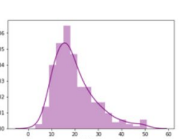
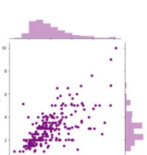
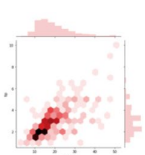
Seaborn

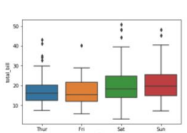
Seaborn

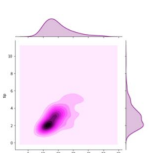
- A library for making statistical graphics in Python
- builds on top of matplotlib and integrates closely with pandas data structures **matplotlib 기반으로 작동**
  - operate on dataframes and arrays containing whole datasets
  - internally perform the necessary semantic mapping and statistical aggregation to produce informative plots
- uses fewer syntax and provides a variety of visual patterns

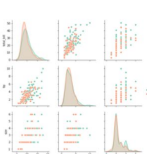

distplot

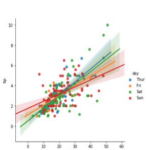

Jointplot

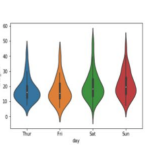

Hexplots


Boxplots


KDE Plot


Pair Plots


LM Plots


Violin Plots

Overview

- Seaborn can do many types of visualizations including:
  - Numerical Data Plotting
    - relplot(), scatterplot(), lineplot()
  - Categorical Data Plotting 순서가 있는 경우
    - catplot(), boxplot(), stripplot(), swarmplot()
  - Visualizing Distribution of the Data
    - distplot(), kdeplot(), jointplot(), rugplot()
  - Linear Regression and Relationship
    - regplot(), lmplot()
  - Controlling Plotted Figure Aesthetics
    - figure styling, axes styling, color palettes

Dataset

- Load a built-in dataset, “tips”

```
[3]  tips = sns.load_dataset('tips')
     tips.tail()
```

	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

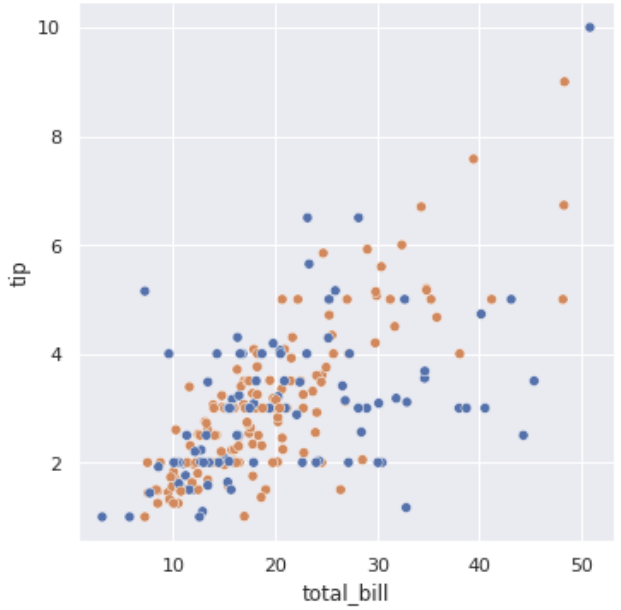
Numerical Data Plotting

Relational Plot

- Visualize a relationship between two variables

```
[5]  sns.relplot (x = 'total_bill', y = 'tip', data = tips, hue = 'smoker')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f0650ffb890>
```



hue를 줌으로써 3차원 데이터로 활용 가능

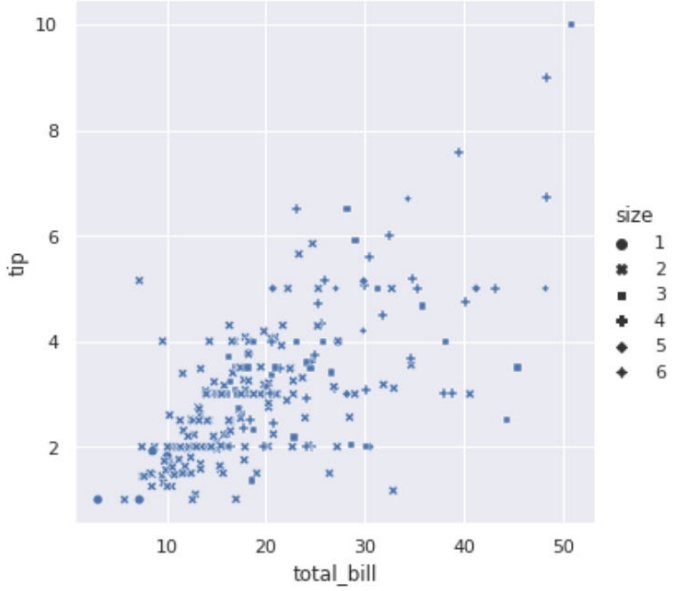
	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

Size as a style

- Use **style** for the ‘size’ variable

```
[5]  sns.relplot(x = 'total_bill', y = 'tip', style = 'size', data = tips)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f3e15c4e850>
```



