DATA VISUALIZATION

Task 1 (Beginner) ~ by Anidipta Pal

➢ What is Data Visualization ?

Data visualization is the technique of translating information or data into a visual context, such as a map or graph, to make data easier for the human brain to understand.

Python is one of the most basic programming languages, other than R, for data visualization. It has libraries like Matplotlib, Seaborn, Plotly, and Bolek for visualization.

In this article, we will discuss Seaborn and Plotly.

Seaborn

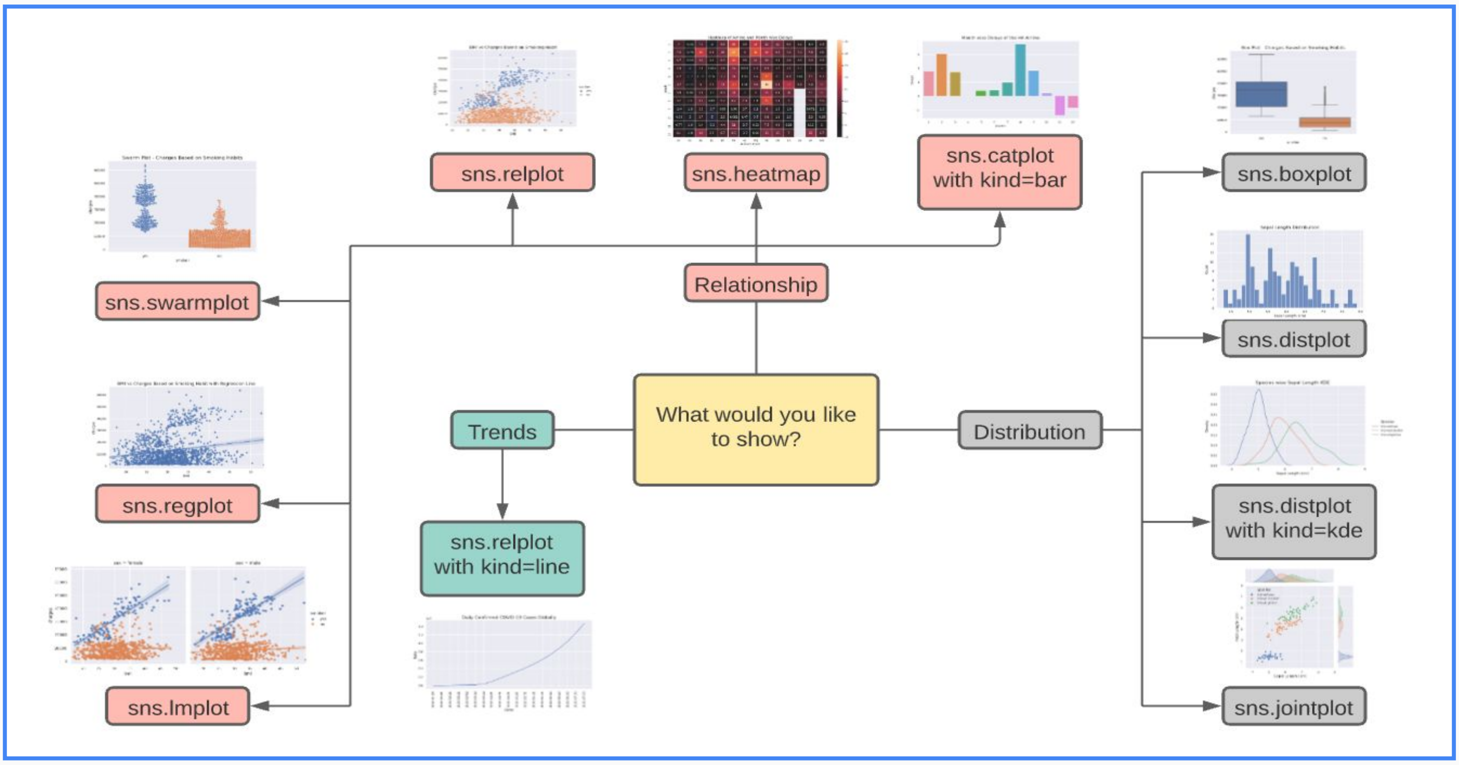
➢ Introduction

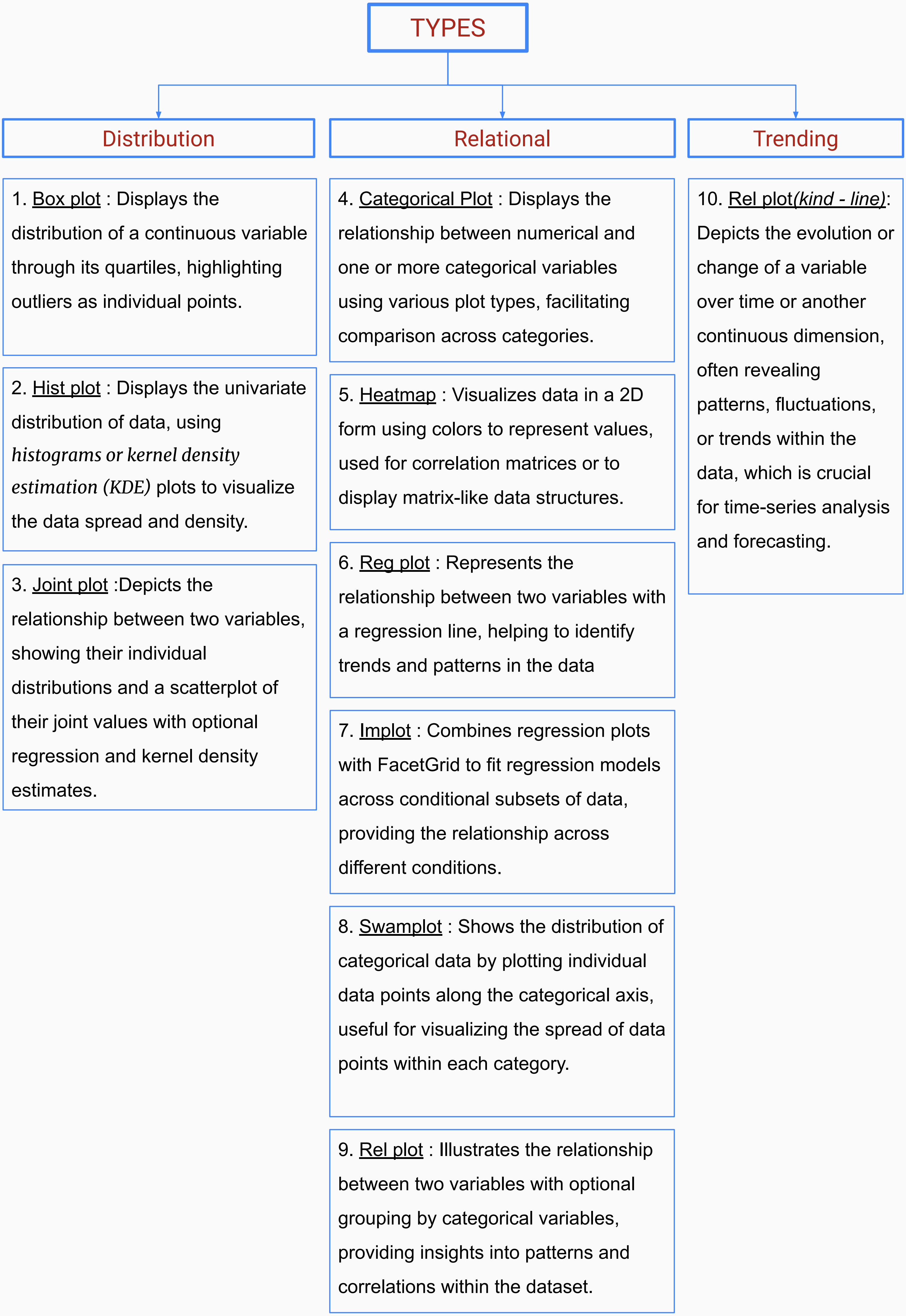
The development of Seaborn began in 2012 by *Michael Waskom* as part of his graduate thesis project. It is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive statistical graphics.

Seaborn is widely used in data analysis and visualization tasks, especially in the fields of data science and machine learning, due to its simplicity and flexibility.

➢ Graphs

Differents Types of Graphs





TYPES

Distribution

1. Box plot : Displays the distribution of a continuous variable through its quartiles, highlighting outliers as individual points.

2. Hist plot : Displays the univariate distribution of data, using *histograms or kernel density estimation (KDE)* plots to visualize the data spread and density.

3. Joint plot :Depicts the relationship between two variables, showing their individual distributions and a scatterplot of their joint values with optional regression and kernel density estimates.

Relational

4. Categorical Plot : Displays the relationship between numerical and one or more categorical variables using various plot types, facilitating comparison across categories.

5. Heatmap : Visualizes data in a 2D form using colors to represent values, used for correlation matrices or to display matrix-like data structures.

6. Reg plot : Represents the relationship between two variables with a regression line, helping to identify trends and patterns in the data

7. lmplot : Combines regression plots with FacetGrid to fit regression models across conditional subsets of data, providing the relationship across different conditions.

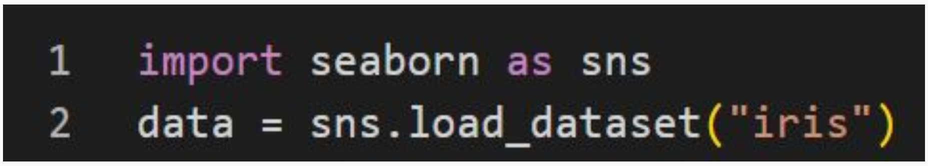
8. Swamplot : Shows the distribution of categorical data by plotting individual data points along the categorical axis, useful for visualizing the spread of data points within each category.

