Visualization

Visualization is the process of representing data or information in a graphical or pictorial format, making it easier to understand patterns, trends, and insights. Instead of analyzing raw data in tables or text form, visualization transforms the data into charts, graphs, maps, and interactive dashboards that help users quickly grasp complex information. It plays a vital role in data analysis, storytelling, and decision-making by making data more accessible and meaningful.

Tools like Plotly, Matplotlib, Seaborn are commonly used to create such visual representations, helping both technical and non-technical audiences explore and communicate data effectively.

Plotly

Plotly is a popular open-source Python library used for creating interactive, publication-quality visualizations. It is widely used in data science, analytics and machine learning for presenting data insights visually and interactively. It supports a wide variety of charts including line plots, scatter plots, bar charts, pie charts, heatmaps and 3D plots.

Note:

- **Interactive by Default:** Built-in support for zoom, pan, tooltips and legends enhances data exploration.
- **Web Integration:** Easily embeddable in web apps and dashboards using frameworks like Dash.

With plotly we can create more than 40 charts and every plot can be created using the plotly.express and plotly.graph_objects class.

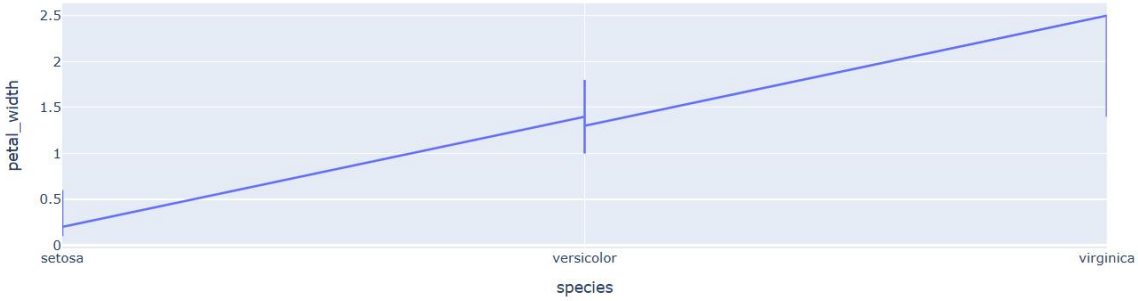
1. **LINE CHART:**

Line plot in Plotly is much accessible and illustrious annexation to plotly which manage a variety of types of data and assemble easy-to-style statistic. With **px.line** each data position is represented as a vertex (which location is given by the x and y columns) of a polyline mark in 2D space.

Code snippet:

```
import plotly.express as px
df=px.data.iris()
g=px.line(df,x="species",y="petal_width")
g.show()
```

Output:



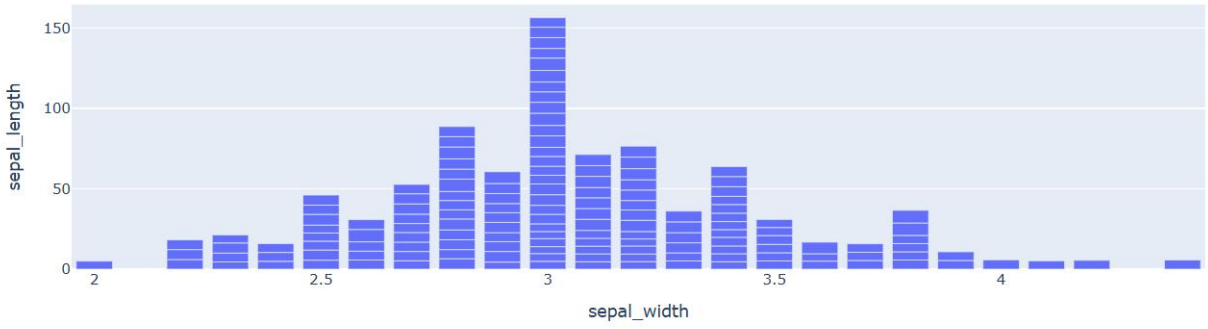
2. **BAR CHART:**

A bar chart is a pictorial representation of data that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent.

Code snippet:

```
import plotly.express as px
df=px.data.iris()
h=px.bar(df,x="sepal_width",y="sepal_length")
h.show()
```

Output:



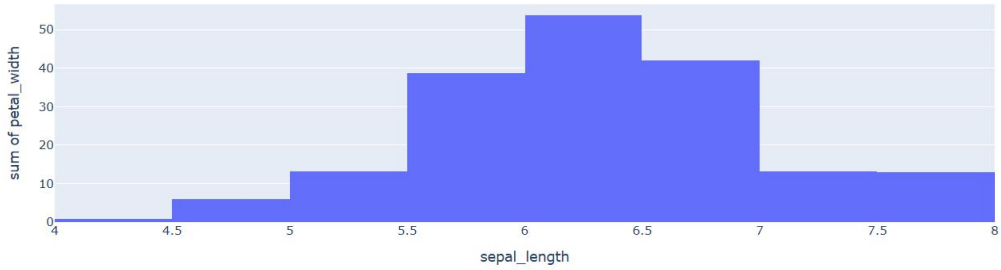
3. **HISTOGRAM:**

A histogram contains a rectangular area to display the statistical information which is proportional to the frequency of a variable and its width in successive numerical intervals. A graphical representation that manages a group of data points into different specified ranges. It has a special feature that shows no gaps between the bars and similar to a vertical bar graph.

Code snippet:

```
import plotly.express as px
df=px.data.iris()
k=px.histogram(df,x="sepal_length",y="petal_width")
k.show()
```

Output:



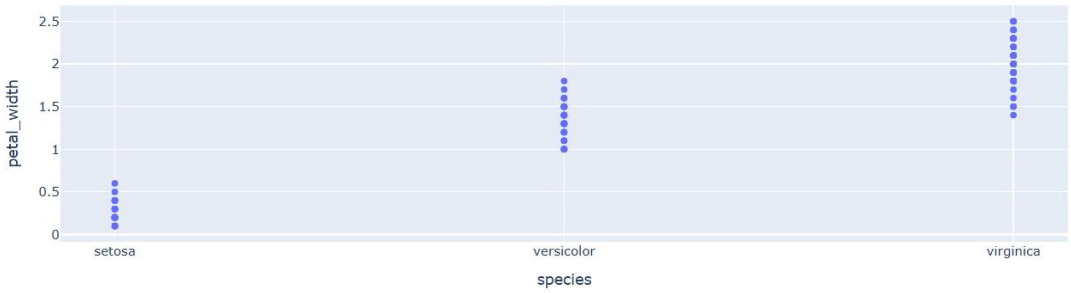
4. **SCATTER PLOT**

A scatter plot is a set of dotted points to represent individual pieces of data in the horizontal and vertical axis.

Code snippet:

```
import plotly.express as px
df=px.data.iris()
t=px.scatter(df,x="species",y="petal_width")
t.show()
```

Output:



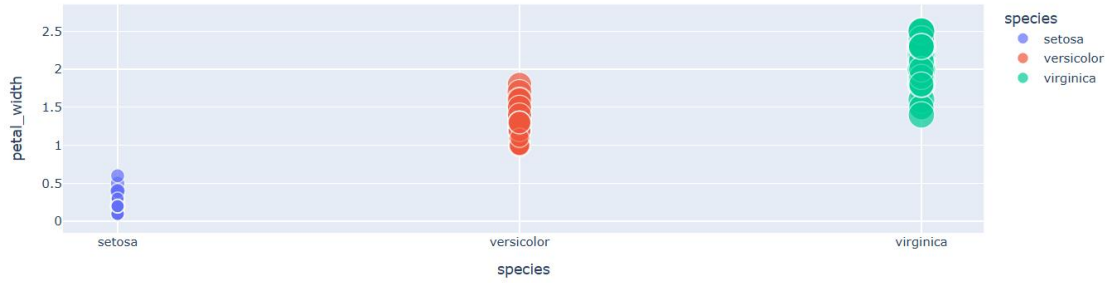
5. **BUBBLE PLOT:**

A bubble plot is a scatter plot with bubbles (color-filled circles). Bubbles have various sizes dependent on another variable in the data. It can be created using the scatter() method of plotly.express.

