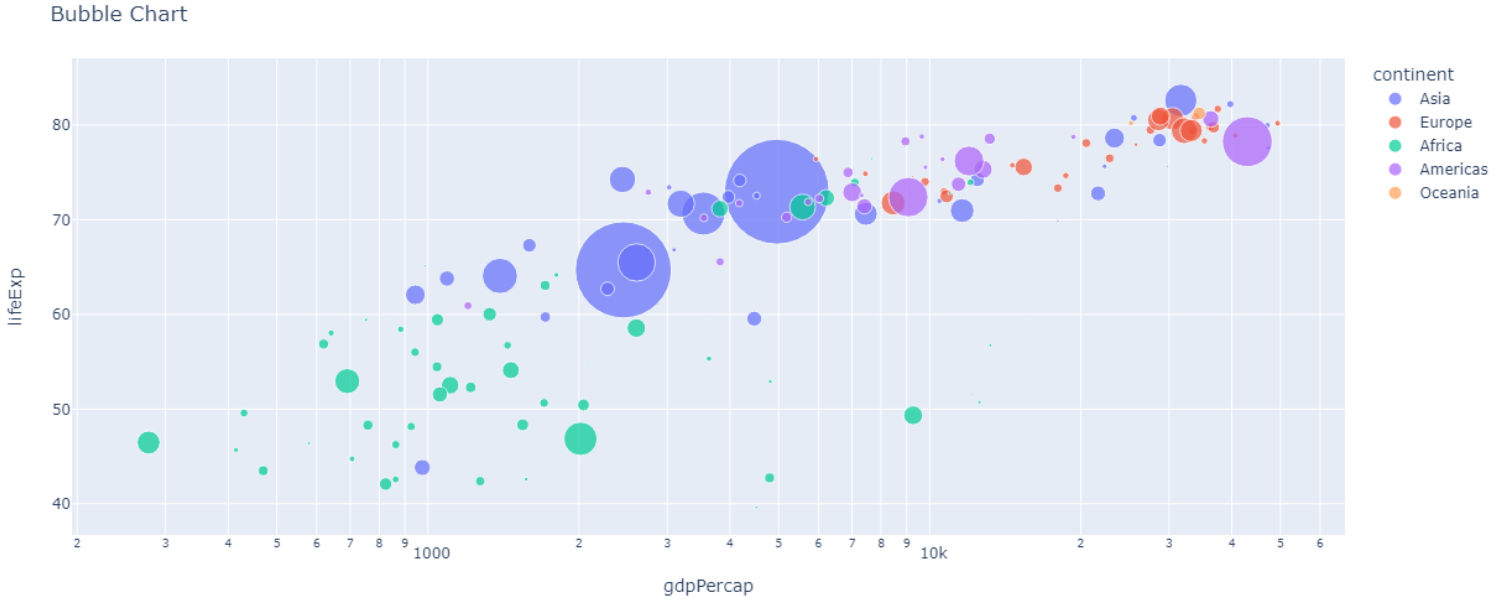
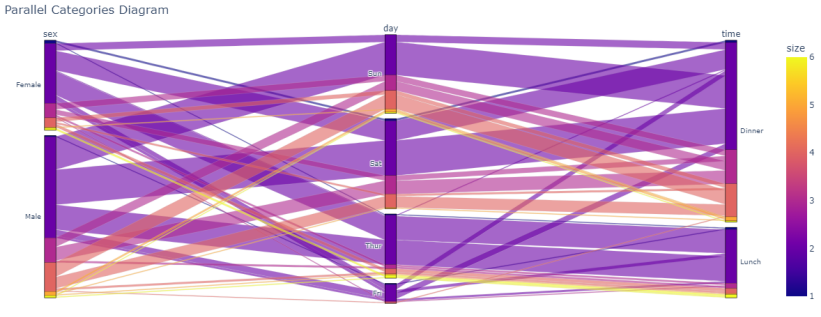
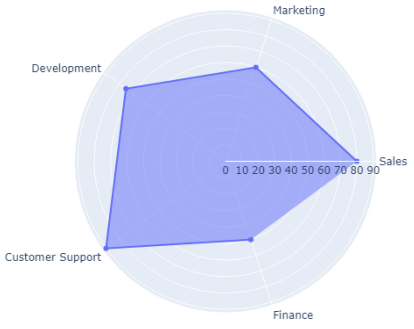
Python Plotly

Plotly is an open-source library for creating interactive and dynamic visualizations in Python and other languages. It offers a wide range of chart types, interactivity, and customization options. Plotly visualizations can be used in Jupyter notebooks, standalone files, or shared online.







Plotly

Plotly is an open-source data visualization library that allows you to create interactive and dynamic visualizations in Python, R, and other programming languages. It provides a wide range of chart types, including scatter plots, bar charts, line charts, pie charts, maps, and more.

Plotly offers a high-level interface through its Plotly Express module, which simplifies the creation of interactive visualizations with a concise syntax. It also provides a lower-level interface through its graph_objects module, allowing for more customization and fine-grained control over the visual elements.

One of the key features of Plotly is its interactivity. You can create interactive visualizations that respond to user interactions such as hover, click, and zoom. Plotly also supports animations, allowing you to create dynamic and engaging visualizations.

Plotly visualizations can be rendered in Jupyter notebooks, as standalone HTML files, or embedded in web applications. You can also share your visualizations online with others or collaborate with team members by using Plotly's cloud-based platform, Plotly Chart Studio.

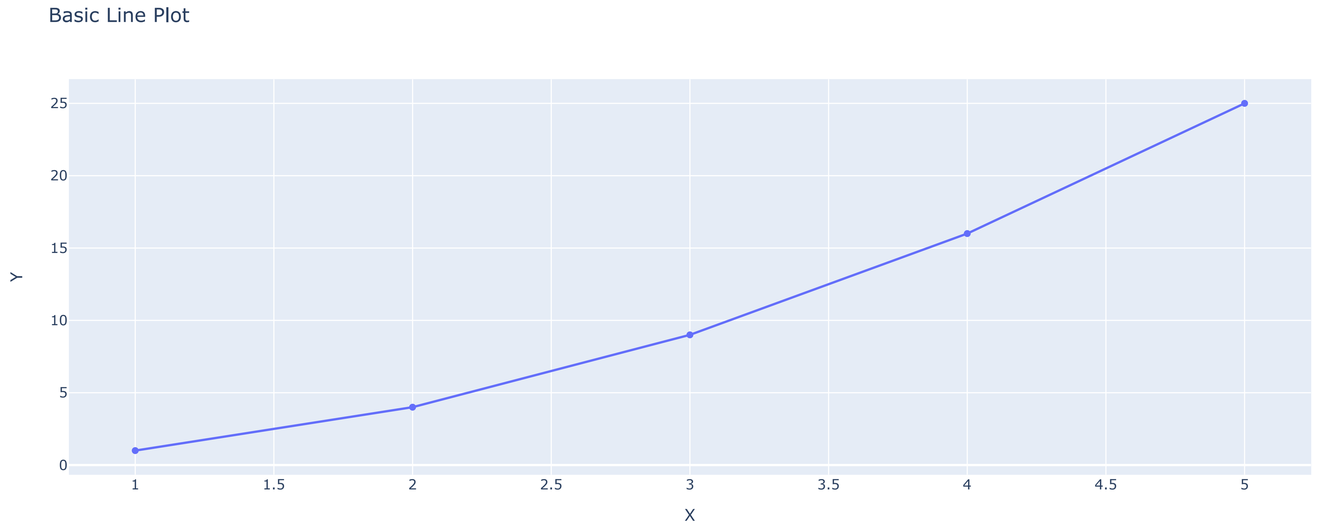
Overall, Plotly is a powerful and versatile data visualization library that empowers you to create professional-looking and interactive visualizations to explore, analyze, and communicate your data effectively.