9. Rel plot : Illustrates the relationship between two variables with optional grouping by categorical variables, providing insights into patterns and correlations within the dataset.

Trending

10. Rel plot*(kind - line)*: Depicts the evolution or change of a variable over time or another continuous dimension, often revealing patterns, fluctuations, or trends within the data, which is crucial for time-series analysis and forecasting.

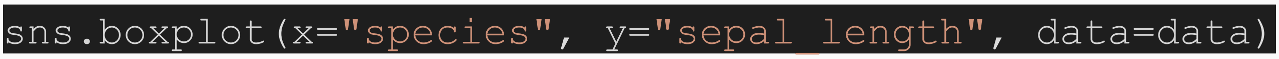
➢ **Code**



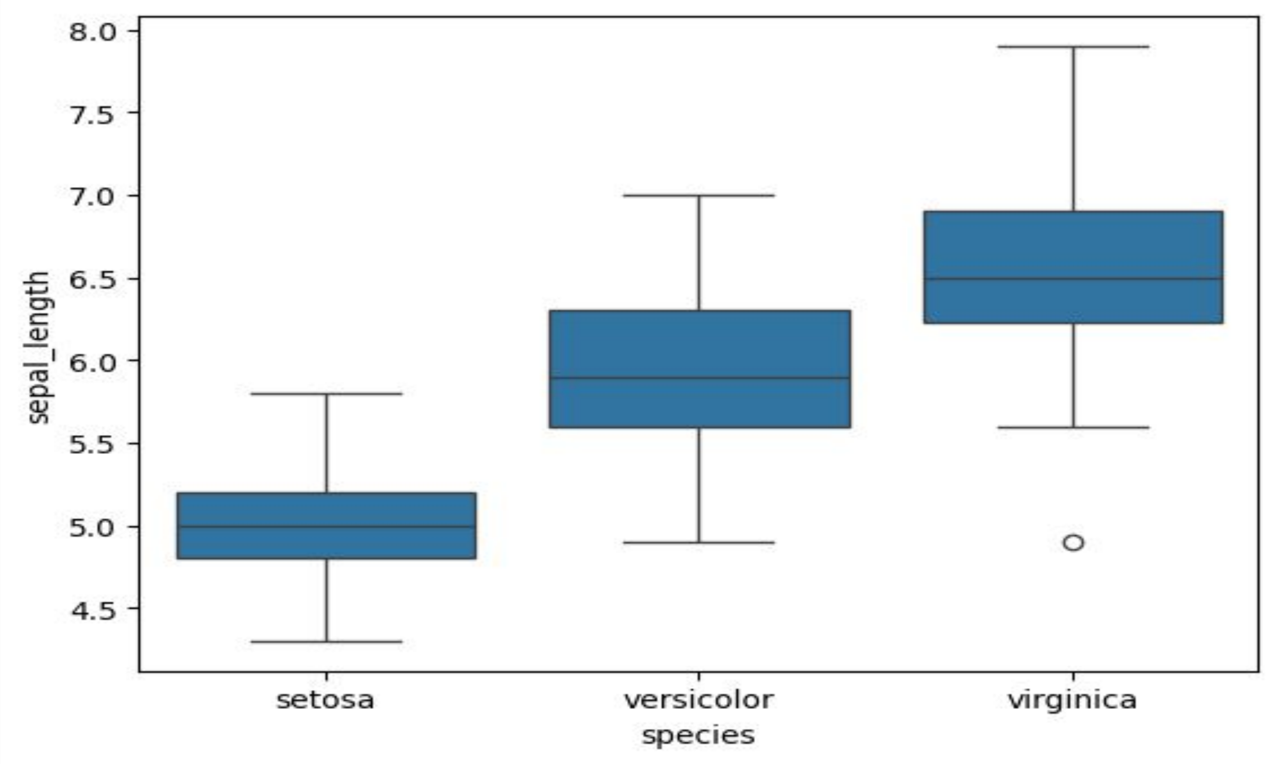
```python
import seaborn as sns
data = sns.load_dataset("iris")
```

Import Dataset and Library

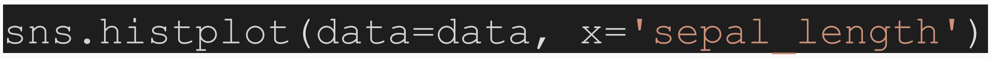
Box Plot



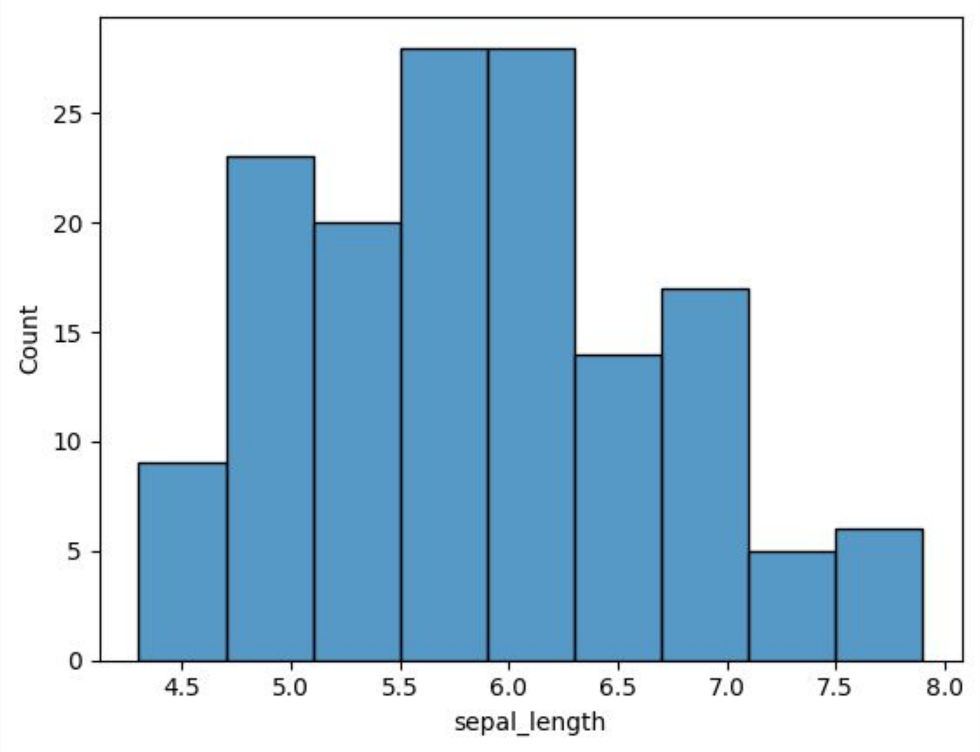
```python
sns.boxplot(x="species", y="sepal_length", data=data)
```



Hist Plot

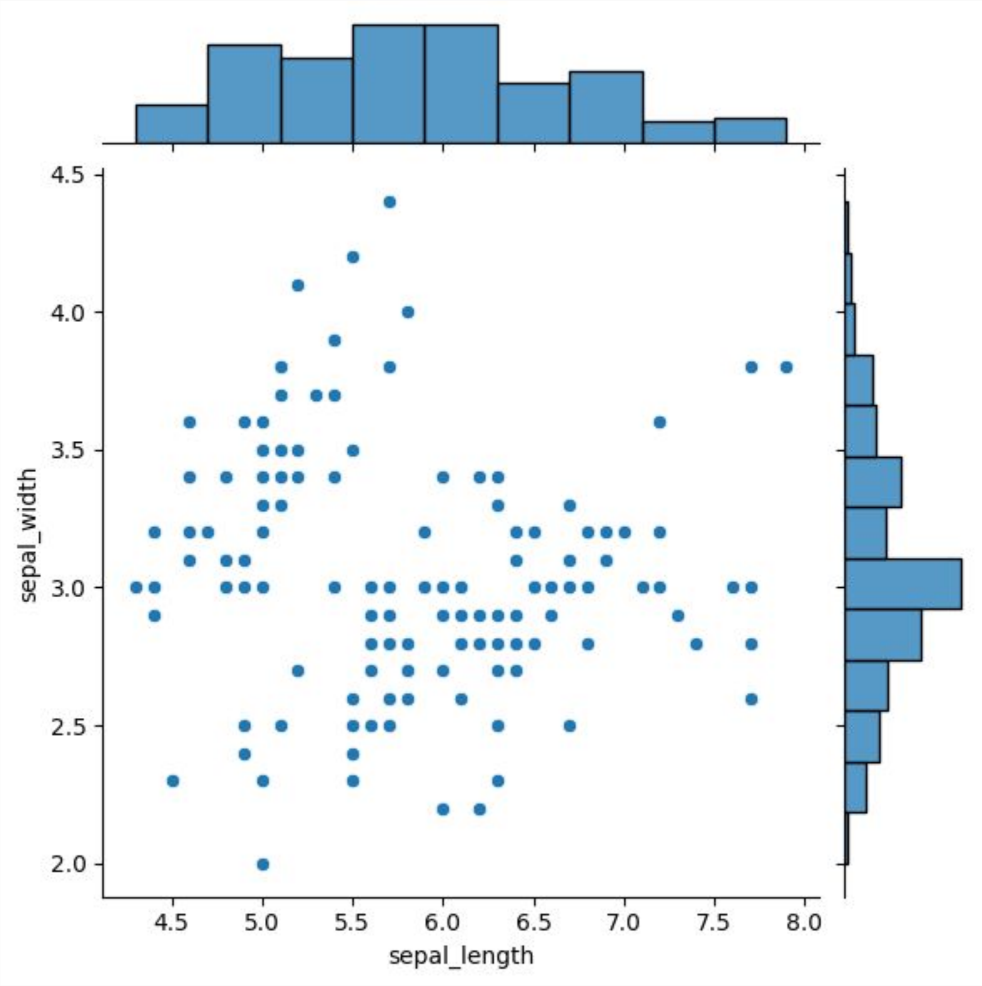


```python
sns.histplot(data=data, x='sepal_length')
```



