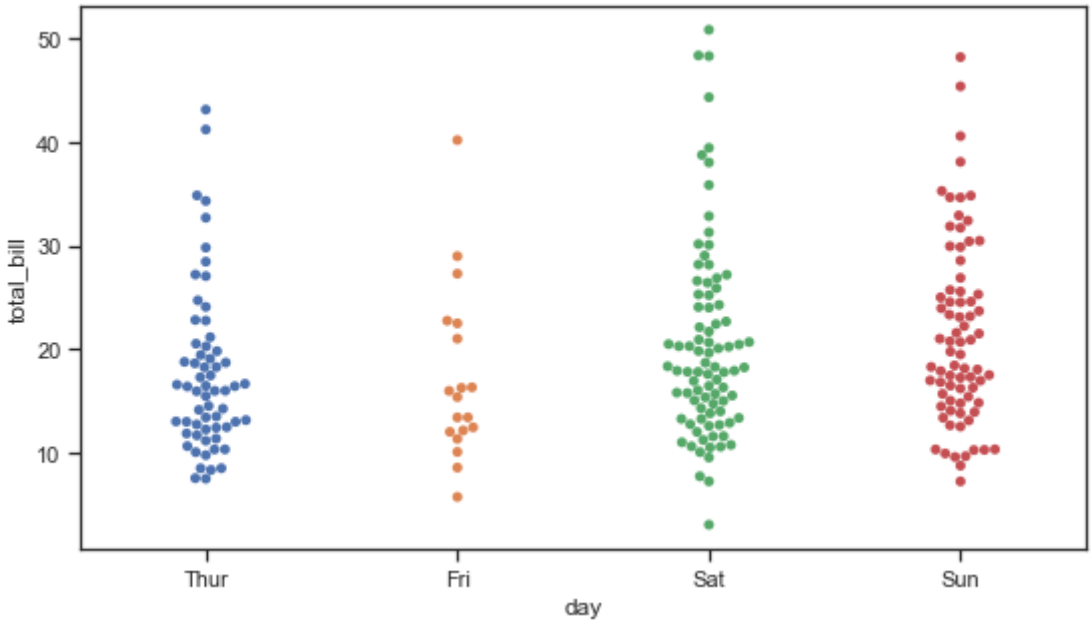
2.6 Swarmplot

It is very similar to the stripplot except the fact that the points are adjusted so that they do not overlap.Some people also like combining the idea of a violin plot and a stripplot to form this plot. One drawback to using swarmplot is that sometimes they dont scale well to really large numbers and takes a lot of computation to arrange them. So in case we want to visualize a swarmplot properly we can plot it on top of a violinplot.

Syntax: `swarmplot([x, y, hue, data, order, …])`

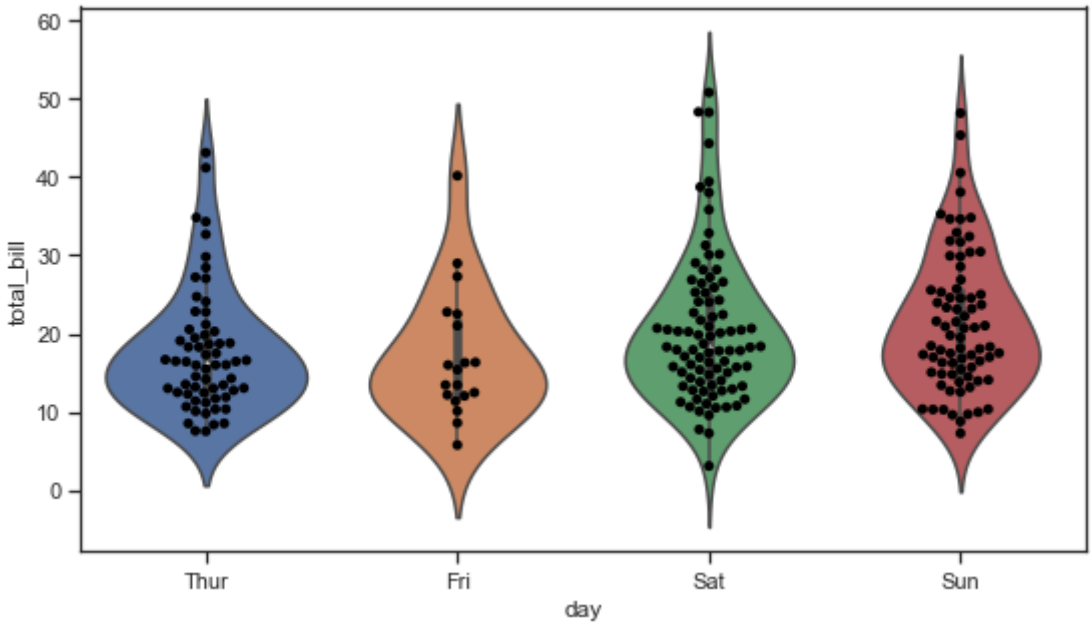
In [46]:
```
plt.figure(figsize = (9,5))
sns.swarmplot(x ='day', y ='total_bill', data = df)
```

Out[46]:
```
<AxesSubplot:xlabel='day', ylabel='total_bill'>
```



In [47]:
```
plt.figure(figsize = (9,5))
sns.violinplot(x ='day', y ='total_bill', data = df)
sns.swarmplot(x ='day', y ='total_bill', data = df, color ='black')
```

Out[47]: <AxesSubplot:xlabel='day', ylabel='total_bill'>



2.7 Factorplot

It is the most general of all these plots and provides a parameter called kind to choose the kind of plot we want thus saving us from the trouble of writing these plots separately. The kind parameter can be bar, violin, swarm etc.

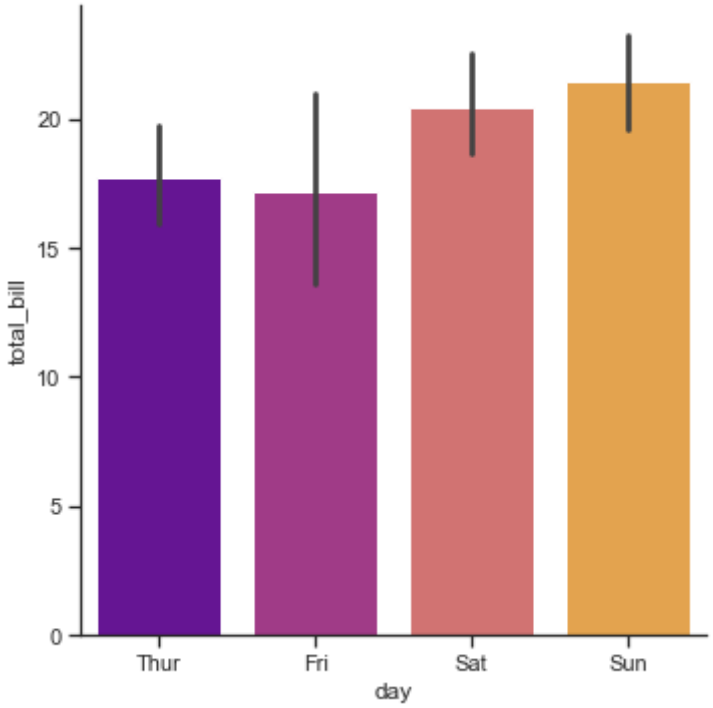
Syntax: `sns.factorplot([x, y, hue, data, row, col, kind…])`

In [48]:
```
sns.factorplot(x ='day', y ='total_bill', data = df, kind ='bar', palette = 'plasma'
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3714: UserWarning:
The `factorplot` function has been renamed to `catplot`. The original name will be r
emoved in a future release. Please update your code. Note that the default `kind` in
```

```
`factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
```

Out[48]: <seaborn.axisgrid.FacetGrid at 0x272b6a10e80>



---

3. Distribution Plots

Distribution Plots are used for examining univariate and bivariate distributions meaning such distributions that involve one variable or two discrete variables, i.e., for examining univariate and bivariate distributions. We will be discussing 4 types of distribution plots namely:

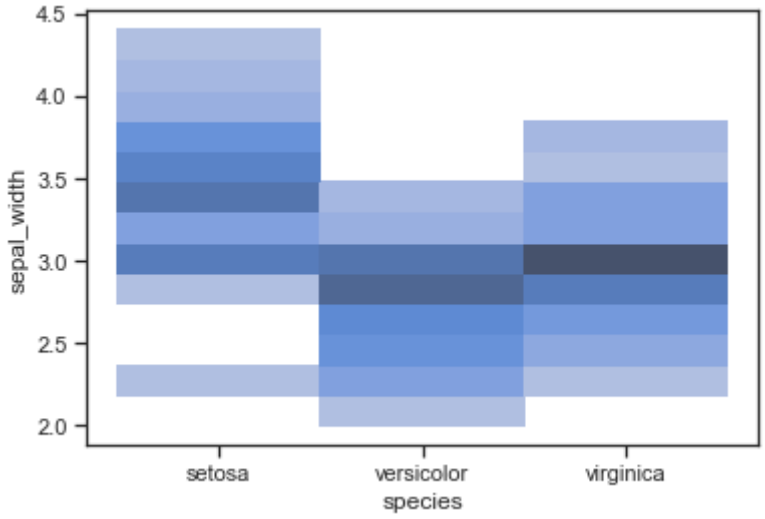
1. `histplot`
2. `distplot`
3. `joinplot`
4. `pairplot`
5. `rugplot`
6. `kdeplot`

3.0 Histplot

A histogram is basically used to represent data provided in a form of some groups.It is accurate method for the graphical representation of numerical data distribution. It can be plotted using the histplot() function.