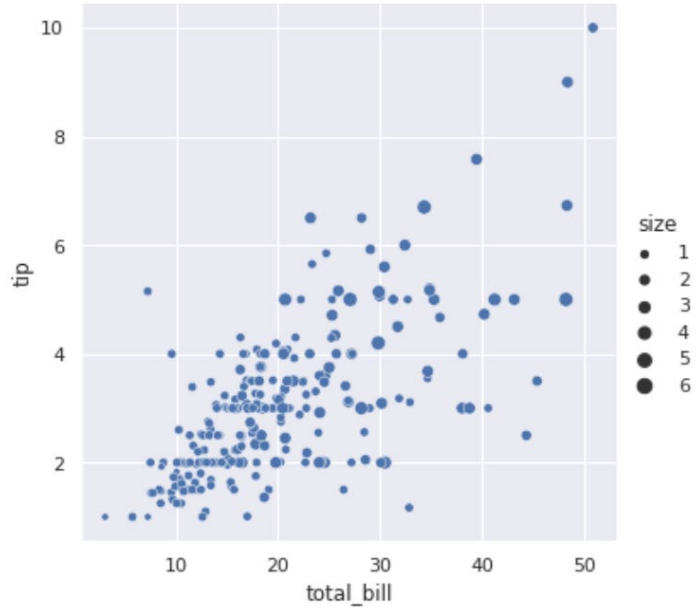
	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

Size in terms of circle radius

- **A larger size has a larger circle**

```
[6] sns.relplot(x = 'total_bill', y = 'tip', data = tips, size = 'size')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f3e1612b5d0>
```



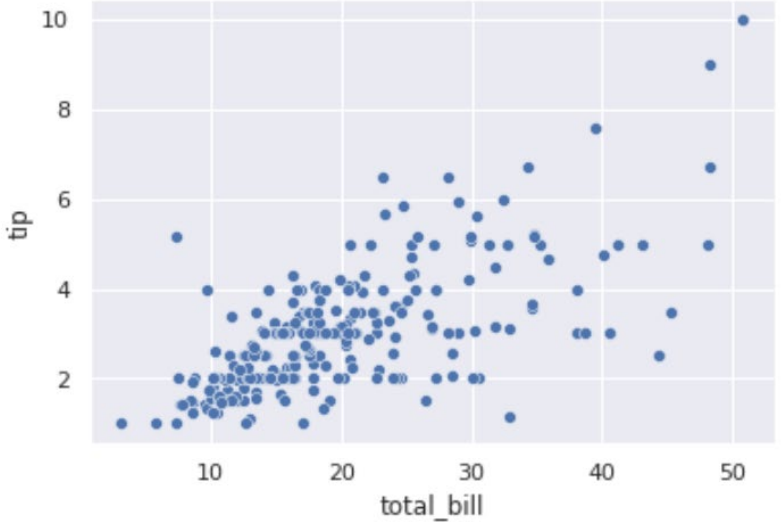
	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

Scatter Plot

- Use dots to represent values for numeric variables

```
[7] sns.scatterplot(x = 'total_bill', y = 'tip', data = tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e15b5a850>
```

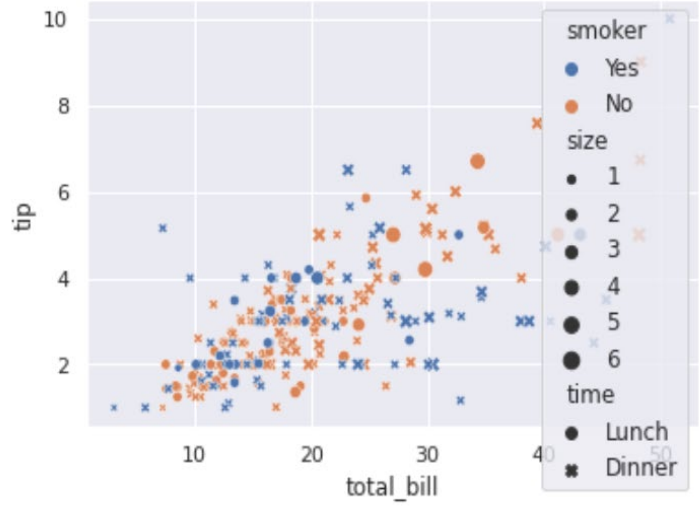


	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

Detailed grouping based on hue, size, style together

```
[8] sns.scatterplot(x = 'total_bill', y = 'tip', data = tips,
                    hue = 'smoker', size = 'size', style = 'time')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0cd5e20d90>
```



- **Style** for the 'time' variable
- **Size** for the 'size' variable
- **Hue**(color) for the 'smoker' variable

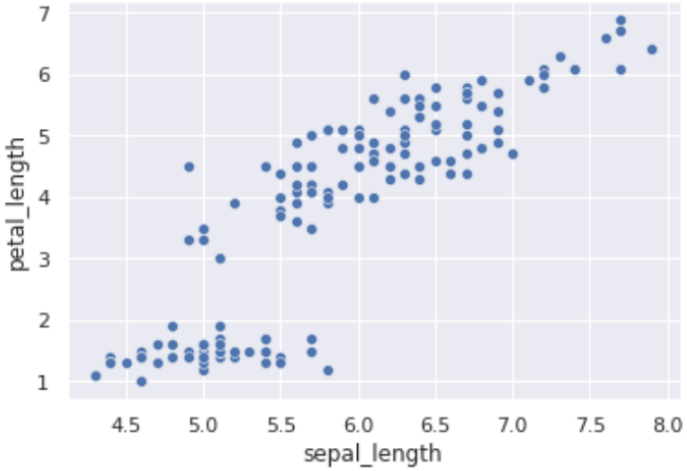
All points can be expressed in 2*6*2 different ways using Style, Size, and Hue!