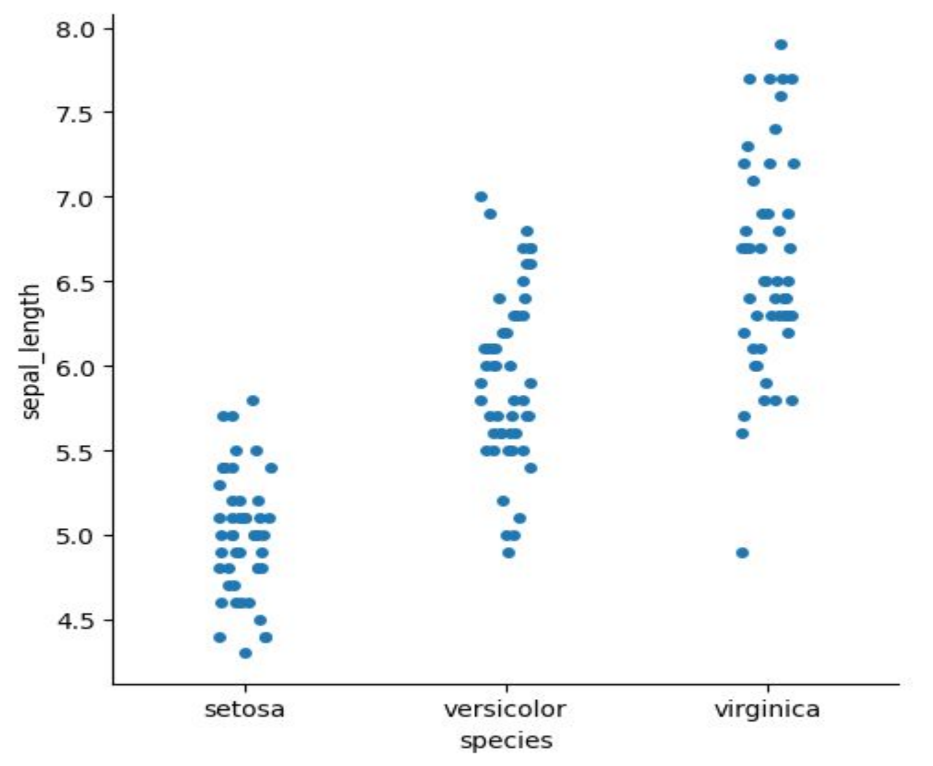
Joint Plot

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



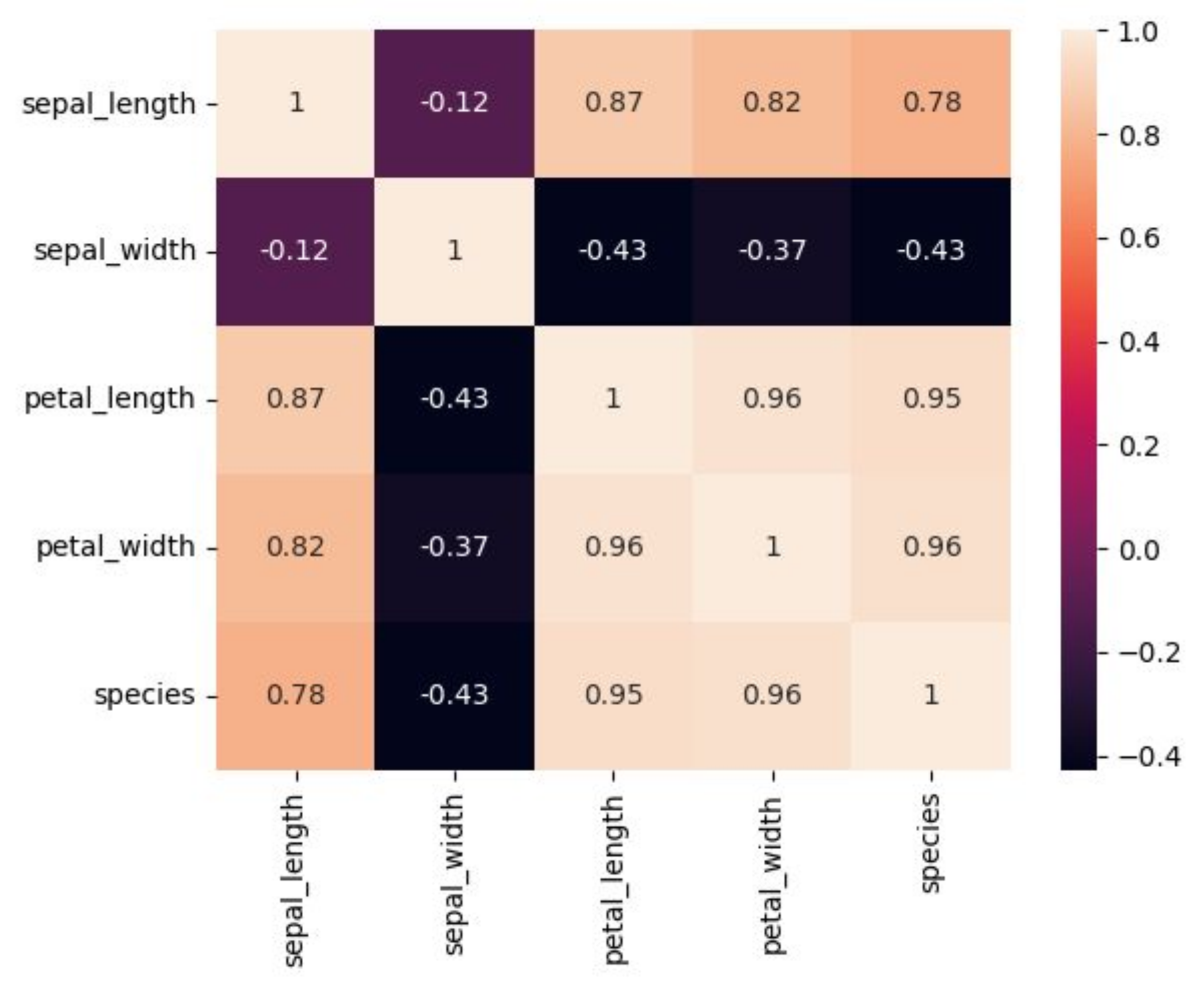
Categorical Plot

```
sns.catplot(x="species", y="sepal_length", data=data)
```



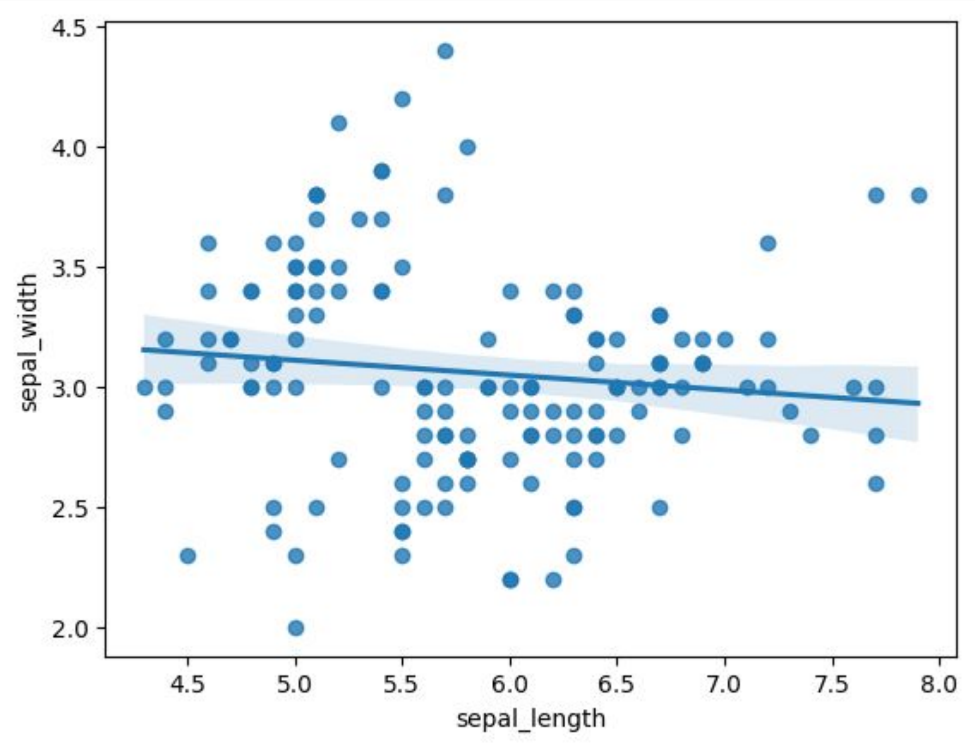
Heatmap

```
sns.heatmap(data=data.corr(), annot=True)
```