Basic Line Plot

```
import plotly.graph_objects as go
import plotly.express as px


x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

fig = go.Figure(data=go.Scatter(x=x, y=y))
fig.update_layout(title="Basic Line Plot", xaxis_title="X", yaxis_title="Y")
fig.show()
```

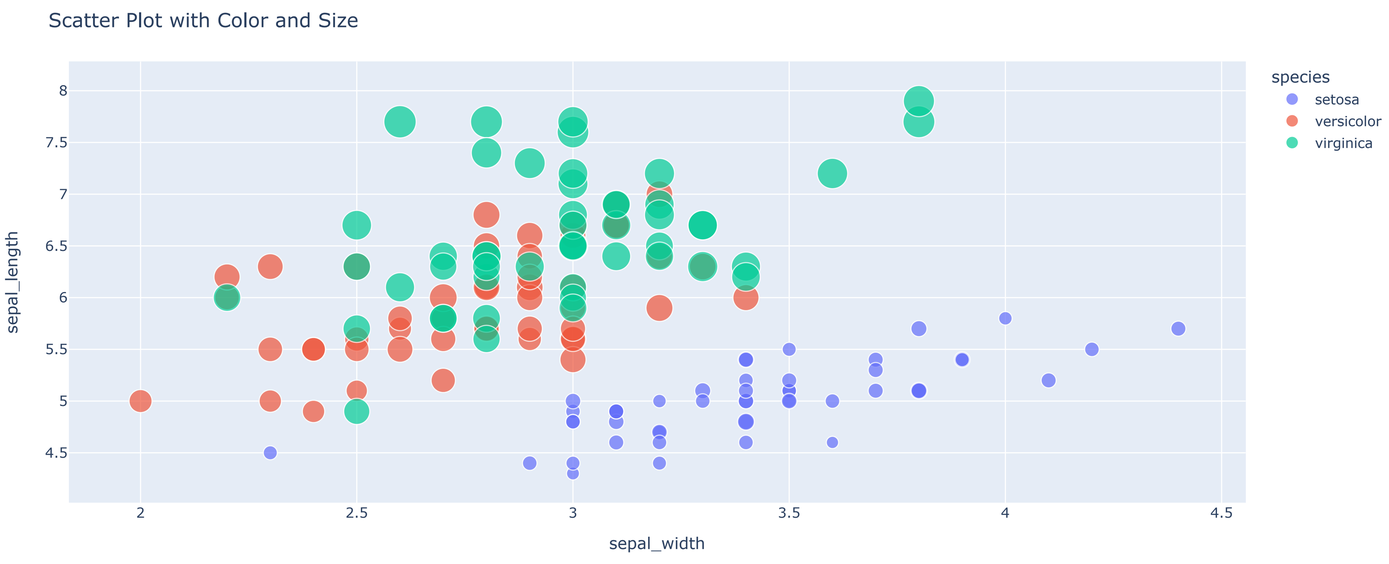


In this example, we create a basic line plot using the go.Figure class from plotly.graph_objects. We define the x and y values as lists representing the x and y coordinates of the points to be plotted. We use the go.Scatter trace to define the data for the line plot. We then update the layout of the figure using the update_layout method to set the title and labels for the x and y axes. Finally, we display the figure using fig.show().

Scatter Plot with Color and Size

```
df = px.data.iris()

fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", size="petal_length",
                 hover_data=["petal_width"])
fig.update_layout(title="Scatter Plot with Color and Size")
fig.show()
```

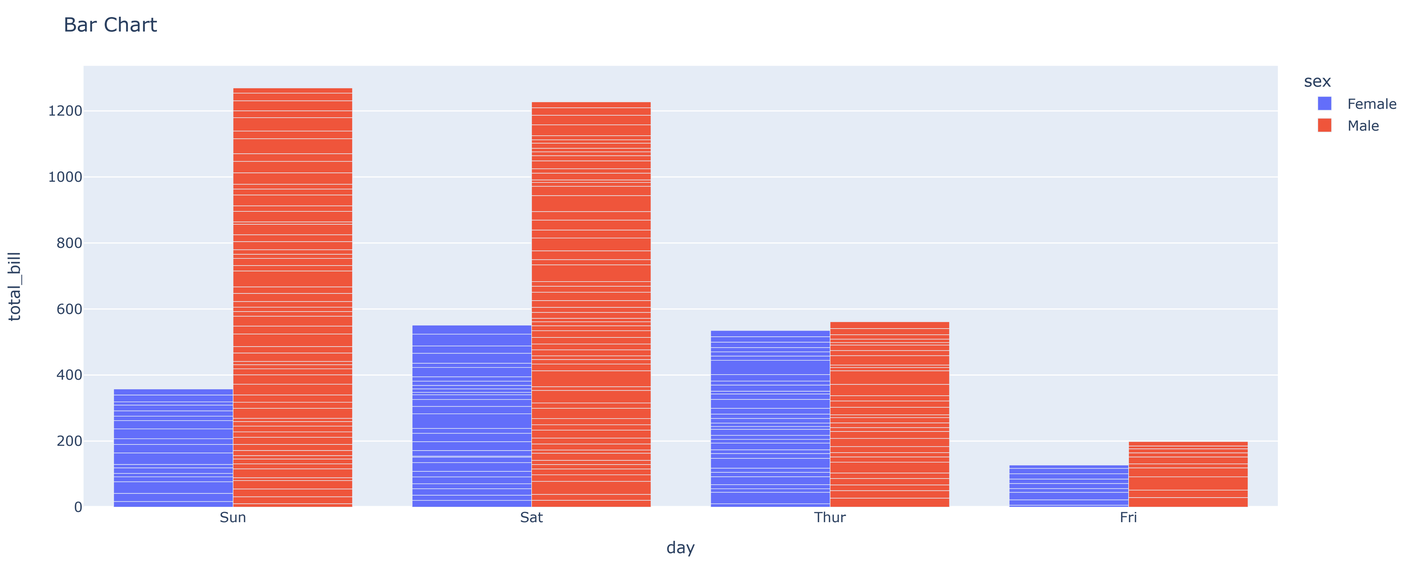


In this example, we create a scatter plot using the px.scatter function from plotly.express. We use the df DataFrame containing the iris dataset. We specify the x and y variables as "sepal_width" and "sepal_length", respectively. We also set the color of the markers based on the "species" column and the size of the markers based on the "petal_length" column. Additionally, we include "petal_width" as hover data, which is displayed when hovering over the markers. We update the layout with a title and display the figure.

