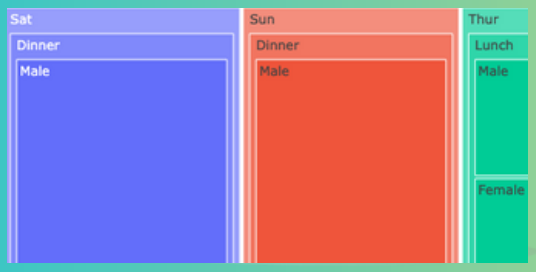
# Visualizing Data with Plotly Library

Transforming data into stunning interactive visual insights.




**Dive into the world of interactive data visualizations with Plotly's Python library, perfect for creating engaging and dynamic charts and graphs.**

PLOTLY
VISUALS

SOLOMON AKINTOLA

# Plotly

Plotly is a powerful and versatile open-source library used for creating interactive and visually appealing data visualizations in Python. It is widely utilized for its ability to generate a variety of high-quality charts and plots, making it an essential tool for data scientists, analysts, and engineers. Plotly is known for its ease of use, flexibility, and the capability to create complex and interactive visualizations with minimal effort.

## Key Features of Plotly:

1. **Interactive Visualizations**:
    - Plotly enables users to create interactive plots that can be embedded in web applications or Jupyter notebooks. Users can zoom, pan, and hover over data points to see additional information.
2. **Wide Range of Plot Types**:
    - Plotly supports a comprehensive range of chart types including but not limited to scatter plots, line charts, bar charts, histograms, box plots, pie charts, heatmaps, choropleth maps, and 3D charts. This wide variety allows users to choose the most appropriate visualization for their data.
3. **High Customizability**:
    - Plotly offers extensive customization options for every aspect of the plot, including colors, labels, axes, and annotations. This allows users to tailor visualizations to their specific needs and preferences.
4. **Integration with Pandas and NumPy**:
    - Plotly works seamlessly with popular data manipulation libraries like Pandas and NumPy, making it easy to convert DataFrame objects into Plotly visualizations.
5. **Compatibility with Dash**:
    - Plotly is fully compatible with Dash, a Python framework for building analytical web applications. This integration enables the creation of highly interactive and dynamic web-based data visualizations.
6. **Open Source and Community Support**:
    - Plotly is open-source and has a large and active community. This ensures continuous improvement, extensive documentation, and a wealth of resources for users to learn from and contribute to.

## Example Use Cases:

1. **Data Exploration and Analysis**:
   - Plotly is ideal for exploratory data analysis (EDA). Users can quickly visualize and understand data distributions, relationships, and patterns.
2. **Business Dashboards**:
   - Businesses use Plotly to create dashboards that display key performance indicators (KPIs) and other critical metrics in real-time, enabling better decision-making.
3. **Scientific Research**:
   - Researchers and scientists use Plotly to visualize experimental results, analyze data trends, and present findings in a clear and interactive manner.
4. **Educational Tools**:
   - Plotly is used in educational settings to teach data visualization and analysis, helping students and professionals alike to develop their data science skills.

