

▼ Import libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

plt.rcParams['figure.figsize'] = 5,3
```

Seaborn is a popular Python data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Here are some of the commonly used plots in Seaborn:

1. Scatter Plot:

- `sns.scatterplot(x, y, data)`: Plot points in a 2D space.

2. Line Plot:

- `sns.lineplot(x, y, data)`: Plot a line connecting data points in 2D space.

3. Bar Plot:

- `sns.barplot(x, y, data)`: Plot categorical data with rectangular bars.

4. Count Plot:

- `sns.countplot(x, data)`: Show the counts of observations in each category.

5. Box Plot:

- `sns.boxplot(x, y, data)`: Display distribution of data based on the interquartile range.

6. Violin Plot:

- `sns.violinplot(x, y, data)`: Similar to a box plot, but it shows the kernel density estimation of the underlying distribution.

7. Strip Plot:

- `sns.stripplot(x, y, data)`: Scatter plot based on one categorical variable.

8. Swarm Plot:

- `sns.swarmplot(x, y, data)`: Similar to a strip plot but with points adjusted so they don't overlap.

9. Histogram:

- `sns.histplot(x, data)`: Plot univariate distribution of data.

10. KDE (Kernel Density Estimation) Plot:

- `sns.kdeplot(x, data)`: Plot the kernel density estimate of a univariate dataset.

11. Heatmap:

- `sns.heatmap(data)`: Plot rectangular data as a color-encoded matrix.

12. Pair Plot:

- `sns.pairplot(data)`: Plot pairwise relationships in a dataset.

13. Joint Plot:

- `sns.jointplot(x, y, data)`: Plot a joint distribution between two variables with marginal distributions.

14. LM Plot (Linear Model Plot):

- `sns.lmplot(x, y, data)`: Plot data and the regression model fit.

15. Factor Plot:

- `sns.factorplot(x, y, data)`: Plot categorical data using a number of different estimators.

16. Relational Plot:

- `sns.relplot(x, y, data)`: Plot data with a semantic mapping.

17. Categorical Plot:

- `sns.catplot(x, y, data)`: Plot data that has one of the variables as categorical.

These are just some of the commonly used plots in Seaborn. Each of these plots can be further customized to suit specific requirements using various parameters available in Seaborn functions.

▼ Load dataset

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

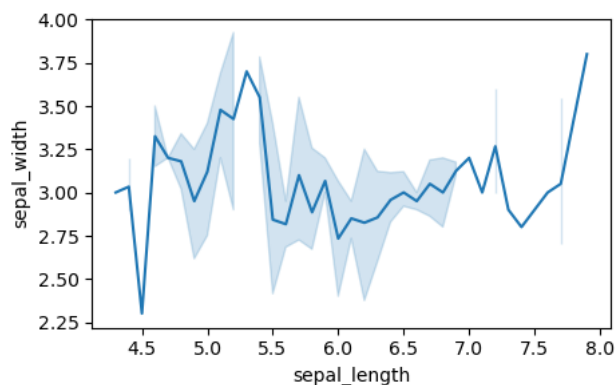
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
iris.dtypes
```

```
sepal_length    float64
sepal_width     float64
petal_length     float64
petal_width     float64
species         object
dtype: object
```

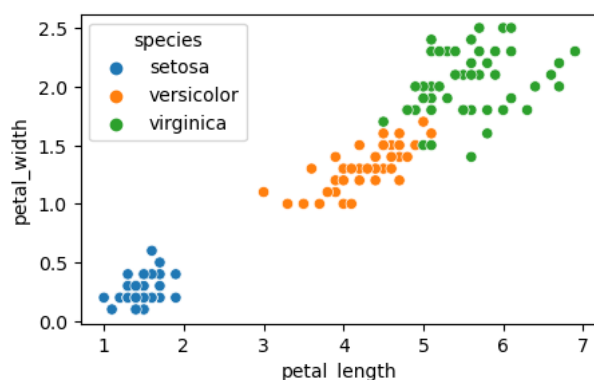
▼ 1. Line plot

```
plt.figure(figsize = (5,3))
sns.lineplot(x='sepal_length',y='sepal_width',data = iris)
plt.show()
```



▼ 2. Scatter plot

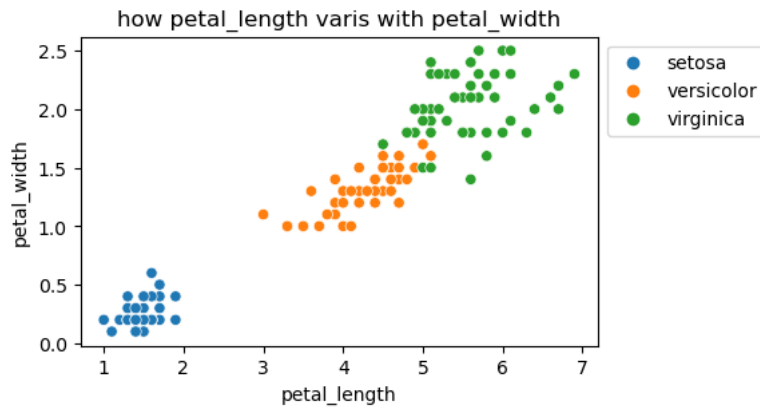
```
sns.scatterplot(x='petal_length',y='petal_width',data = iris,hue = 'species')
plt.show()
```