Bar Chart

```
df = px.data.tips()

fig = px.bar(df, x="day", y="total_bill", color="sex", barmode="group")
fig.update_layout(title="Bar Chart")
fig.show()
```

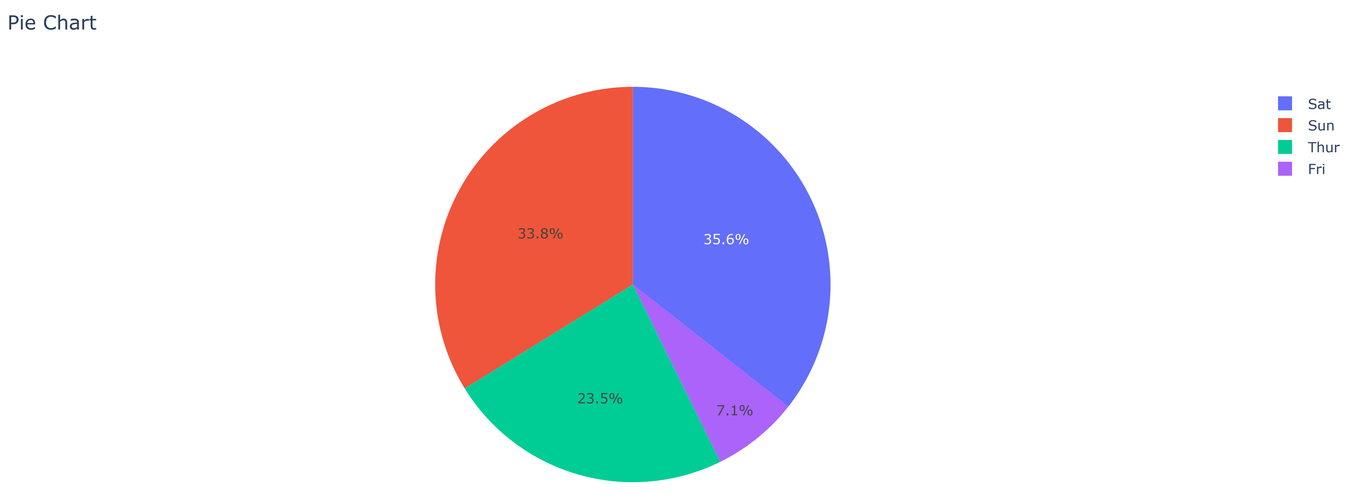


In this example, we create a bar chart using the px.bar function. We use the df DataFrame containing the tips dataset. We set the x and y variables as "day" and "total_bill", respectively. We also color the bars based on the "sex" column and set the barmode to "group" for grouped bars. We update the layout with a title and display the figure.

Pie Chart

```
df = px.data.tips()

fig = px.pie(df, values="tip", names="day", title="Pie Chart")
fig.show()
```

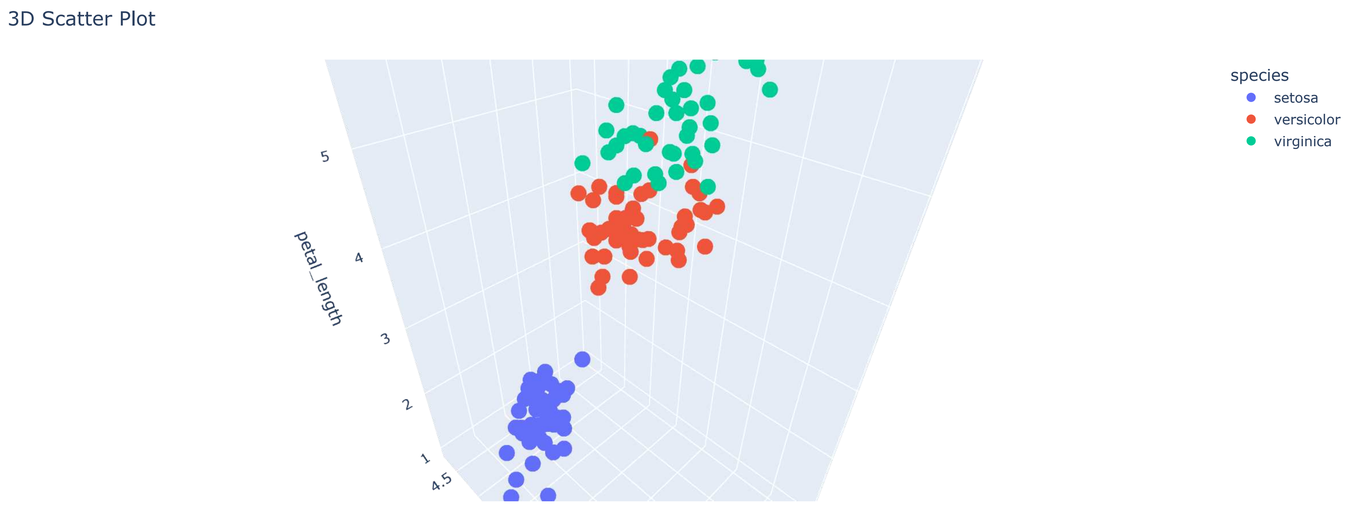


In this example, we create a pie chart using the px.pie function. We use the df DataFrame containing the tips dataset. We specify the values as the "tip" column and the names as the "day" column. We set the title of the chart and display the figure.

3D Scatter Plot

```
df = px.data.iris()

fig = px.scatter_3d(df, x="sepal_width", y="sepal_length", z="petal_length", color="species")
fig.update_layout(title="3D Scatter Plot")
fig.show()
```



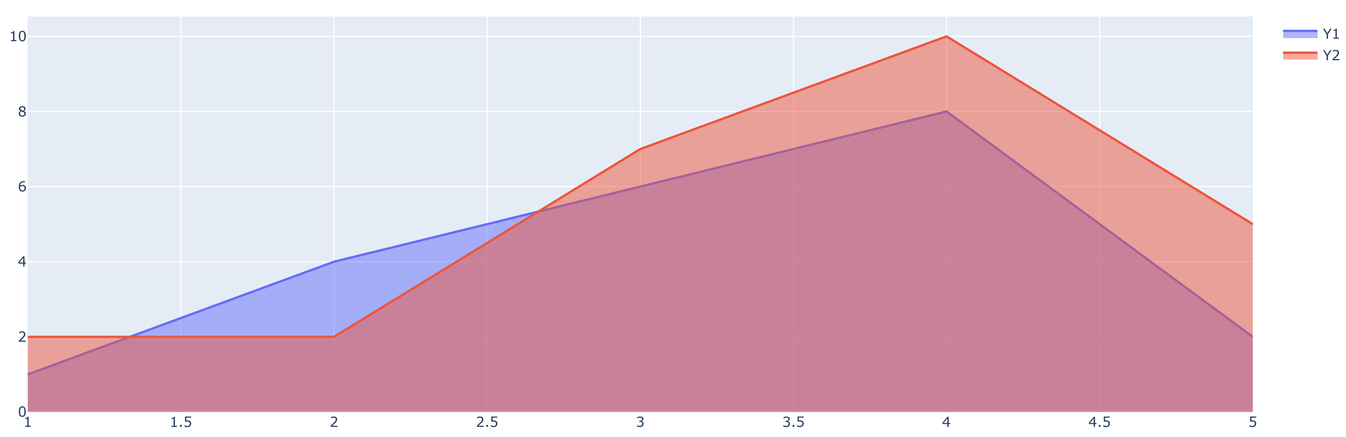
In this example, we create a 3D scatter plot using the px.scatter_3d function. We use the df DataFrame containing the iris dataset. We specify the x, y, and z variables as"sepal_width", "sepal_length", and "petal_length", respectively. We also set the color of the markers based on the "species" column. We update the layout with a title and display the figure.