## Basic and few advanced examples

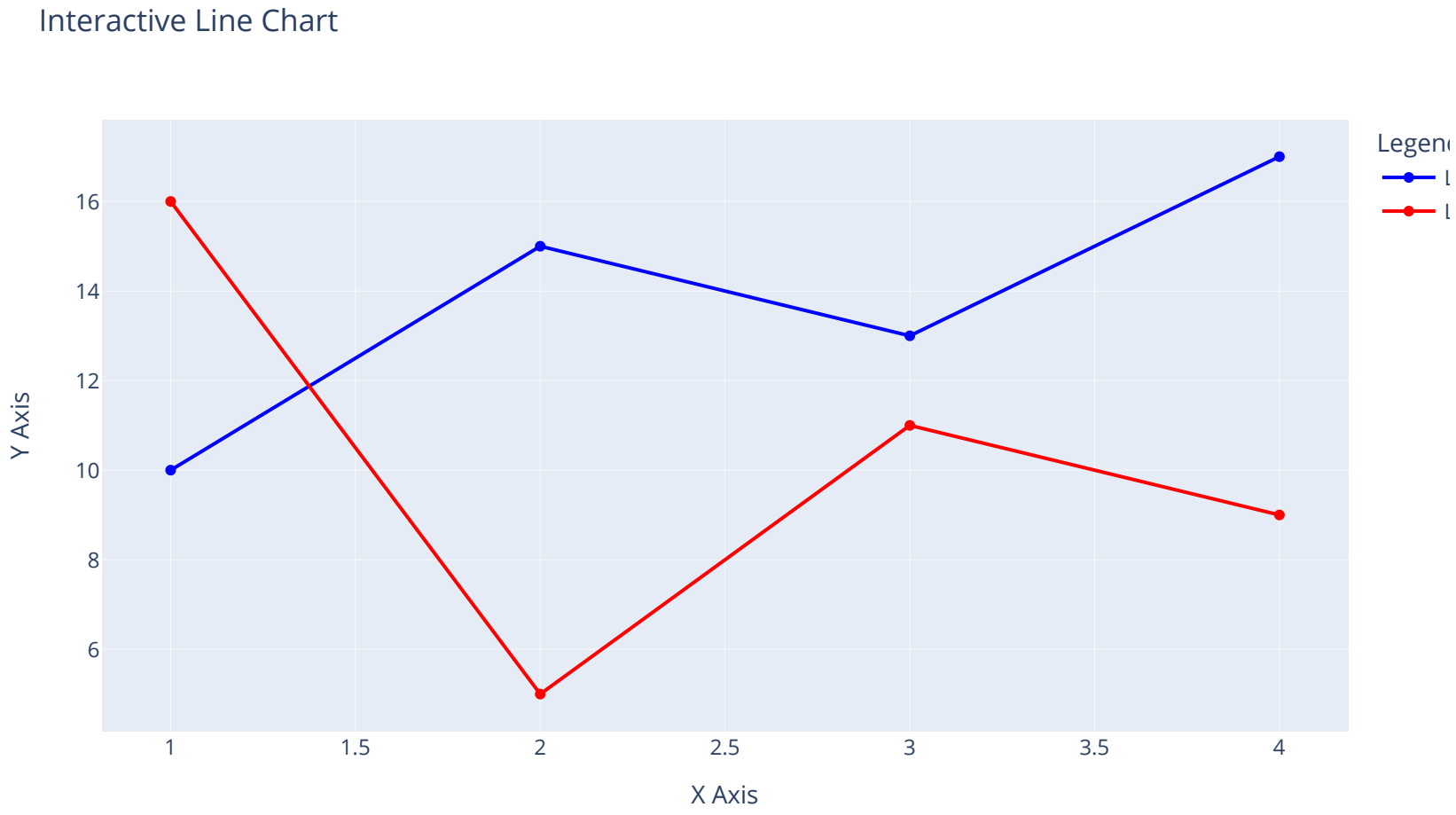
In [1]:
```
import numpy as np
import plotly.graph_objects as go
import plotly.express as px
```

### 1. Line Chart

- **Category**: Time Series Analysis, Trend Analysis
- **Usage**: Used to display trends over time.

In [2]:
```
# Create traces
trace1 = go.Scatter(
    x=[1, 2, 3, 4],
    y=[10, 15, 13, 17],
    mode='lines+markers',
    name='Line 1',
```