▼ Back ground ==> white, dark, whitegrid, darkgrid, ticks

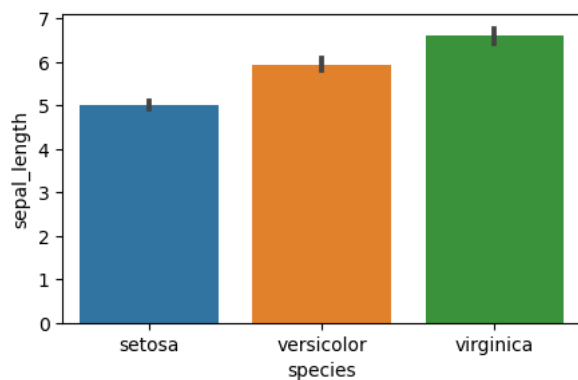
```
plt.figure(figsize = ( 5,3))

sns.scatterplot(x='petal_length',y='petal_width',data = iris,color='red',hue = 'species')
plt.title('how petal_length varis with petal_width')
plt.legend(bbox_to_anchor=(1,1))
plt.show()
```

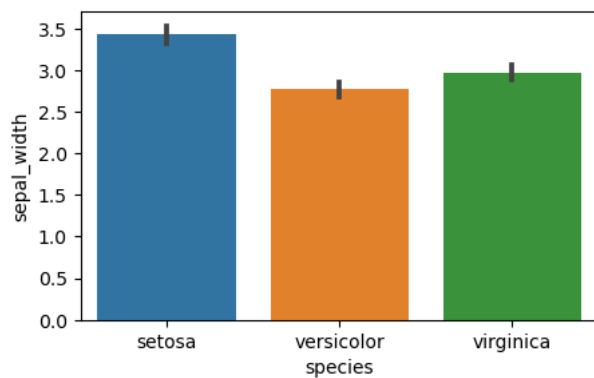


▼ 3. Bar plot

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

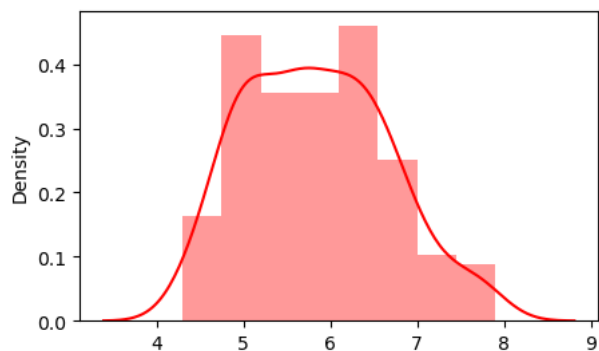


```
sns.barplot(x='species',y='sepal_width', data = iris)
plt.show()
```

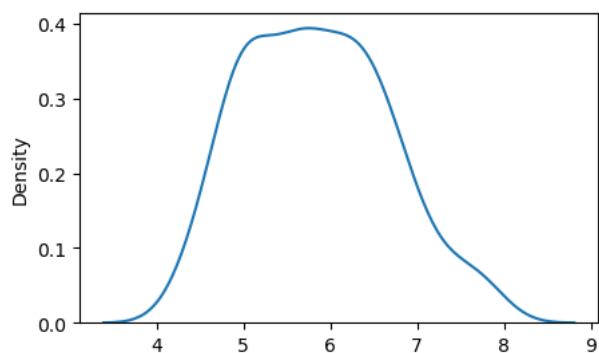


▼ 4.Hist plot

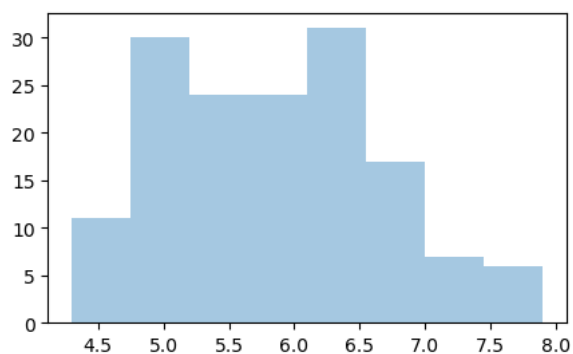
```
sns.distplot(x=iris['sepal_length'],color = 'red')
plt.show()
```



```
sns.distplot(x=iris['sepal_length'],hist = False)  
plt.show()
```

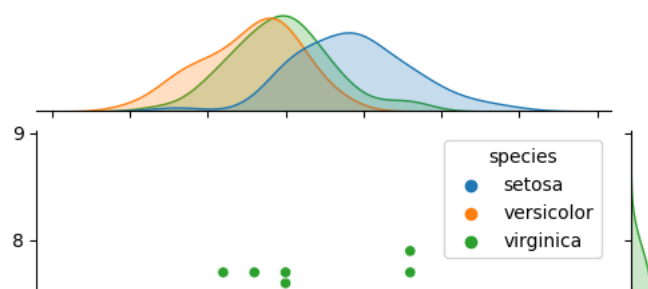


```
sns.distplot(x=iris['sepal_length'],kde = False)  
plt.show()
```