	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

Scatter Plot (cont.)

```
[9]  iris = sns.load_dataset('iris')
     sns.scatterplot(x = 'sepal_length', y = 'petal_length', data = iris)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0cd4ce4550>
```

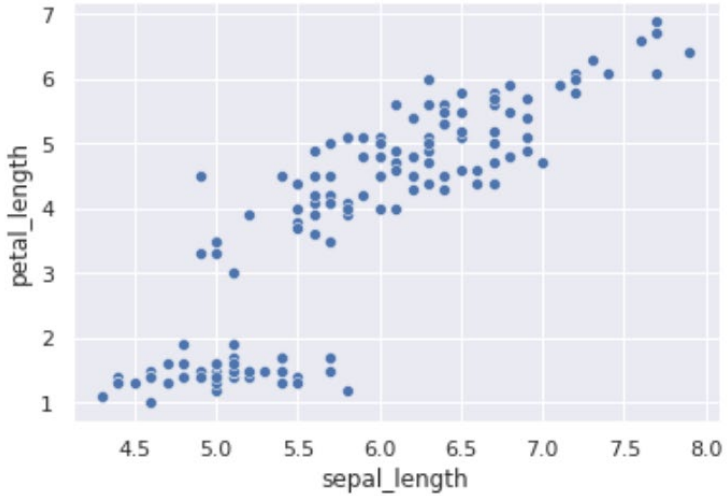


- Load another inbuilt dataset ‘iris’

Scatter Plot (cont.)

```
[17] sns.scatterplot(x = iris['sepal_length'], y = iris['petal_length'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5a5ba44910>
```



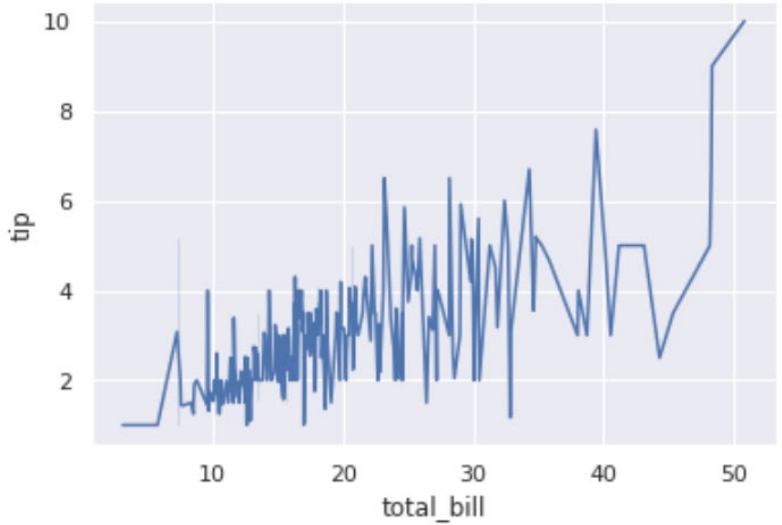
- Instead of passing the data = iris we can even set x and y in the way shown in the figure

Line Plot

- Display data along a number line

```
[8]  sns.lineplot(x = 'total_bill', y = 'tip', data = tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e15b11910>
```

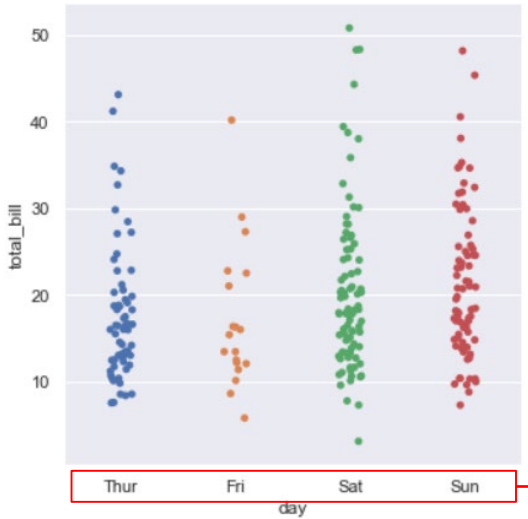


Categorical Data Plotting

Categorical Plot

- Visualize the relationship between a numerical and one or more categorical variables

```
sns.catplot(x = 'day', y = 'total_bill', data = tips)
```

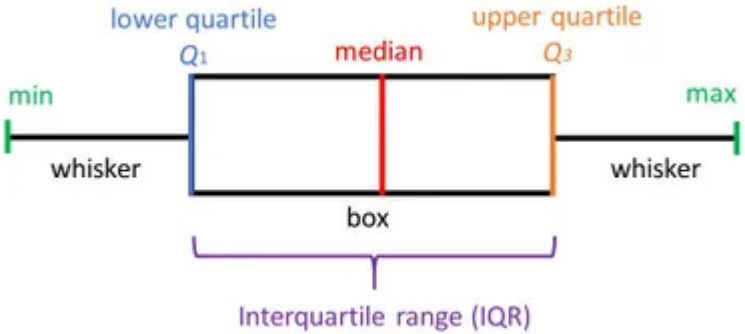


- We can even interchange the variables on `x` and `y` axis to get a horizontal catplot plot.