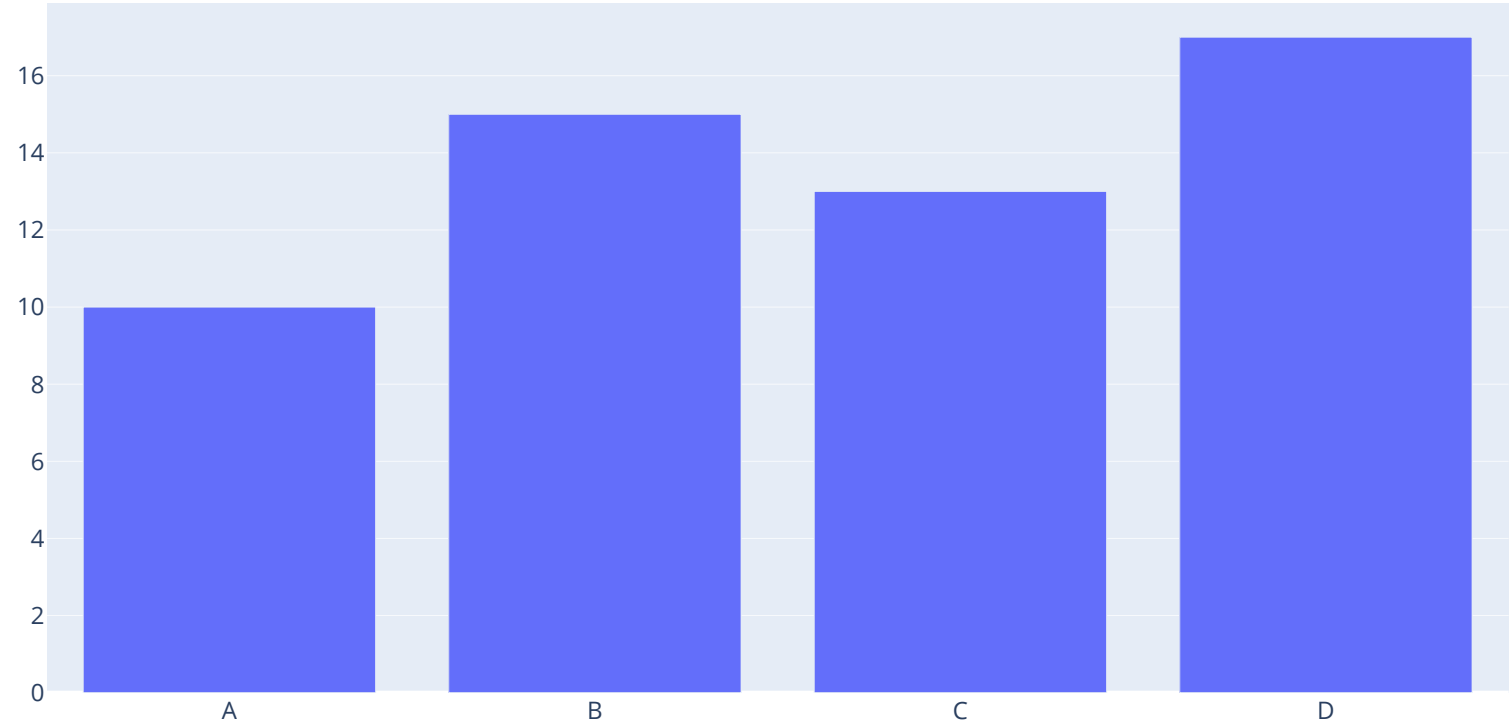
```
    line=dict(color='blue', width=2)
)

trace2 = go.Scatter(
    x=[1, 2, 3, 4],
    y=[16, 5, 11, 9],
    mode='lines+markers',
    name='Line 2',
    line=dict(color='red', width=2)
)

# Create a figure
fig = go.Figure(data=[trace1, trace2])

# Update layout
fig.update_layout(
    title='Interactive Line Chart',
    xaxis_title='X Axis',
    yaxis_title='Y Axis',
    legend_title='Legend',
    hovermode='x unified'
)

# Show the figure
fig.show()
```




## 2. **Bar Chart**

- **Category**: Categorical Data Analysis
- **Usage**: Used to compare quantities across categories.