Syntax: `histplot(data=None, *, x=None, y=None, hue=None,  **kwargs)`

In [49]:
```
data = sns.load_dataset("iris")

sns.histplot(x='species', y='sepal_width', data=data)
plt.show()
```



3.1 Displot

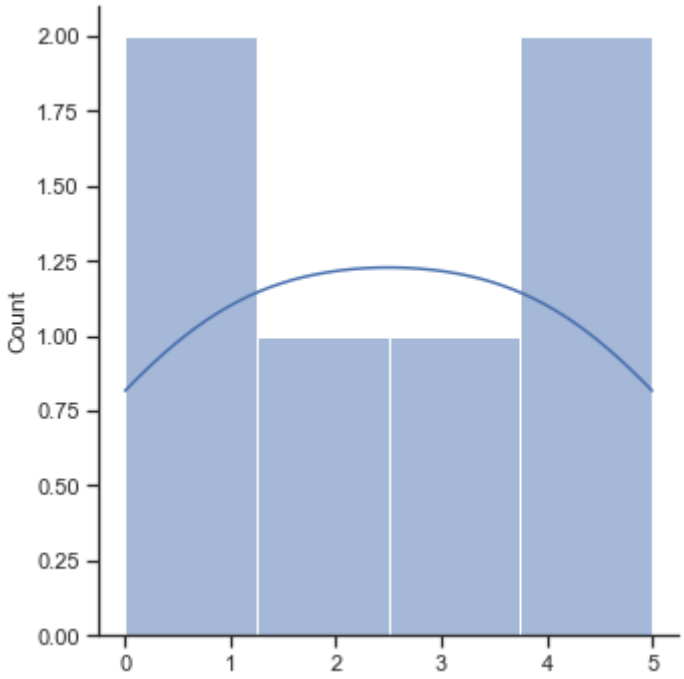
It is used basically for univariant set of observations and visualizes it through a histogram i.e. only one observation and hence we choose one particular column of the dataset.

Syntax: `displot(dist, bins, kde, rug, fit, ...])` (newer one doesnt accepts `hist` as parameter instead uses `kde`)

: `distplot(dist, bins, hist, rug, fit, ...])` (older one accepts `hist` as parameter)

In [50]:
```
sns.displot([0, 1, 2, 3, 4, 5], kde = True)
```

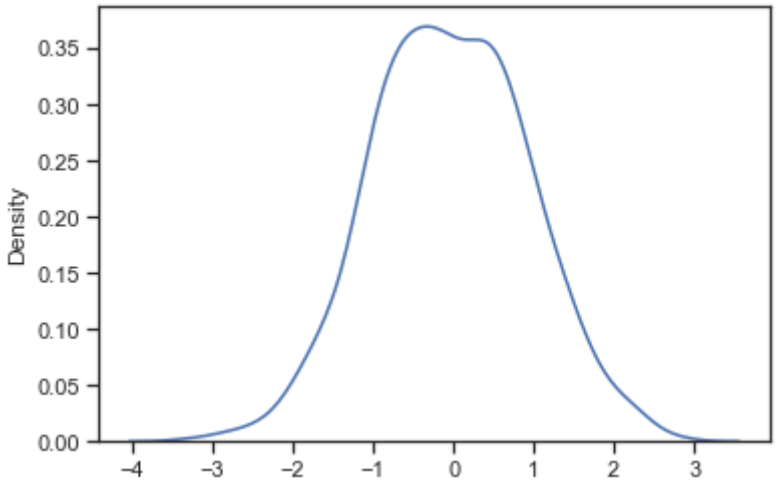
Out[50]: `<seaborn.axisgrid.FacetGrid at 0x272b6a69730>`