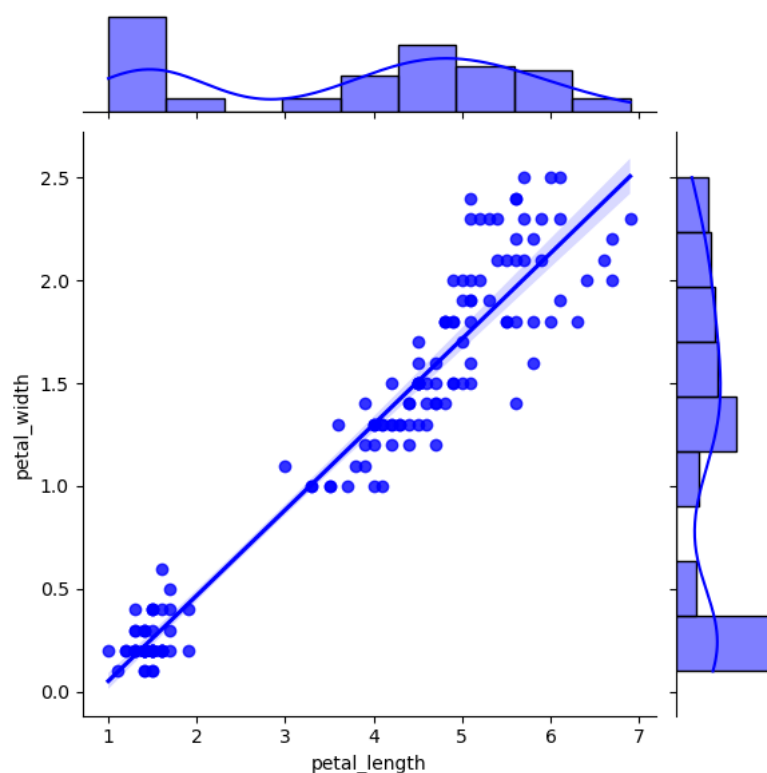


▼ 5. Join plot

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

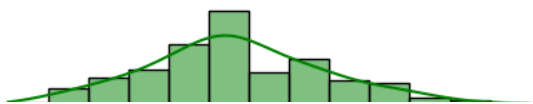


```
sns.jointplot(x='petal_length',y='petal_width',data=iris,color='blue',kind='reg')
plt.show()
```



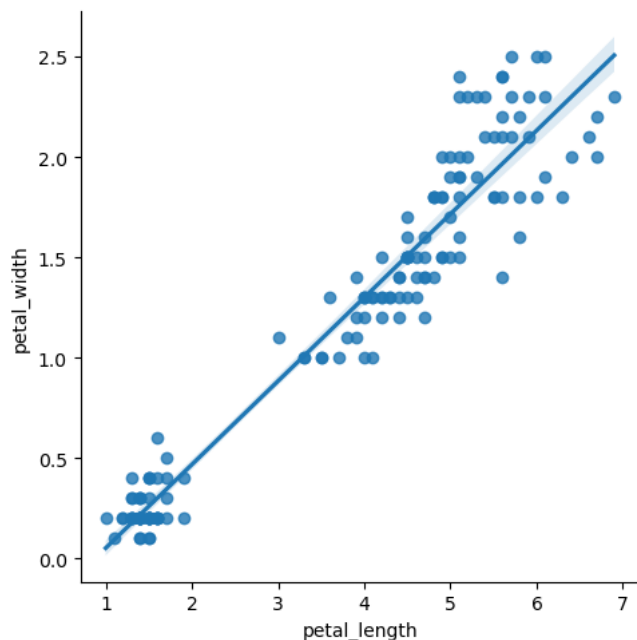
Unsupported Cell Type. Double-Click to inspect/edit the content.

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

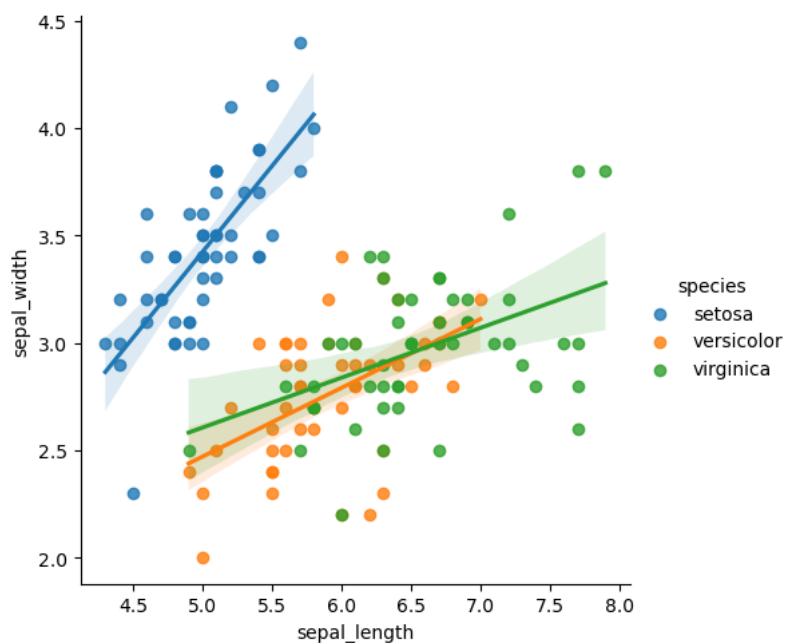


6. Lm plot

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

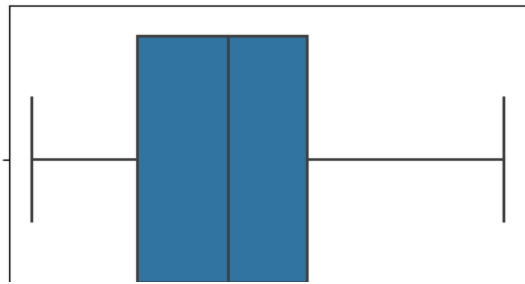


```
sns.lmplot(x='sepal_length',y='sepal_width',data=iris,fit_reg = True,hue = 'species')
plt.show()
```