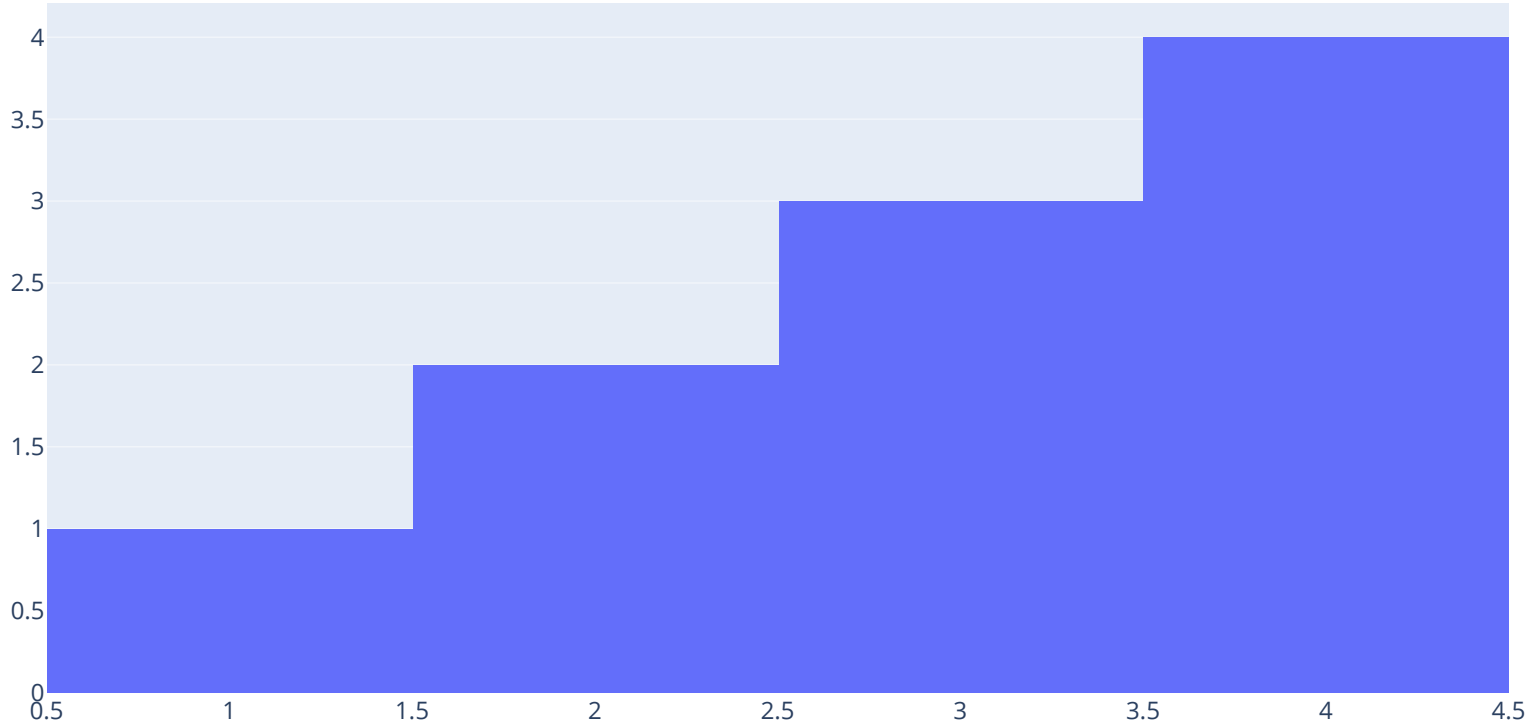
In [3]:
```
fig = go.Figure(data=go.Bar(x=['A', 'B', 'C', 'D'], y=[10, 15, 13, 17]))
fig.show()
```




### 3. **Histogram**

- **Category**: Distribution Analysis
- **Usage**: Used to visualize the distribution of a dataset.