Code snippet:

```
import plotly.express as px
df=px.data.iris()
b=px.scatter(df,x="species",y="petal_width",size="petal_length",color="species")
b.show()
```

Output:



6. **PIE CHART:**

A pie chart is a circular statistical graphic, which is divided into slices to illustrate numerical proportions. It depicts a special chart that uses “pie slices”, where each sector shows the relative sizes of data.

Code snippet:

```
import plotly.express as px
tf=px.data.tips()
e=px.pie(tf,values="total_bill",names="day")
e.show()
```

Output:

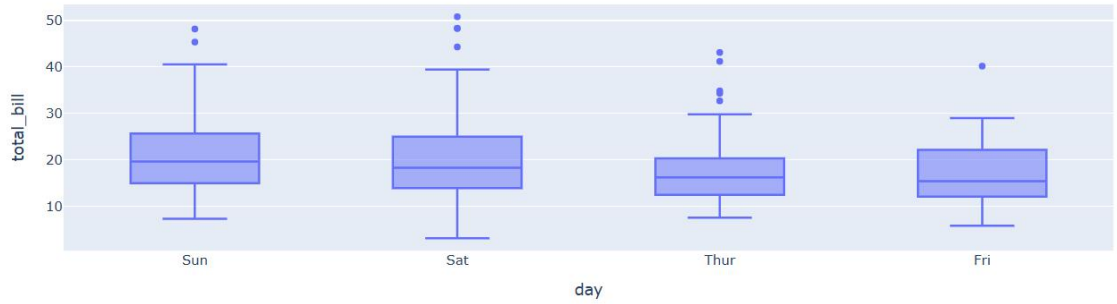
7. **BOX PLOT:**

A Box plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum.

Code snippet:

```
import plotly.express as px
tf=px.data.tips()
r=px.box(tf,x="day",y="total_bill")
r.show()
```

Output:



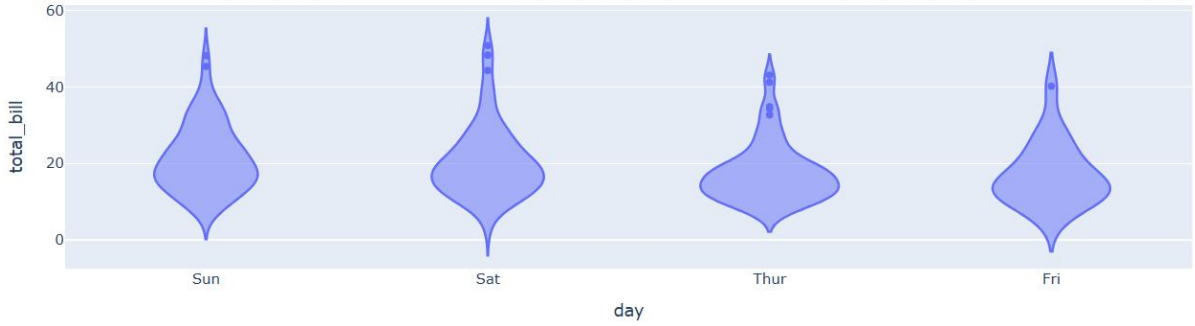
8. **VIOLIN PLOT:**

A violin plot is a method to visualize the distribution of numerical data of different variables. It is similar to Box Plot but with a rotated plot on each side, giving more information about the density estimate on the y-axis. The density is mirrored and flipped over and the resulting shape is filled in, creating an image resembling a violin.

Code snippet:

```
import plotly.express as px
tf=px.data.tips()
v=px.violin(tf,x="day",y="total_bill")
v.show()
```

Output:



9.GANTT CHART:

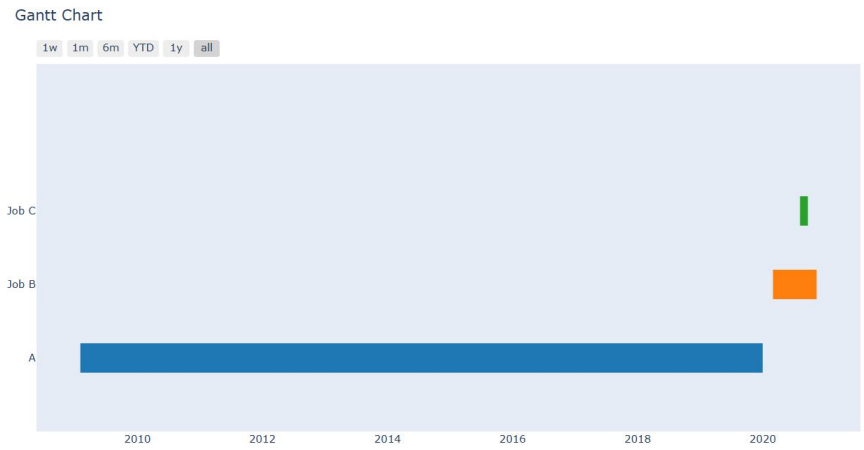
Generalized Activity Normalization Time Table (GANTT) chart is type of chart in which series of horizontal lines are present that show the amount of work done or production completed in given period of time in relation to amount planned for those projects.

Code snippet:

```
import plotly.figure_factory as ff

yf=[dict(Task="A", Start='2020-01-01', Finish='2009-02-02'),
    dict(Task="Job B", Start='2020-03-01', Finish='2020-11-11'),
    dict(Task="Job C", Start='2020-08-06', Finish='2020-09-21')]
w=ff.create_gantt(yf)
w.show()
```

Output:



10.CONTOUR PLOT:

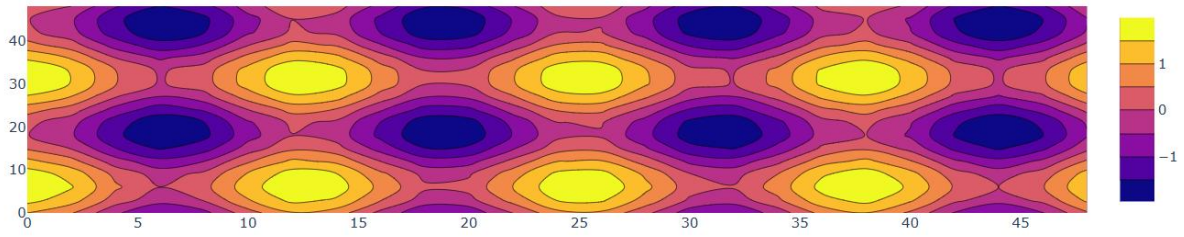
A Contour plots also called level plots are a tool for doing multivariate analysis and visualizing 3-D plots in 2-D space.