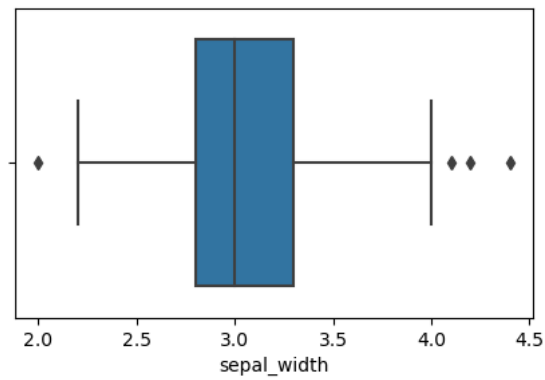


7.Box plot

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

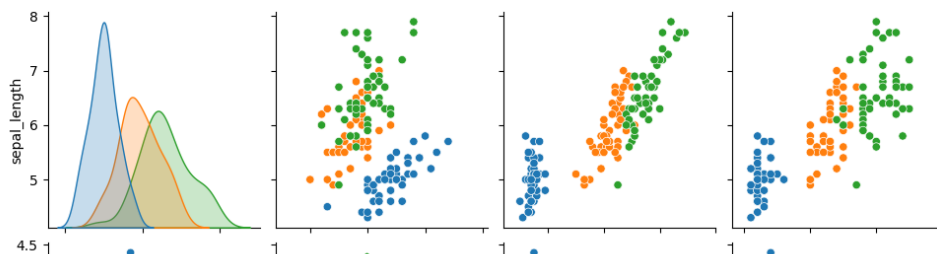


```
sns.boxplot(iris['sepal_width'])  
plt.show()
```



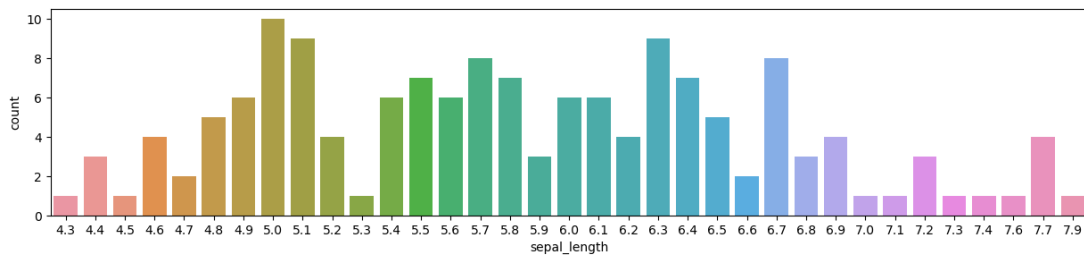
▼ 8.Pair plot

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



9.Count plot

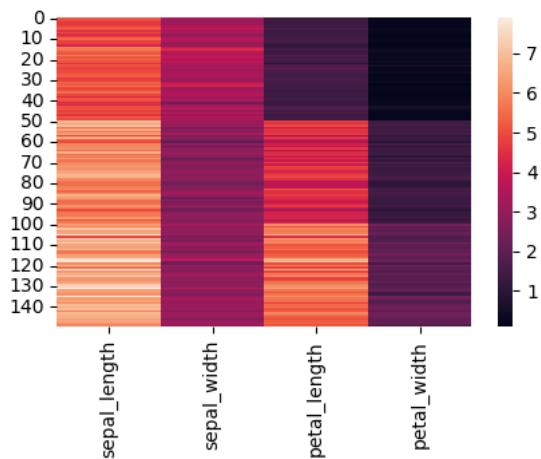
```
plt.figure(figsize=(15,3))
sns.countplot(iris['sepal_length'])
plt.show()
```



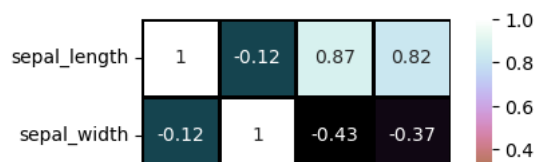
10.Heat map

```
iris_num=iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
```

```
sns.heatmap(iris_num)
plt.show()
```



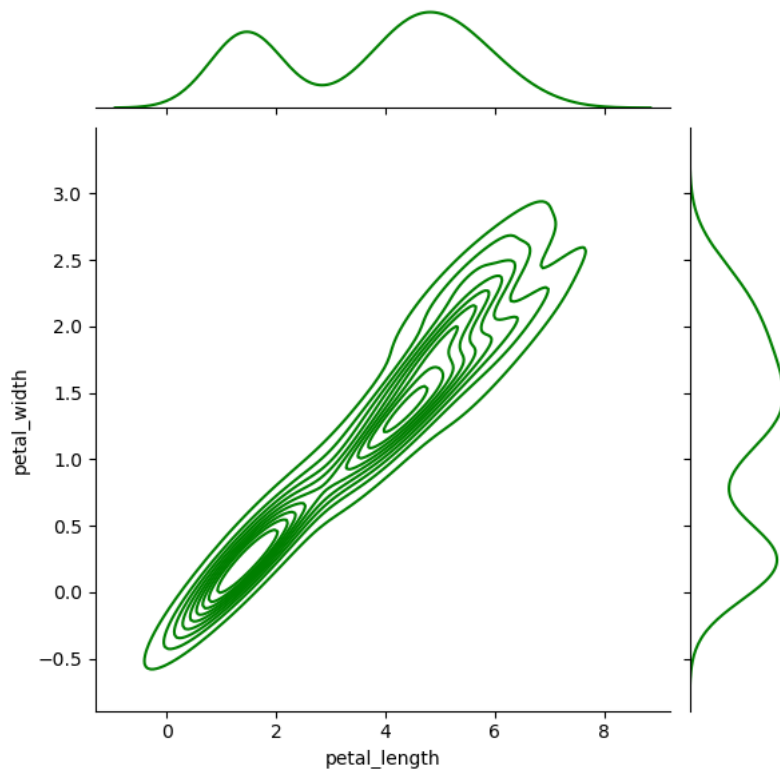
```
sns.heatmap(iris.corr(),annot = True, cmap = 'cubehelix' , linewidth = 1, linecolor = 'k', square = True , mask = False)
plt.show()
```