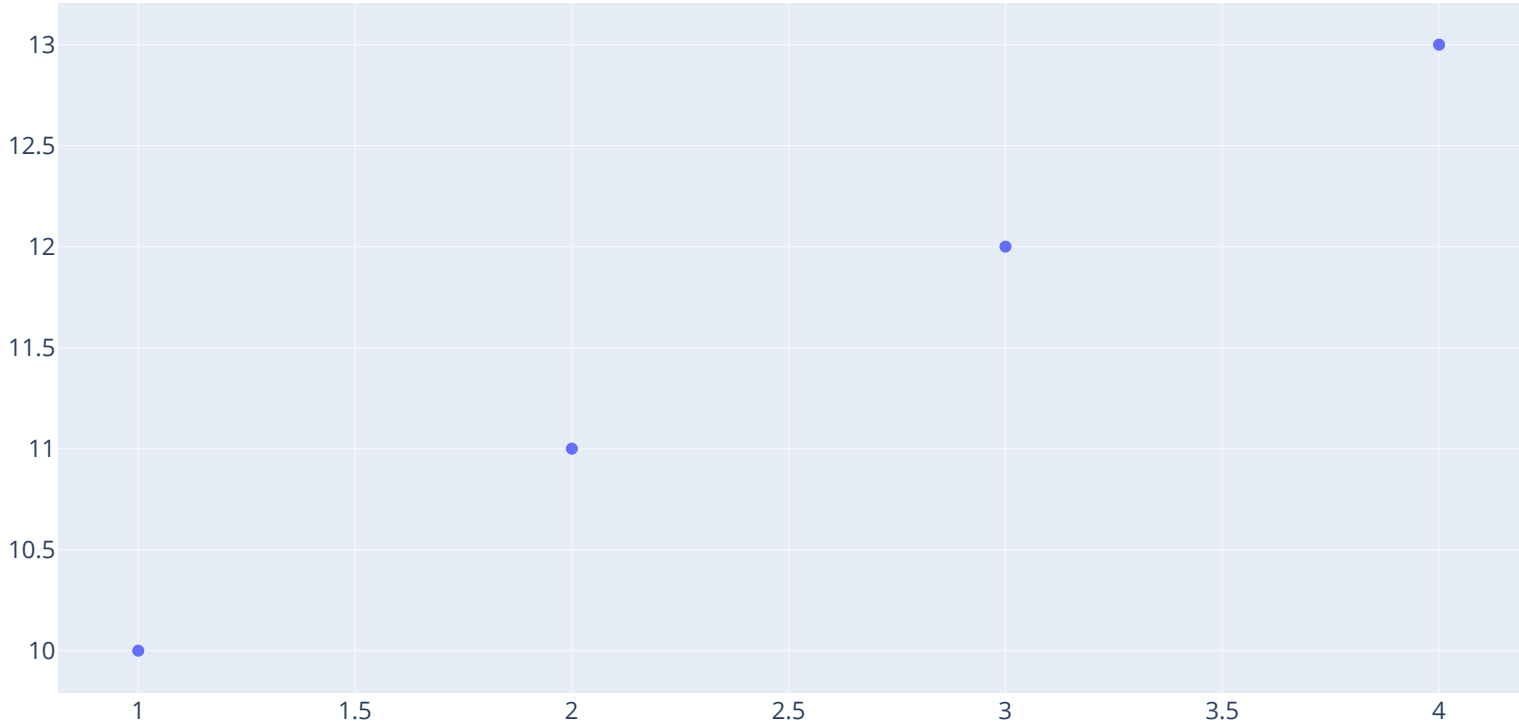
In [4]:
```
fig = go.Figure(data=go.Histogram(x=[1, 2, 2, 3, 3, 3, 4, 4, 4, 4]))
fig.show()
```