Code snippet:

```
import plotly.graph_objects as go
import numpy as np
f_x=np.arange(0,50,2)
f_y=np.arange(0,50,3)
[X,Y]=np.meshgrid(f_x,f_y)
u = np.cos(X / 2) + np.sin(Y / 4)
i = go.Figure(data =
    go.Contour(x = f_x, y = f_y, z = u))

i.show()
```

Output:



11. **HEATMAPS**

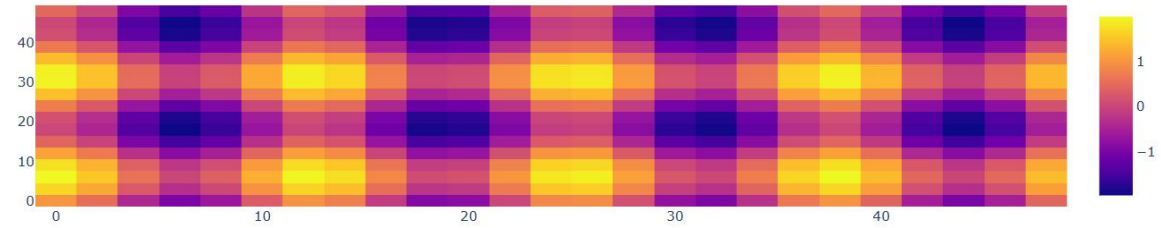
Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix.

Code snippet:

```
import plotly.graph_objects as go
import numpy as np
f_x=np.arange(0,50,2)
f_y=np.arange(0,50,3)
[X,Y]=np.meshgrid(f_x,f_y)
u = np.cos(X / 2) + np.sin(Y / 4)
ig = go.Figure(data =
    go.Heatmap(x = f_x, y = f_y, z = u))

ig.show()
```

Output:



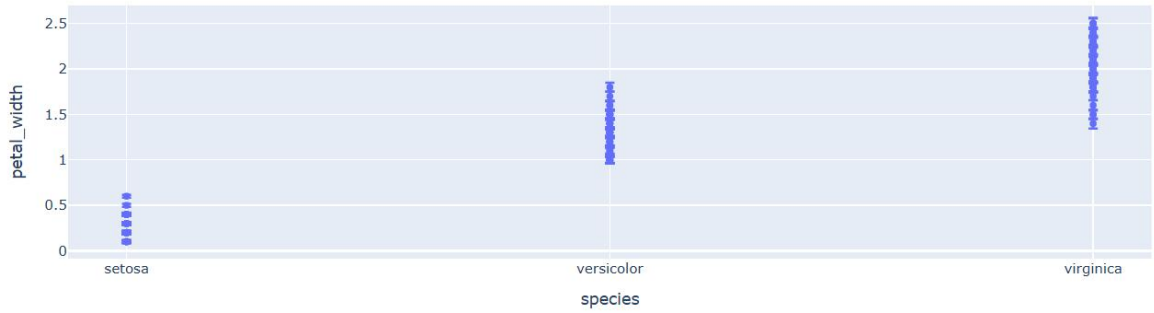
12. **ERROR BARS:**

Error bars are the graphical presentation alternation of data and used on graphs to imply the error or uncertainty in a reported capacity.

Code snippet:

```
import plotly.express as px
df = px.data.iris()
df["error"] = df["petal_length"]/100
fig = px.scatter(df, x="species", y="petal_width",
                 error_x="error", error_y="error")
fig.show()
```

Output:



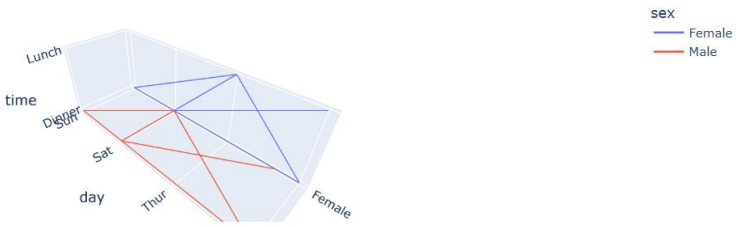
13.**3D_LINE CHART:**

With **px.line_3d** each data position is represented as a vertex (which location is given by the x, y and z columns) of a polyline mark in 3D space.

Code snippet:

```
import plotly.express as px
tf=px.data.tips()
hk=px.line_3d(tf,x="sex",y="day",z="time",color="sex")
hk.show()
```

Output:



14.**3D_SCATTER PLOT:**

3D Scatter plot can plot two-dimensional graphics that can be enhanced by mapping up to three additional variables while using the semantics of hue, size and style parameters.

Code snippet:

```
import plotly.express as px
df = px.data.iris()
gm=px.scatter_3d(df,x="sepal_width",y="sepal_length",z="sepal_width",color="species")
gm.show()
```

Output:

15.3D_SURFACE PLOT:

Surface plot is those plot which has three-dimensions data which is X, Y and Z. Rather than showing individual data points, the surface plot has a functional relationship between dependent variable Y and have two independent variables X and Z. This plot is used to distinguish between dependent and independent variables.

Code snippet:

```
import plotly.graph_objects as go
import numpy as np
x = np.outer(np.linspace(-2, 2, 50), np.ones(50))
y = x.copy().T
z = np.cos(x ** 2 + y ** 2)
fig = go.Figure(data=[go.Surface(x=x, y=y, z=z)])
fig.show()
```

Output:

Seaborn

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

1. Scatter Plot

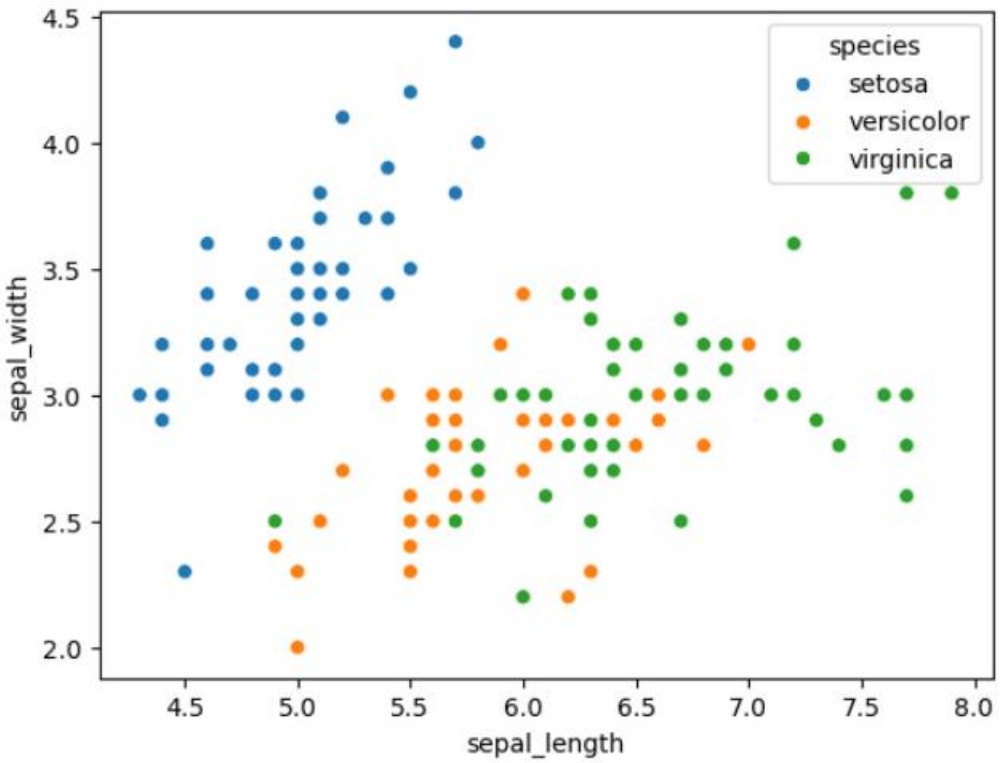
Purpose: Shows the relationship between two numeric variables.