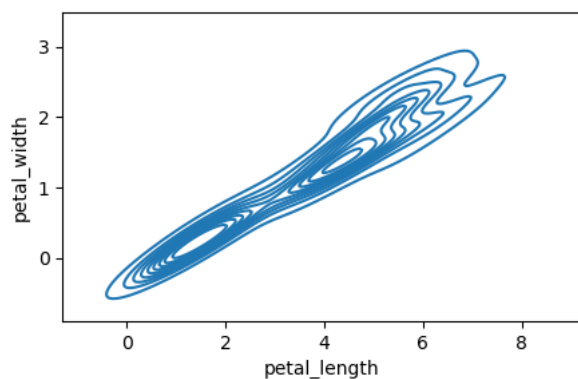
11. KDE plot (Kernel Density Estimation)

```
plt.figure(figsize=(10,4))
sns.jointplot(x='petal_length',y='petal_width',data=iris,kind='kde',color='green')
plt.show()
```

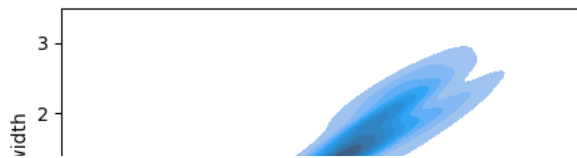
<Figure size 1000x400 with 0 Axes>



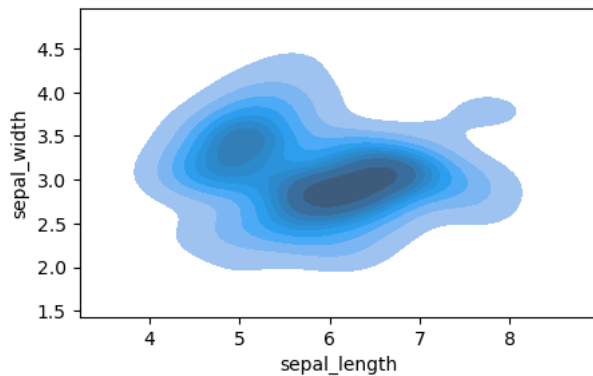
```
sns.kdeplot(x='petal_length',y='petal_width',data=iris)#,shade = True)
plt.show()
```



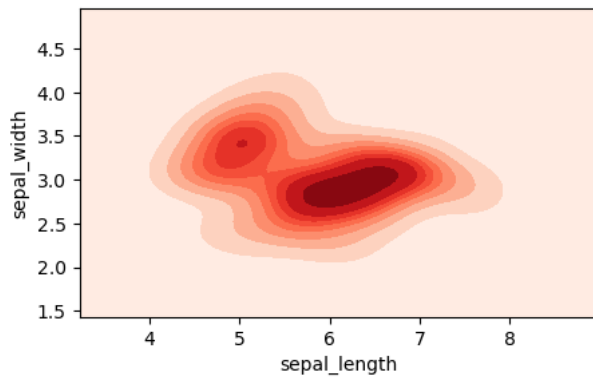
```
sns.kdeplot(x='petal_length',y='petal_width',data=iris,shade = True)
plt.show()
```



```
sns.kdeplot(x='sepal_length',y='sepal_width',data=iris,shade = True)
plt.show()
```

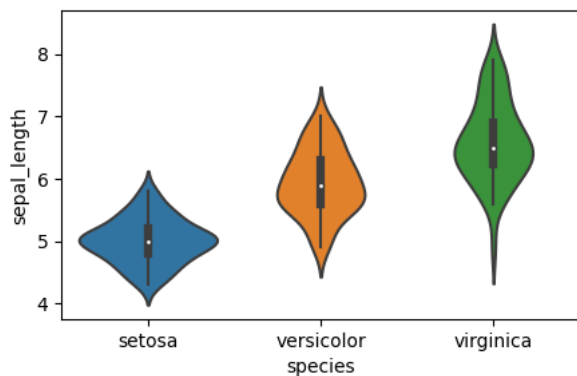


```
sns.kdeplot(x='sepal_length',y='sepal_width',data=iris,shade = True,shade_lowest=True,cmap='Reds')
plt.show()
```



12. Violinplot

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

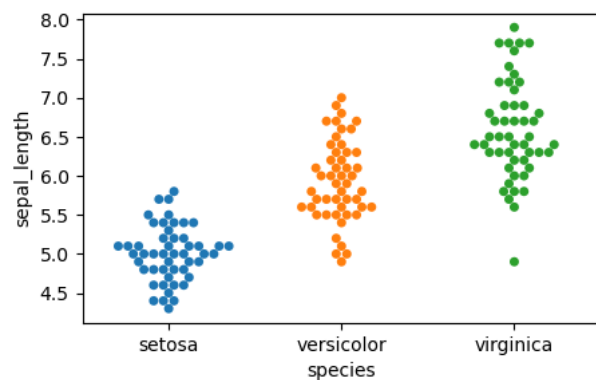


13. Swarm plot

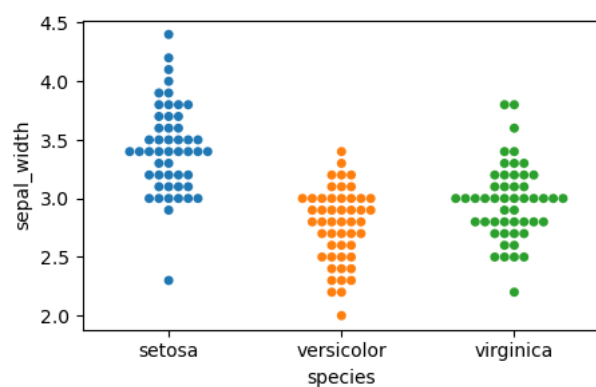
The `sns.swarmplot()` function is used to create a categorical scatter plot, which is particularly useful when you want to visualize the distribution of data points for different categories along a continuous axis. It helps you see the individual data points and how they are distributed within each category.