### 4. Scatter Plot

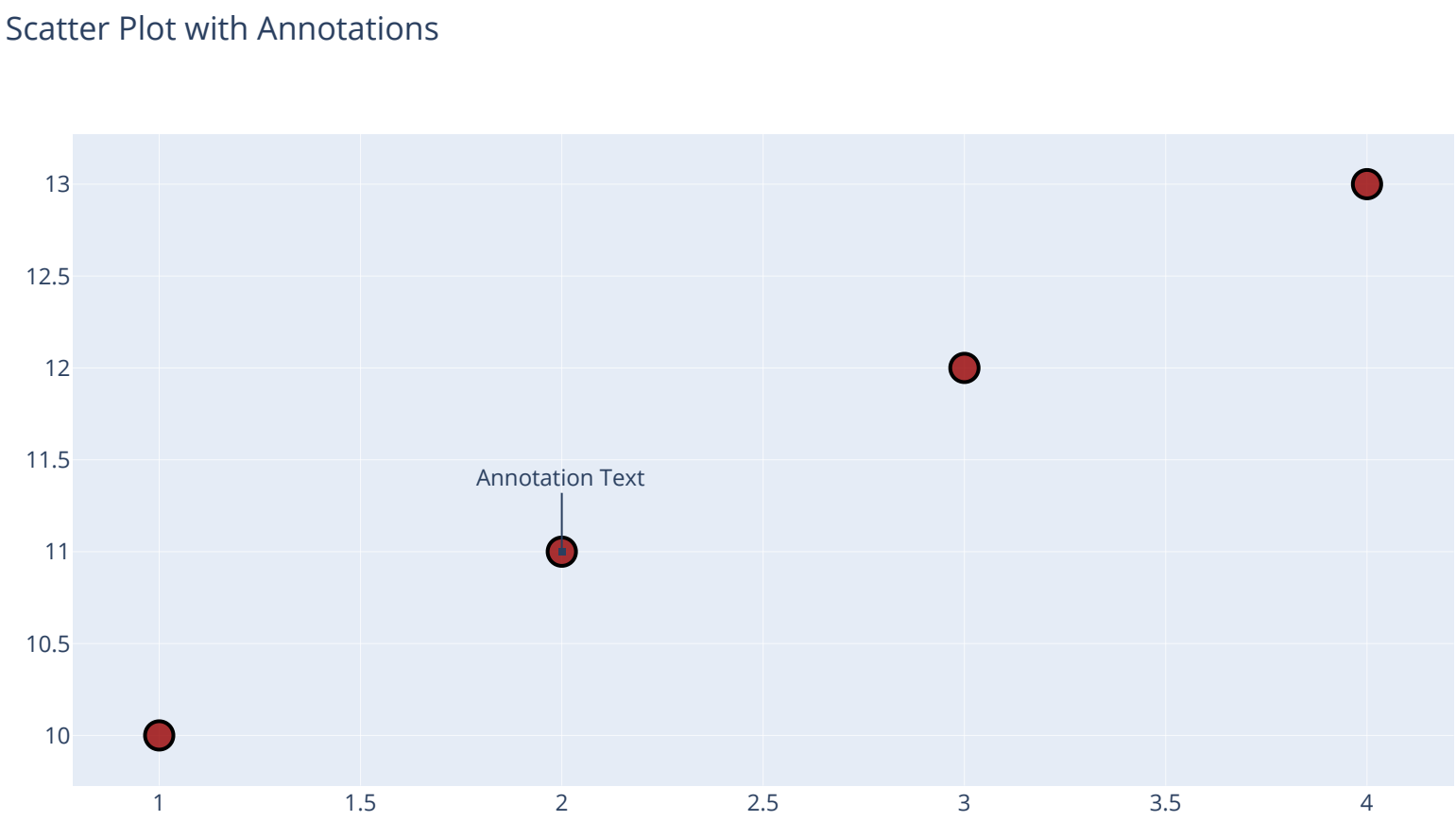
- **Category**: Data Visualization, Statistical Analysis
- **Usage**: Used to observe relationships between variables.