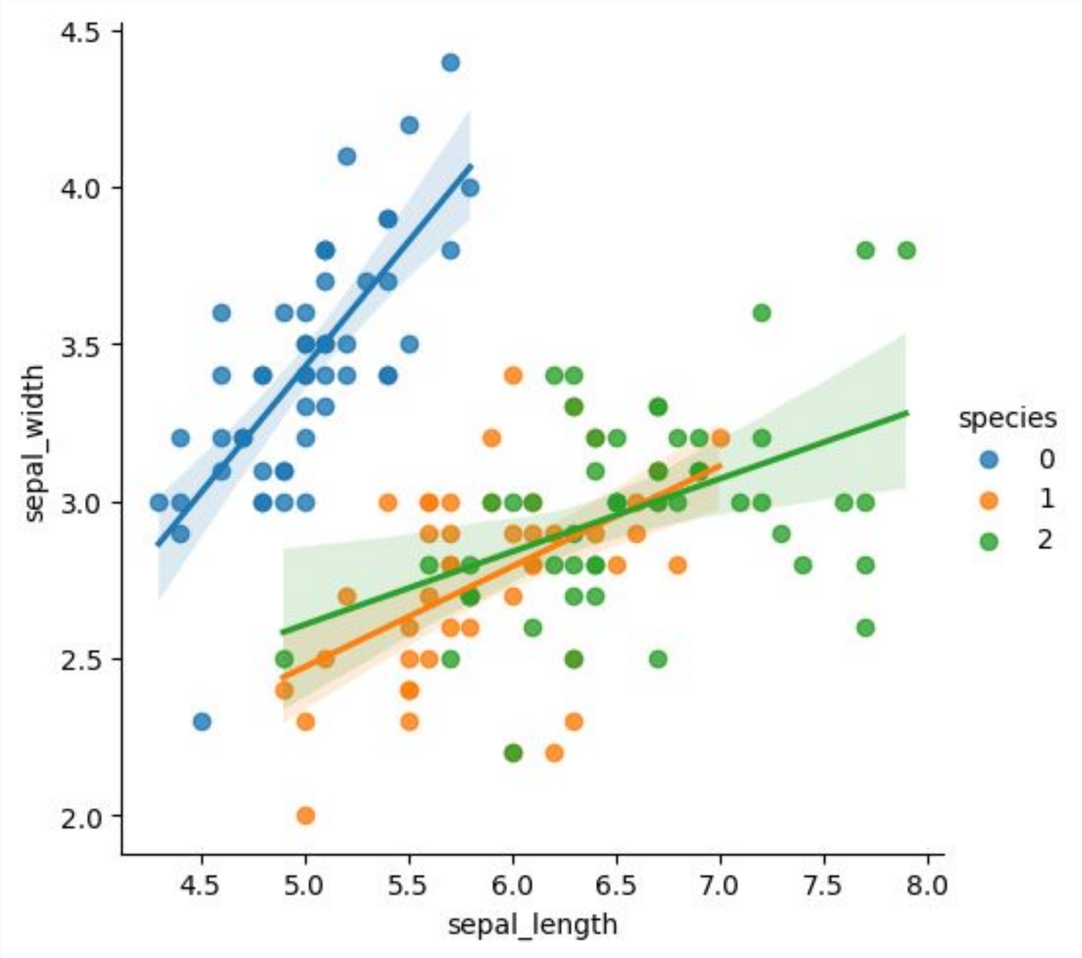
Reg Plot

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



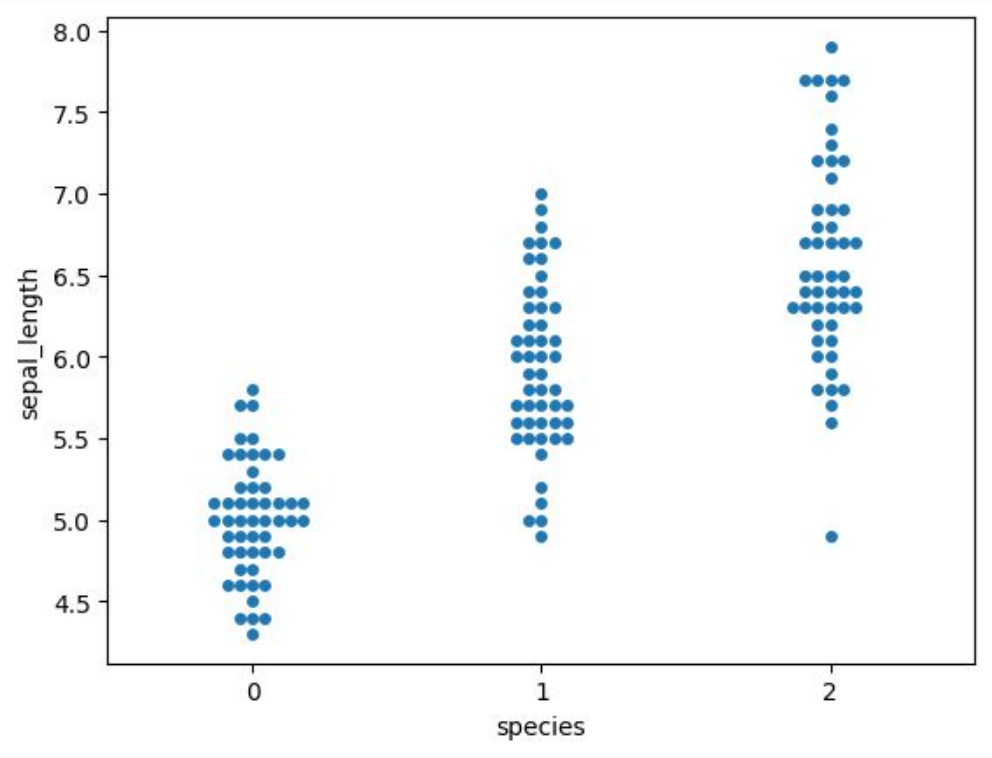
lmPlot

```
sns.lmplot(data=data, x='sepal_length', y='sepal_width',
hue='species')
```