In [51]:
```
sns.distplot(rnd.normal(size=1000), hist=False)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl
ease adapt your code to use either `displot` (a figure-level function with similar f
lexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```

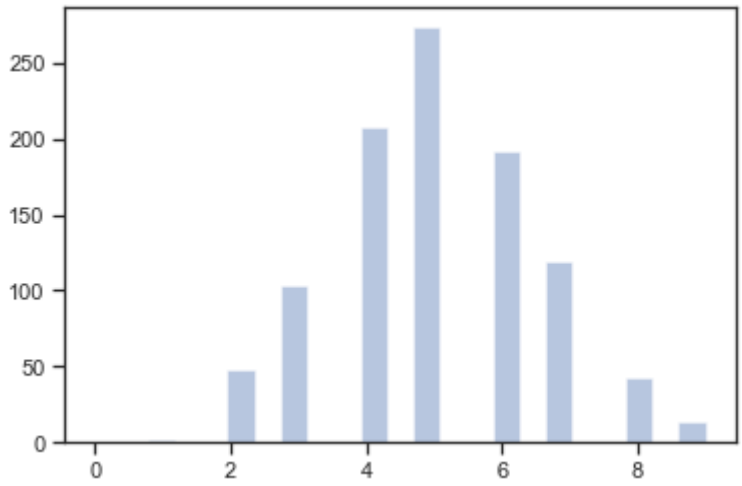
Out[51]: <AxesSubplot:ylabel='Density'>



In [52]:
```
sns.distplot(rnd.binomial(n=10, p=0.5, size=1000), hist=True, kde=False)
```

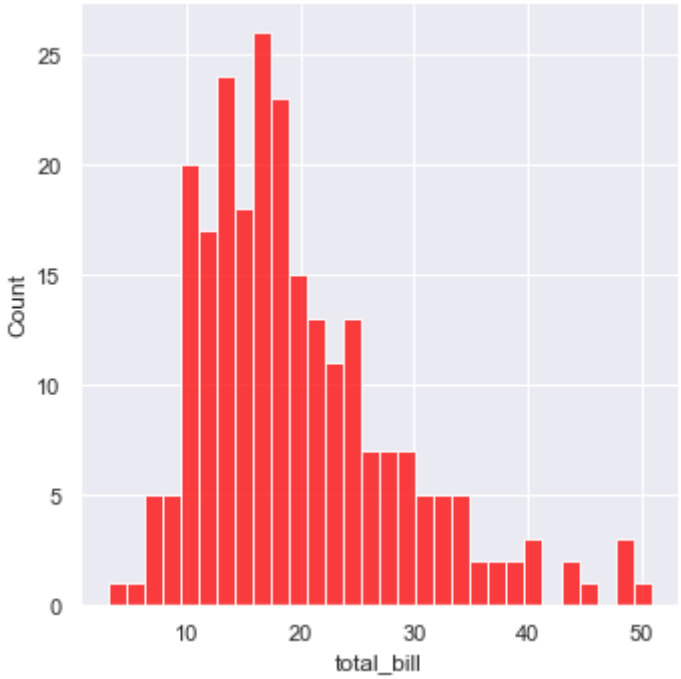
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl
ease adapt your code to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[52]: <AxesSubplot:>



In [53]:
```
sns.set_style('darkgrid')
sns.displot(df['total_bill'], kde = False, color ='red', bins = 30)
```

Out[53]: <seaborn.axisgrid.FacetGrid at 0x272b6d0a3d0>



- `KDE` stands for Kernel Density Estimation and that is another kind of the plot in seaborn.
- `bins` is used to set the number of bins/rectangular bars you want in your plot and it actually depends on your dataset.
- `color` is used to specify the color of the plot

Now looking at this we can say that most of the total bill given lies between 10 and 20.

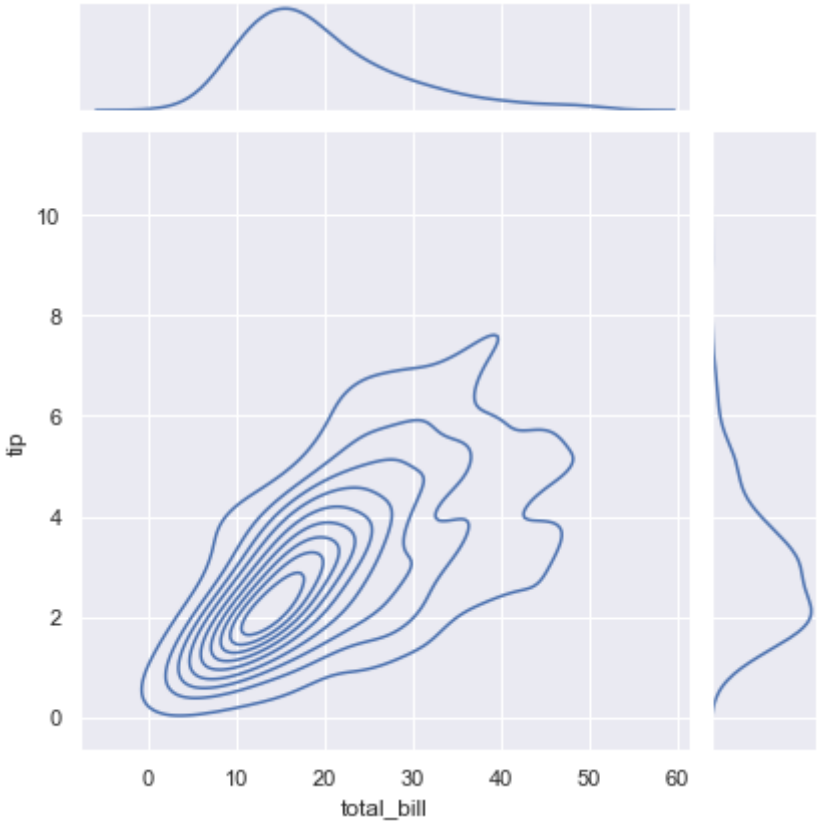
3.2 Joinplot

It is used to draw a plot of two variables with bivariate and univariate graphs. It basically combines two different plots. **KDE shows the density where the points match up the most**

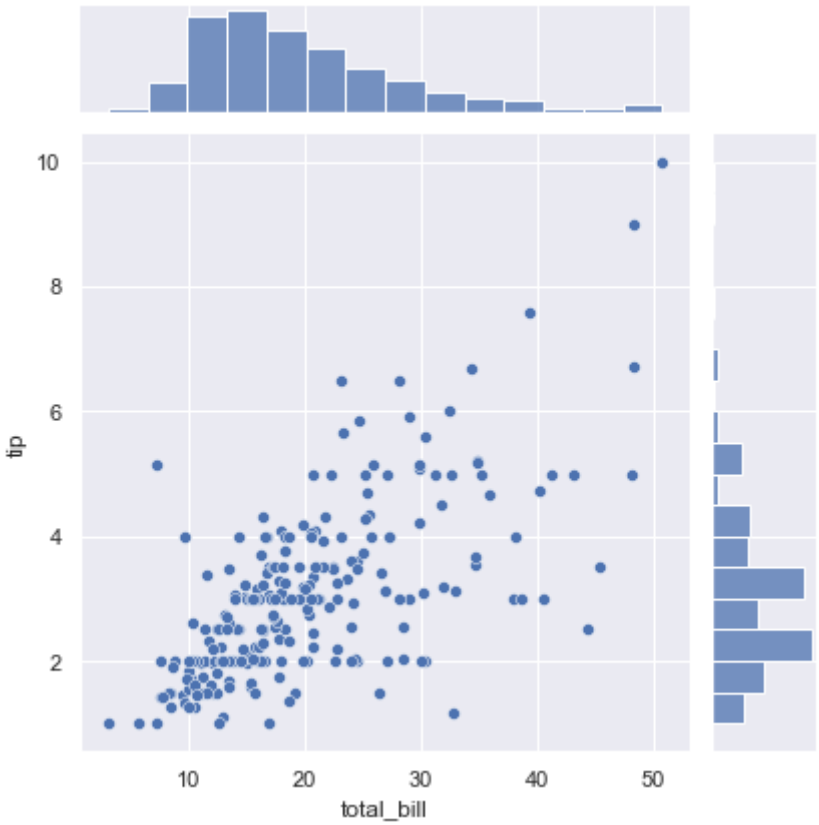
Syntax: `jointplot(x, y, data, kind, stat_func, ...])`

In [54]:
```
sns.set_style('darkgrid')
sns.jointplot(x ='total_bill', y ='tip', data = df, kind = 'kde')
```

Out[54]:
```
<seaborn.axisgrid.JointGrid at 0x272b6df74c0>
```

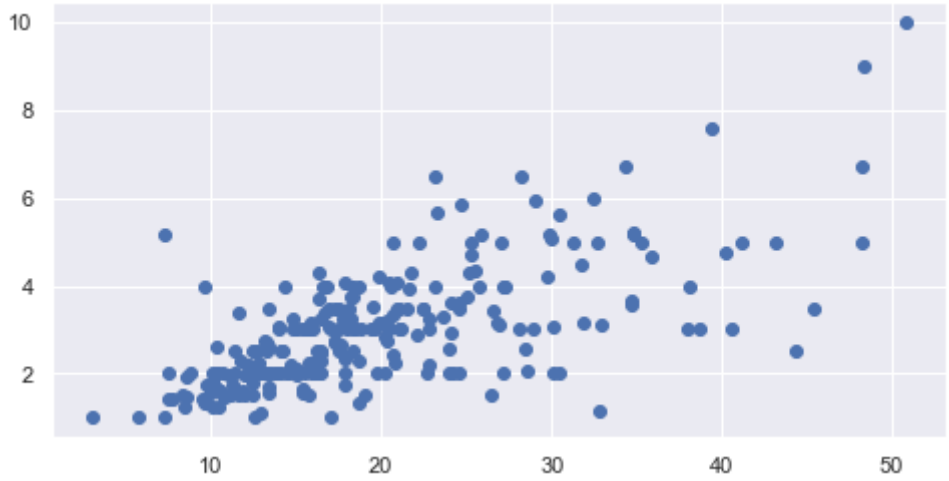


In [55]:
```
sns.jointplot(x ='total_bill', y ='tip', data = df)
plt.show()
```



In [56]:
```
# Scatter Plot of matplotlib can also be used but the points are overlapped over eac
plt.figure(figsize = (8,4))
plt.scatter(x = df.total_bill, y = df.tip)
```

Out[56]: <matplotlib.collections.PathCollection at 0x272b6e054f0>