Area Plot

```
import plotly.graph_objects as go
import plotly.express as px


x = [1, 2, 3, 4, 5]
y1 = [1, 4, 6, 8, 2]
y2 = [2, 2, 7, 10, 5]

fig = go.Figure()
fig.add_trace(go.Scatter(x=x, y=y1, mode='lines', fill='tozeroy', name='Y1'))
fig.add_trace(go.Scatter(x=x, y=y2, mode='lines', fill='tozeroy', name='Y2'))
fig.update_layout(title='Area Plot')
fig.show()
```

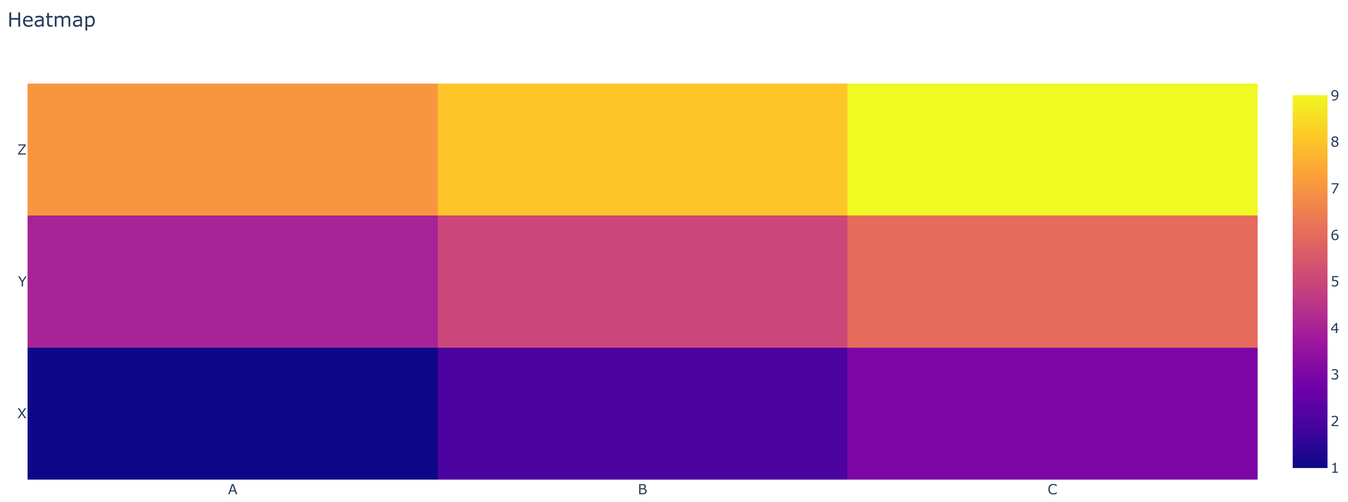


In this example, we create an area plot using the go.Scatter trace. We define two sets of y-values (y1 and y2) and specify the x-values (x). We use the mode parameter to set the line mode to 'lines', and the fill parameter to 'tozeroy' to fill the area under the lines. We add the traces to the figure and update the layout with a title before displaying the figure.

Heatmap

```
z = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]

fig = go.Figure(data=go.Heatmap(z=z, x=['A', 'B', 'C'], y=['X', 'Y', 'Z']))
fig.update_layout(title='Heatmap')
fig.show()
```

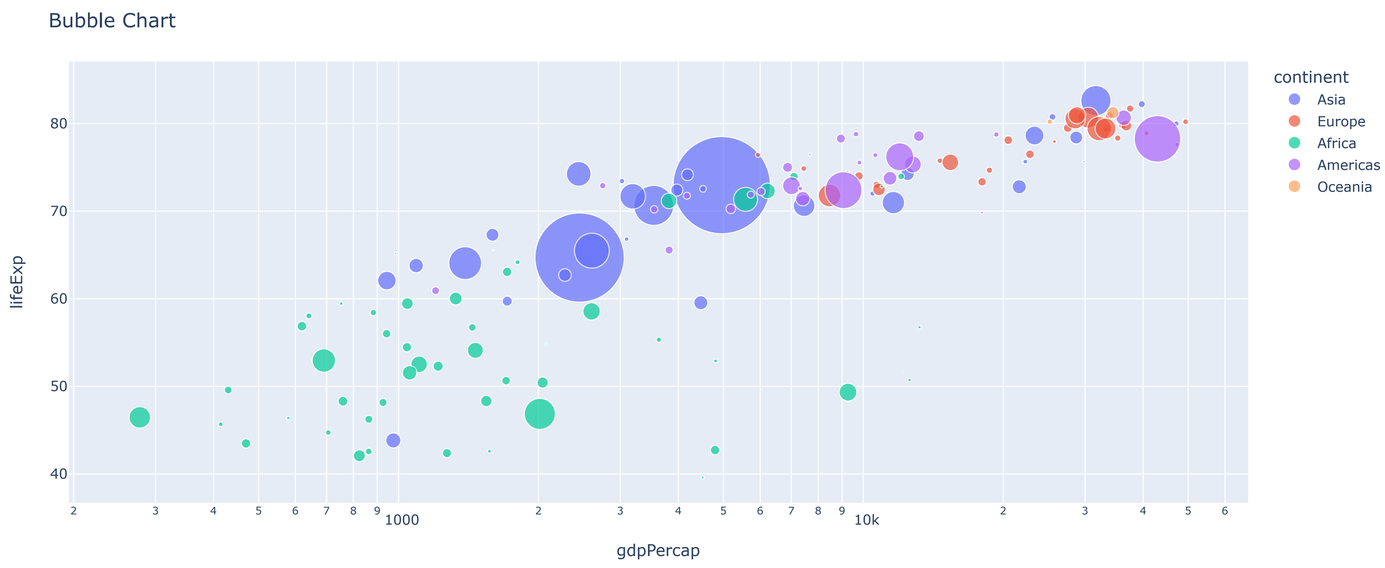


Here, we create a heatmap using the go.Heatmap trace. We specify the values (z) as a 2D array, and the x and y labels. We create a figure with the heatmap trace, update the layout with a title, and display the figure.

