Seaborn

Content

- **Introduction**
- **Installing and Importing**
- **Loading Dataset using seaborn**
- **Plots using Seaborn**
  - Histogram
  - KDE Plot
  - Scatterplot
  - Joint Plot
  - Pair Plot
- **Categorical Plots**
  - Count Plot
  - Box Plot
  - Voilin Plot
- **Finding correlations among attributes**
  - `corr()`
  - Heat Map
- **Choosing right visualization for a given purpose**
- **Use Case: Visualizing Tips Dataset**
- **Challenge: Visualize Titanic Dataset**

Agenda

- Here, we'll cover another Data Visualization Library - **Seaborn**
- We'll also see some **interesting things** we can do **using Seaborn** to visualize our data

Introduction

- **Seaborn** is also a Python **Data Visualisation Library**
- Just **like Matplotlib**
- Infact, it is **built on top of matplotlib**
- It uses **Pandas DataFrame** to work with datasets

Installing Seaborn

You must know this by now

Using `!pip install` command

In [1]:

```
!pip install seaborn
```

```
Requirement already satisfied: seaborn in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (0.11.2)
Requirement already satisfied: pandas>=0.23 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from seaborn)
(1.4.2)
Requirement already satisfied: numpy>=1.15 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from seaborn)
(1.21.5)
Requirement already satisfied: matplotlib>=2.2 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from seabor
n) (3.5.1)
Requirement already satisfied: scipy>=1.0 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from seaborn)
(1.7.3)
Requirement already satisfied: python-dateutil>=2.7 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from m
atplotlib>=2.2->seaborn) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from matplotl
ib>=2.2->seaborn) (9.0.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from matp
lotlib>=2.2->seaborn) (1.3.2)
Requirement already satisfied: fonttools>=4.22.0 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from matp
lotlib>=2.2->seaborn) (4.25.0)
Requirement already satisfied: cycler>=0.10 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from matplotli
b>=2.2->seaborn) (0.11.0)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from matpl
otlib>=2.2->seaborn) (3.0.4)
Requirement already satisfied: packaging>=20.0 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from matplo
tlib>=2.2->seaborn) (21.3)
Requirement already satisfied: pytz>=2020.1 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from pandas>=
0.23->seaborn) (2021.3)
Requirement already satisfied: six>=1.5 in /Users/shivank/opt/anaconda3/lib/python3.9/site-packages (from python-dateut
il>=2.7->matplotlib>=2.2->seaborn) (1.16.0)
```

Importing Seaborn

- You should be able to import Seaborn after installing it
- We'll import `seaborn` as its **alias name** **`sns`**

In [4]:

```
import seaborn as sns
```

We'll also import `numpy`

- To make use of its **numeric calculations**

We'll import `matplotlib.pyplot` as well

- **Only to display final plots** generated using seaborn's functionality
- Just for **`plt.show()`**

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
```

Loading Dataset using `seaborn`

- Seaborn contains some **datasets in-built within the library itself**
- We can use its **`load_dataset()`** function to **read the data**
- Works just like Pandas's `read_csv()`

Have you heard of the Iris Dataset?

- It's like the **"Hello World" of datasets**
- Mostly used by **beginners** as a practice
- It's the dataset about **3 species of Iris flower**
- Feel free to read more about the dataset here:

  https://en.wikipedia.org/wiki/Iris_flower_data_set (https://en.wikipedia.org/wiki/Iris_flower_data_set)

Let's load and read Iris Dataset using `seaborn`

In [5]:

```
iris = sns.load_dataset('iris')
```

Now, Let's check what all is there in the dataset

In [4]:

```
type(iris)
```

Out[4]:

```
pandas.core.frame.DataFrame
```

In [5]:

```
iris.head()
```

Out[5]:

	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

As you can see:

- The iris dataset is **loaded as a Pandas DataFrame**
- The iris dataset has **5 columns (attributes)**

    1. `sepal_length`
    2. `sepal_width`
    3. `petal_length`
    4. `petal_width`
    5. `species`

We'll explore these 5 variables (features) one by one

- The last attribute `species` tells us which **species** the flower is **based on those measurements**

Let's check it out in a bit more detail

In [6]:

```
iris['species'].unique()
```

Out[6]:

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

So, the flowers in dataset can belong to one of the 3 categories

1. `setosa`
2. `versicolor`
3. `virginica`

- However, we will not go into the details of the dataset

The purpose of this lecture is to see the functionality of `seaborn`

- Feel free to explore the iris dataset more on your own

Let's plot our first graph using `seaborn` for the variable `petal_length`

In [7]:

```
iris['petal_length']
```

Out[7]:

```
0      1.4
1      1.4
2      1.3
3      1.5
4      1.4
      ...
145    5.2
146    5.0
147    5.2
148    5.4
149    5.1
Name: petal_length, Length: 150, dtype: float64
```

Quiz:

Look at this `petal_length` variable and tell Which graph/plot will be most appropriate to plot this variable?

Ans. B

A. Bar Graph

B. Histogram

C. ScatterPlot

D. Box Plot

Histogram

- We want to check the **distribution of this variable `petal_length`**
- So, we use Histogram and not any other plot !!
- For **Scatter Plot**, we need **2 variables** - one on x-axis and one on y-axis
- For **Bar Chart**, we need a **categorical variable**
  - Like if we want to count how many employees in a company have Bachelors, Masters or PhD.

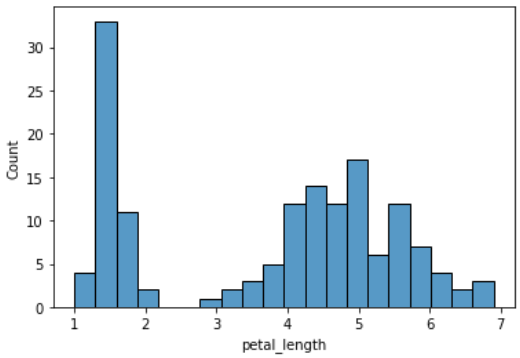
Remember histogram from the matplotlib lecture?

- We can use seaborn's `histplot()` to plot a histogram
- We can also set the number of bins we want in our plot

Let's plot the distribution for the `petal_length` column

In [8]:

```
sns.histplot(iris['petal_length'], bins= 20)
plt.show()
```



As you can see here

- We got a **histogram for `petal_length` attribute**
- **More than 30 flowers** have `petal_length` **between approx 1.25 and 1.5**
- **No flowers** have `petal_length` between approx **2.25 and 2.75**
- ... and so on. You can make similar other observations

See how neatly each bar is separated from the other

- Like we mentioned `seaborn` is built on top of `matplotlib`
- It has **some features** (not all) that are an **enhancement** over `matplotlib.pyplot`

Now, let's check out another plot using `seaborn`

Kernel Density Estimate (KDE) Plot

- A KDE plot is a method for visualizing the distributions
- Just like histogram
- But instead of bars, KDE represents data using a **continuous probability density curve**
- You can check documentation for more details

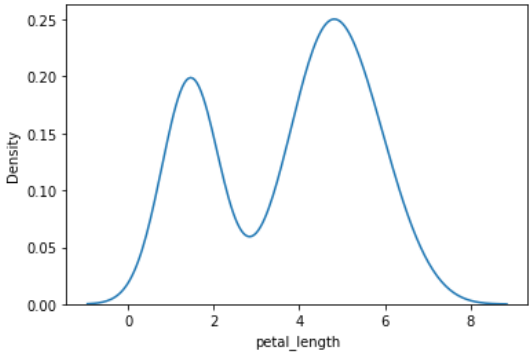
Now, Why do we even need KDE plots?

- Compared to histogram, KDE produces a plot which is **less cluttered** and **more interpretable**
- Especially when drawing multiple distributions.

Let's plot KDE using `seaborn`'s `kdeplot`

In [9]:

```
1 sns.kdeplot(iris['petal_length'])
2 plt.show()
```



As you can see

- We got a **curve** instead of bars
- **y-axis** has the **probabilities** instead of actual count

It gives the same information as histogram, but in a smoother way

- Probability of flowers having petal_length between approx 1.25 and 1.5 is high (~ 0.2)
- Probability of flowers having petal_length between approx 2.25 and 2.75 is low (~ 0.05)
- ... and so on

Let's move on to scatterplot now

Scatterplot

Remember scatterplot from Matplotlib?

Scatter Plot is mostly used to visualize relationship b/w 2 variables (Bi-Variate Analysis)

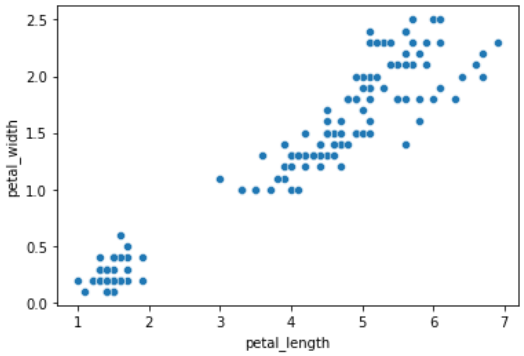
- A scatter plot displays each (x, y) coordinate point separately
- The points are scattered over the graph

Let's plot scatterplot now using `seaborn`

- We will take **`petal_length` as x-coordinates** and **`petal_width` as y-coordinates**
- These (x, y) pairs of points will be plotted as a scatter plot

In [9]:

```
sns.scatterplot(x= iris['petal_length'], y = iris['petal_width'])
plt.show()
```



Did you notice?

- We did not have to specifically mention `plt.xlabel()` or `plt.ylabel()`
- **Seaborn automatically labelled axes for us**, unlike matplotlib.
- So, we don't have to write many lines of code in this case

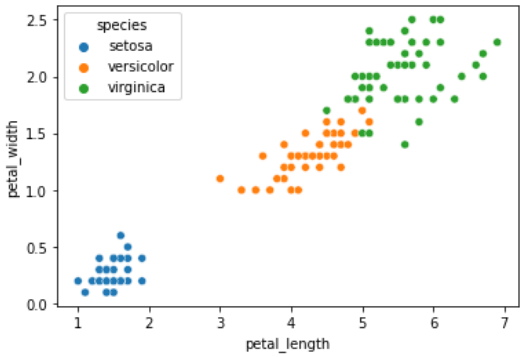
There's one more interesting thing we can do with `seaborn` 's `scatterplot`

- We can visualize different species in different colours

Let's see how we can do it

In [11]:

```
sns.scatterplot(x ='petal_length', y ='petal_width' , data= iris, hue='species')
plt.show()
```



- So, we provide our iris dataset as **input `data` parameter**
- We **set the `hue`** to `species` column

Each different species is plotted in a different colour

- The default colours are chosen automatically by the plot
- Check the **top left corner**
  - **Colour Legends** for each category are also shown automatically

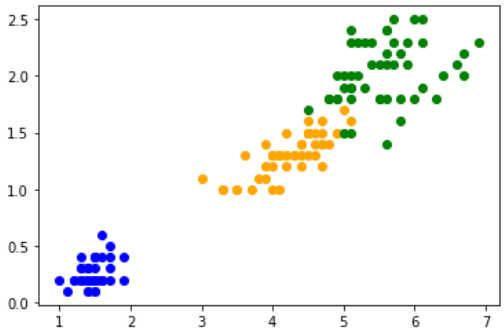
Now, What if we had to draw the same Scatter Plot using Matplotlib?

- We'd have to separate the datasets for the 3 flowers first
- Then, we'd have to write scatterplot code for each individual dataset

Let's try and draw the same Scatter Plot using Matplotlib and compare it with Seaborn

In [8]:

```
setosa = iris[iris['species'] == 'setosa']
versicolor = iris[iris['species'] == 'versicolor']
virginica = iris[iris['species'] == 'virginica']

plt.scatter(x=setosa['petal_length'], y=setosa['petal_width'], c = 'blue')
plt.scatter(x=versicolor['petal_length'], y=versicolor['petal_width'], c = 'orange')
plt.scatter(x=virginica['petal_length'], y=virginica['petal_width'], c = 'green')

plt.show()
```



- As you can see, we got the same plot.
- But with seaborn, the **code is just so much simpler and smaller**
- That's the **convenience** we have with seaborn

Let's see a few more plots that we can visualize using `seaborn`

Joint Plot

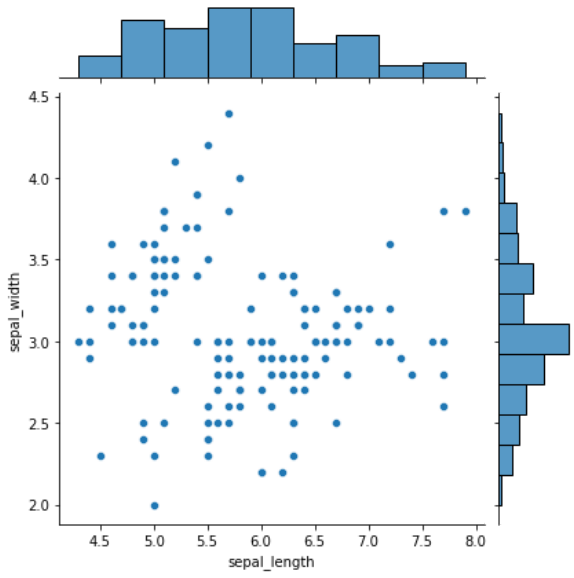
- It draws a plot of two variables
- It shows scatter, histogram and KDE graphs in the same plot.

Let's check it out

- We will take **`sepal_length` as x-coordinates** and **`sepal_width` as y-coordinates**
- Again, we will pass iris dataset as **input `data` parameter**
- We can select from different values for **parameter `kind`** and it **will plot accordingly**
  - "scatter" | "kde" | "hist" | "hex" | "reg" | "resid"
- We will set **parameter `kind`** to **`'reg'`** here

In [8]:

```
sns.jointplot(x= 'sepal_length', y = 'sepal_width', data= iris, kind='scatter')
plt.show()
```



As we can see here:

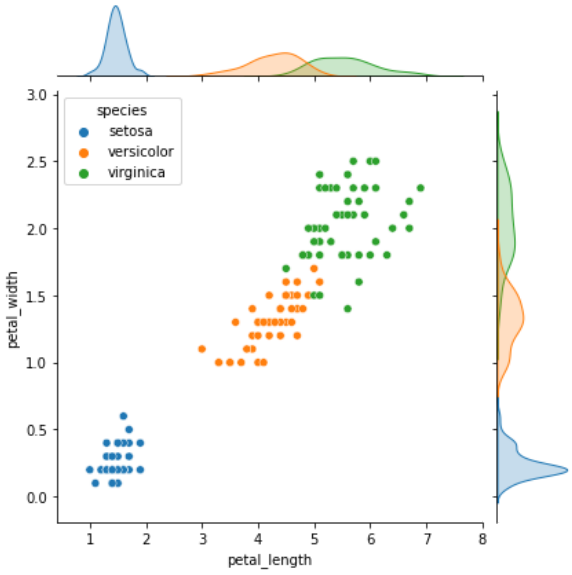
- `jointplot` plots **scatter, histogram and KDE in the same graph** when we set **`kind=reg`**
- Scatter shows the **scattering of (`sepal_length` , `sepal_width`) pairs as (x, y) points**
- Histogram and KDE shows the separate distributions of `sepal_length` and `sepal_width` in the data

We can also add hue to Joint Plot

- Let's check how the 3 species of flowers are distributed in terms of `petal_length` and `petal_width`
- We'll take **`petal_length`** as **x-coordinates** and **`petal_width`** as **y-coordinates**

In [6]:

```
sns.jointplot(x= 'petal_length', y = 'petal_width', data= iris, hue='species')
plt.show()
```



Question: Can you obtain some observations from this Joint Plot?

- **`Setosa` flowers are clearly separable** from rest of the flowers **in terms of `petal_length` and `petal_width`**
- There is some **overlap b/w `Versicolor` and `Virginica`** as can be seen in both Scatter part and KDE part of the plot
- **`petal_length` and `petal_width`** of **`Setosa`** is **smaller** than **`Versicolor` and `Virginica`**
- ... and so on, you can make many more observations from this plot.

Pair Plot

- `pairplot()` in `seaborn` creates a **grid of Axes by default**
- Each numeric attribute in `data` is shared across the y-axes across a single row and the x-axes across a single column.
- It displays a **scatterplot between each pair of attributes in the data** with different **hue** for each category

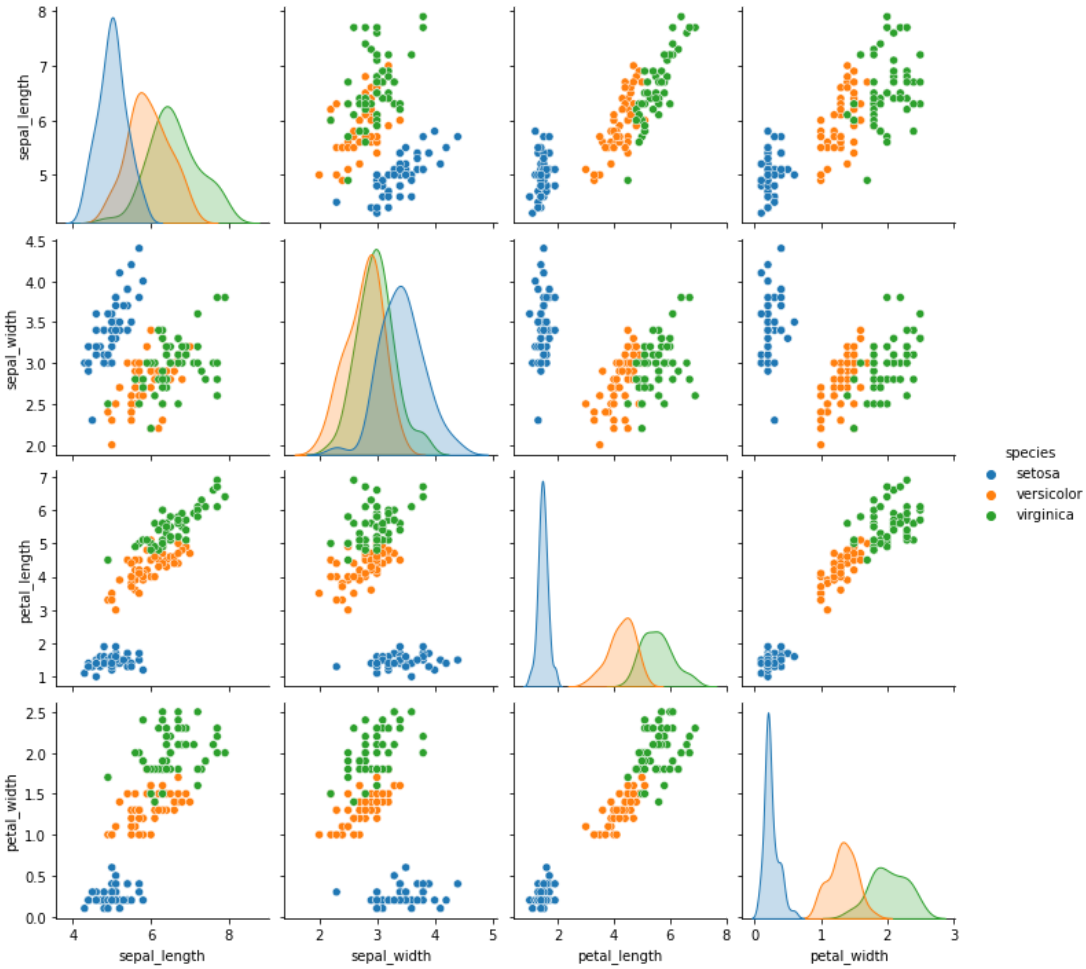
Since, the diagonal plots belong to same attribute at both x and y axis, they are treated differently

- A univariate distribution plot is drawn to show the marginal distribution of the data in each column.

Let's check it out

In [19]:

```
sns.pairplot(data = iris, hue= 'species')
plt.show()
```



Notice that:

- It is **like a scatterplot of iris with `hue='species'`**
- But the **scatter is plotted between every pair of attributes**
- **Colour Legends** for each species category are given on **right side**
- It shows **relation between each pair of attributes**

Diagonal plots are different from scatterplots

- Because x and y axis have same attribute
- Diagonal plots show a univariate curve category-wise for each attribute

It is also possible to show a subset of variables or plot different variables on the rows and columns

- Feel free to experiment this on your own

Quiz

From this pair plot, which 2 features can separate the data points category-wise?

Ans. A

A. `petal_length` and `petal_width`

B. `sepal_length` and `sepal_width`

C. `sepal_length` and `petal_width`

D. `petal_length` and `sepal_width`

Categorical Plots

Now, we'll see some Categorical Plots using `seaborn`

- Categorical Plots are the **plots based on categories**

Do we have a categorical variable in our dataset?

- **`species` in the case of iris dataset**

Which plots, in general, do you think are suitable to visualize categorical variables?

- Bar Plots
- Because we want to see the **frequency/count of data points belonging to each category** of that categorical variable

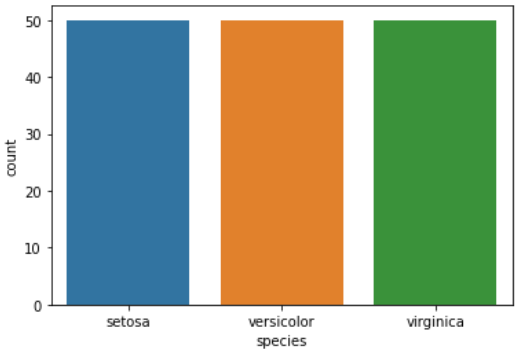
Count Plot

- It's like a bar plot
- It plots a simple bar graph displaying **count of datapoints (rows) belonging to each category**
- We provide **column label on x-axis**
- **Count of data** (no. of flowers) **belonging to each category** in the column is **on y-axis**

Let's see it in action

In [20]:

```
sns.countplot(x = 'species', data = iris)
plt.show()
```



What do you notice?

How many flowers in dataset belong to each species?

- **All the 3 species** - `setosa`, `versicolor` and `virginica` have the **same number of flowers**
- There are **50 flowers belonging to each species in the Iris Dataset**

Box Plot

- It draws a box plot to show distributions with respect to categories.

But what exactly is a Box Plot?

- A box plot or **box-and-whisker plot** shows the **distribution of quantitative data** in a way that **facilitates comparisons between attributes** or **across levels** of a categorical attribute.
- The **box** shows the **quartiles** of the dataset
- While the **whiskers** extend to show the **rest of the distribution**
- Except for points that are determined to be "outliers" using a method that is a function of the **inter-quartile range**.

Getting confused with the new terminology?

- Don't worry
- We'll explain each term one-by-one

Let's start with understanding what's a quartile

- Box plot shows distributions of numerical data and skewness through displaying the **data percentiles**, called **quartiles**

Box plots show the five-number summary of data:

1. Minimum score,
2. first (lower) quartile
3. Median
4. Third (upper) quartile
5. maximum score

Minimum Score

- It is the **lowest value**, excluding outliers
- It is shown at the **end of bottom whisker**

Lower Quartile

- **25% of values fall below the lower quartile value**
- It is also known as the **first quartile**.

Median

- Median marks the **mid-point of the data**
- It is shown by the **line that divides the box into two parts**
- It is sometimes known as the **second quartile**.
- **Half the scores are greater than or equal to this value and half are less**.

Upper Quartile

- **75% of the values fall below the upper quartile value**
- It is also known as the **third quartile**.
- So, **25% of data are above this value**.

Maximum Score

- It is the **highest value**, excluding outliers
- It is shown at the **end of upper whisker**.

Whiskers

- The upper and lower whiskers represent **values outside the middle 50%**
- That is, the **lower 25% of values** and the **upper 25% of values**.

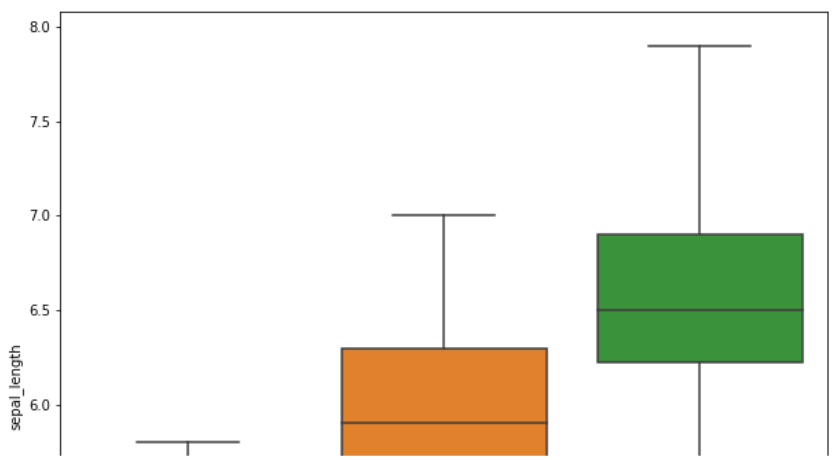
Interquartile Range (or IQR)

- This is the box plot showing the **middle 50% of scores**
- It is the **range between the 25th and 75th percentile**.

Now, Let's plot a box plot to check variation of `sepal_length` among the 3 species of iris

In [21]:

```
plt.figure(figsize=(10,10))
sns.boxplot(x = 'species', y = 'sepal_length', data = iris)
plt.show()
```



As you can see:

- **Different species have different range of `sepal_length`**

Each species' box plot shows:

- what is the **lowest `sepal_length`** in data for that species

- what is the **25th percentile (lower quartile) value of** `sepal_length` for that species
- what is the **median** `sepal_length` in data for that species
- what is the **75th percentile (upper quartile) value of** `sepal_length` for that species
- what is the **highest** `sepal_length` in data for that species

Whiskers show the

- `sepal_length` **outside the middle 50% of values**
- The **lower 25% of** `sepal_length` and the **upper 25% of** `sepal_length` .

Voilin Plot

- Its a **combination of Box Plot and Distribution Plot**.
- It works similar to a box and whisker plot.
- It shows the distribution of quantitative data across several levels of categorical attribute such that those distributions can be compared.

How is it different from box plot?

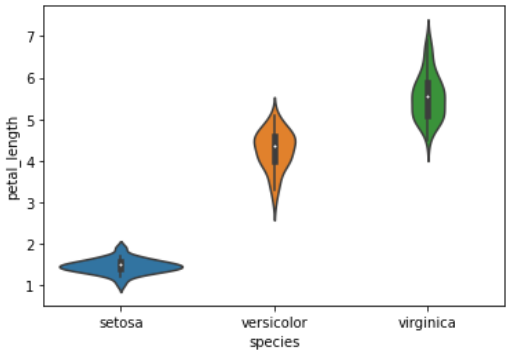
- In a box plot, all of the plot components correspond to actual datapoints
- Whereas, the violin plot features a KDE of the underlying distribution.

This can be an effective and attractive way to show multiple distributions of data at once

Let's draw a Violin Plot for `petal_length` **for the 3 species of iris**

In [24]:

```
sns.violinplot(x = 'species', y = 'petal_length', data = iris)
plt.show()
```



Observe:

- Inside violin, you will see a boxplot.
- Left and right side represents the **distributions of** `petal_length` **for each species**.

Finding correlations among attributes

- We can find the level of correlation b/w different attributes (variables)

But what exactly is a correlation?

- Two variables are correlated when **they change in same/opposite direction**

We can check coefficient of correlation using `corr()`

In [26]:

```
iris.corr()
```

Out[26]:

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.117570	0.871754	0.817941
sepal_width	-0.117570	1.000000	-0.428440	-0.366126
petal_length	0.871754	-0.428440	1.000000	0.962865
petal_width	0.817941	-0.366126	0.962865	1.000000

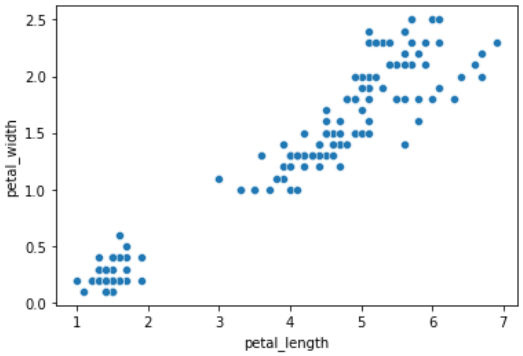
- Higher the **MAGNITUDE** of coefficient of correlation, more the variables are **correlated**
- The **sign just determines the direction of change**
  - + means increase in value of one variable causes increase in value of other variable
  - - means increase in value of one variable causes decrease in value of other variable, and vice versa

As you can see, `petal_length` and `petal_width` have the highest correlation coeff of 0.96

Let's plot it using scatter plot

In [30]:

```
sns.scatterplot(x= 'petal_length', y= 'petal_width', data = iris)
plt.show()
```



- When `petal_length` increases, `petal_width` also increases

But Remember

Correlation does NOT mean Causation

- We cannot conclude that change in values of a variable is causing change in values of other variable

Now, Let's look at a way to visualize correlation among variables

Heat Map

- A heat map plots rectangular data as a color-encoded matrix.
- **Stronger the colour, stronger the correlation b/w the variables**

Let's plot a Heat Map using correlation coefficient matrix generated using `corr()`

In [32]:

```
sns.heatmap(iris.corr(), cmap= "Blues", annot=True)
plt.show()
```

- **`annot=True`** is for writing correlation coeff inside each cell

You can change the colours of cells in Heat Map if you like

- There are a lot of options available!

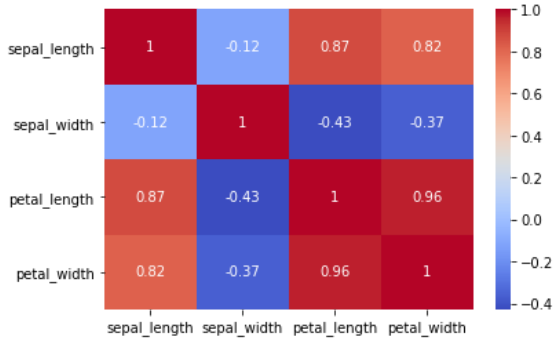
In [31]:

```
print(plt.colormaps())
```

```
['Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_r', 'BuPu', 'BuPu_r', 'CMRmap', 'CMRmap_r',
'Dark2', 'Dark2_r', 'GnBu', 'GnBu_r', 'Greens', 'Greens_r', 'Greys', 'Greys_r', 'OrRd', 'OrRd_r', 'Oranges', 'Oranges_
r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1', 'Pastel1_r', 'Pastel2', 'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu',
'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_r', 'RdBu', 'RdBu_r', 'RdGy',
'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2', 'Se
t2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_r', 'YlGn_r', 'YlOrB
r', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'bone', 'bone_
r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r', 'copper', 'copp
er_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_he
at', 'gist_heat_r', 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_stern_r', 'gist_y
arg', 'gist_yarg_r', 'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'hot', 'hot_r', 'hsv', 'hsv_
r', 'icefire', 'icefire_r', 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r', 'mako', 'mako_r', 'nipy_spectra
l', 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_r', 'prism', 'prism_r', 'rainbow', 'rain
bow_r', 'rocket', 'rocket_r', 'seismic', 'seismic_r', 'spring', 'spring_r', 'summer', 'summer_r', 'tab10', 'tab10_r',
'tab20', 'tab20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terrain', 'terrain_r', 'twilight', 'twilight_r', 'twil
ight_shifted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'vlag', 'vlag_r', 'winter', 'winter_r']
```

In [31]:

```
sns.heatmap(iris.corr(), cmap= "coolwarm", annot=True)
plt.show()
```



Quiz

What is the larget correlation coefficient b/w 2 different variables?

Ans. C

A. 1

B. 0.98

c. 0.96

D. 0.86

Choosing right visualization for a given purpose

- There's a whole bunch of charts and plots we've seen
  - Bar chart
  - Historgam
  - Box Plot
  - Violin Plot
  - Scatterplot
  - Count Plot
  - Heat Map
  - ... and so on
- But we always need to **select the right plot for every purpose**

What is the right chart to use for a given problem?

- We need to decide the right chart/plot to use for a dataset at-hand
- We can't just blindly use any chart for any data that's available to us
- There are certain **thumb rules** that we need to consider

First, Let's see how much understanding of plots you have got after all the discussion we've done so far

Quiz

What is the right chart to use to **find dependency between one Continuous Variable and Categorical variable**?

Ans. C

A: Bar Chart

B: Histogram

C: Box Plot

D: Violin Plot

Now, What exactly is the process of selecting the right chart?

- First you need to look at what is the type of variable you're dealing with

Let's divide this step in into:

1. 1-Dimensional
2. 2-Dimensional
3. Multi-Dimensional

1-Dimensional Visualization

Which one will we choose when we just want to analyze 1 variable?

- 1-D
- We look at only 1 variable at a time

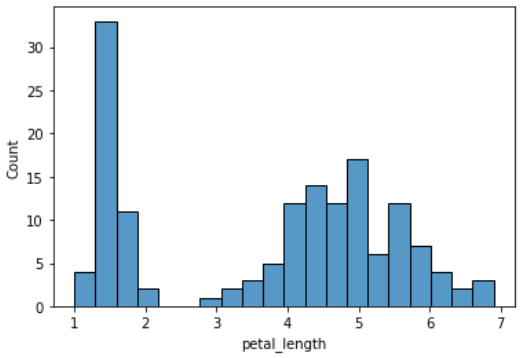
Now, Which plot to use if the variable is continuous (numeric)?

- **Histogram** ---> To see **distribution** of that continuous variable
- **Box Plot** ---> To see the **inter-quartile range of values** of that continuous variable

For example: What we saw with `petal_length` above:

In [12]:

```
sns.histplot(iris['petal_length'], bins= 20)
plt.show()
```

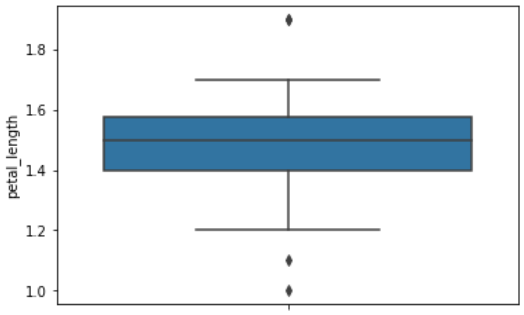


Let's say we want to visualize how `petal_length` varies for just `setosa` species

- We can utilize the **Box Plot** here we saw earlier

In [16]:

```
setosa = iris[iris['species'] == 'setosa']
sns.boxplot(y = 'petal_length', data = setosa)
plt.show()
```



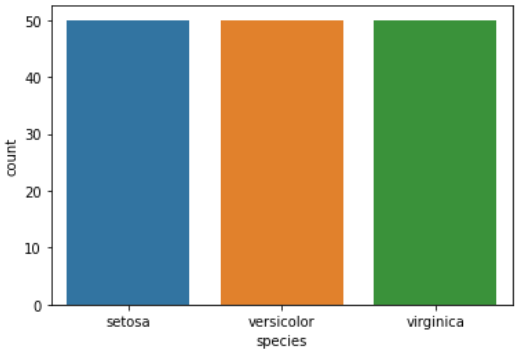
What if the variable is categorical?

- Categorical variable will have **discrete unique values**
- **Bar Chart / Count Plot** ---> To see **how many datapoints belong to each category**
- **Pie Chart** ---> To see **ratio (%age) of datapoints** belonging to each category

For example: Bar/Count Plot we saw above to check no. of flowers belonging to each species:

In [17]:

```
sns.countplot(x = 'species', data = iris)
plt.show()
```

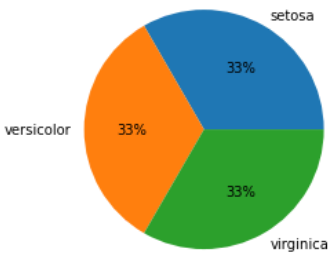


We can also check the proportions / %ages of flowers belonging to each species using a Pie Chart:

- **Seaborn doesn't have a direct function to create a pie chart**
- So, we'll use **matplotlib** here

In [22]:

```
n_setosa = iris[iris['species'] == 'setosa'].shape[0]  # We are getting the number of datapoints in each species
n_versicolor = iris[iris['species'] == 'versicolor'].shape[0]
n_virginica = iris[iris['species'] == 'virginica'].shape[0]

data = [n_setosa, n_versicolor, n_virginica]
labels=['setosa','versicolor','virginica']

plt.pie(data,
        labels=labels,
        autopct='%.0f%%') # To show the portions in %ages

plt.show()
```



2-Dimensional Visualization

Now, What if we want to analyze 2 variables at a time?

- We usually do this to check the relationship b/w 2 variables
- Its a **Bi-Variate Analysis**

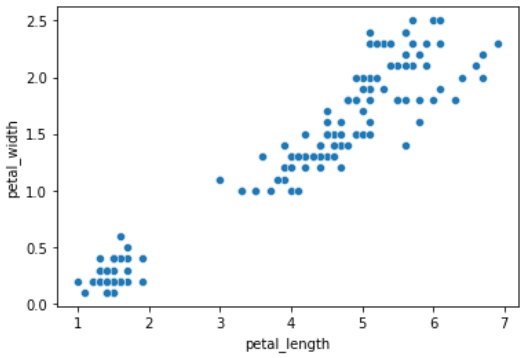
Now, Which chart will we use if both variables are continuous (numeric)?

- **Scatter Plot** ---> To see how the 2 continuous variables are **dependent on each other or vary with each other**
- **Line Chart** ---> To see the **approximate relationship (dependency) b/w the 2 variables** represented by a line

For example: When we analyzed how `petal_length` and `petal_width` vary with each other

In [23]:

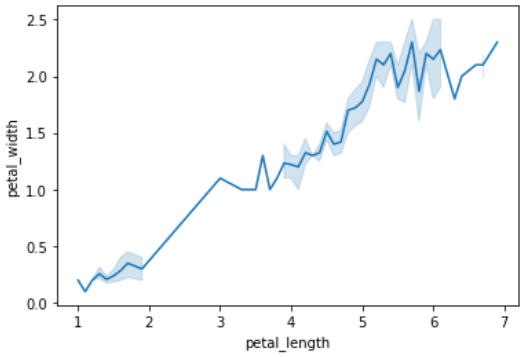
```
sns.scatterplot(x= 'petal_length', y= 'petal_width', data = iris)
plt.show()
```



We can also use Line Chart to get an approximate relationship b/w `petal_length` and `petal_width`

In [25]:

```
sns.lineplot(x= 'petal_length', y= 'petal_width', data = iris)
plt.show()
```



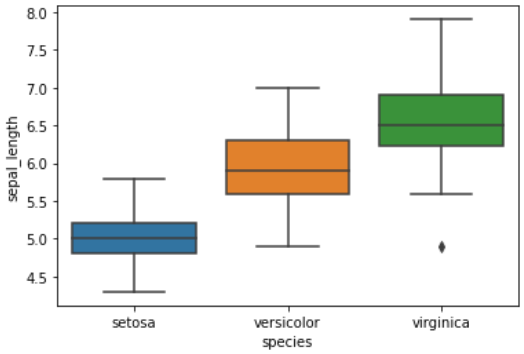
Now, Which chart to use when we have 1 continuous and 1 discrete variable?

- You should know this by now!
- **Box Plot** ---> To see **distribution of numeric variable across each category of categorical variable**

For example: When we checked above how `sepal_length` varies for each species of iris

In [27]:

```
sns.boxplot(x = 'species', y = 'sepal_length', data = iris)
plt.show()
```



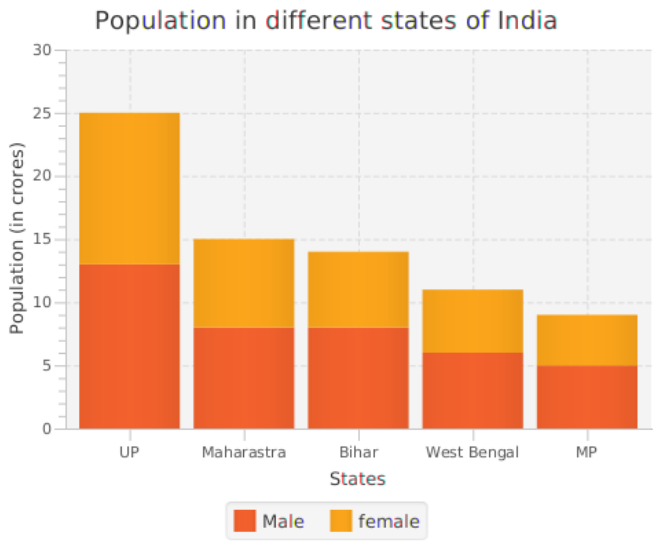
- It helps us answer questions like **"is there a change/difference in `sepal_length` for different species?"**
- Box Plot allows these types of **comparisons of a numerica variable among different categories**

Now, What if both our variables are categorical?

- We can use a Bar Chart (Stacked or Dodged)

For example:

- Let's say we have Population data of different states
- Now, we want to check **how many Males and Females are there in each states**
- A **Stacked/Dodged Bar Chart can be appropriately used here**



In [1]:

```
#<img src="https://drive.google.com/uc?export=view&id=1TyWzzJPAsYCIZzDhEvjrORR4JScQ8xK7" width="400"/>
```

Multi-Dimensional Visualization

Now, Let's talk about multi-dimensional visualization

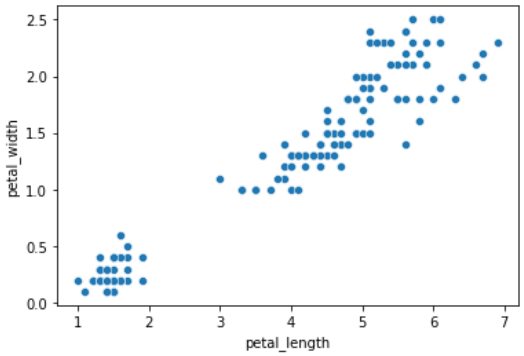
- It's not very straightforward
- But if plotted in a **constructive way**, a multi-dimensional plot can reveal a lot of useful information

Let's start with 2 dimensions, i.e., 2 variables, and then we'll add more dimensions to the plot

Do you remember the Scatter Plot b/w `petal_length` and `petal_width` ?

In [28]:

```
sns.scatterplot(x= 'petal_length', y= 'petal_width', data = iris)
plt.show()
```



We can make use of

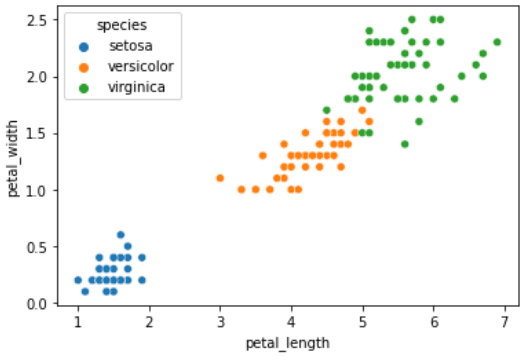
- Colour
- Shape
- Size

to add more dimensions to our Scatter Plot

We have already used "colour" when we plotted Scatter Plot b/w `petal_length` and `petal_width` and added `species` as `hue` to the Scatter Plot

In [29]:

```
sns.scatterplot(x ='petal_length', y ='petal_width' , data= iris, hue='species')
plt.show()
```



- Different species are now represented by different colours
- We had a 2-D Scatter Plot b/w `petal_length` and `petal_width`
- The, we added **`species` as a 3rd dimension** using the **colour**

Now, It's a multi-dimensional plot

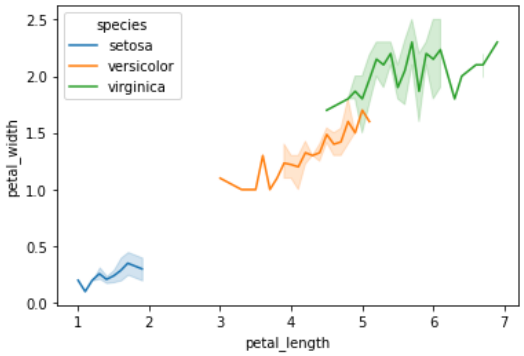
By looking at it, Can you now tell "which points belong to which species of iris"?

- You can also tell what's the **range (clusters) of `petal_length` and `petal_width`** of each species

We can also add this dimension of species in our Line Plot we saw earlier:

In [30]:

```
sns.lineplot(x= 'petal_length', y= 'petal_width', data = iris, hue='species')
plt.show()
```



- Same way we can add more dimensions to the same plot using **shape** and **size** to both Scatter Plots
- We'll cover them when we come to **Exploratory Data Analysis (EDA)**

- For now, just remember that we can always **constructively plot a multi-dimensional plot to reveal important information from our data**

Visualizing Tips Dataset

- Let's move on to visualizing another dataset from `seaborn` library
- **Tips Dataset**
- Its some tipping data where one waiter recorded information about each tip he/she received over a period of a few months working in one restaurant

Let's go ahead and load the data

In [13]:

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

Let's see what all the tips data includes

In [33]:

```
tips.shape
```

Out[33]:

```
(244, 7)
```

In [34]:

```
tips.head()
```

Out[34]:

	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

It has 7 columns

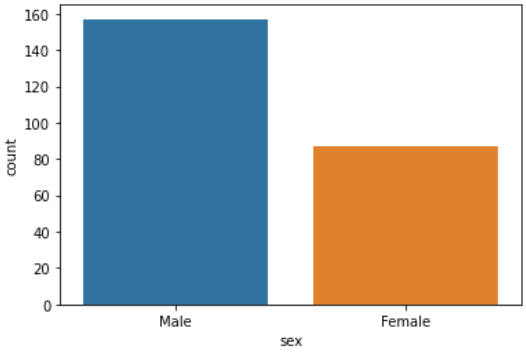
- How much was the total bill
- How much did the customer tip
- Was the customer male or female
- Was the customer a smoker or not
- What day was it
- What meal time was it
- `size` represents no. of people in the party

Now, we'll answer a few questions related to dataset using visualization through `seaborn`

Can you visualize in how many cases the customer was male and in how many cases the customer was female?

In [39]:

```
sns.countplot(x = tips['sex'])
plt.show()
```



What all days does the dataset has?

In [40]:

```
tips['day'].unique()
```

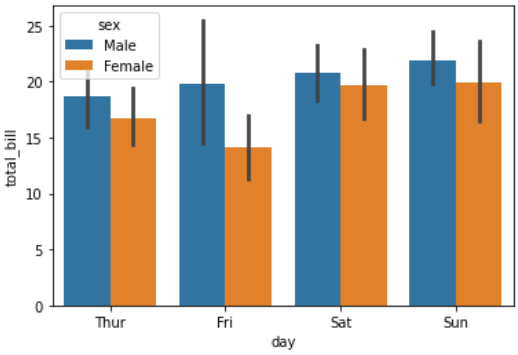
Out[40]:

```
[Sun, Sat, Thur, Fri]
Categories (4, object): [Sun, Sat, Thur, Fri]
```

Can you tell out of all the days which day gets the highest bill amount for both male and female customers?

In [36]:

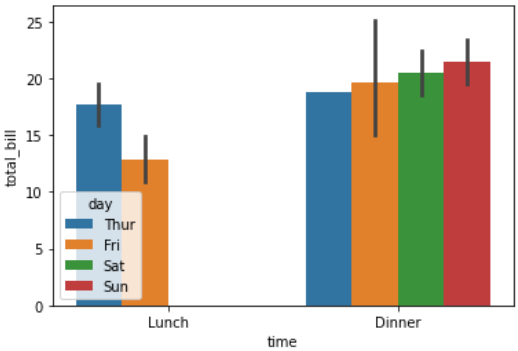
```
sns.barplot(x = 'day', y = 'total_bill', data=tips, estimator=np.mean, hue = 'sex')
plt.show()
```



Can you tell when do people prefer coming to restaurant more - lunch time or dinner time? And on what days?

In [35]:

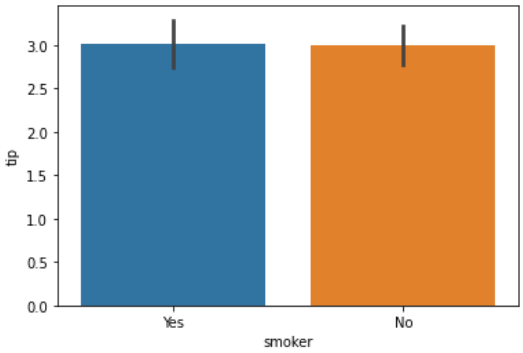
```
sns.barplot(x = 'time', y = 'total_bill', data = tips, hue = 'day')
plt.show()
```



Can you tell whether people who smoke give more tip or not?

In [37]:

```
sns.barplot(x = 'smoker', y = 'tip', data = tips)
plt.show()
```

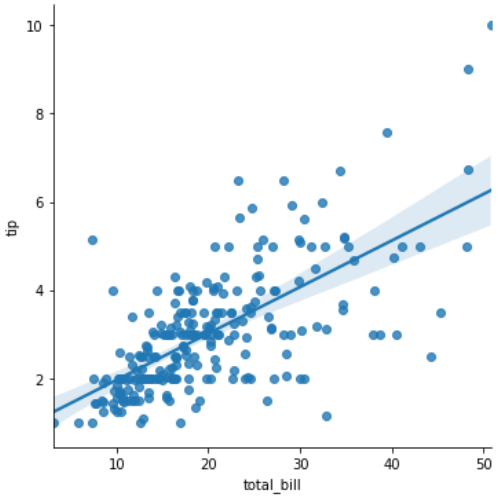


How are `total_bill` and `tip` related? Is there a correlation b/w them?

- We can use `lmplot()`

In [38]:

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



Challenge for you - Visualize Titanic Dataset

In [56]:

```
titanic = sns.load_dataset('titanic')
```

In [57]:

```
titanic.head()
```

Out[57]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [58]:

```
titanic.shape
```

Out[58]:

```
(891, 15)
```

Visualize the Titanic Data on your own

- Frame questions on your own and find answers
- We can take doubts and queries in next class

In [1]:

```
d1 = df[df["type"] == "TV Show"]
d2 = df[df["type"] == "Movie"]

col = "year_added"

vc1 = d1[col].value_counts().reset_index()
vc1 = vc1.rename(columns = {col : "count", "index" : col})
vc1['percent'] = vc1['count'].apply(lambda x : 100*x/sum(vc1['count']))
vc1 = vc1.sort_values(col)

vc2 = d2[col].value_counts().reset_index()
vc2 = vc2.rename(columns = {col : "count", "index" : col})
vc2['percent'] = vc2['count'].apply(lambda x : 100*x/sum(vc2['count']))
vc2 = vc2.sort_values(col)

trace1 = go.Scatter(x=vc1[col], y=vc1["count"], name="TV Shows", marker=dict(color="#a678de"))
trace2 = go.Scatter(x=vc2[col], y=vc2["count"], name="Movies", marker=dict(color="#6ad49b"))
data = [trace1, trace2]
layout = go.Layout(title="Content added over the years", legend=dict(x=0.1, y=1.1, orientation="h"))
fig = go.Figure(data, layout=layout)
fig.show()
```

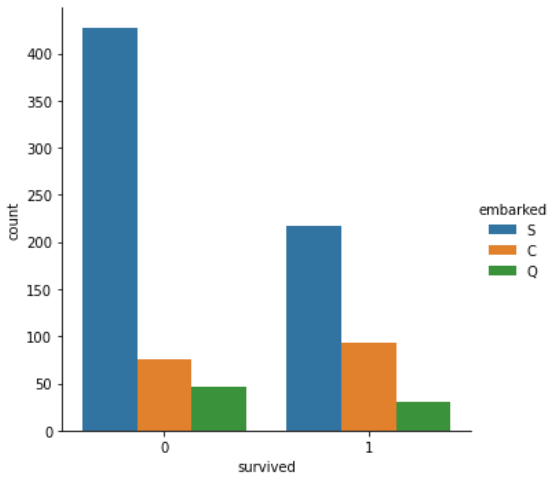
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\SHELEN~1\AppData\Local\Temp/ipykernel_14024/1858746412.py in <module>
----> 1 d1 = df[df["type"] == "TV Show"]
      2 d2 = df[df["type"] == "Movie"]
      3
      4 col = "year_added"
      5

NameError: name 'df' is not defined
```

In [1]:

```
import seaborn as sns
titanic = sns.load_dataset('titanic')
#Use this plot to draw insights
sns.catplot(x ="survived", hue ="embarked", kind ="count", data = titanic)
```

Out[1]:

```
<seaborn.axisgrid.FacetGrid at 0x18159a6d310>
```



In [3]:

```
import seaborn as sns
titanic = sns.load_dataset('titanic')
titanic
```

Out[3]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns

In [ ]:

```
import seaborn as sns
titanic = sns.load_dataset(“titanic”)
# Hint: Use countplot to see majority class in both survived and non-survived people.
```