- `kind` is a variable that helps us play around with the fact as to how do you want to visualise the data. It helps to see whats going inside the joinplot. The default is scatter and can be hex, reg(regression) or kde.
- `x` and `y` are two strings that are the column names and the data that column contains is used by specifying the `data` parameter.
- Here we can see tips on the Y axis and total bill on the X axis as well as a linear relationship between the two that suggests that the total bill increases with the tips.

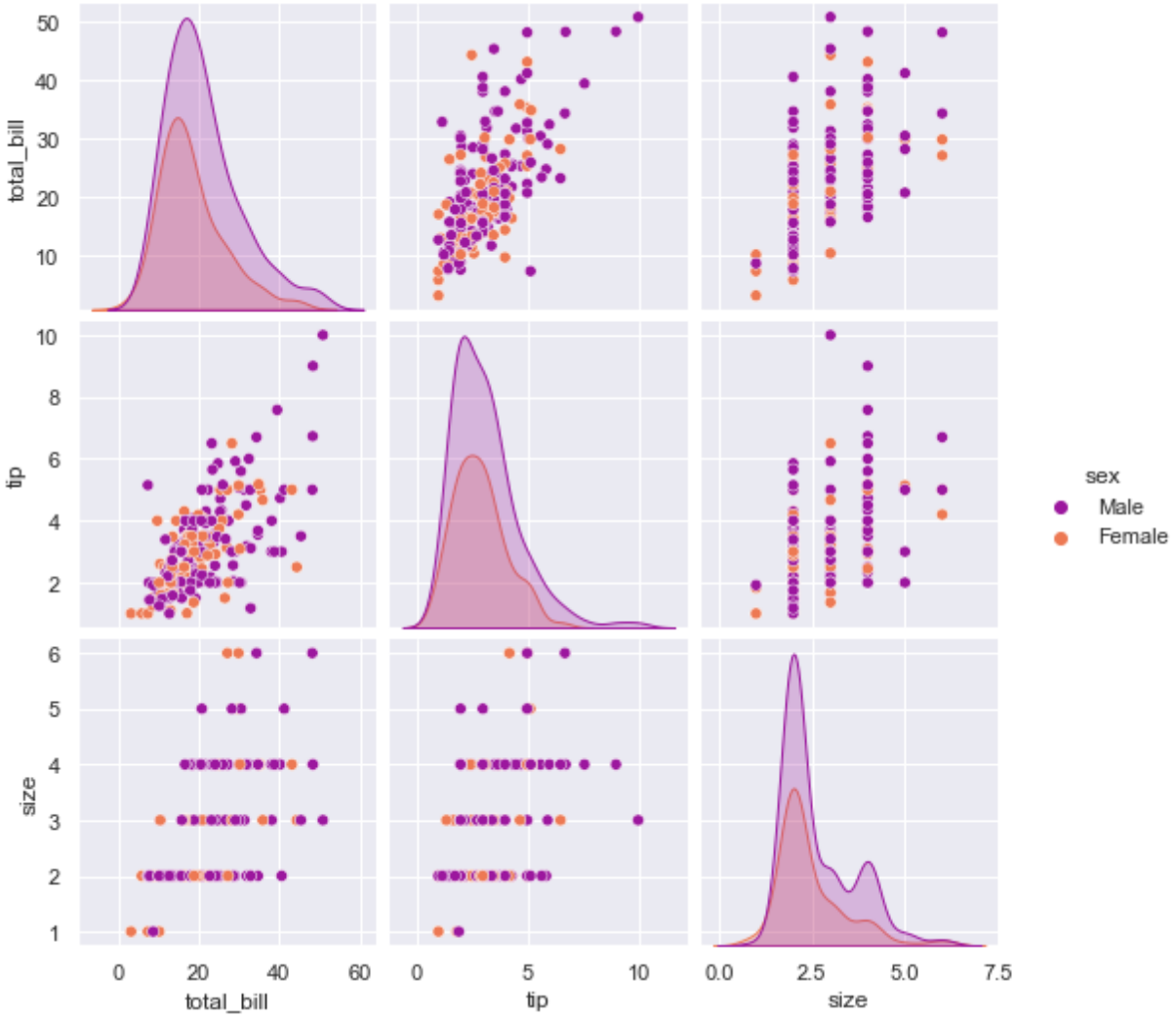
3.3 Pairplot

It represents pairwise relation across the entire dataframe and supports an additional argument called hue for categorical separation. What it does basically is create a jointplot between every possible numerical column and takes a while if the dataframe is really huge.

Syntax: `pairplot(data, hue, hue_order, palette, …])`

In [57]:
```
sns.pairplot(df, hue ="sex", palette ='plasma')
```

Out[57]:
```
<seaborn.axisgrid.PairGrid at 0x272b6de8370>
```



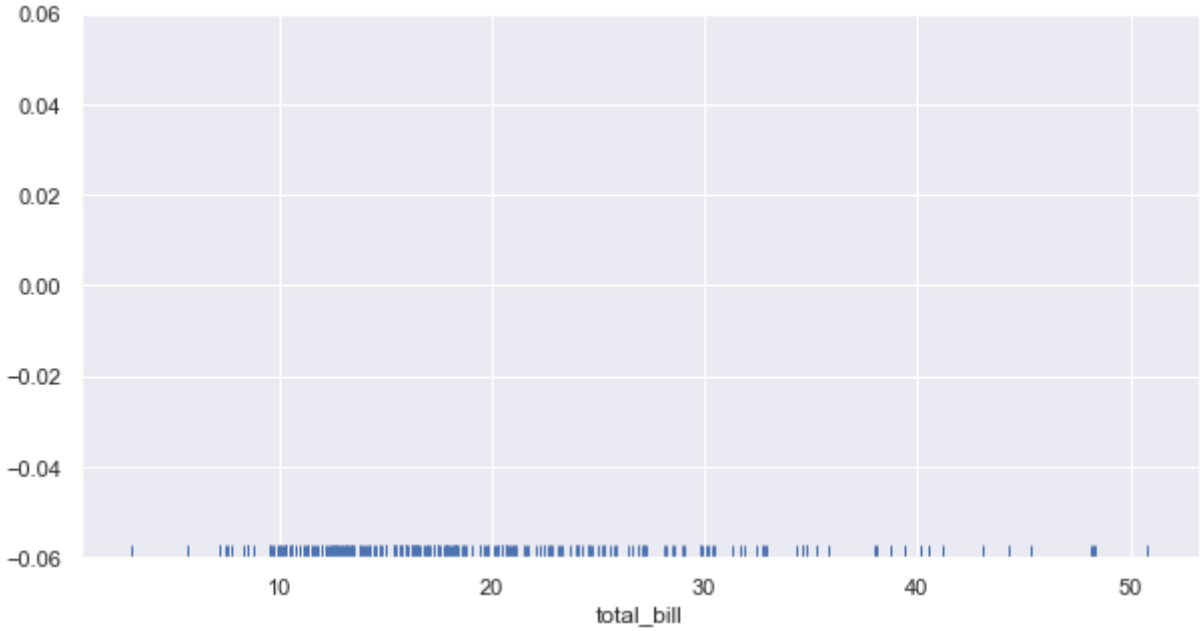
3.4 Rugplot

It plots datapoints in an array as sticks on an axis.Just like a distplot it takes a single column. Instead of drawing a histogram it creates dashes all across the plot. If you compare it with the joinplot you can see that what a jointplot does is that it counts the dashes and shows it as bins.

Syntax: `rugplot(a, height, axis, ax)`

In [58]:
```
df = sns.load_dataset('tips')
plt.figure(figsize = (10,5))
sns.rugplot(df['total_bill'])
```

Out[58]:
```
<AxesSubplot:xlabel='total_bill'>
```



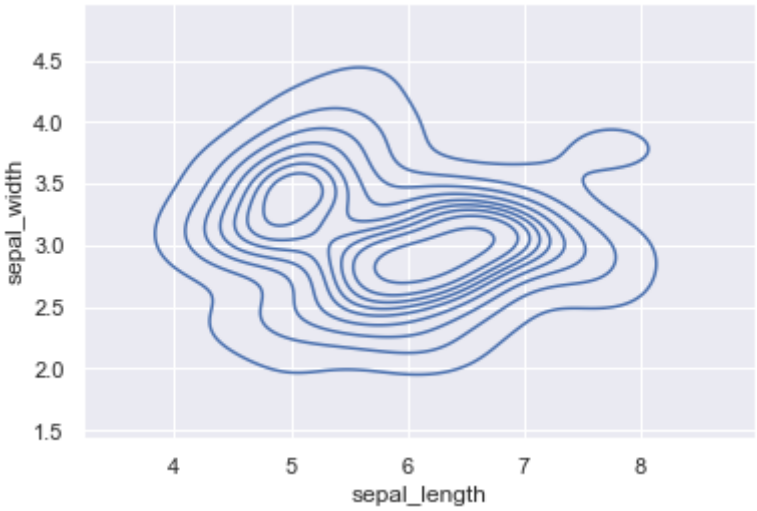
3.5 KDE Plot

KDE Plot described as Kernel Density Estimate is used for visualizing the Probability Density of a continuous variable. It depicts the probability density at different values in a continuous variable. We can also plot a single graph for multiple samples which helps in more efficient data visualization.

Syntax: `seaborn.kdeplot(x = None, *, y = None, vertical = False, palette = None, **kwargs)`

In [59]:
```
data = sns.load_dataset("iris")