Bubble Chart

```
df = px.data.gapminder().query("year == 2007")

fig = px.scatter(df, x='gdpPercap', y='lifeExp', size='pop', color='continent', hover_name='country',
                 log_x=True, size_max=60)
fig.update_layout(title='Bubble Chart')
fig.show()
```

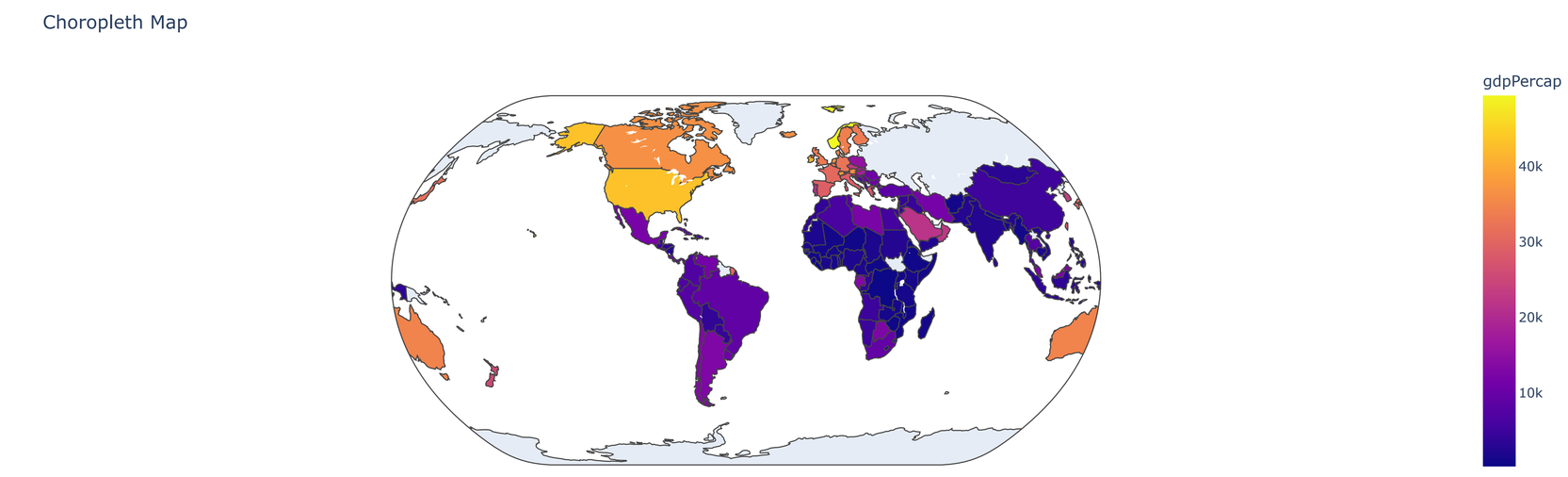


In this example, we create a bubble chart using the px.scatter function. We use the df DataFrame containing the gapminder dataset for the year 2007. We specify the x and y variables as 'gdpPercap' and 'lifeExp', respectively. We set the size of the bubbles based on the 'pop' column, color them based on the 'continent' column, and include the country names as hover information. We use the log_x parameter to set a logarithmic x-axis scale, and size_max to control the maximum size of the bubbles. We update the layout with a title and display the figure.

Choropleth Map

```
df = px.data.gapminder().query("year == 2007")

fig = px.choropleth(df, locations='iso_alpha', color='gdpPercap',
                    hover_name='country', projection='natural earth')
fig.update_layout(title='Choropleth Map')
fig.show()
```



In this example, we create a choropleth map using the px.choropleth function. We use the df DataFrame containing the gapminder dataset for the year 2007. We specify the locations as 'iso_alpha' and the color as 'gdpPercap'. We include the country names as hover information and set the projection to 'natural earth'. We update the layout with a title and display the figure.

Sankey Diagram

```
labels = ["A", "B", "C", "D", "E"]
source = [0, 1, 0, 2, 3]
target = [2, 3, 3, 4, 4]
value = [10, 15, 7, 12, 9]

fig = go.Figure(data=[go.Sankey(node=dict(label=labels), link=dict(source=source, target=target, value=value))])
fig.update_layout(title='Sankey Diagram')
fig.show()
```

Sankey Diagram

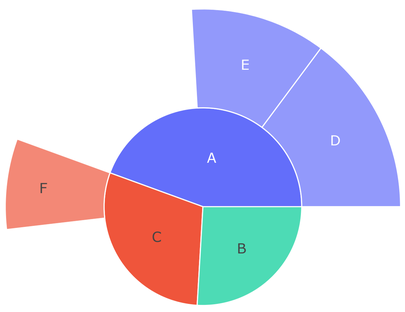
In this example, we create a Sankey diagram using the go.Sankey trace. We define the labels, sources, targets, and values for the diagram. We pass this data to the go.Figure constructor to create the figure. We then update the layout with a title before displaying the figure.

Sunburst Chart

```
import plotly.graph_objects as go
import plotly.figure_factory as ff
import plotly.express as px


fig = go.Figure(go.Sunburst(
    labels=["A", "B", "C", "D", "E", "F"],
    parents=["", "", "", "A", "A", "C"],
    values=[10, 14, 12, 8, 6, 4],
))
fig.update_layout(title='Sunburst Chart')
fig.show()
```

Sunburst Chart



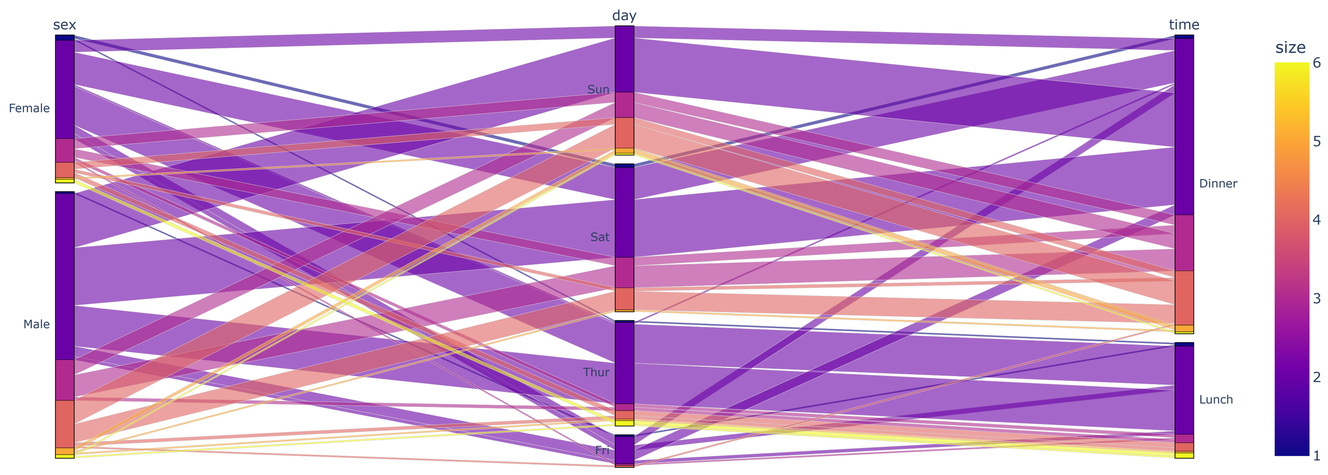
In this example, we create a Sunburst chart using the go.Sunburst trace. We define the labels, parents, and values for each level of the Sunburst chart. We pass this data to the go.Figure constructor to create the figure. We then update the layout with a title before displaying the figure.