Code snippet:

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

df = sns.load_dataset("iris")
sns.scatterplot(data=df, x="sepal_length", y="sepal_width", hue="species")
plt.show()
```

Output:



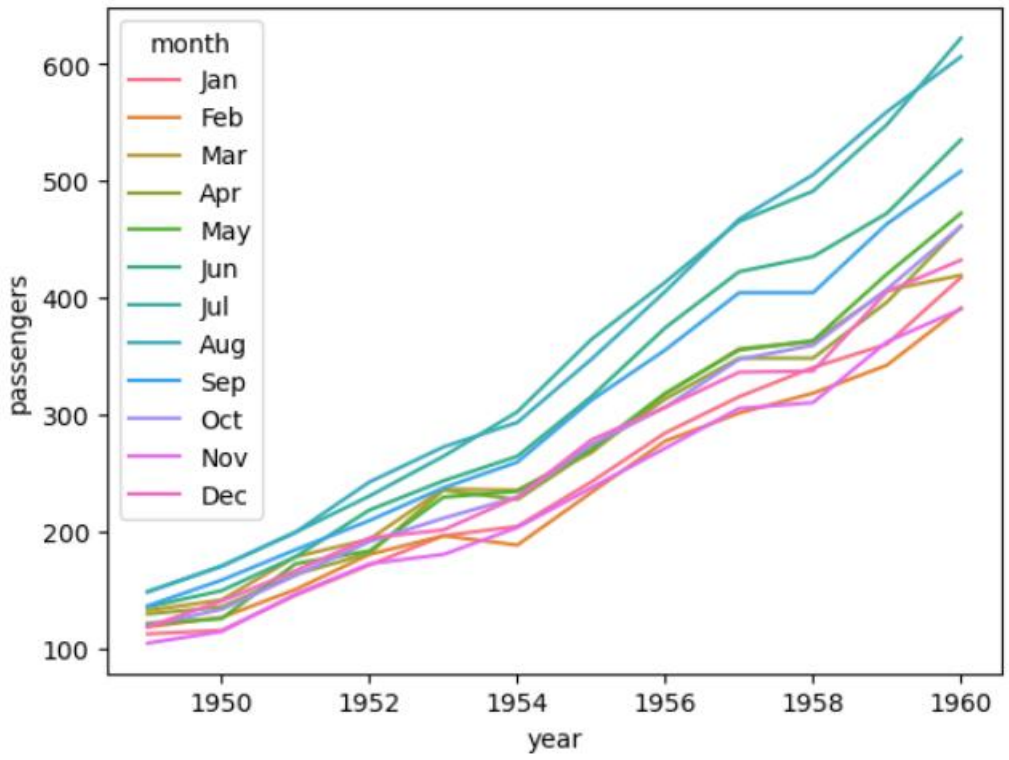
2. Line Plot

Purpose: Displays trends over continuous data (e.g., time series).

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
yf = sns.load_dataset("flights")
sns.lineplot(data=yf, x="year", y="passengers", hue="month")
plt.show()
```

Output:



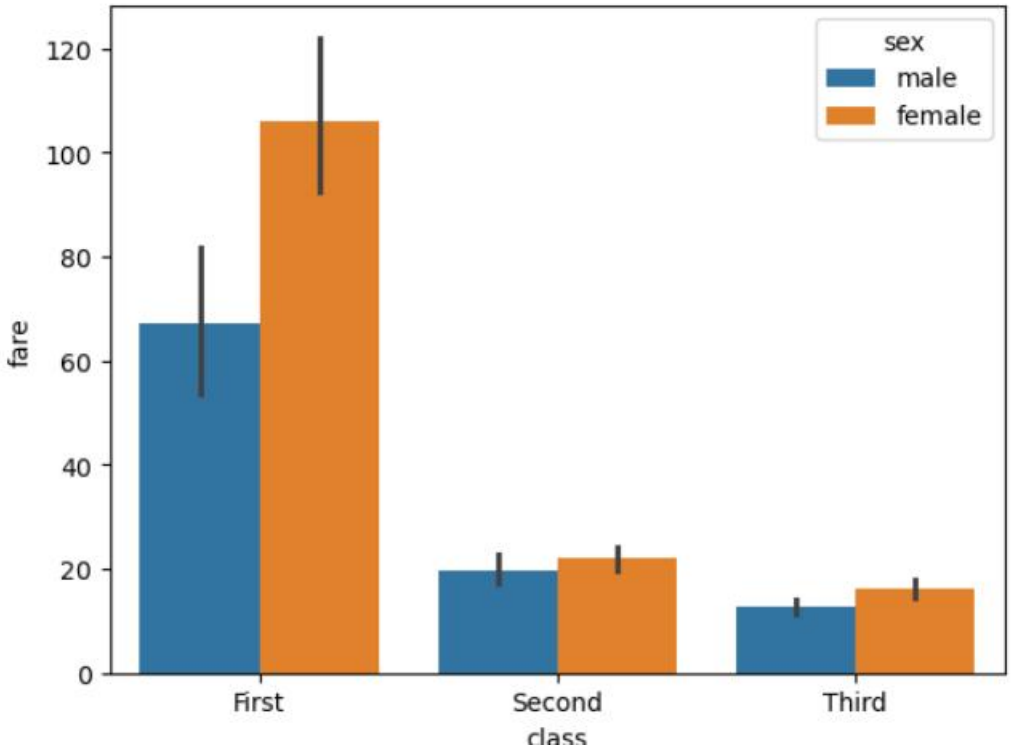
3. Bar Plot

Purpose: Compares values across categories.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
hf = sns.load_dataset("titanic")
sns.barplot(data=hf, x="class", y="fare", hue="sex")
plt.show()
```

Output:



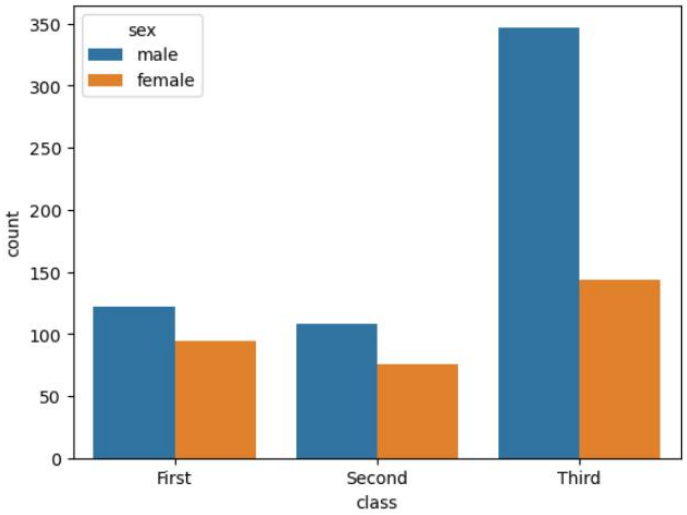
4. Count Plot

Purpose: Shows counts of each category.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(data=df, x="class", hue="sex")
plt.show()
```

Output:



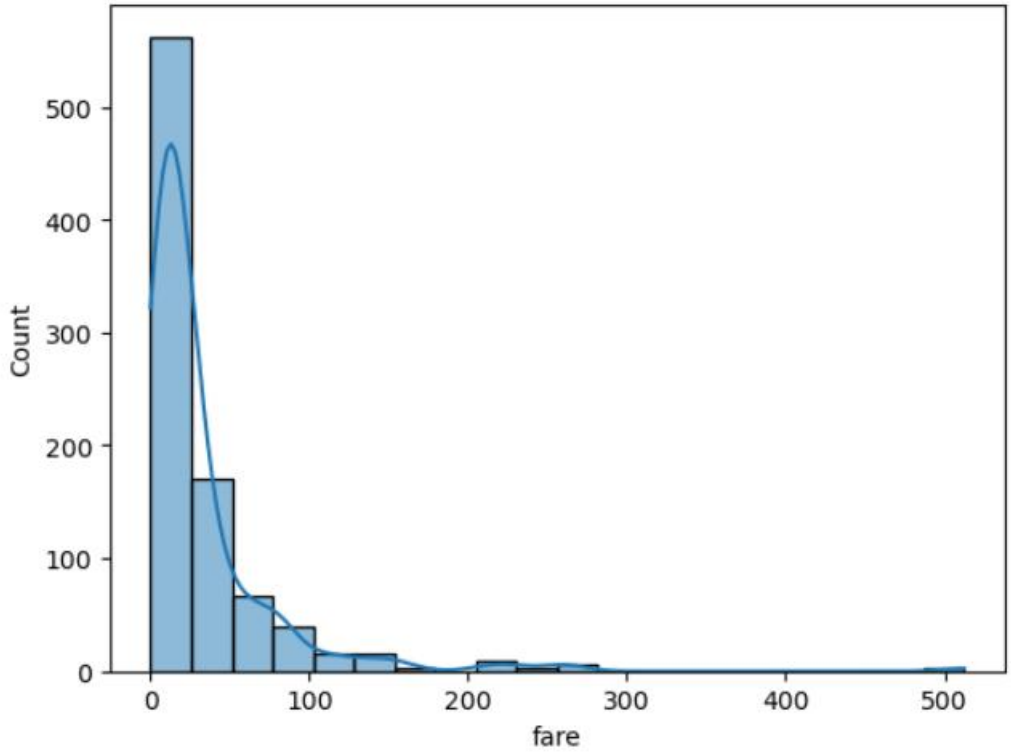
5. Histogram

Purpose: Shows distribution of a single numeric variable.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.histplot(data=hf, x="fare", bins=20, kde=True)
plt.show()
```

Output:



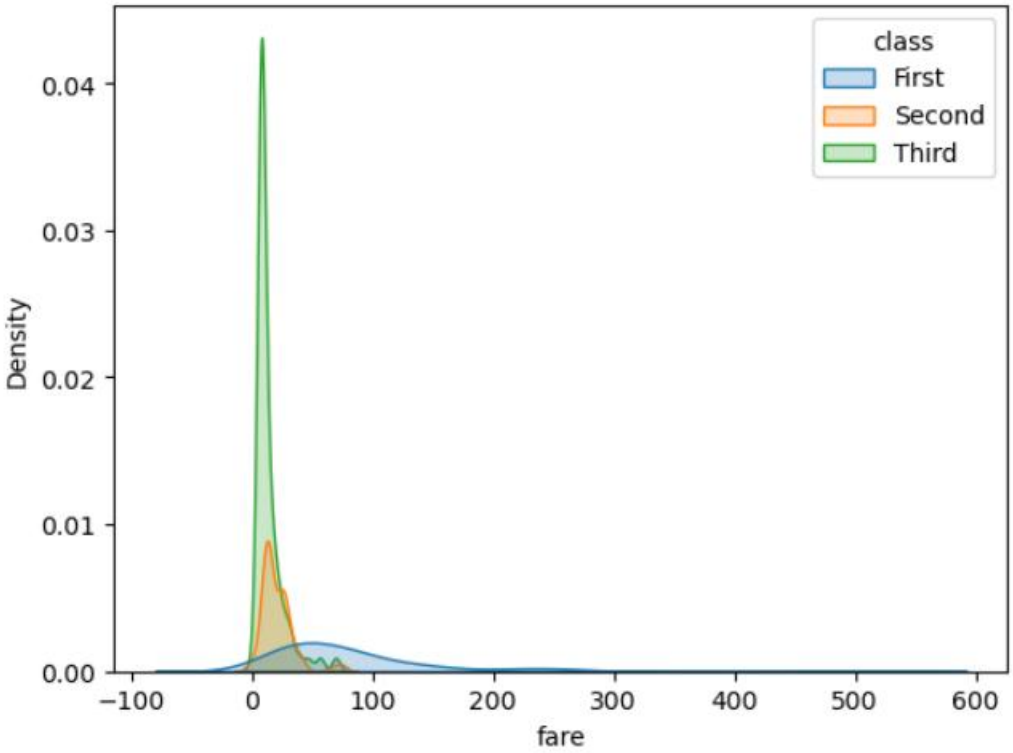
6. KDE Plot (Kernel Density Estimate)

Purpose: Smooth version of a histogram, shows probability density.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.kdeplot(data=hf, x="fare", hue="class", fill=True)
plt.show()
```

Output:



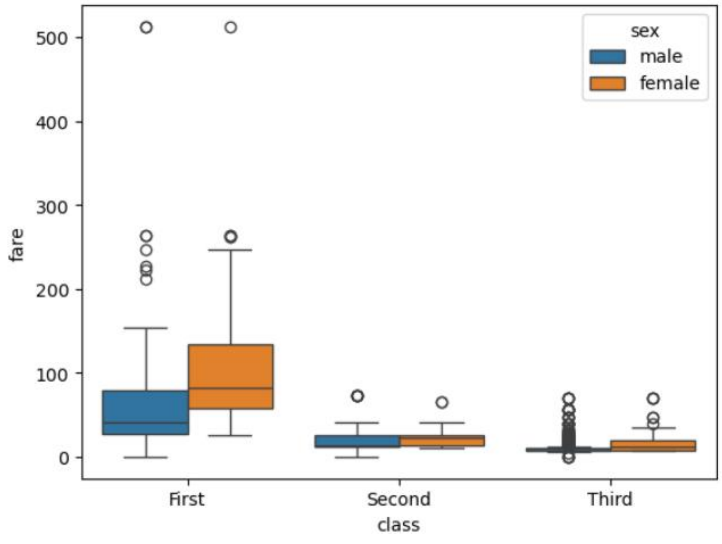
7. Box Plot

Purpose: Shows data spread, median, and outliers.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
hf = sns.load_dataset("titanic")
sns.boxplot(data=hf, x="class", y="fare", hue="sex")
plt.show()
```

Output:



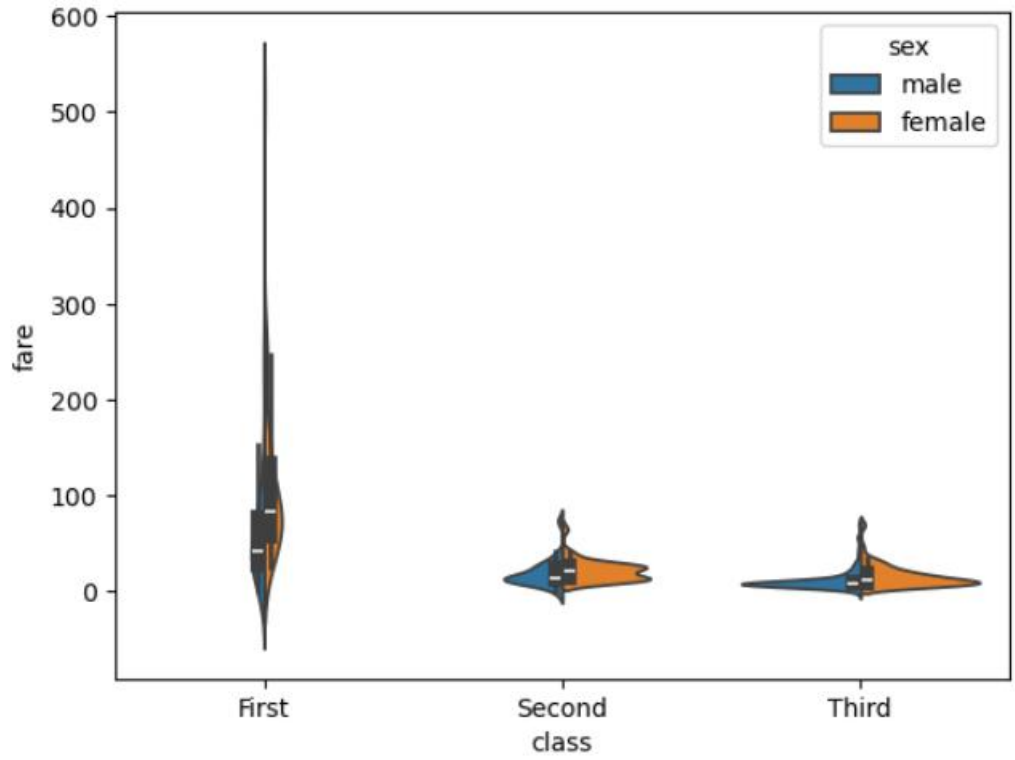
8. Violin Plot

Purpose: Combines box plot with KDE distribution shape.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
hf = sns.load_dataset("titanic")
sns.violinplot(data=hf, x="class", y="fare", hue="sex", split=True)
plt.show()
```

Output:



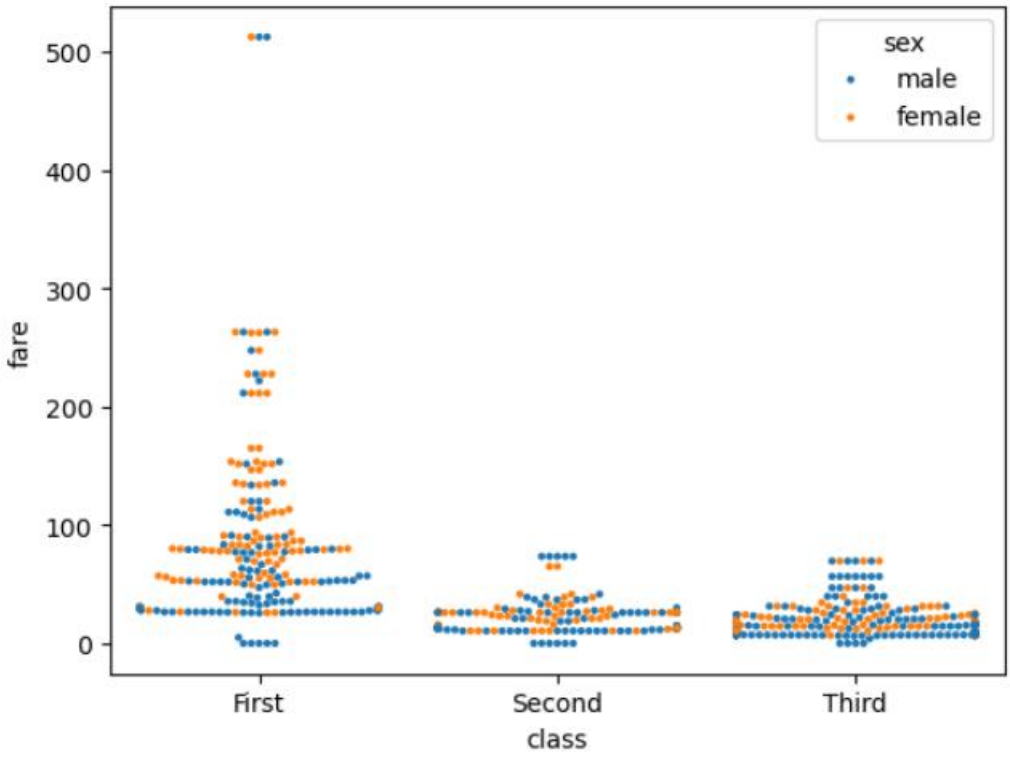
9. Swarm Plot

Purpose: Shows all data points along with distribution.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
hf = sns.load_dataset("titanic")
sns.swarmplot(data=hf, x="class", y="fare", hue="sex", size=3)
plt.show()
```

Output:



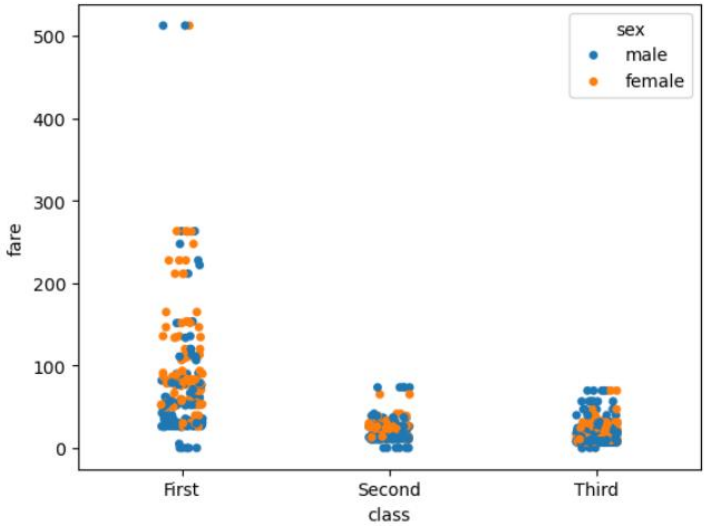
10. Strip Plot

Purpose: Like swarm plot but with overlapping points.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
hf = sns.load_dataset("titanic")
sns.stripplot(data=hf, x="class", y="fare", hue="sex", jitter=True)
plt.show()
```

Output:



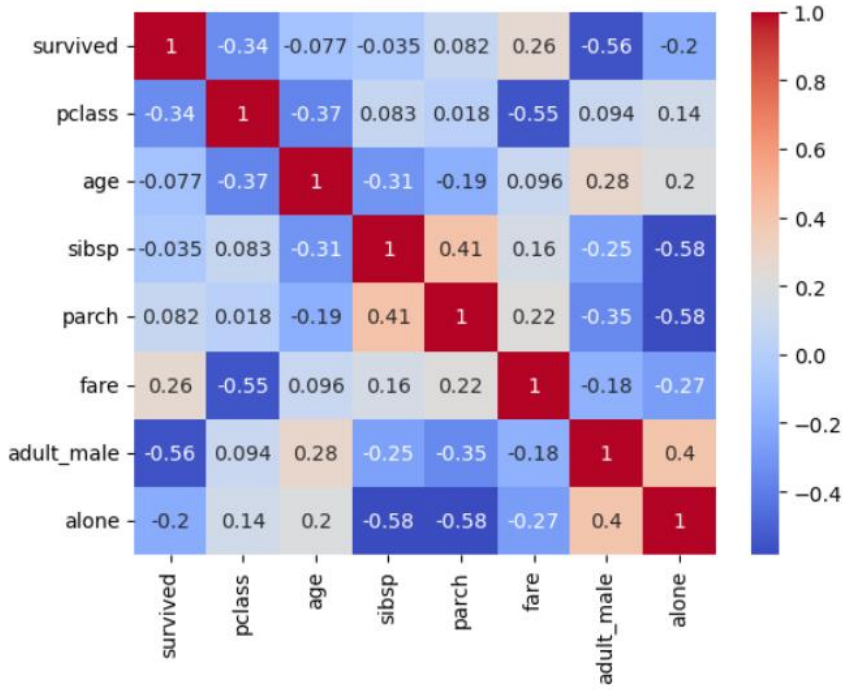
11. Heatmap

Purpose: Shows values in a grid, often for correlations or matrices.