sns.kdeplot(x='sepal_length', y='sepal_width', data=data)
```

Out[59]: `<AxesSubplot:xlabel='sepal_length', ylabel='sepal_width'>`



---

4. Regression Plot

The regression plots are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses. Regression plots as the name suggests creates a regression line between two parameters and helps to visualize their linear relationships.

There are two main functions that are used to draw linear regression models. These functions are `lmplot()` , and `regplot()` , are closely reled to each other. They even share their core functionality.

4.1 `lmplot( )`

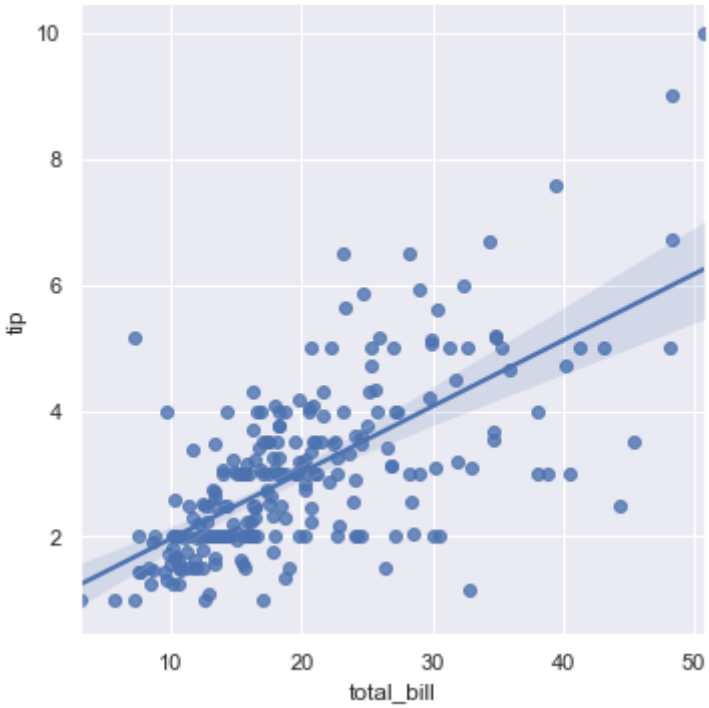
`seaborn.lmplot()` method is used to draw a scatter plot onto a FacetGrid. This method can be understood as a function that basically creates a linear model plot. It creates a scatter plot with a linear fit on top of it.

Syntax: `seaborn.lmplot(x, y, data, hue=None, col=None, row=None, **kwargs)`

Note that passing column names via dot operator as argument values of x and y doent works here

In [60]:
```
df = sns.load_dataset('tips')
# to remove regression line pass the value of fit_reg (optional) parameter as false
sns.lmplot(x = 'total_bill', y = 'tip', data = df)
```

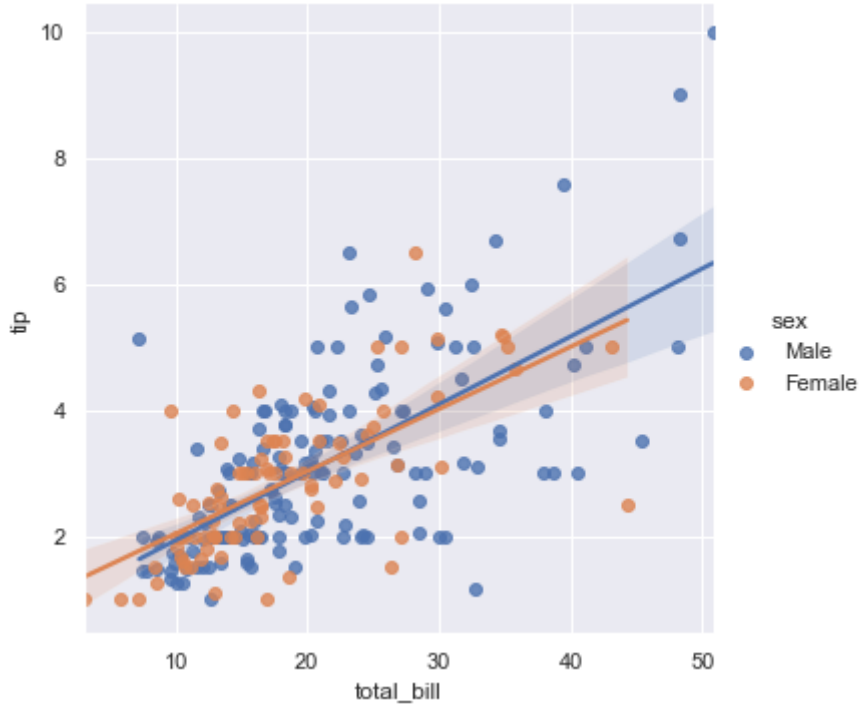
Out[60]: `<seaborn.axisgrid.FacetGrid at 0x272b6d65160>`



Seperating on basis of some category

In [61]:
```
sns.lmplot(x = 'total_bill', y = 'tip', hue = 'sex', data = df)
```

Out[61]: `<seaborn.axisgrid.FacetGrid at 0x272b8c9af40>`

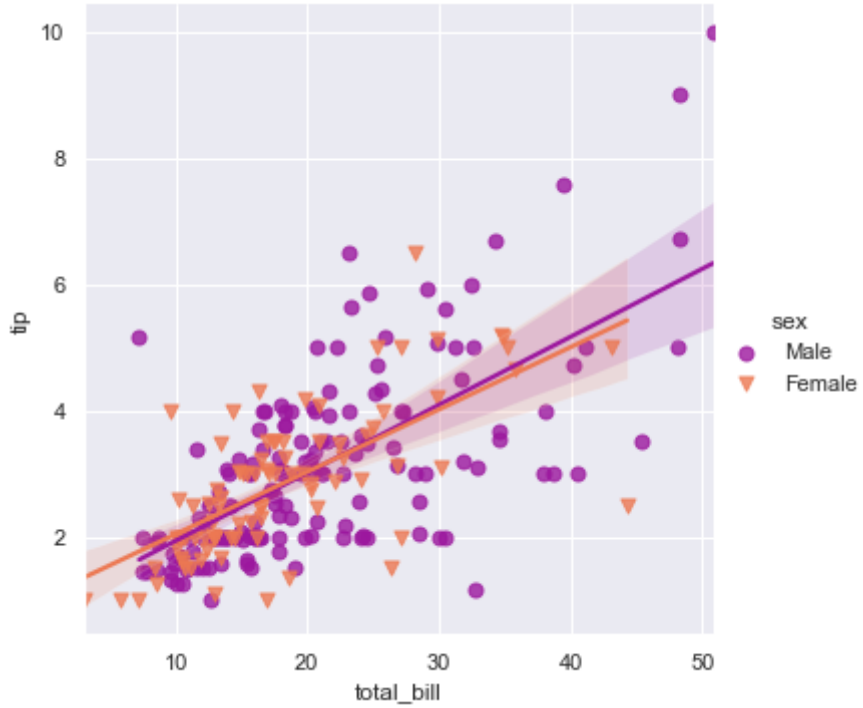


Changing marker size and colors

In order to have a better analysis capability using these plots, we can specify hue to have a categorical separation in our plot as well as use markers that come from the matplotlib marker symbols. Since we have two separate categories we need to pass in a list of symbols while specifying the marker. We specify a parameter called `scatter_kws`. We must note that the `scatter_kws` parameter changes the size of only the scatter plots and not the regression lines. *The regression lines remain untouched*. We also use the palette parameter to change the color of the plot.

In [62]:
```
sns.lmplot(x ='total_bill', y ='tip', data = df, hue ='sex', markers =['o','v'], sca
```

Out[62]:
```
<seaborn.axisgrid.FacetGrid at 0x272b8bbdd00>
```



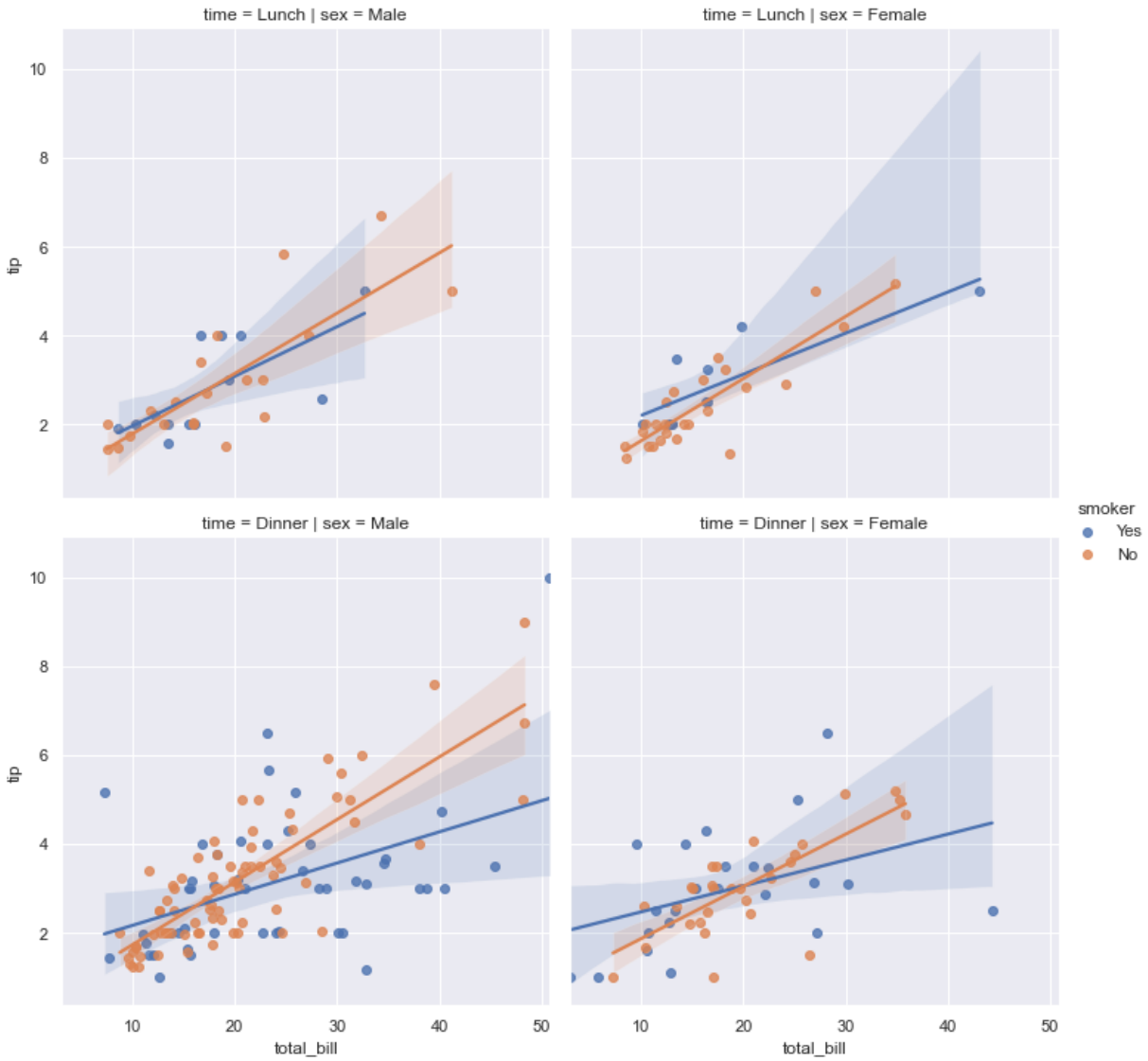
Displaying Multiple Plots