Parallel Categories Diagram

```
df = px.data.tips()

fig = px.parallel_categories(df, dimensions=['sex', 'day', 'time'], color="size")
fig.update_layout(title='Parallel Categories Diagram')
fig.show()
```

Parallel Categories Diagram



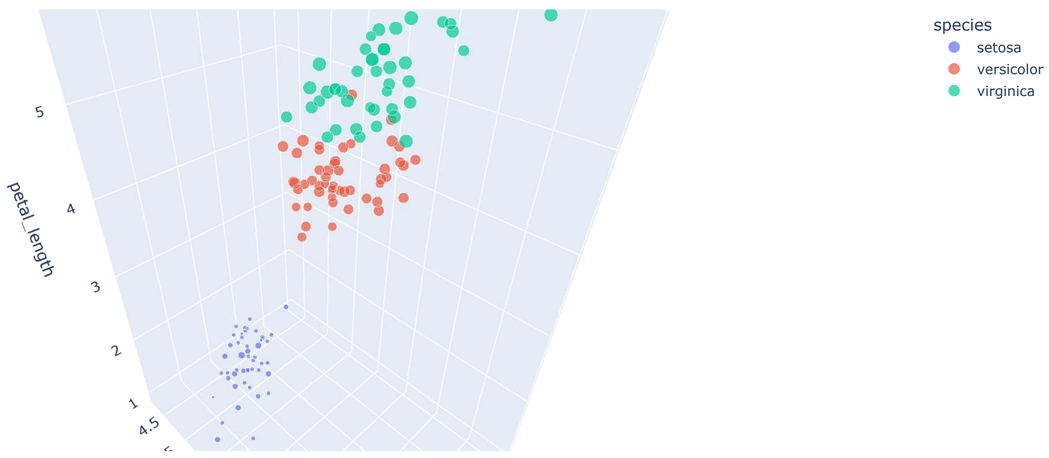
Here, we create a Parallel Categories diagram using the px.parallel_categories function. We use the df DataFrame containing the tips dataset. We specify the dimensions as ['sex', 'day', 'time'], and set the color based on the "size" column. We update the layout with a title and display the figure.

3D Scatter Plot with Colored Markers and Size

```
df = px.data.iris()

fig = px.scatter_3d(df, x="sepal_width", y="sepal_length", z="petal_length", color="species", size="petal_width")
fig.update_layout(title='3D Scatter Plot with Colored Markers and Size')
fig.show()
```

3D Scatter Plot with Colored Markers and Size



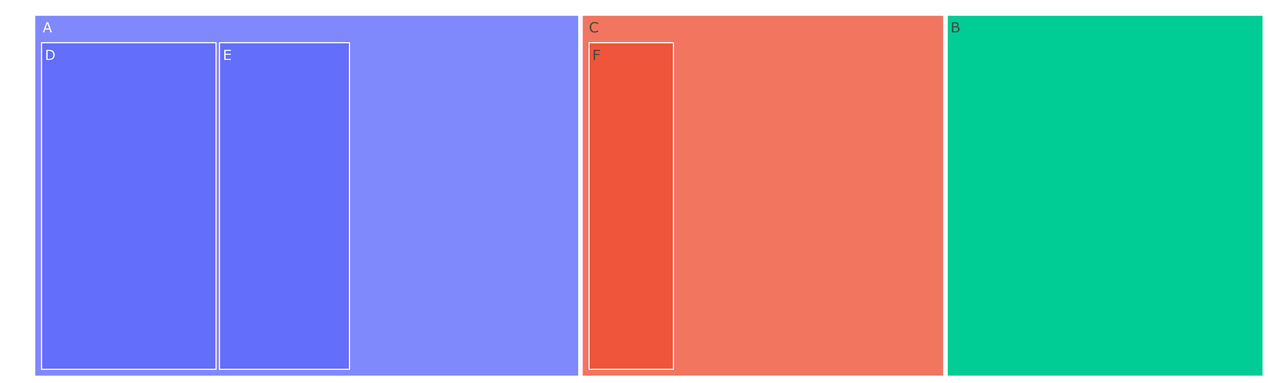
In this example, we create a 3D scatter plot with colored markers and size using the px.scatter_3d function. We use the df DataFrame containing the iris dataset. We specify the x, y, and z variables as "sepal_width", "sepal_length", and "petal_length", respectively. We color the markers based on the "species" column and set the size based on the "petal_width" column. We update the layout with a title and display the figure.

Treemap

```
import plotly.graph_objects as go
import plotly.figure_factory as ff
import plotly.express as px


labels = ["A", "B", "C", "D", "E", "F"]
parents = ["", "", "", "A", "A", "C"]
values = [10, 14, 12, 8, 6, 4]
```

```
fig = go.Figure(go.Treemap(
    labels=labels,
    parents=parents,
    values=values,
))
fig.update_layout(title='Treemap')
fig.show()
```

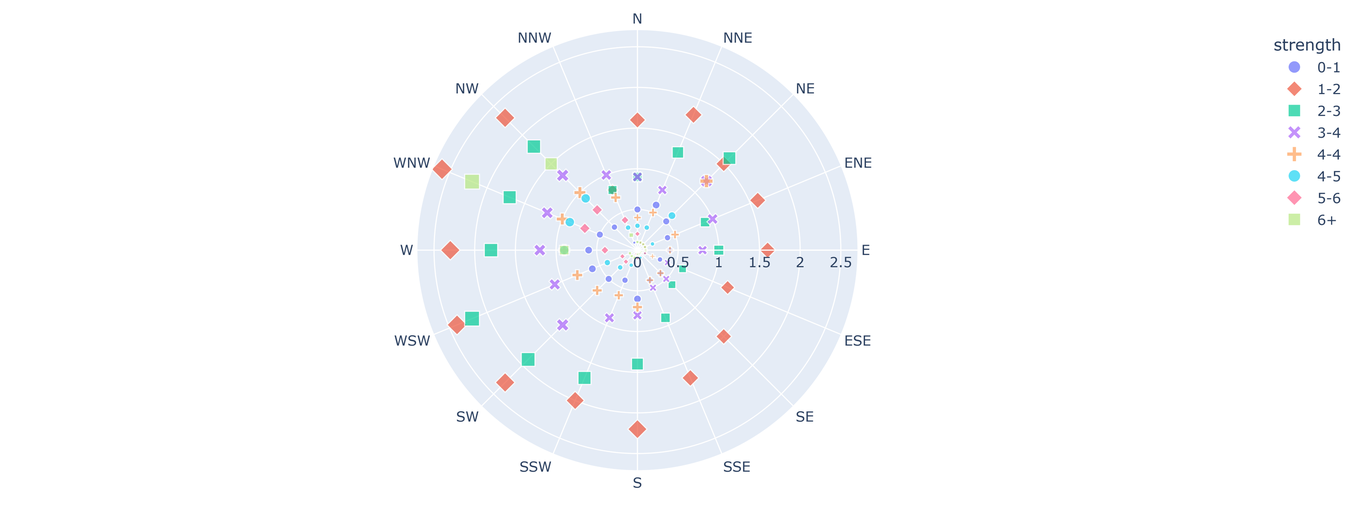


In this example, we create a Treemap using the go.Treemap trace. We define the labels, parents, and values for each level of the Treemap. We pass this data to the go.Figure constructor to create the figure. We then update the layout with a title before displaying the figure.