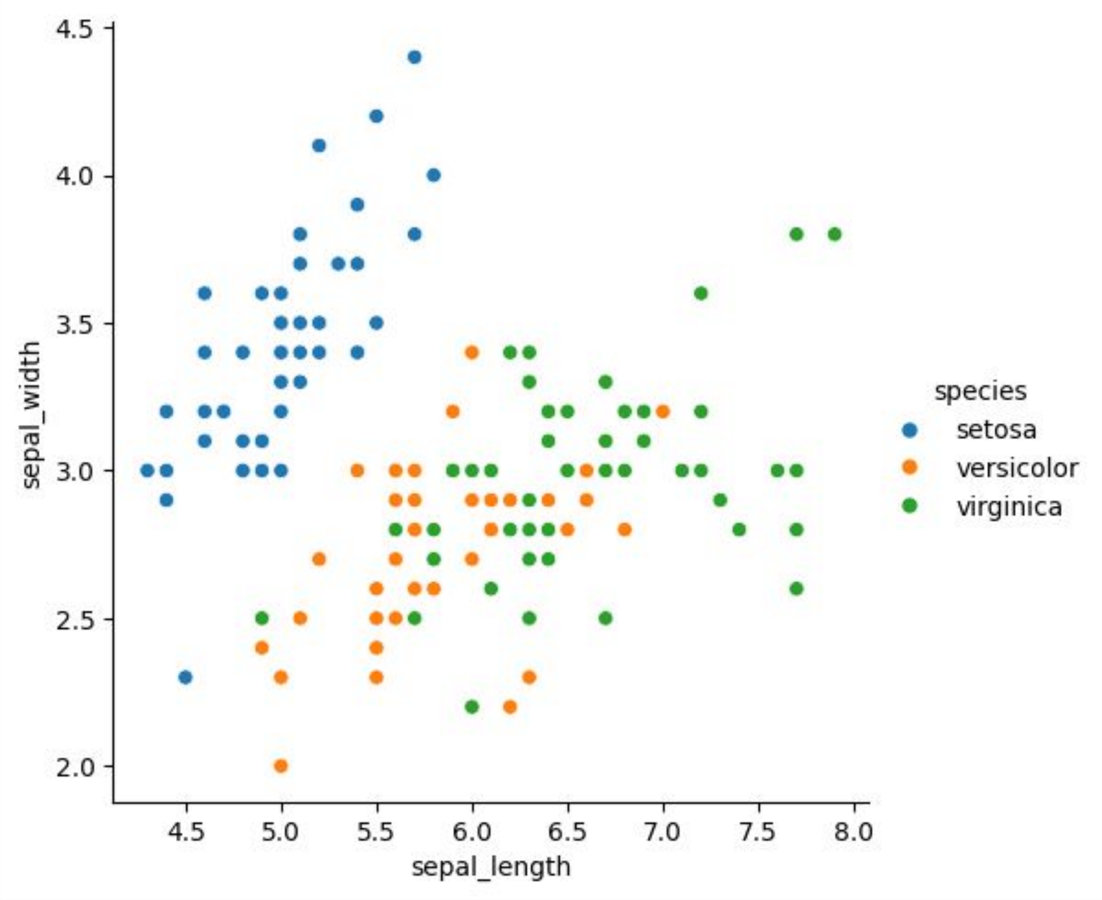
Swarm Plot

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



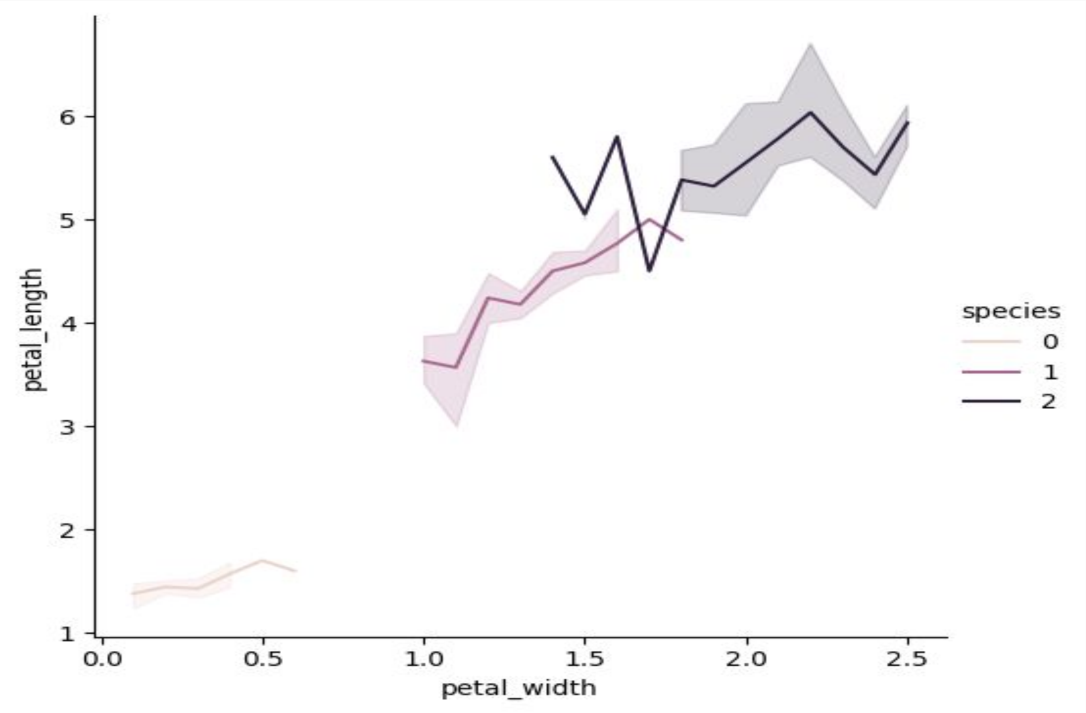
Rel Plot

```
sns.relplot(data=data,x='sepal_length',y='sepal_width',
hue='species')
```



Rel Plot (kind = “line”)

```
sns.relplot(data=data, x='petal_width', y='petal_length',
hue='species', kind='line')
```



Plotly

➢ Introduction

The development of Plotly began in 2012 by *Alex Johnson*, *Jack Parmer*, and *Chris Parmer* as a collaborative effort to create a versatile, interactive data visualization library.

Plotly offers a high-level interface for creating a wide variety of charts, including statistical graphics, 3D plots, and geographical maps. It excels in producing interactive visualizations that can be embedded in web applications and dashboards.

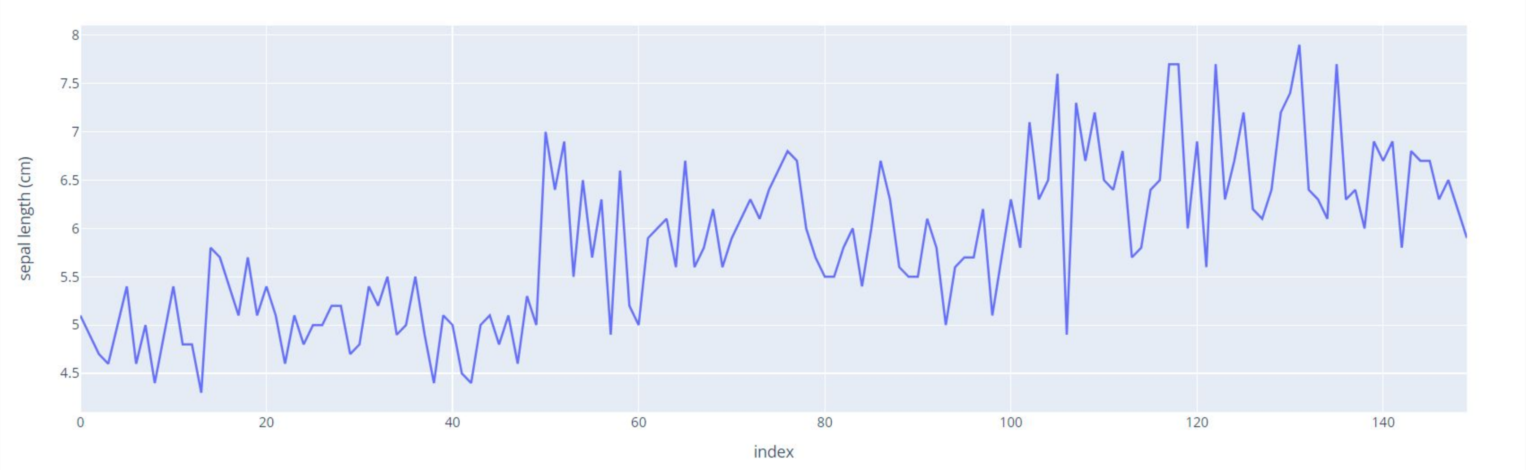
➢ Graphs

Import Dataset and Library

```
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data,columns=iris.feature_names)
iris_df['target'] = iris.target
```

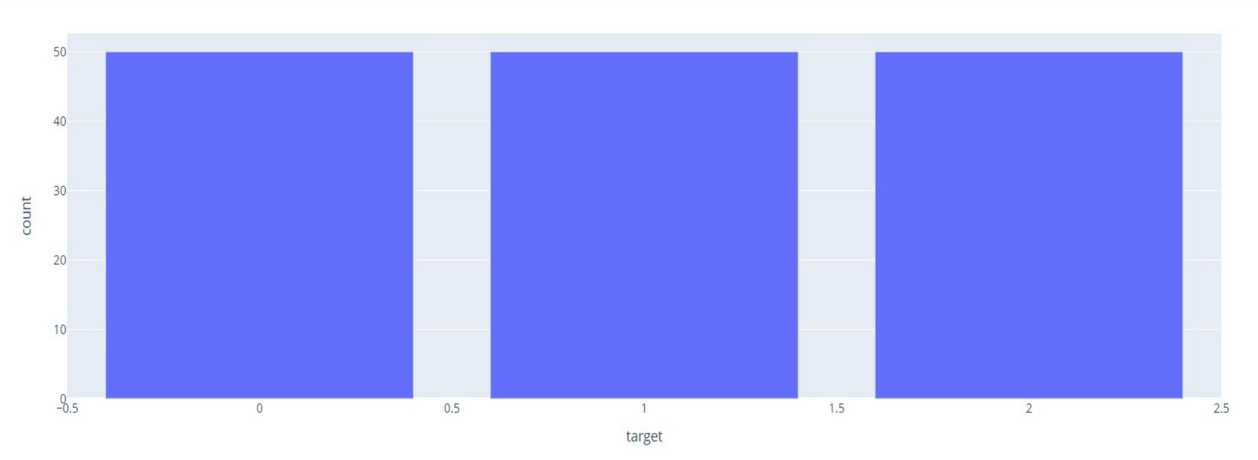
Line Plot → Ideal for showing trends over time.

```
fig = px.line(iris_df, x=iris_df.index, y='sepal length (cm)')
fig.show()
```



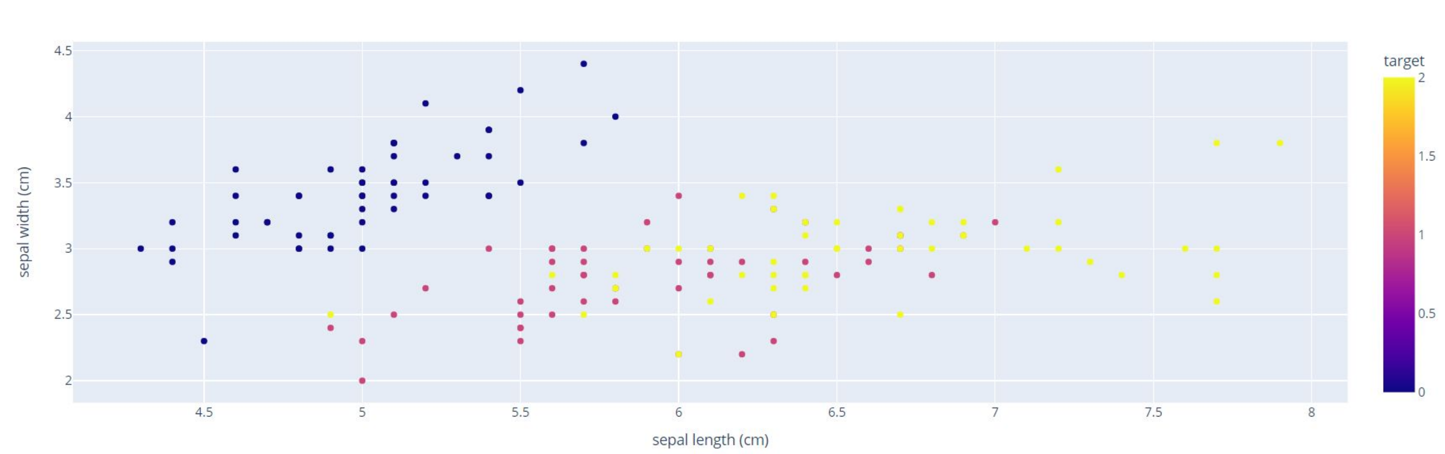
Bar Plot → for comparing quantities across categories.

```
fig = px.bar(
iris_df.groupby('target').size().reset_index(name='count'),
 x='target', y='count')
fig.show()
```