We can draw multiple plots by specifying a separation with the help of the `rows` and `columns`. Each row contains the plots of tips vs the total bill for the different times specified in the dataset. Each column contains the plots of tips vs the total bill for the different genders. A further separation is done by specifying the `hue` parameter on the basis of whether the person smokes.

For instance the 1st plot shows the tip vs total bill relationship for lunchtime of males with blue dots marking male smokers and yellow dots marking non-smokers.

In [63]:
```
sns.lmplot(x ='total_bill', y ='tip', data = df, col ='sex', row ='time', hue ='smok
```

Out[63]:
```
<seaborn.axisgrid.FacetGrid at 0x272b8eb48b0>
```



During dinner time there are less smokers among males (3rd plot: less blue dots) while during lunchtime there are less smokers among females (2nd plot: less blue dots).

During Lunch time, there are less smokers among males (1st plot: less blue dots) while during dinnertime the #smokers and #non-smokers are almost same among females. (4th plot: almost same #dots from each category)

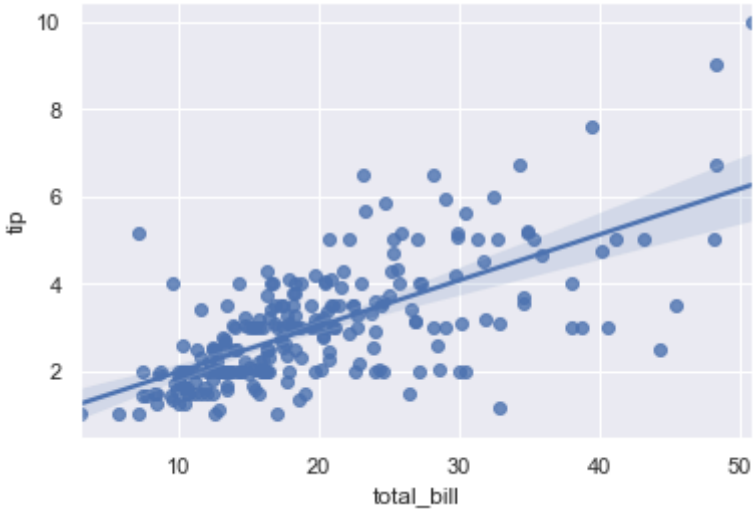
4.2 `regplot( )`

This method is used to plot data and a linear regression model fit. There are a number of mutually exclusive options for estimating the regression model. This method is also similar to `lmplot()` which creates linear regression model.

Syntax: `seaborn.regplot( x,  y,  data=None, x_estimator=None, **kwargs)`

In [64]:
```
sns.regplot(x = 'total_bill', y = 'tip', data = df)
```

Out[64]:
```
<AxesSubplot:xlabel='total_bill', ylabel='tip'>
```



Note: The difference between both the function is that `regplot()` accepts the x, y variables in different format inlcuding NumPy arrays, Pandas objects, whereas, the `lmplot()` **only accepts the value as strings.**

4.3 `residplot( )`

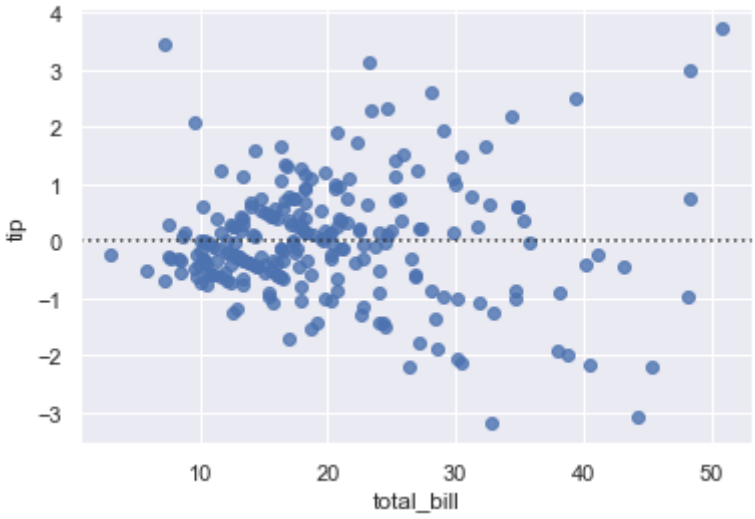
This method is used to plot the residuals of linear regression. This method will regress y on x and then draw a scatter plot of the residuals. You can optionally fit a lowess smoother to the residual plot, which can help in determining if there is a structure to the residuals.

Syntax: `seaborn.residplot(x, y, data, lowess = T/F, dropna = T/F, scatter_kws, line_kws)`

- `lowess` : (optional) Fit a lowess smoother to the residual scatterplot.
- `dropna` : (optional) This parameter takes boolean value. If True, ignore observations with missing data when fitting and plotting.

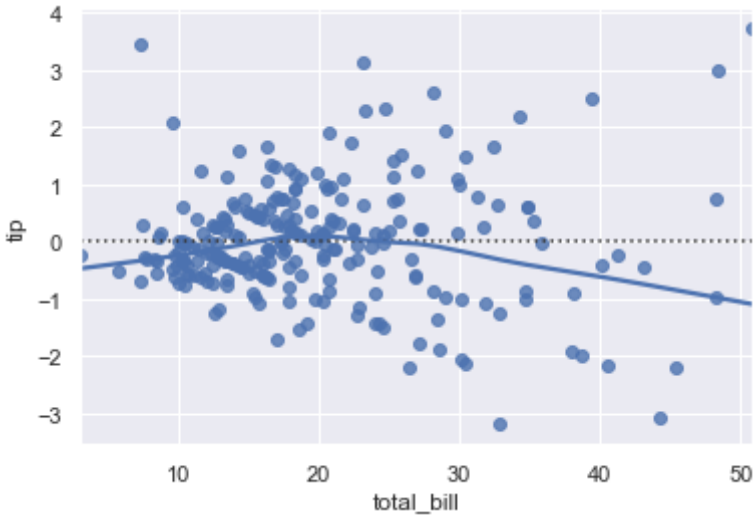
In [65]:
```
data = sns.load_dataset("tips")