Polar Scatter Plot

```
df = px.data.wind()

fig = px.scatter_polar(df, r="frequency", theta="direction", color="strength",
                       symbol="strength", size="frequency", size_max=10)
fig.update_layout(title='Polar Scatter Plot')
fig.show()
```



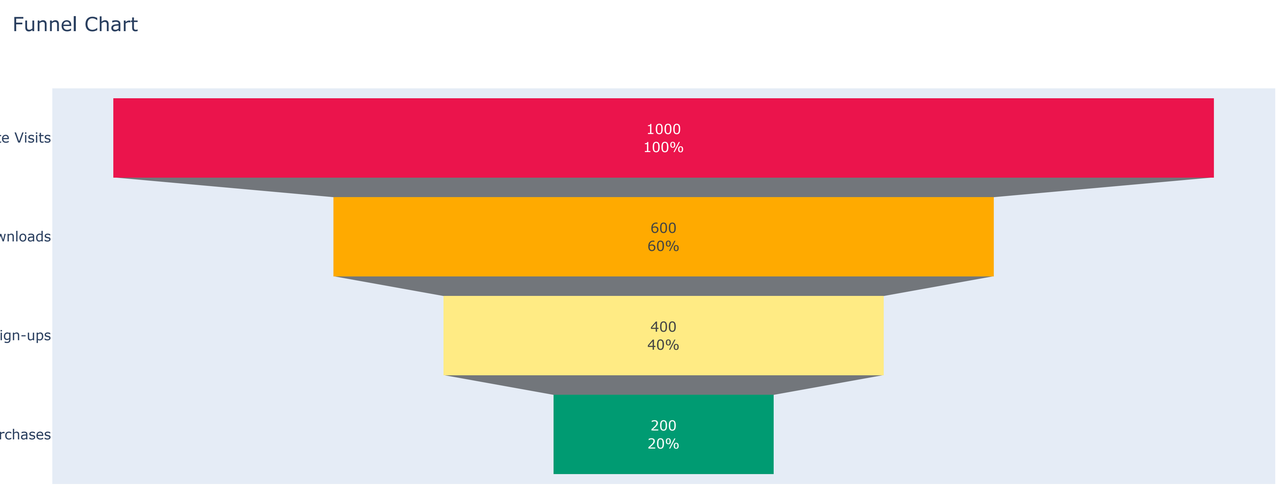
In this example, we create a Polar Scatter Plot using the px.scatter_polar function. We use the df DataFrame containing wind data. We specify the radial axis as "frequency" and the angular axis as "direction". We color the markers based on the "strength" column and use the "strength" column for symbol and size parameters. We update the layout with a title and display the figure.

Funnel Chart

```
import plotly.graph_objects as go
import plotly.figure_factory as ff
import plotly.express as px


labels = ["Website Visits", "Downloads", "Sign-ups", "Purchases"]
values = [1000, 600, 400, 200]

fig = go.Figure(go.Funnel(
    y=labels,
    x=values,
    textinfo="value+percent initial",
    marker=dict(color=["#EB144C", "#FFAA00", "#FFEB84", "#009B72"])
))
fig.update_layout(title='Funnel Chart')
fig.show()
```

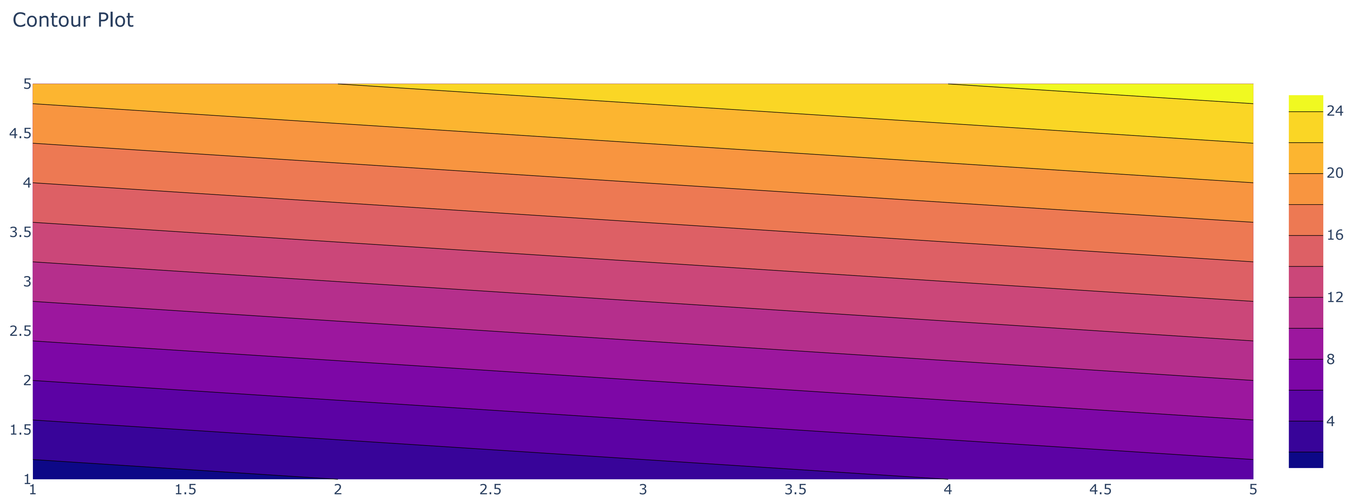


In this example, we create a Funnel chart using the go.Funnel trace. We define the labels and values for each stage of the funnel. We specify the textinfo parameter to display both the value and the percentage at each stage. We customize the colors of the funnel stages using the marker parameter. We update the layout with a title and display the figure.