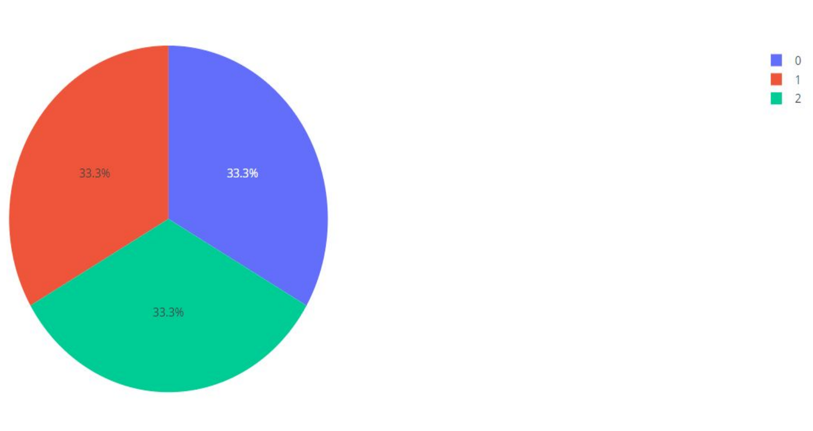
Scatter Plot → for identifying relationships between variables.

```
fig = px.scatter(iris_df, x='sepal length (cm)', y='sepal width
(cm)', color='target')
fig.show()
```



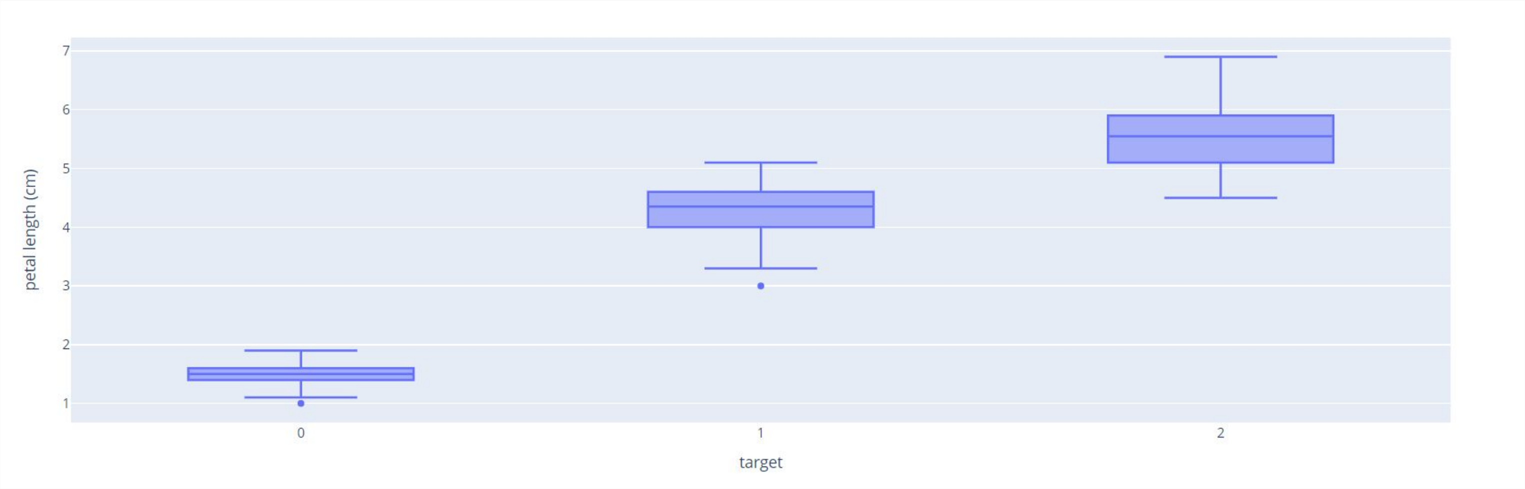
Pie Plot → for displaying proportions of a whole.

```
fig = px.pie(
iris_df.groupby('target').size().reset_index(name='count'),
values='count', names='target')
fig.show()
```



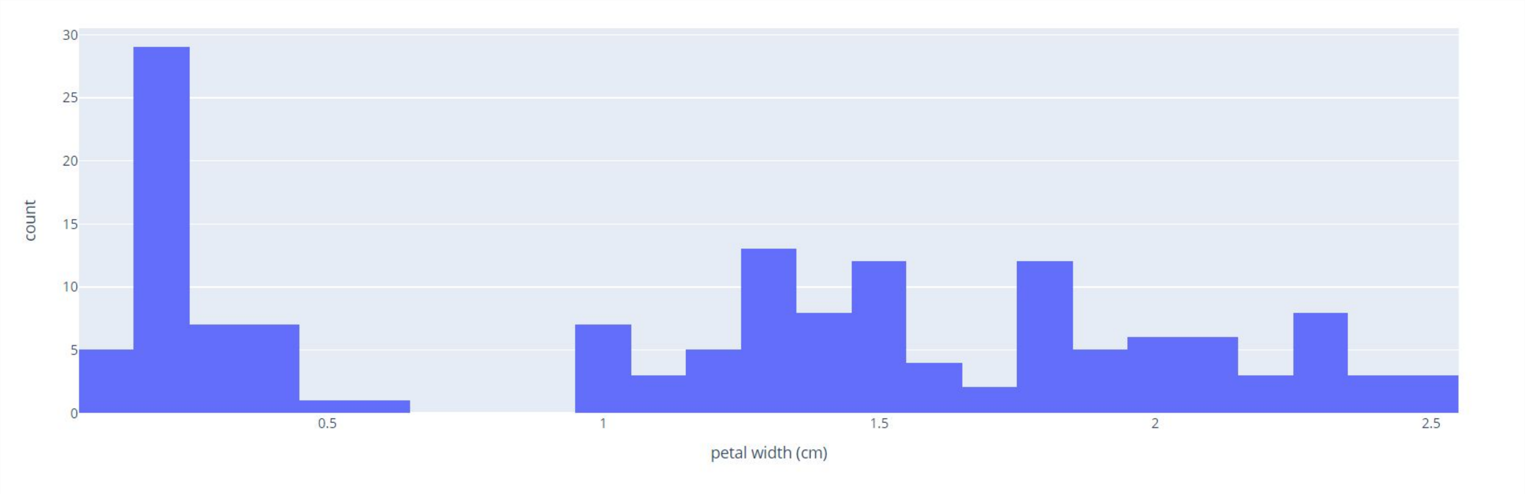
Box Plot → Show distribution of data

```
fig = px.box(iris_df, x='target', y='petal length (cm)')
fig.show()
```



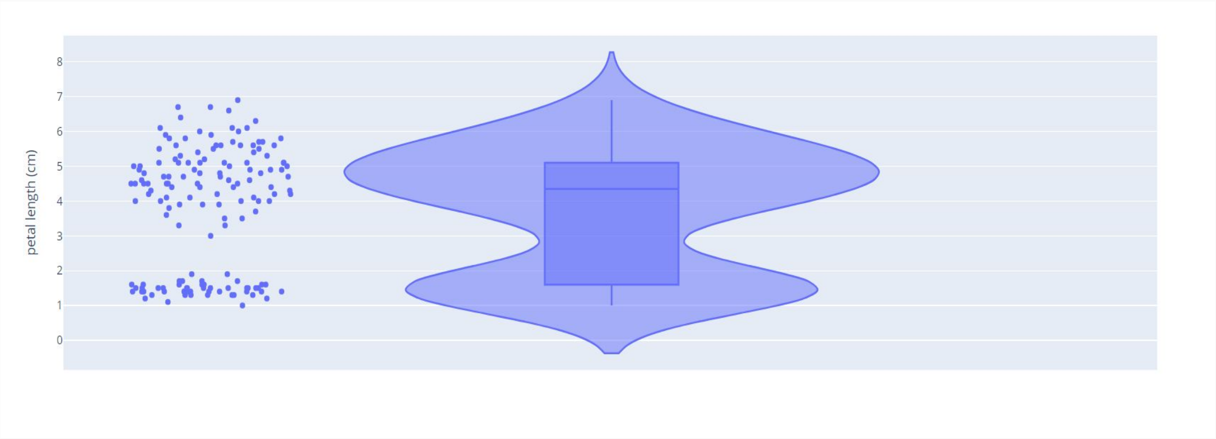
Histogram → Display the distribution of a dataset.

```
fig = px.histogram(iris_df, x='petal width (cm)', nbins=30)
fig.show()
```



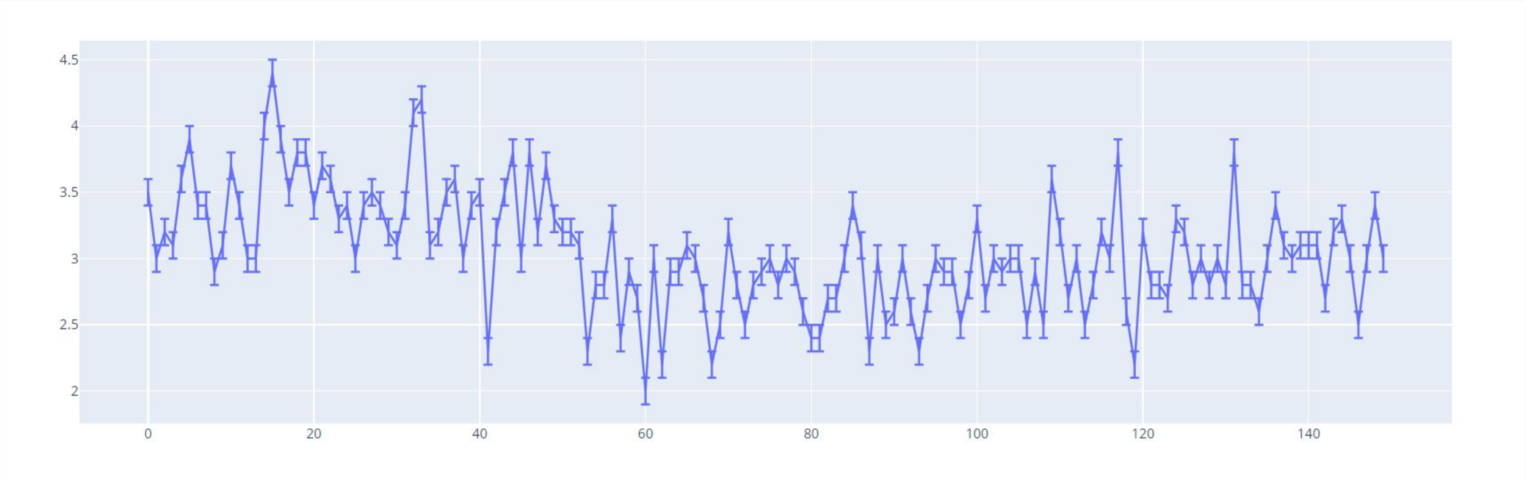
Violin Plot → Combine box plot and density plot to show data distribution.

```
fig = px.violin(iris_df, y='petal length (cm)', box=True)
fig.show()
```