Keep in mind that when dealing with large datasets, `sns.swarmplot()` might not be the best choice, as the data points can overlap and make it hard to see the distribution. In such cases, you may want to consider using other types of plots or adjusting the plot settings to avoid overlapping.

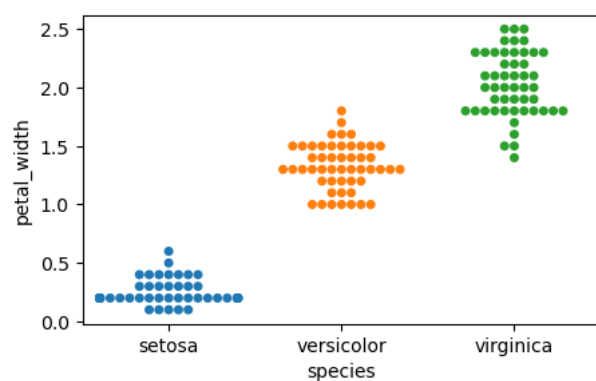
```
sns.swarmplot(x='species', y = 'sepal_length', data=iris)
plt.show()
```



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




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



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



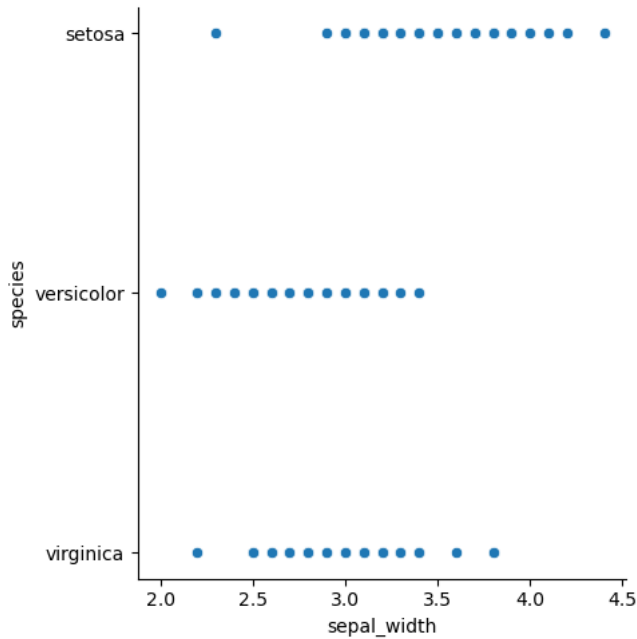
14.Relational Plot

al |  |

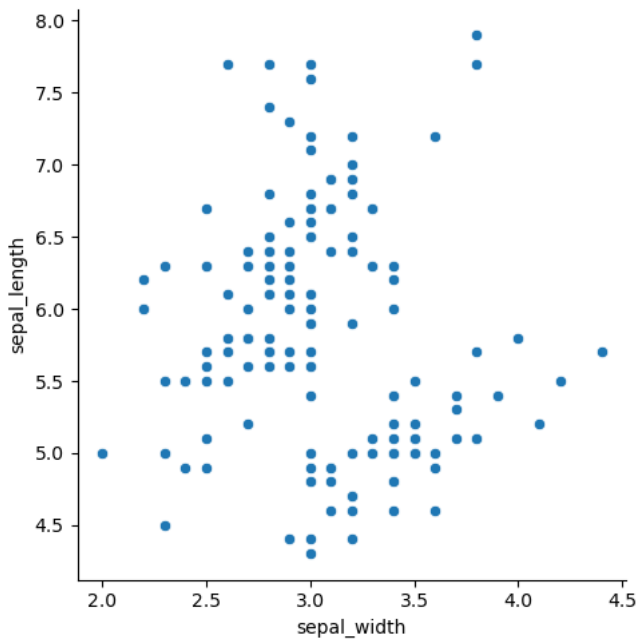
`sns.relplot()` is a function provided by the Seaborn library, which is a popular data visualization library in Python. It is used to create a scatter plot or line plot with optional additional information such as color-coded groups or a regression line.

al |  |

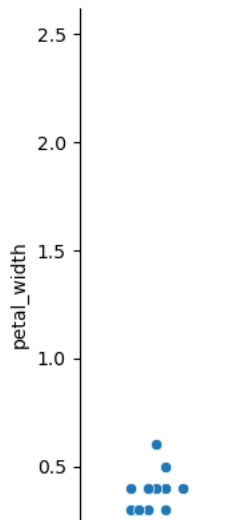
```
sns.relplot(x='sepal_width', y="species", data=iris, kind="scatter")
plt.show()
```



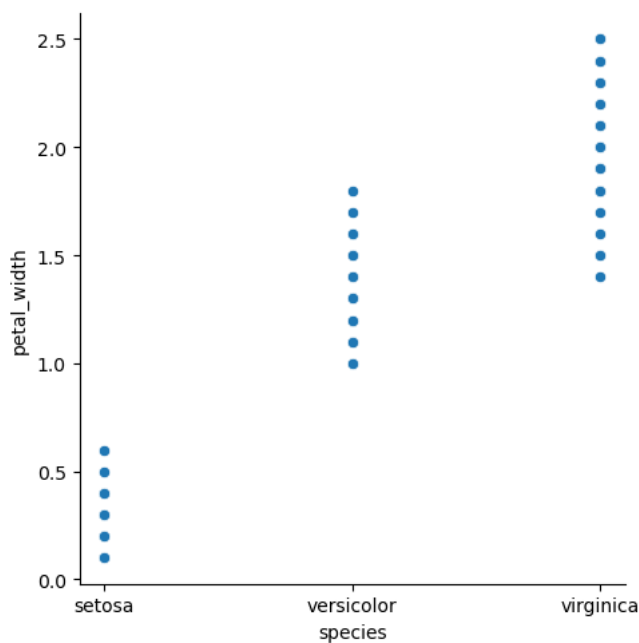
```
sns.relplot(x='sepal_width', y="sepal_length", data=iris, kind="scatter")
plt.show()
```



```
sns.relplot(x="petal_length", y="petal_width", data=iris)
plt.show()
```



```
sns.relplot(x="species", y="petal_width", data=iris)
plt.show()
```



15. Categorical Plot

In Seaborn, a categorical plot is a type of data visualization that is used to show the relationship between one categorical variable and one continuous or categorical variable. It is particularly useful when you want to compare the distribution or relationship between different groups in your data.