Contour Plot

```
x = [1, 2, 3, 4, 5]
y = [1, 2, 3, 4, 5]
z = [[1, 2, 3, 4, 5],
     [6, 7, 8, 9, 10],
     [11, 12, 13, 14, 15],
     [16, 17, 18, 19, 20],
     [21, 22, 23, 24, 25]]

fig = go.Figure(data=[go.Contour(x=x, y=y, z=z)])
fig.update_layout(title='Contour Plot')
fig.show()
```



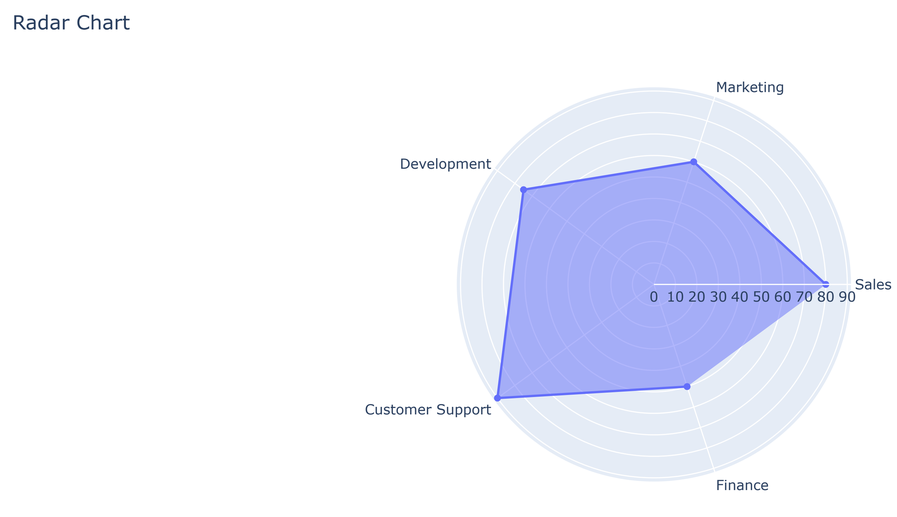
Here, we create a Contour plot using the go.Contour trace. We define the x, y, and z values as lists and a 2D array, respectively. We pass this data to the go.Figure constructor to create the figure. We then update the layout with a title before displaying the figure.

Radar Chart

```
import plotly.graph_objects as go
import plotly.figure_factory as ff
import plotly.express as px

categories = ['Sales', 'Marketing', 'Development', 'Customer Support', 'Finance']
values = [80, 60, 75, 90, 50]

fig = go.Figure(data=go.Scatterpolar(
    r=values,
    theta=categories,
    fill='toself'
))
fig.update_layout(polar=dict(
    radialaxis=dict(visible=True)),
    title='Radar Chart'
)
fig.show()
```



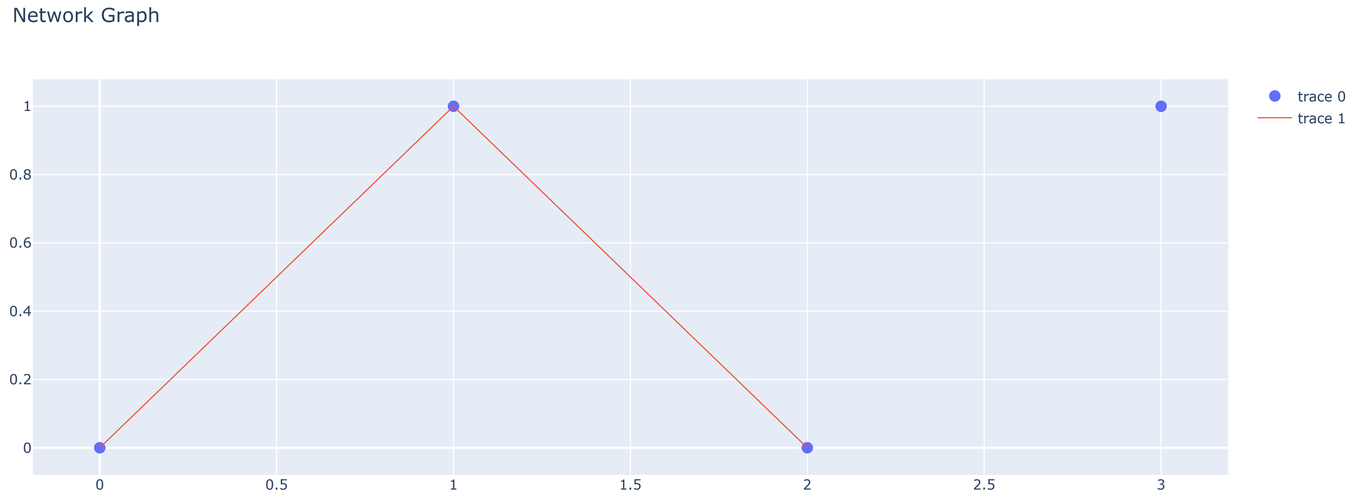
In this example, we create a Radar chart using the go.Scatterpolar trace. We define the categories (axes) and corresponding values for each category. We pass this data to the go.Figure constructor to create the figure. We use fill='toself' to connect the data points and fill the area. We update the layout with a polar configuration and a title before displaying the figure.

Network Graph

```
nodes = [
    go.Scatter(
        x=[0, 1, 2, 3],
        y=[0, 1, 0, 1],
        mode='markers',
        marker=dict(size=10),
        text=['Node A', 'Node B', 'Node C', 'Node D']
    )
]

edges = [
    go.Scatter(
        x=[0, 1, 2],
        y=[0, 1, 0],
        mode='lines',
        line=dict(width=1),
        hoverinfo='none'
    )
]

fig = go.Figure(data=nodes + edges)
fig.update_layout(title='Network Graph')
fig.show()
```



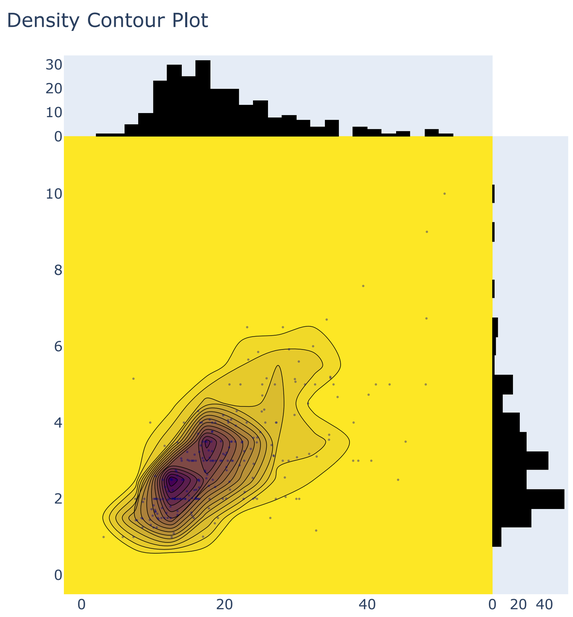
Here, we create a Network graph using the go.Scatter trace. We define the nodes and their coordinates as scatter points, and the edges connecting the nodes as lines. We pass this data to the go.Figure constructor to create the figure. We update the layout with a title before displaying the figure.

Density Contour Plot

```
df = px.data.tips()

fig = ff.create_2d_density(
    x=df['total_bill'],
    y=df['tip'],
    colorscale='Viridis'
```

```
    colorscale='Viridis',
    hist_color='rgba(0,0,0,0)'
)
fig.update_layout(title='Density Contour Plot')
fig.show()
```



In this example, we create a Density contour plot using the ff.create_2d_density function from plotly.figure_factory. We pass in the x and y values from a DataFrame (df). We specify the colorscale and set hist_color to 'rgba(0,0,0,0)' to make the histogram background transparent. We update the layout with a title before displaying the figure.

1s completed at 4:29 PM