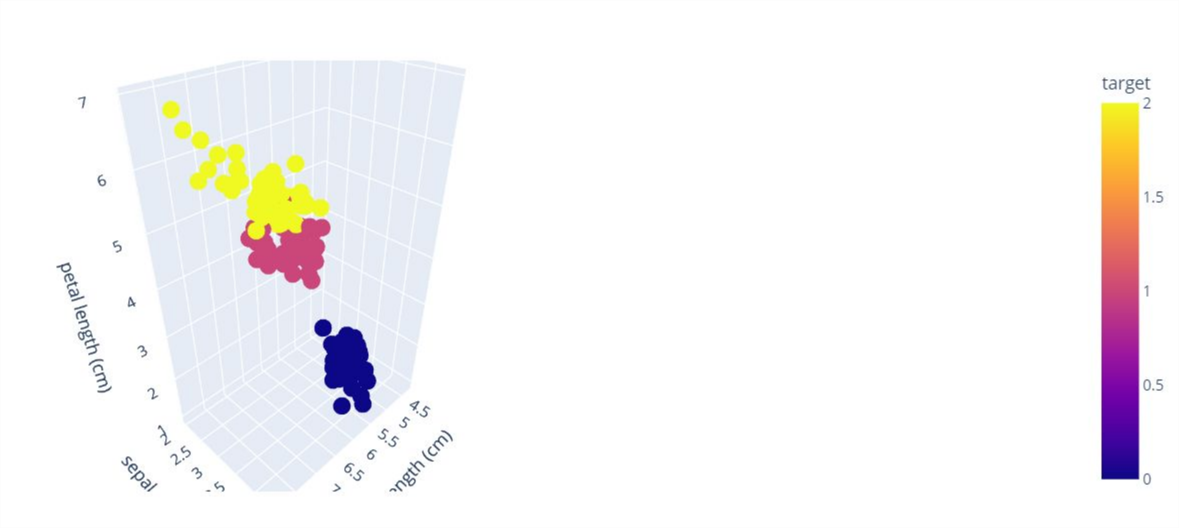
Error Bar → Show variability

```
fig = go.Figure(data=go.Scatter(x=iris_df.index, y=iris_df['sepal
width (cm)'], error_y=dict(type='constant', value=0.1)))
fig.show()
```



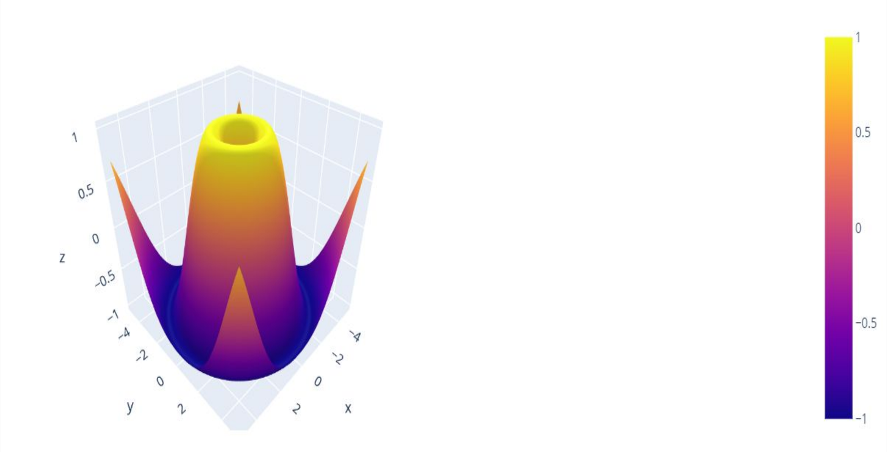
3D Scatter Plot → Extend scatter plots into three dimensions.

```
fig = px.scatter_3d(iris_df, x='sepal length (cm)', y='sepal
width (cm)', z='petal length (cm)',
color='target')
fig.show()
```



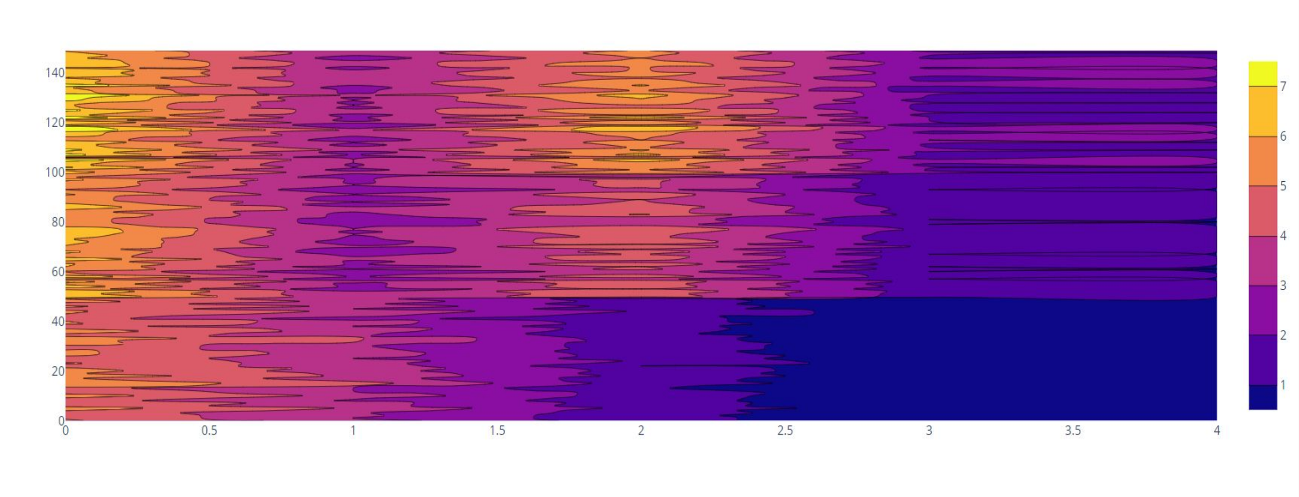
3D Surface Plot → Represent three-dimensional data as a surface.

```
x, y = np.linspace(-5, 5, 100), np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))
fig= go.Figure(data=
[go.Surface(z=Z, x=x, y=y)])
fig.show()
```



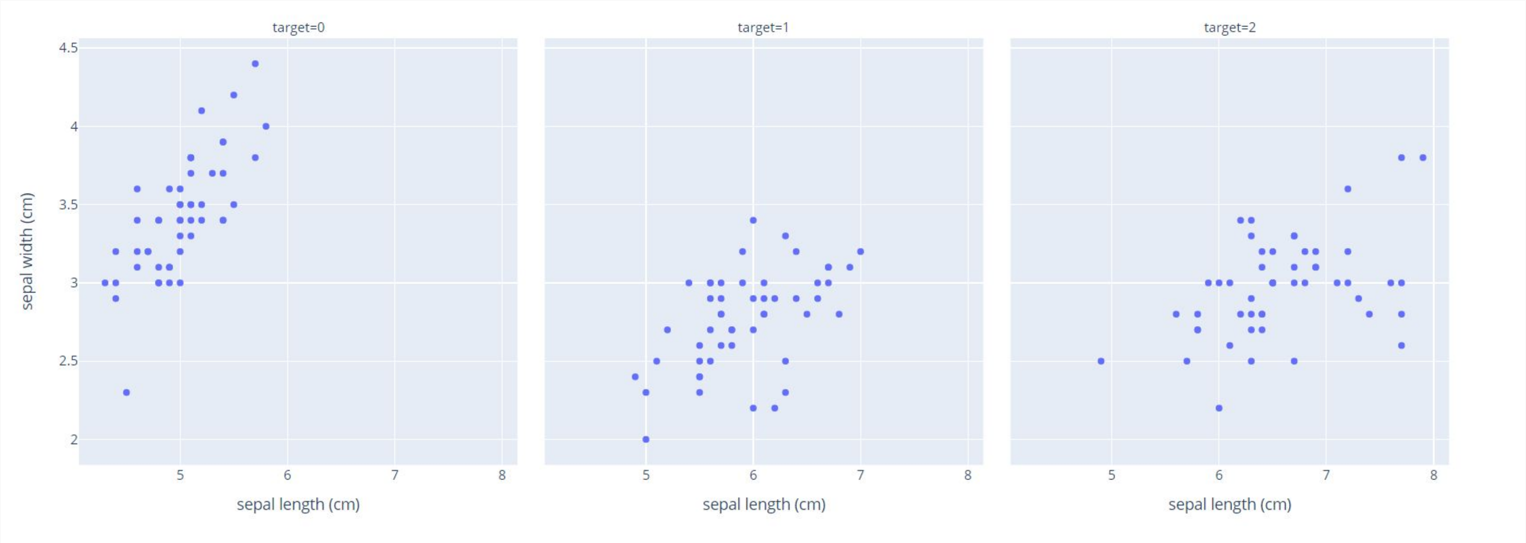
Contour Plot → Display contour lines to show 3D in 2D

```
fig = go.Figure(data=[go.Contour(z=iris_df.values)])
fig.show()
```