no ordering

Box Plot

- Box plots show the five-number summary of a set of data
  - the minimum, first (lower) quartile, median, third (upper) quartile, and maximum
- We can draw a box plot in catplot()



IQR: Q3-Q1

Max: Q3+1.5IQR보다 작은 값 중에서 가장 큰 값
MIN: Q3-1.5IQR보다 큰 값 중에서 가장 작은 값

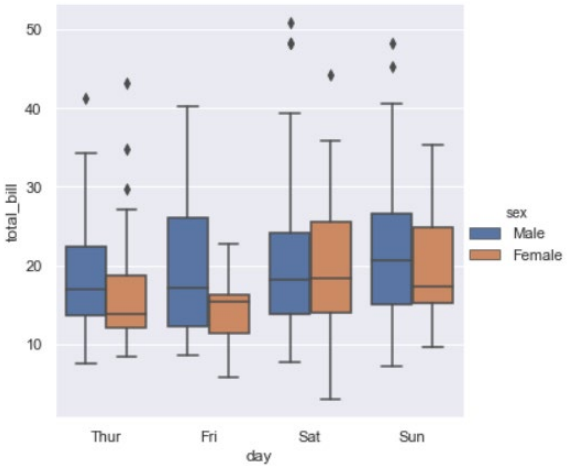
Example

Q2: 68
Q1: 52
Q3: 87
IQR: Q3-Q1=35

Box Plot Example

- If we want detailed characteristics of data we can use box plot by setting *kind = 'box'*

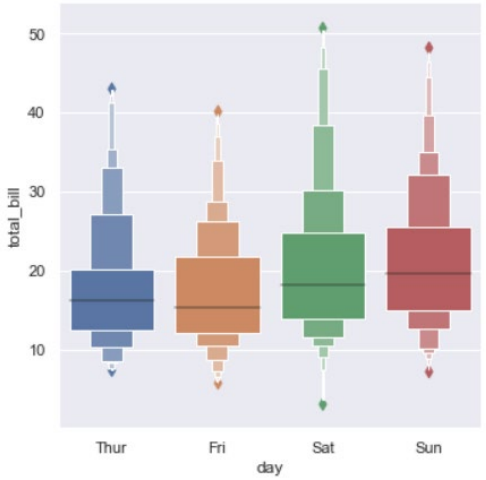
```
sns.catplot(x = 'day', y = 'total_bill', kind = 'box', data = tips, hue = 'sex')
```



Boxen Plot

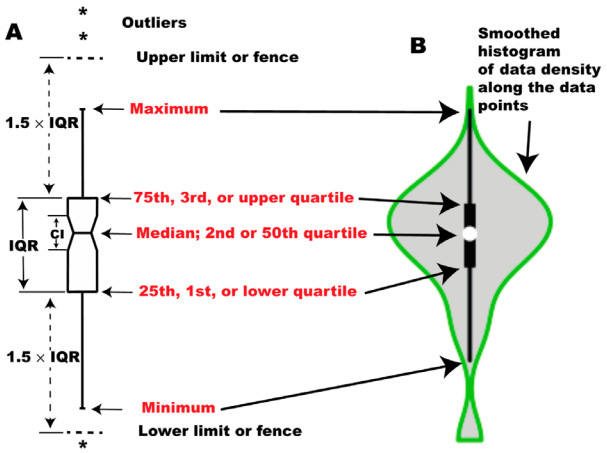
- If you want more visualize detailed information you can use boxen plot
- It is similar to a box plot in plotting a nonparametric representation of a distribution in which all features correspond to actual observations

```
sns.catplot(x = 'day', y = 'total_bill', kind = 'boxen', data =
tips, dodge = False)
```



Violin Plot

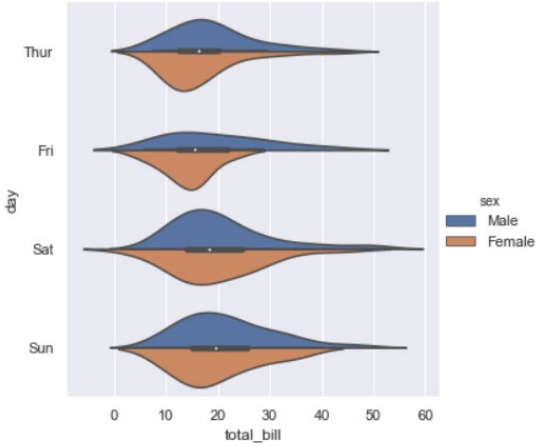
- Violin plot shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared
- Unlike a box plot, in which all of the plot components correspond to actual data points, the violin plot features a kernel density estimation of the underlying distribution.