There are several types of categorical plots available in Seaborn, including:

1. `sns.catplot()` : This function is a general categorical plotting function in Seaborn. It can create various types of categorical plots, such as strip plots, swarm plots, box plots, violin plots, bar plots, and point plots.
2. `sns.barplot()` : It is used to create bar plots to show the central tendency (usually the mean) of a numeric variable for different categories.
3. `sns.countplot()` : This function is used to display the count of observations in each category in a bar plot.
4. `sns.boxplot()` : It creates box-and-whisker plots to show the distribution of data in different categories.
5. `sns.violinplot()` : It creates violin plots, which are a combination of a box plot and a kernel density plot.
6. `sns.stripplot()` : It shows individual data points as dots to visualize the distribution across categories.
7. `sns.swarmplot()` : It is similar to a strip plot, but the data points are adjusted so they do not overlap, providing a better view of the data distribution.

Here's a basic example of using `sns.catplot()` to create a bar plot:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset("tips")

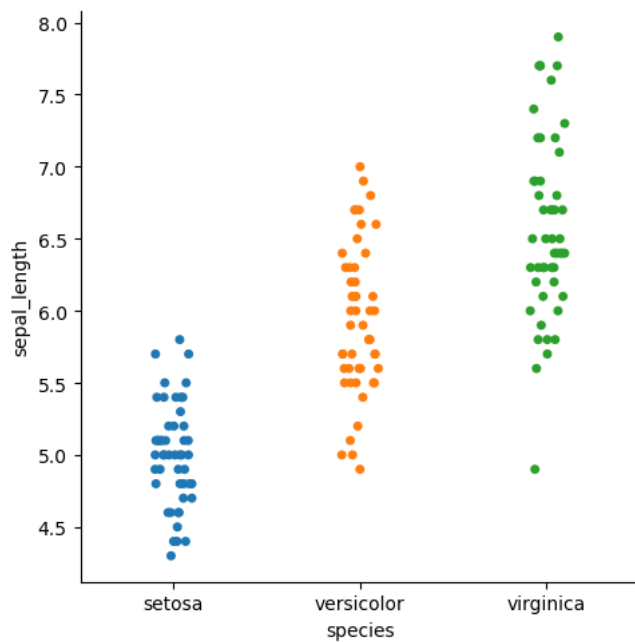
# Create a bar plot with 'day' on the x-axis and 'total_bill' on the y-axis
sns.catplot(x="day", y="total_bill", data=data, kind="bar")

# Show the plot
plt.show()
```

This example uses the "tips" dataset from Seaborn, which contains information about restaurant tips, including the day of the week and the total bill. The `kind="bar"` argument specifies that we want to create a bar plot.

You can explore other types of categorical plots by changing the `kind` parameter in `sns.catplot()` to "box", "violin", "swarm", etc., depending on the specific type of plot you want to create.

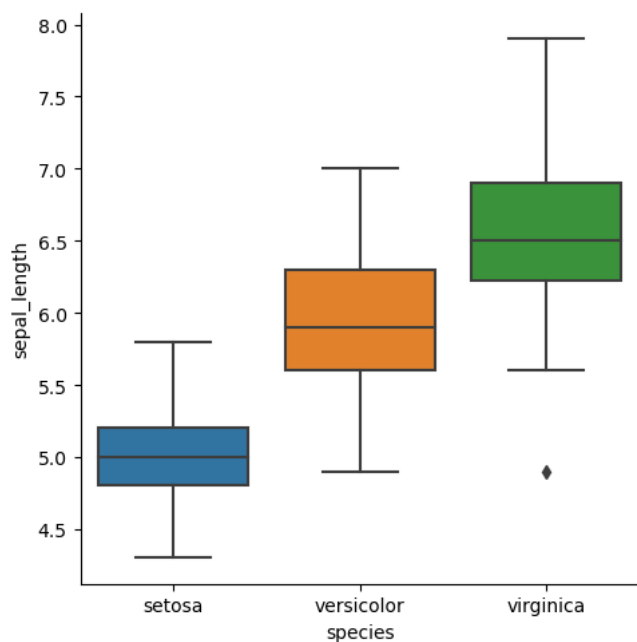
```
sns.catplot(x="species", y="sepal_length", data=iris, kind="bar")
plt.show()
```



```
sns.catplot(x="species", y="sepal_length", data=iris, kind="bar")
plt.show()
```

```
strip == "strip", "swarm", "box", "violin", "boxen", "point", "bar", or "count".
```

```
|
sns.catplot(x="species", y="sepal_length", data=iris, kind="box")
plt.show()
```



▼ 16. Factor Plot

`factorplot` is a function that was part of the Seaborn data visualization library, a popular Python library for creating statistical graphics based on Matplotlib. However, as of Seaborn version 0.11.0, released in September 2020, `factorplot` has been deprecated and replaced by the `catplot` function.

The main purpose of `factorplot` (and now `catplot`) is to create categorical plots, which allow you to visualize the distribution of a variable or the relationship between two variables based on categories. It provides a high-level interface to create various types of categorical plots like bar plots, count plots, point plots, box plots, etc., with a simple API.

Here is an example of how you can use `catplot` to create a bar plot:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Example data
tips = sns.load_dataset("tips")

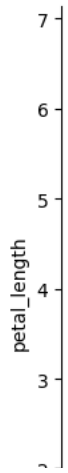
# Create a bar plot using catplot (formerly factorplot)
sns.catplot(x="day", y="total_bill", hue="sex", data=tips, kind="bar")
plt.show()
```

In this example, we use the `catplot` function to create a bar plot to visualize the relationship between the "day" and "total_bill" columns from the "tips" dataset, with different colors representing the "sex" category.