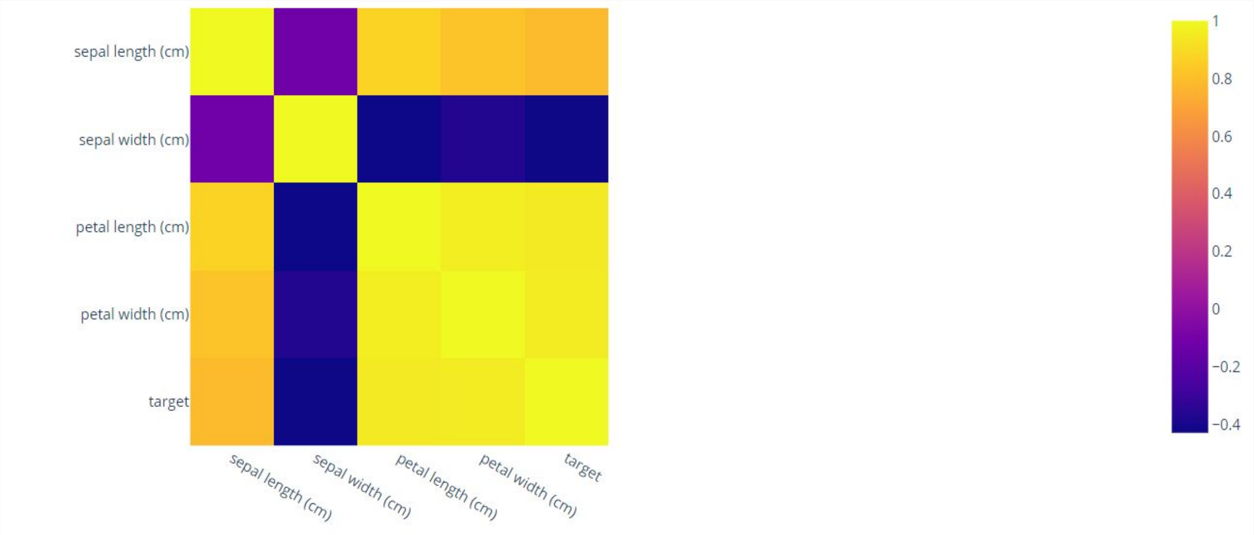
Facet Plot → Small multiples for comparing subsets of data.

```
fig = px.scatter(iris_df, x='sepal length (cm)', y='sepal width
(cm)', facet_col='target')
fig.show()
```



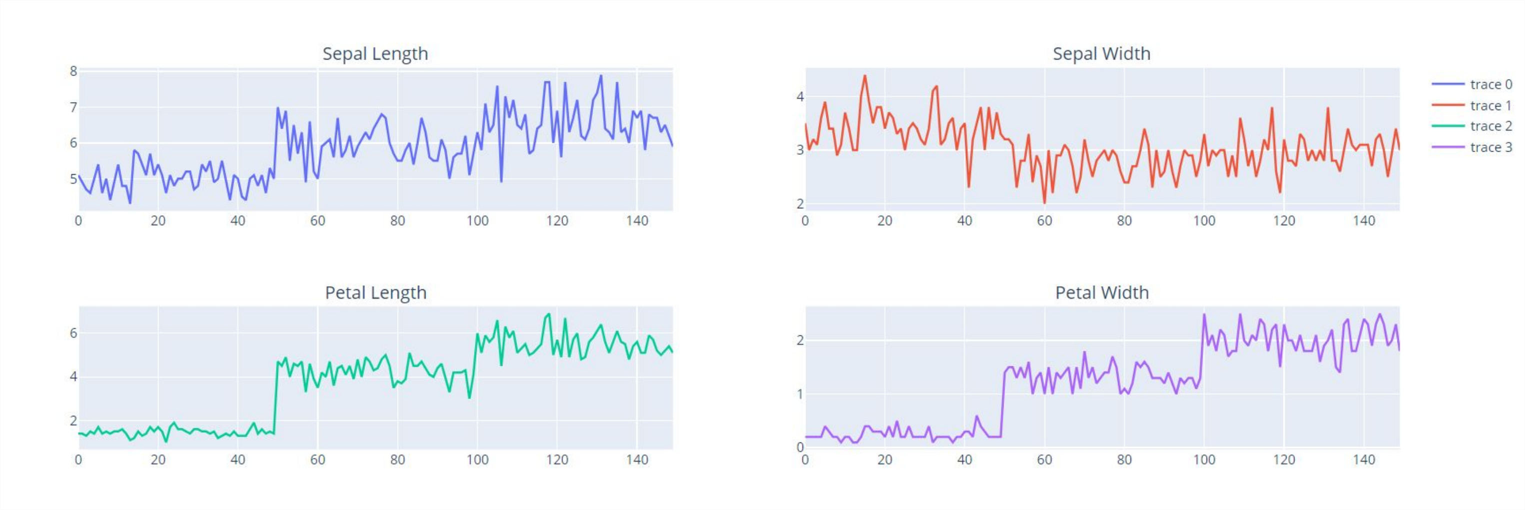
Heatmap → Represent data values with color coding.

```
fig = px.imshow(iris_df.corr())
fig.show()
```



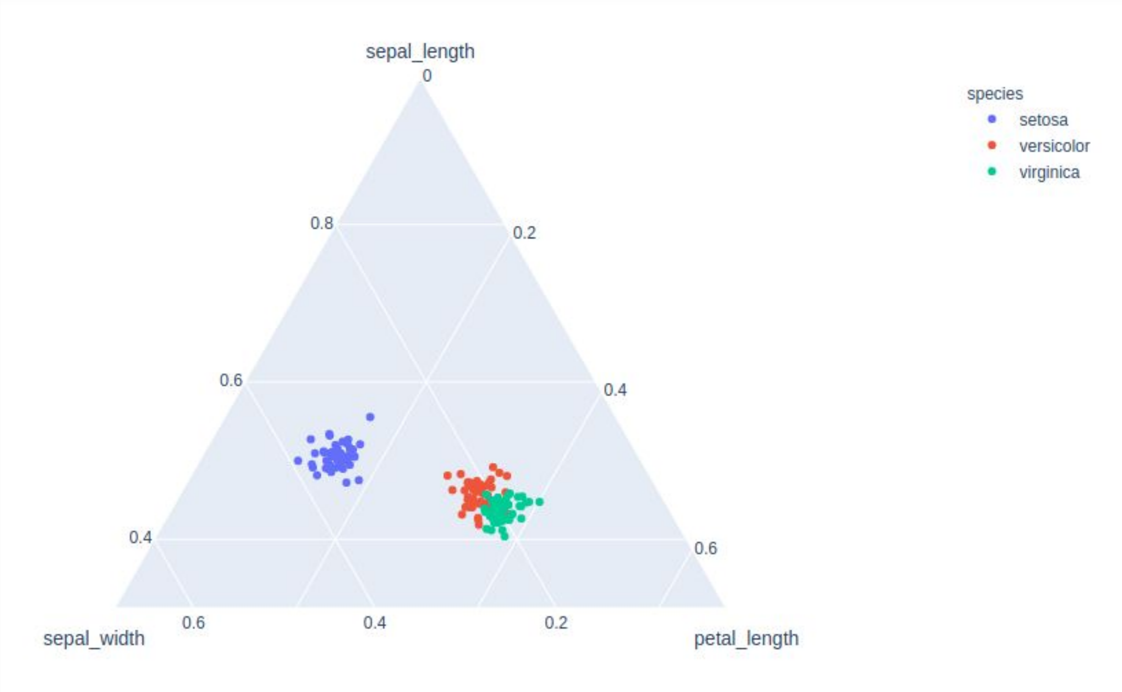
Subplot → Combine multiple plots into a single figure.

```
fig = make_subplots(rows=2, cols=2, subplot_titles=(
'Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width'))
fig.add_trace(go.Scatter(x=iris_df.index, y=iris_df['sepal length
(cm)']), row=1, col=1)
fig.add_trace(go.Scatter(x=iris_df.index, y=iris_df['sepal width
(cm)']), row=1, col=2)
fig.add_trace(go.Scatter(x=iris_df.index, y=iris_df['petal length
(cm)']), row=2, col=1)
fig.add_trace(go.Scatter(x=iris_df.index, y=iris_df['petal width
(cm)']), row=2, col=2)
fig.show()
```



Ternary Plot → Visualizes proportions in three-component systems.

```
fig = px.scatter_ternary(df, a="sepal_length",
b="sepal_width",
c="petal_length",
color="species",
size_max=20)
fig.show()
```



Differences

Feature	Plotly	Seaborn
Interactivity	Highly interactive, ideal for web applications	Static image, limited interactivity
Ease of Use	requires more code for basic plots	User-friendly, concise syntax, easy for basic plots
Customization	Extensive, detailed control over all aspects	High-level interface, less granular control
Integration	Integrates well with web applications (Dash), Jupyter	Primarily for Jupyter notebooks
Types	Wide range including 3D, maps, complex interactive charts	Focuses on common statistical visualizations

When to Use :

Plotly :

1. Highly interactive visualizations needed like dashboards.
2. Complex Visualizations needed like 3D charts, geographical maps etc.
3. Fine-grained Customization required In case of extensive control over the appearance .

Seaborn :

1. Statistical data visualization, especially when working with DataFrames.
2. Creating static images suitable for publication or reports.
3. For quick prototyping and integration into data workflows.

Summary

- ❖ Plotly shines in interactivity, complex visualizations, and fine-grained customization.
- ❖ Seaborn excels in statistical analysis, producing publication-quality images, and ease of use within the Python ecosystem.