Code snippet:

```
import matplotlib.pyplot as plt
hf = sns.load_dataset("titanic")
corr = hf.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap="coolwarm")
plt.show()
```

Output:



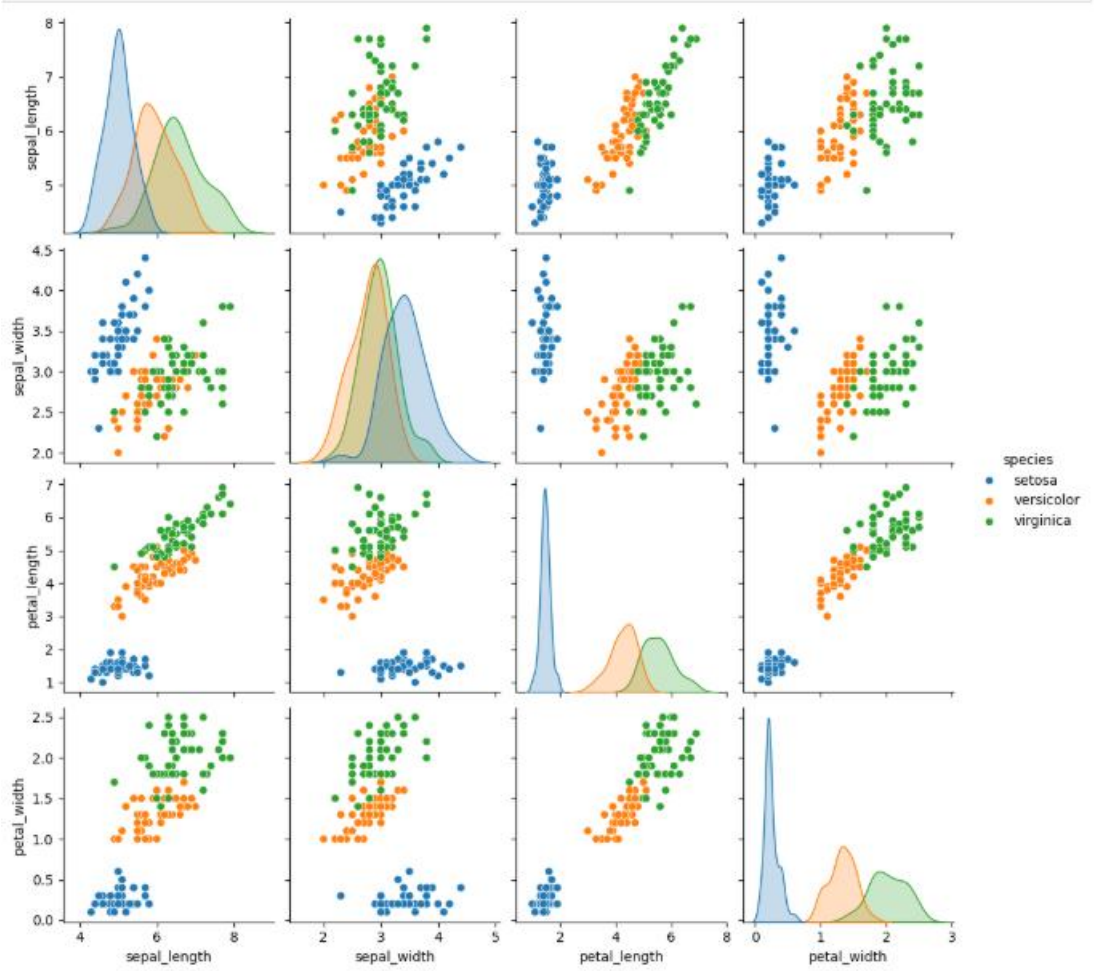
12. Pair Plot

Purpose: Shows pairwise relationships between all numeric variables.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset("iris")
sns.pairplot(df, hue="species")
plt.show()
```

Output:



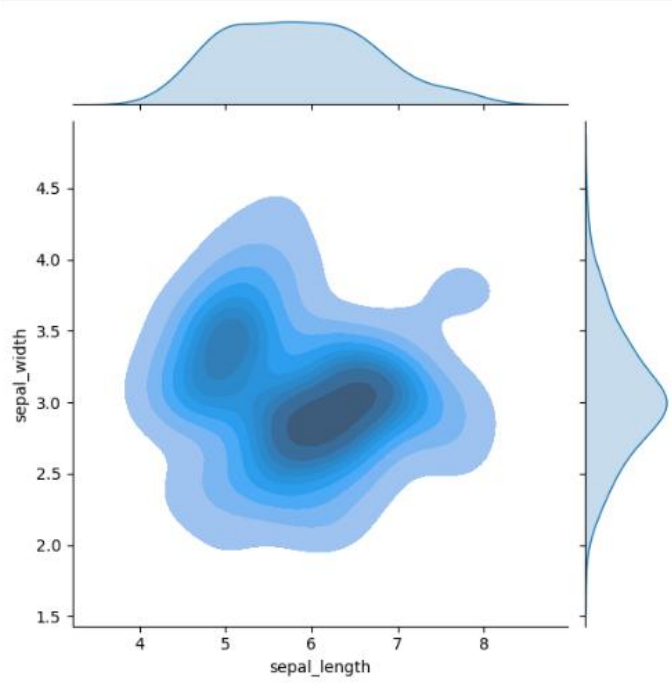
13. Joint Plot

Purpose: Combines scatter plot with marginal histograms or KDEs.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset("iris")
sns.jointplot(data=df, x="sepal_length", y="sepal_width", kind="kde", fill=True)
plt.show()
```

Output:



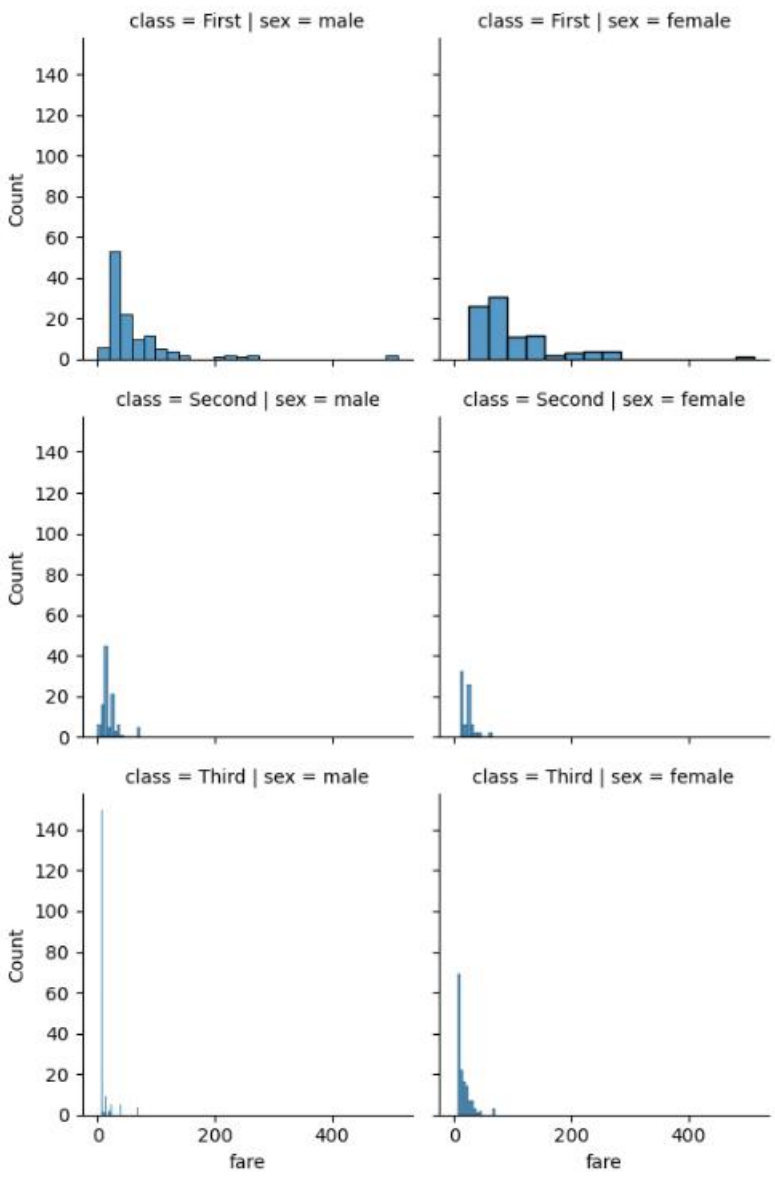
14. Facet Grid

Purpose: Plots multiple subplots based on categorical splits.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
hf = sns.load_dataset("titanic")
g = sns.FacetGrid(hf, col="sex", row="class")
g.map_dataframe(sns.histplot, x="fare")
plt.show()
```

Output:



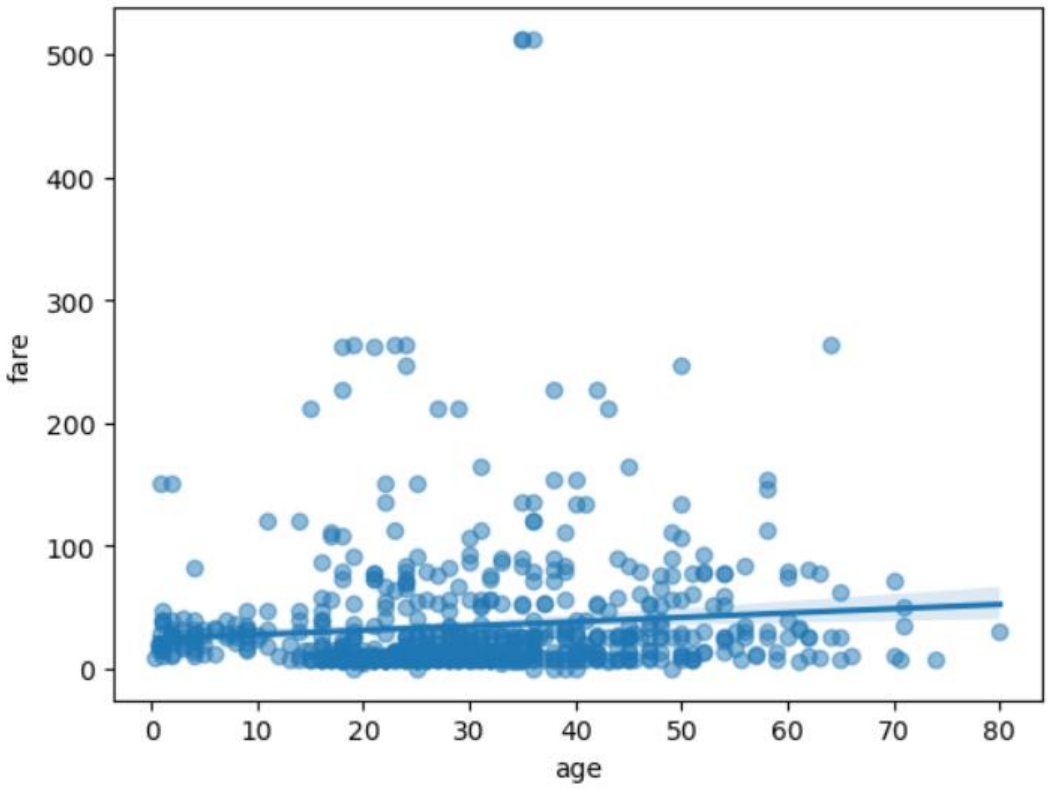
15. Regression Plot

Purpose: Scatter plot with regression line fit.

Code snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
hf = sns.load_dataset("titanic")
sns.regplot(data=hf, x="age", y="fare", scatter_kws={"alpha":0.5})
plt.show()
```

Output:



Comparison Between Seaborn and Plotly for Data Visualization

Data visualization is an essential aspect of data science and analytics, allowing complex datasets to be transformed into clear and meaningful visuals that help uncover trends, patterns, and insights. Among the many Python libraries available for this purpose, Seaborn and Plotly are two widely used tools, each with its own strengths and ideal use cases.

Seaborn

It is a statistical data visualization library built on top of Matplotlib. It is designed to simplify the creation of beautiful and informative plots with minimal code. Seaborn integrates seamlessly with pandas DataFrames, making it particularly effective for exploratory data analysis (EDA). It comes with built-in themes and color palettes that make visualizations aesthetically pleasing by default. The library excels at producing a variety of advanced statistical plots, including heatmaps, violin plots, pair plots, and regression plots. While Seaborn's plots are static and lack interactivity, its ease of use, clear syntax, and professional appearance make it a top choice for quick analysis, academic reports, and research papers.

Plotly

In contrast, is a modern interactive visualization library that enables the creation of dynamic, web-based charts. Its plots respond to user actions such as hovering, zooming, panning, and clicking, making it ideal for dashboards, data exploration tools, and web applications. Plotly supports a broad range of visualizations, from standard statistical plots to 3D charts, animations, and geographical maps. It integrates well with pandas DataFrames and can export visualizations as standalone HTML files, allowing interactive graphs to be shared without additional software. For projects requiring advanced interactivity and seamless web integration, Plotly is a preferred choice, especially when used alongside Dash, Plotly's framework for building interactive applications.

When comparing the two libraries, Seaborn is often seen as the simpler tool for statistical analysis, providing clean static charts with minimal effort. Its customization is achieved through Matplotlib's backend, which offers flexibility for fine-tuned designs. Plotly, while slightly more complex for beginners, offers a higher level of customization using its JSON-style layout system and is better suited for larger datasets when rendered in a browser. Furthermore, Plotly includes capabilities that

Seaborn does not, such as animations, interactive legends, 3D surfaces, and choropleth maps.

In conclusion, Seaborn is best suited for exploratory analysis, research, and situations where clean static visualizations are sufficient, while Plotly excels in building dynamic, presentation-ready dashboards and interactive applications. A data professional who learns both can cover the full spectrum of visualization needs, from quick statistical summaries to fully interactive analytical tools.