Keep in mind that the syntax or functionality might change in future versions of Seaborn, so it's always a good idea to check the official documentation for the most up-to-date information.

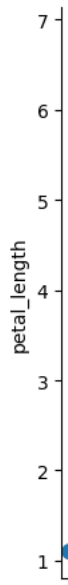
```
plt.figure(figsize=(35,25))
sns.factorplot(x='sepal_length', y='petal_length', data=iris)
plt.show()
```

<Figure size 3500x2500 with 0 Axes>



```
plt.figure(figsize=(35,25))
sns.factorplot(x='sepal_length',y='petal_length',data = iris)
plt.show()
```

<Figure size 3500x2500 with 0 Axes>



4.3.4.5.6.7.8.9.0.1.2.3.4.5.6.7.8.9.0.1.2.3.4.5.6.7.8.9.0.1.2.3.4.5.6.7.9

▼ 17. Strip Plot

A strip plot is a data visualization technique used to display individual data points along a one-dimensional axis. It is particularly useful when you have a relatively small dataset and want to observe the distribution and density of the data points.

In a strip plot, each data point is represented as a dot along the axis, with no overlapping points. This allows you to see the distribution of the data, identify potential outliers, and get a sense of how the data is spread out.

Strip plots are often used in exploratory data analysis and can be helpful for comparing groups or categories. They can be combined with other visualizations, such as box plots or violin plots, to provide a more comprehensive view of the data.

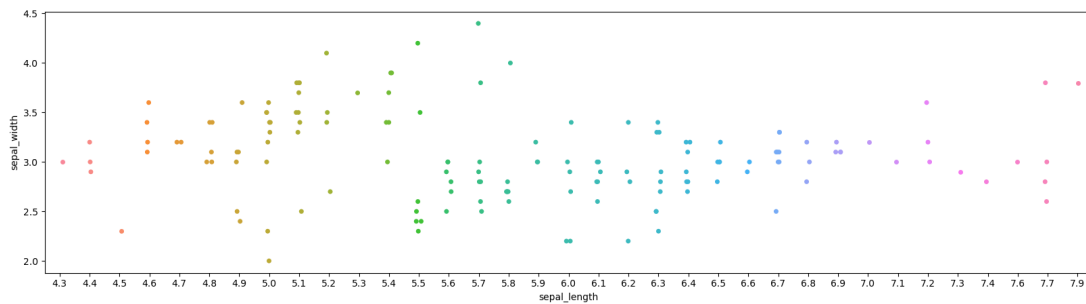
Here's a simple example of how a strip plot could look like:

```
|      |  
|      x      |  
|     x x     |  
|    x        x   |  
|   x          x   |  
|  x           x   |  
| x            x   |  
|x             |  
|              |
```

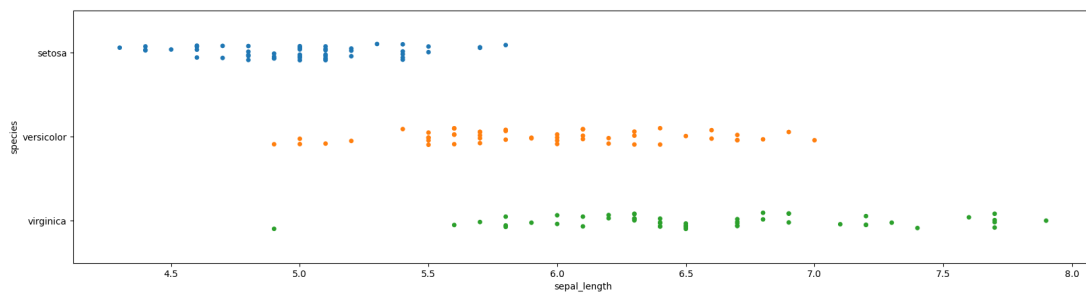

In this example, each "x" represents an individual data point along the axis.

Keep in mind that when the dataset is large, strip plots can become crowded, making it difficult to interpret the data accurately. In such cases, other visualization techniques like box plots or violin plots might be more appropriate to summarize the data effectively.

```
plt.figure(figsize= (20,5))
sns.stripplot(x='sepal_length', y='sepal_width', data=iris)
plt.show()
```

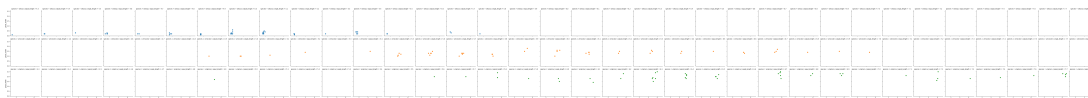


```
plt.figure(figsize= (20,5))
sns.stripplot(x='sepal_length', y='species', data=iris)
plt.show()
```



▼ 18. FacetGrid

```
F = sns.FacetGrid(iris , row = 'species' , col = 'sepal_length' , hue = 'species')
F = F.map(plt.scatter, 'petal_length', 'petal_width')
```



```
g = sns.FacetGrid (iris, row = 'species', col = 'sepal_length', hue = 'species')
g = g.map(plt.scatter, 'petal_length', 'petal_width' )
```