Violin Plot Example

- We can draw a violin plot by setting *kind = 'violin'*
- When using hue nesting with a variable that takes two levels, setting split to True will draw half of a violin for each level

```
sns.catplot(x = 'total_bill', y = 'day', hue = 'sex', kind =
'violin', data = tips, split = True,)
```



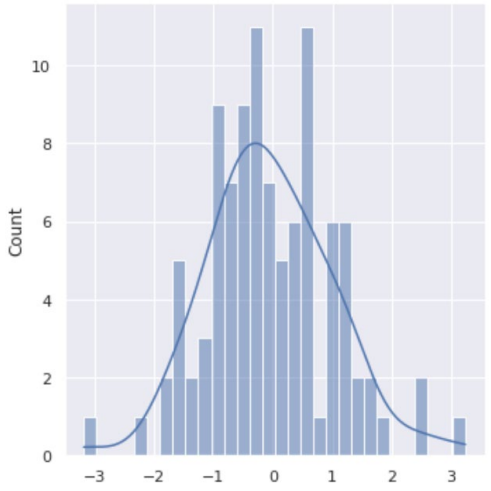
Visualizing Distribution of the Data

Displot() for Univariate Distribution

- Draw a histogram for a specified attribute
  - # of bins can be controlled by 'bins'
- Additionally draw a kernel density estimate (KDE) by setting kde to True

```
[38] x = randn(100)
     sns.displot(x, kde = True, rug= False, bins= 30)
```

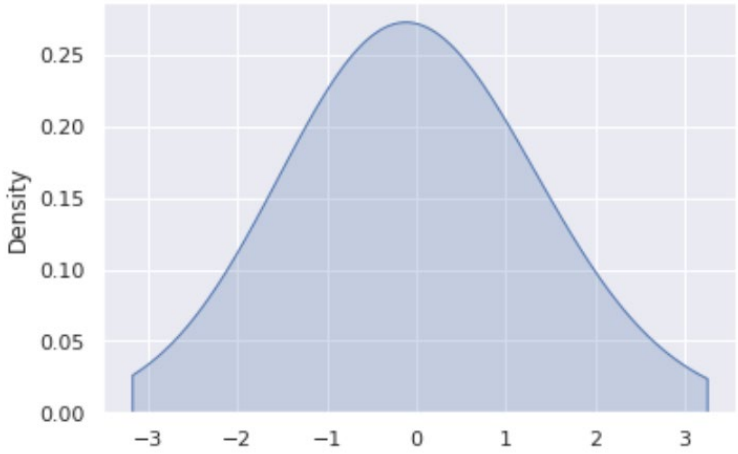
```
<seaborn.axisgrid.FacetGrid at 0x7f5a59127810>
```



Kernel Density Estimate (KDE) Plot

- A method for visualizing the distribution of observations in a dataset, analogous to a histogram

```
[41] sns.kdeplot(x, shade=True, cbar = True, bw = 1, cut = 0)
```



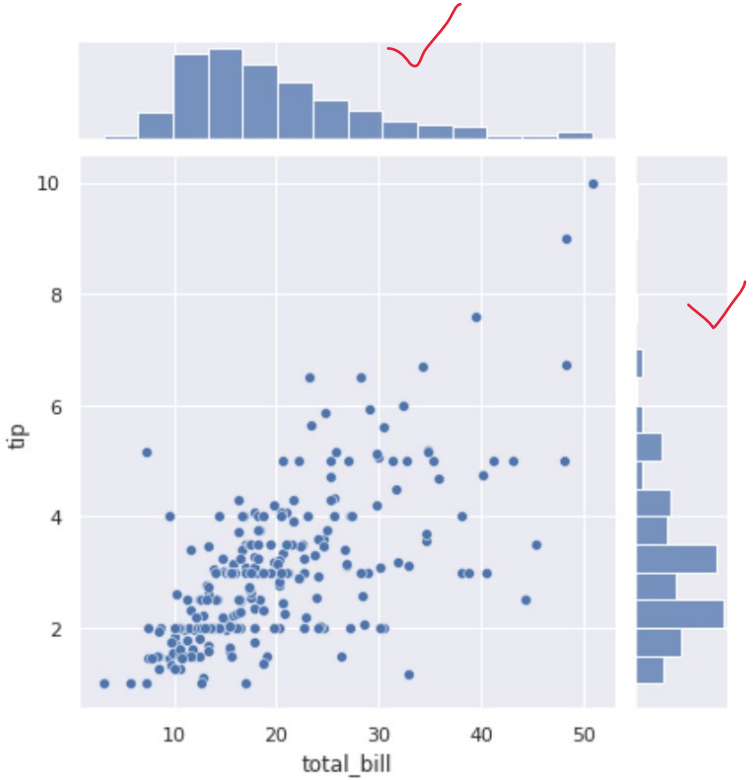
- **Shade = True** shades in the area under the KDE curve
- **Bw** is smoothing parameter
- When set **cut** to 0, truncate the curve at the data limits. it cuts the plot and zooms it.