# draw residplot
sns.residplot(x = "total_bill", y = "tip", data = data)
```

Out[65]:
```
<AxesSubplot:xlabel='total_bill', ylabel='tip'>
```



In [66]:
```
sns.residplot(x = "total_bill", y = "tip", data = data, lowess = True)
```

Out[66]: <AxesSubplot:xlabel='total_bill', ylabel='tip'>



4.4 Matrix Plot

A matrix plot means plotting matrix data where color coded diagrams shows rows data, column data and values. It can shown using the heatmap and clustermap.

4.4.1 Heatmap

Heatmap is a way to show some sort of matrix plot. To use a heatmap the data should be in a matrix form. By matrix we mean that the index name and the column name must match in some way so that the data that we fill inside the cells are relevant. A correlation heatmap is a rectangular representation of data and it repeats the same data description twice because the categories are repeated on both axis for computing analysis. The values of the first dimension appear as the rows of the table while of the second dimension as a column. The *color of the cell is proportional to the number of measurements that match the dimensional value.* This makes correlation heatmaps ideal for data analysis since it makes patterns easily readable and highlights the differences and variation in the same data. A correlation heatmap, like a regular heatmap is assisted by a colorbar making data easily readable and comprehensible. Hence, the same result is obtained twice. So we can say that all a heatmap does is color the cells based on the gradient and uses some parameters to increase the data visualization.

`corr()` itself eliminates columns which will be of no use while generating a correlation heatmap and selects those which can be used.

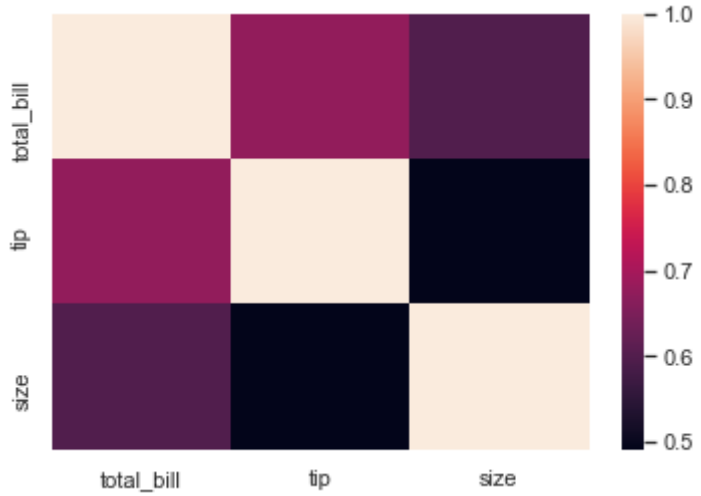
In [67]:
```
# correlation between the different parameters
tc = df.corr()
tc
```

Out[67]:

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

In [68]:
```
# plot a heatmap of the correlated data
sns.heatmap(tc)
```

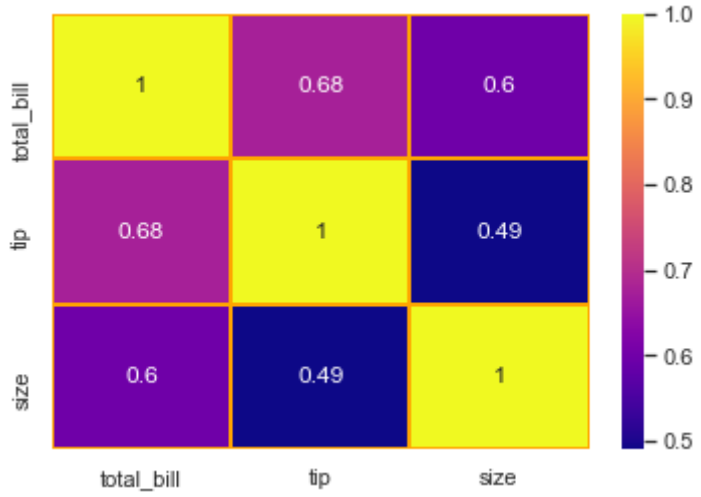
Out[68]: <AxesSubplot:>



- `annot` is used to annotate the actual value that belongs to these cells
- `cmap` is used for the colour mapping you want like coolwarm, plasma, magma etc.
- `linewidth` is used to set the width of the lines separating the cells.
- `linecolor` is used to set the colour of the lines separating the cells.
- `cbar_kws = {'shrink': size}` is used to set the size of colorbar. To make the colorbar small, shrink should be given a value smaller than 1 and to increase its size it should be given a value greater than 1. Default value is 1.
- `annot_kws = {'size': 15}` is used to set size of annotation of each color.

In [69]:
```
sns.heatmap(tc, annot = True, cmap ='plasma', linecolor ='orange', linewidths = 1)
```

Out[69]: <AxesSubplot:>



Triangular Correlation Heatmap

A correlation heatmap that presents data only once without repetition that is categories are correlated only once is known as a triangle correlation heatmap. Since data is symmetric across the diagonal from left-top to right bottom the idea of obtaining a triangle correlation heatmap is to remove data above it so that it is depicted only once. The elements on the diagonal are the parts where categories of the same type correlate.

Mask is a heatmap attribute that takes a dataframe or a boolean array as an argument and displays only those positions which are marked as False or where masking is provided to be False.

For masking, here an array using NumPy is being generated as shown below:

`np.triu(np.ones_like())`

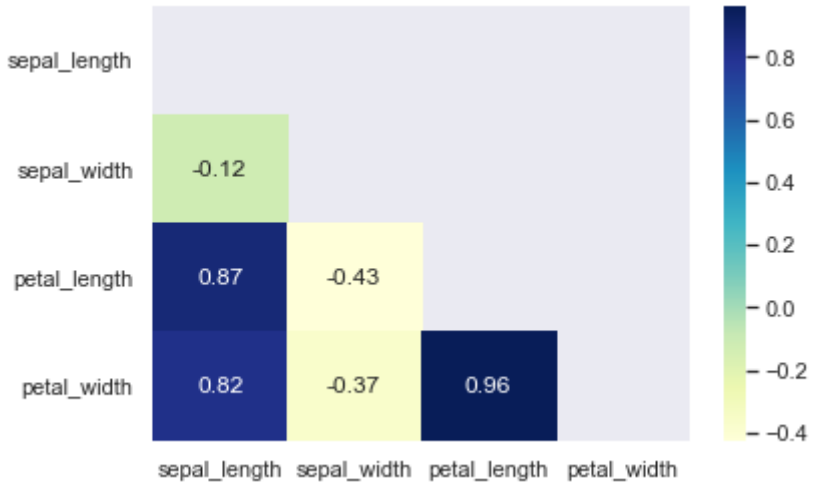
First, the `ones_like()` method of NumPy module will generate an array of size same as that of our data to be plotted containing only number one. Then, `triu()` method of the NumPy module will turn the matrix so formed into an upper triangular matrix, i.e. elements above the diagonal will be 1 and below, and on it will be 0. **Masking will be applied to places where `1(True)` is set.**

The following steps show how a triangle correlation heatmap can be produced:

- Import all required modules first
- Import the file where your data is stored
- Plot a heatmap
- Mask the part of the heatmap that shouldn't be displayed
- Display it using matplotlib

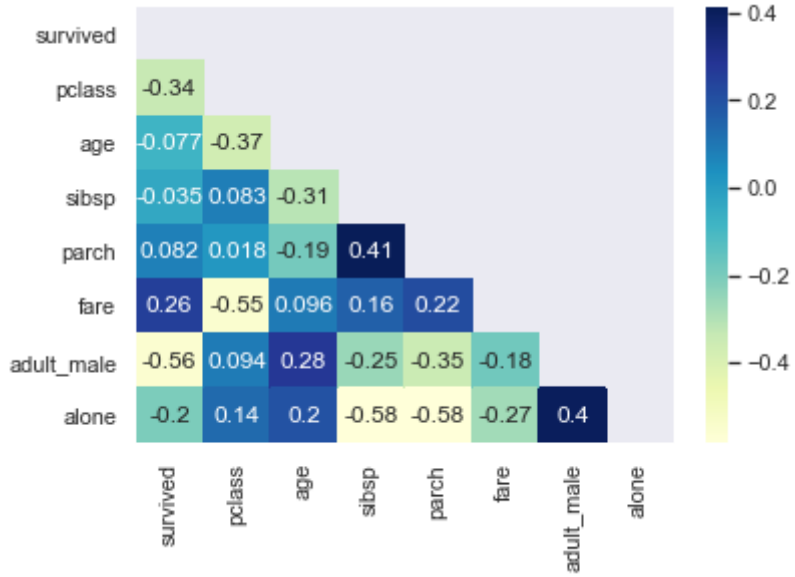
In [70]:
```
data = sns.load_dataset('iris')
sns.heatmap(data.corr(), cmap="YlGnBu", annot = True, mask = np.triu(np.ones_like(da
```

Out[70]:
```
<AxesSubplot:>
```



In [71]:
```
data = sns.load_dataset('titanic')
sns.heatmap(data.corr(), cmap="YlGnBu", annot = True, mask = np.triu(np.ones_like(da
```

Out[71]: <AxesSubplot:>



4.4.2 Clustermap

Cluster maps use hierarchical clustering. It performs the clustering based on the similarity of the rows and columns. Clustering simply means grouping data based on relationship among the variables in the data. To create clustermp we need a pivot table. A pivot table is a table of grouped values that aggregates the individual items of a more extensive table within one or more discrete categories.

Syntax: `clustermap(data, *, pivot_kws=None,  **kwargs)`

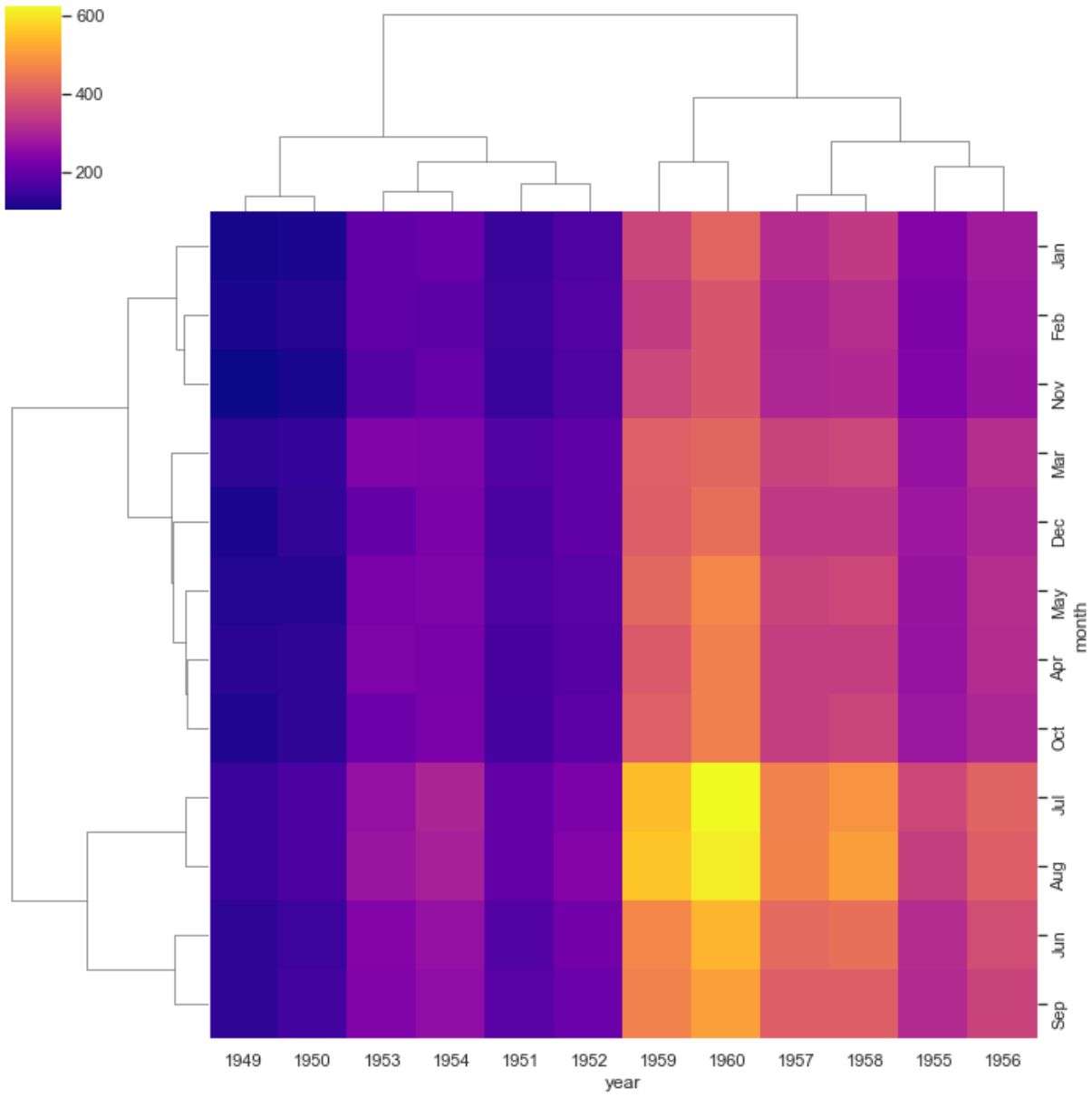
In [72]:
```
import pandas as pd
# load the flights dataset
fd = sns.load_dataset('flights')

# make a dataframe of the data
df = pd.pivot_table(values ='passengers', index ='month', columns ='year', data = fd

# first five entries of the dataset
df.head()
```

Out[72]:

year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
month												
Jan	112	115	145	171	196	204	242	284	315	340	360	417
Feb	118	126	150	180	196	188	233	277	301	318	342	391
Mar	132	141	178	193	236	235	267	317	356	362	406	419
Apr	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472

In [73]:
```
sns.clustermap(df, cmap ='plasma')
```

Out[73]: <seaborn.matrix.ClusterGrid at 0x272b86c1b20>

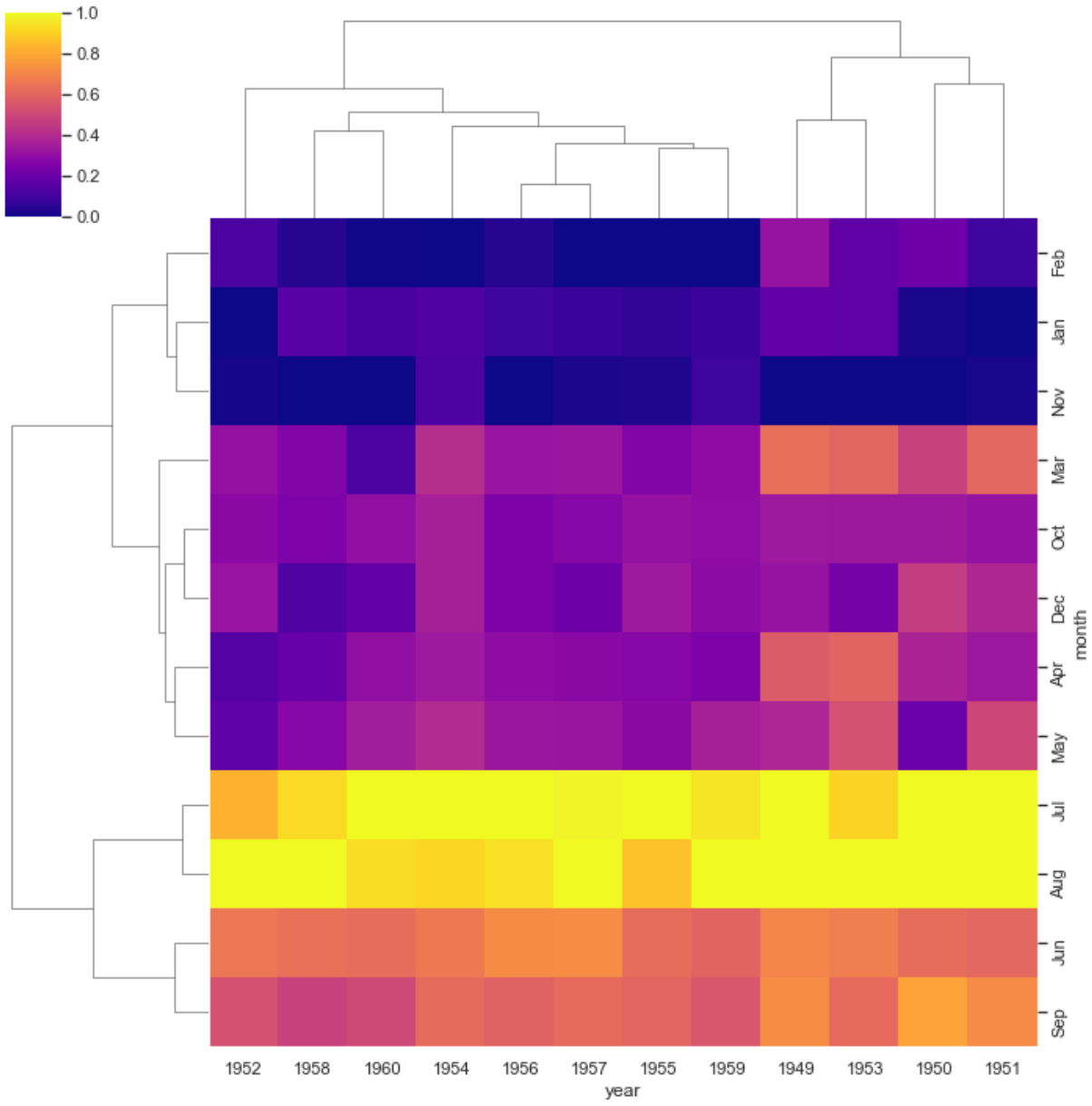


We can also change the scale of the color bar by using the `standard_scale` parameter. `standard_scale = 1` normalizes the data from 0 to 1 range. We can see that the months as well as years are no longer in order as they are clustered according to the similarity in case of clustermaps.

So we can conclude that a **heatmap will display things in the order we give** whereas the **cluster map clusters the data based on similarity.**

In [74]:
```
sns.clustermap(df, cmap ='plasma', standard_scale = 1)
```

Out[74]:
```
<seaborn.matrix.ClusterGrid at 0x272ba8526a0>
```



4.5 Point Plot

- This method is used to show point estimates and confidence intervals using scatter plot glyphs. A point plot represents an estimate of central tendency for a numeric variable by the position of scatter plot points and provides some indication of the uncertainty around that estimate using error bars.
- This function always treats one of the variables as categorical and draws data at ordinal positions (0, 1, … n) on the relevant axis, even when the data has a numeric or date type.

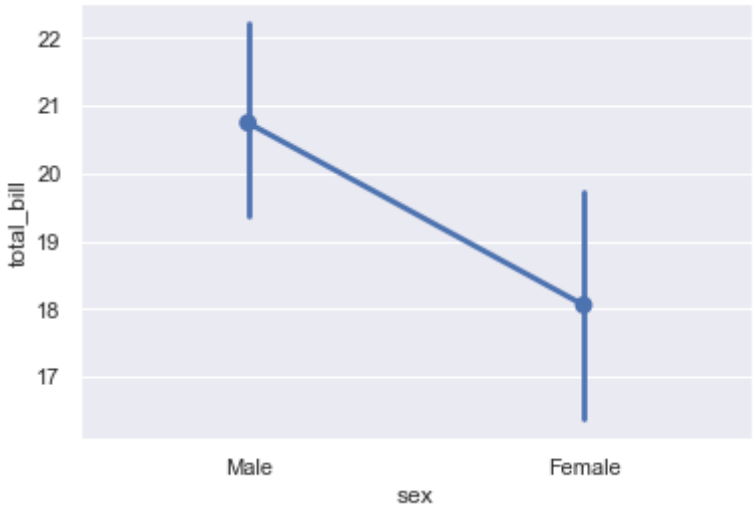
Syntax: `seaborn.pointplot(x, y, hue, data, markers, linestyles, dodge = T/F, color, palette=None, capsize=None, **kwargs)`

Parameters: The description of some main parameters are given below:

1. dodge: (optional) Amount to separate the points for each level of the 'hue' variable along the categorical axis.
2. color: (optional) Color for all the elements, or seed for a gradient palette.
3. capsize: (optional) Width of the 'caps' on error bars.

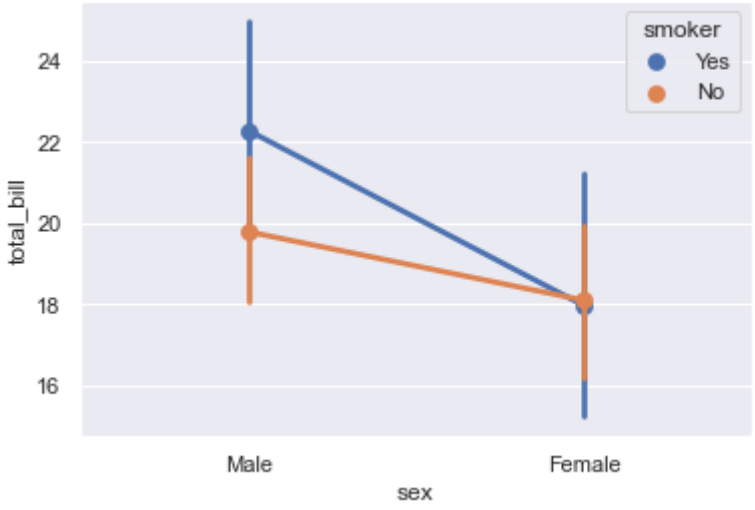
In [75]:
```
sns.pointplot(x = "sex", y = "total_bill", data = sns.load_dataset("tips"))
```

Out[75]:
```
<AxesSubplot:xlabel='sex', ylabel='total_bill'>
```



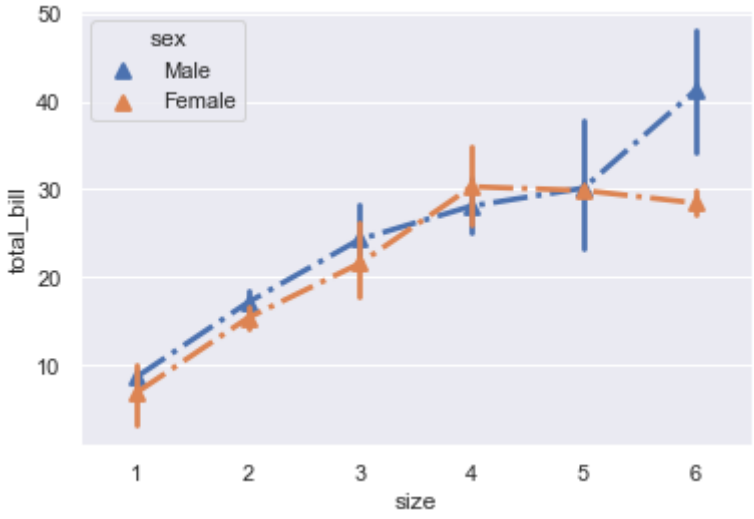
In [76]:
```
sns.pointplot(x = "sex", y = "total_bill", data = sns.load_dataset("tips"), hue = "s
```

Out[76]:
```
<AxesSubplot:xlabel='sex', ylabel='total_bill'>
```



In [77]:
```
sns.pointplot(x = "size", y = "total_bill", linestyles = '-.', markers = '^', hue =
```

Out[77]:
```
<AxesSubplot:xlabel='size', ylabel='total_bill'>
```



END