Jointplot() for Bivariate Distribution

- displays relationship between 2 variables (bivariate) as well as 1D profiles (univariate) in the margins

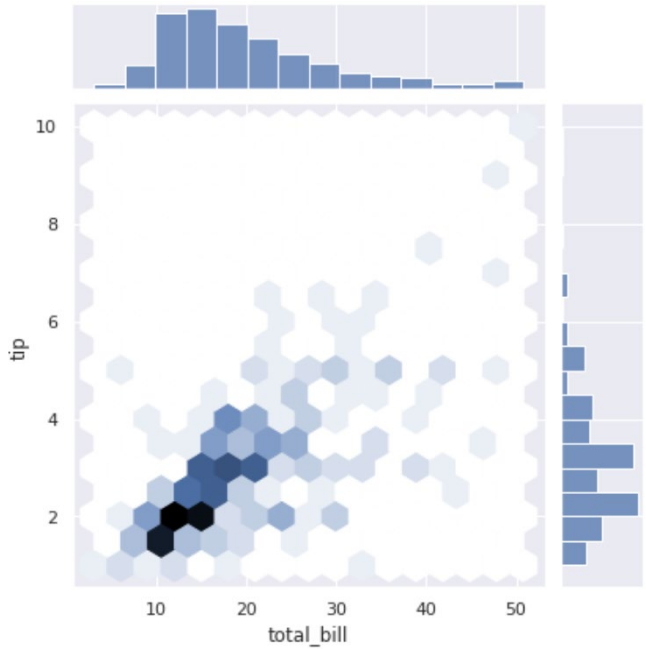
```
[54] x = tips['total_bill']
     y = tips['tip']

     sns.jointplot(x = x, y = y)
```

	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

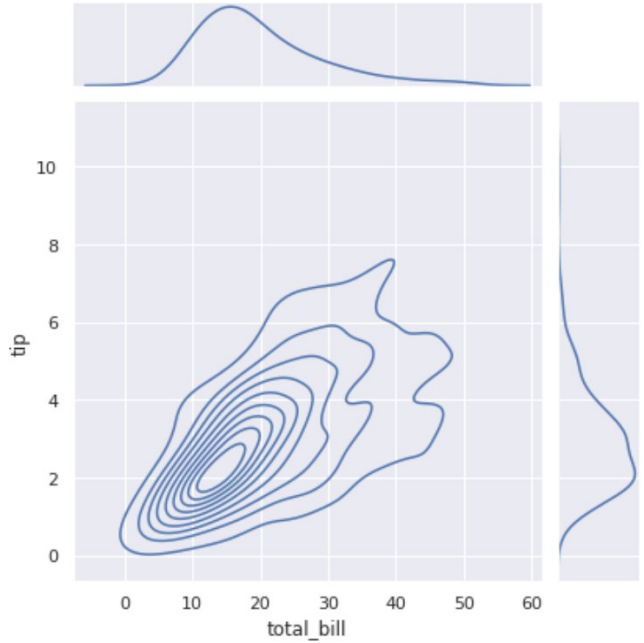


Bivariate Distribution (cont.)

- By using **kind** we can select the kind of plot to draw




```
kind = 'hex'
```

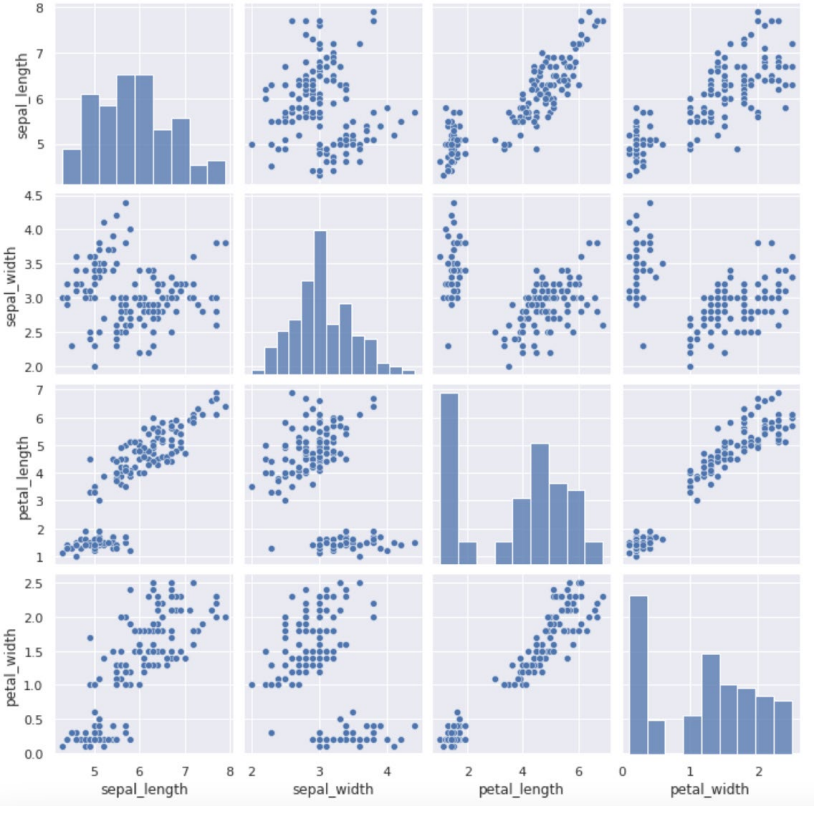



```
kind = 'kde'
```

Pair plot

- sns.pairplot() plots pairwise relationships in a dataset

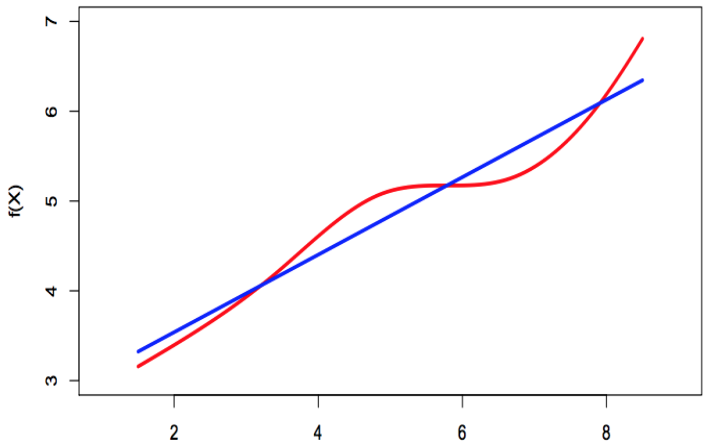
```
[57] sns.pairplot(iris)
```



Linear Regression and Relationship

Linear Regression

- Simple approach to supervised learning
- Assume that the dependence of Y on X_1, X_2, …, X_p is linear

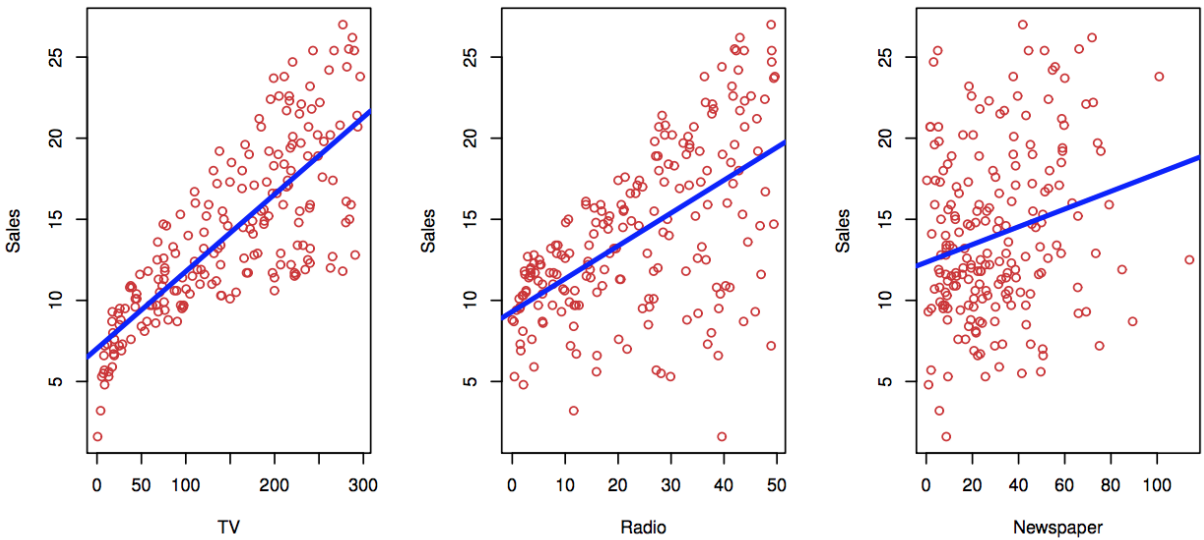


e.g.

$$y = f(x) = \begin{bmatrix} w_1 & w_2 & w_3 & \dots \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \end{bmatrix} = 1 + w_1 x_1 + w_2 x_2 + \dots$$

- Although it may seem overly simplistic, linear regression is extremely useful both conceptually and practically

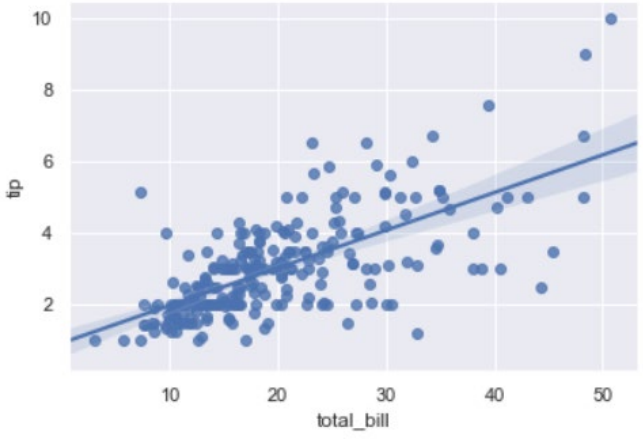
Example


Sales
5
10
15
20
25
0
50
100
200
300
TV
Sales
5
10
15
20
25
0
10
20
30
40
50
Radio
Sales
5
10
15
20
25
0
20
40
60
80
100
Newspaper

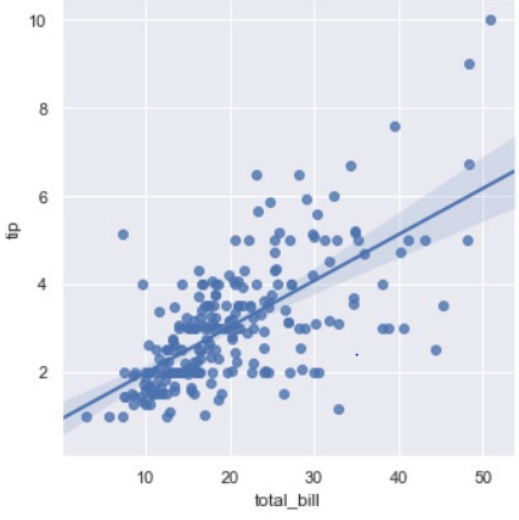
Regression Plot

- We can draw regression plots with the help of *regplot()* and *lmplot()*
- The plot drawn below shows the relationship between total_bill and tip

```
sns.regplot(x = 'total_bill', y = 'tip', data = tips)
```




```
sns.lmplot(x = 'total_bill', y= 'tip', data = tips)
```



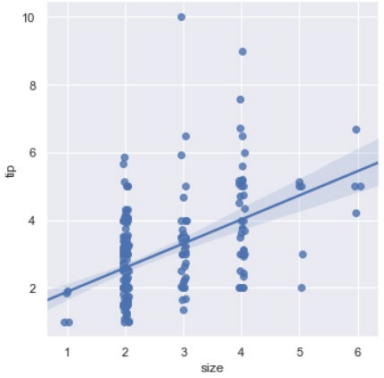
Linear Relationship

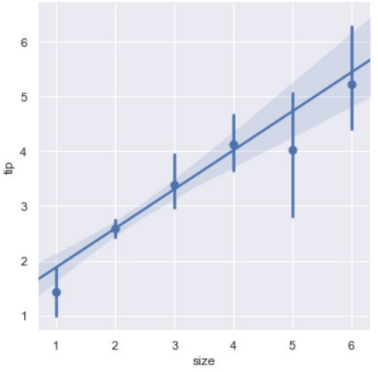
- We can draw a plot which shows the linear relationship between size and tips.

```
sns.lmplot(x = 'size', y = 'tip', data = tips, x_jitter = 0.05)
```

- If we set *x_estimator = np.mean* the dots in the above plot will be replaced by the mean and a confidence line.

```
sns.lmplot(x = 'size', y = 'tip', data = tips, x_estimator = np.mean)
```





Nonlinear Regression

- We will see how to draw a plot for the data which is not linearly related
- To do this we will load the *anscombe* dataset

```
data = sns.load_dataset('anscombe')
data.head()
```

	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33

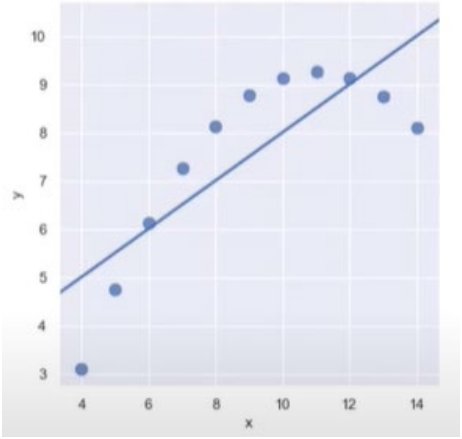
- This dataset contains 4 types of data and each type contains 11 values

```
data['dataset'].value_counts()

II     11
I      11
III    11
IV     11
Name: dataset, dtype: int64
```

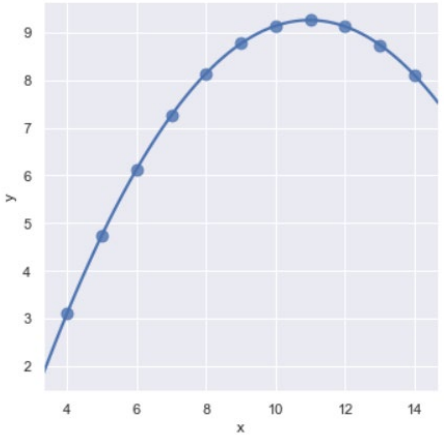
Nonlinear Regression (cont.)

- We can see that the dataset II is not a linear relation
- In order to fit such type of dataset, we can use the *order* parameter
- If an order is greater than 1, it estimates a polynomial regression

```
sns.lmplot(x = 'x', y = 'y', data = data.query("dataset == 'II'"),
ci = None, scatter_kws={'s': 80}, order = 2)
```



order = 1

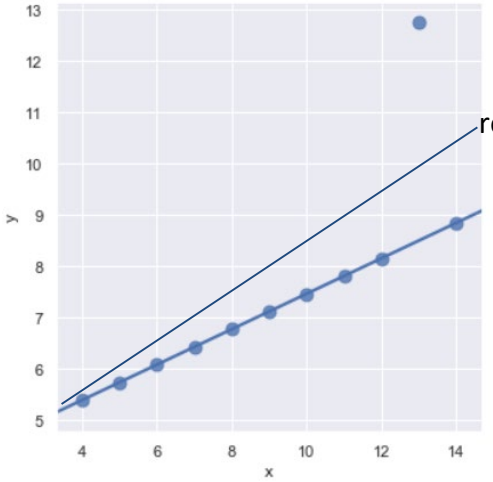


order = 2

Handling Outliers

- An outlier is a data point that differs significantly from other observations
- We can go and manually remove the outlier from the dataset or we can set *robust = True* to nullify its effect while drawing the plot

```
sns.lmplot(x = 'x', y = 'y', data = data.query("dataset == 'III'"), ci = None, scatter_kws={'s': 80}, robust=True)
```



Conclusion

- Visualizing data should be the first step before making any complex analysis
- Visualize how the data looks like and what kind of correlation is held by the attributes of data
- Visualization help us find patterns (clues) by exploring if there are any clusters within data or if data are linearly separable/too much overlapped