In [5]:
```
fig = go.Figure(data=go.Scatter(x=[1, 2, 3, 4], y=[10, 11, 12, 13], mode='markers'))
fig.show()
```




In [6]:
```
#Advance Scattered plot with annotation

# Create data
trace = go.Scatter(
    x=[1, 2, 3, 4],
    y=[10, 11, 12, 13],
    mode='markers',
    marker=dict(
        size=15,
        color='rgba(152, 0, 0, .8)',
        line=dict(
```

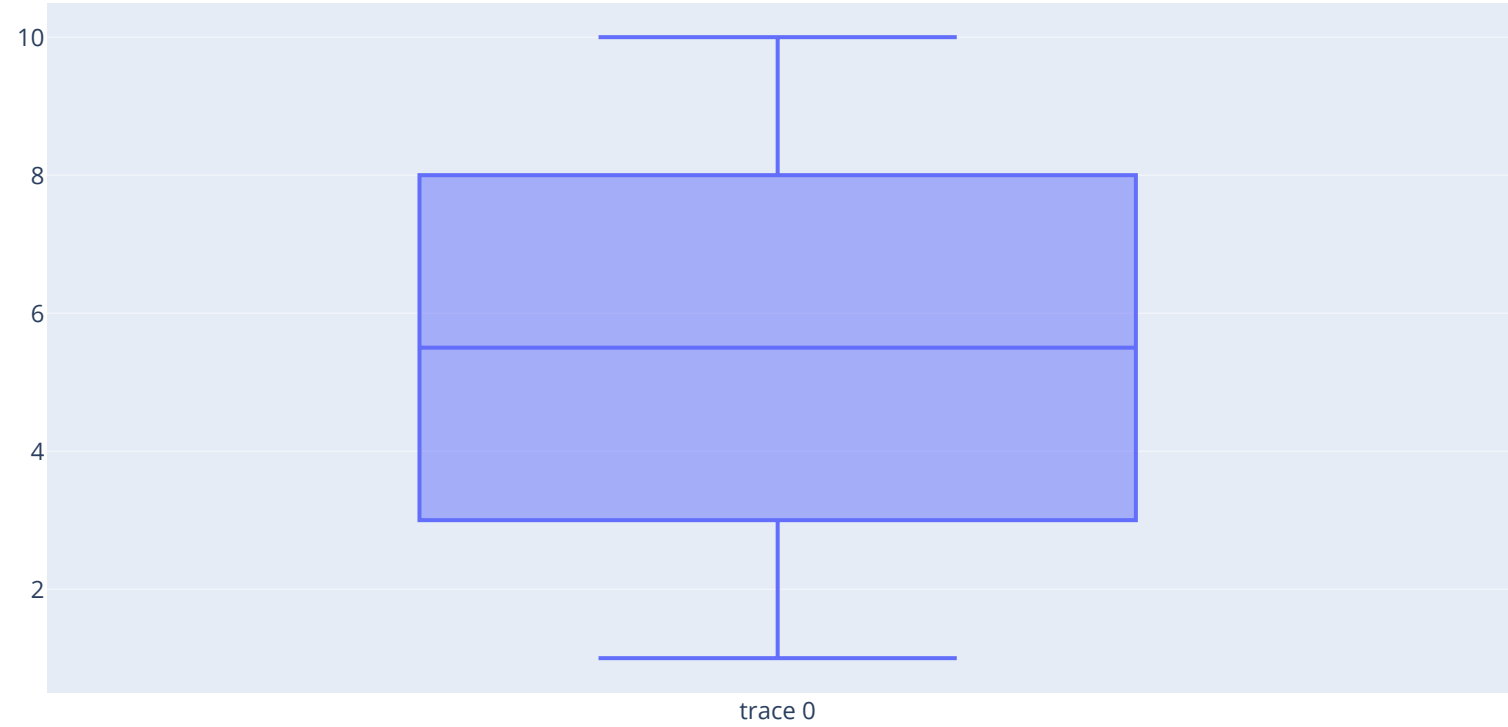
```
            width=2,
            color='rgb(0, 0, 0)'
        )
    )
)

# Create a figure
fig = go.Figure(data=[trace])

# Add annotations
fig.update_layout(
    title='Scatter Plot with Annotations',
    annotations=[
        dict(
            x=2,
            y=11,
            xref='x',
            yref='y',
            text='Annotation Text',
            showarrow=True,
            arrowhead=7,
            ax=0,
            ay=-40
        )
    ]
)

# Show the figure
fig.show()
```




### 5. **Box Plot**

- **Category**: Statistical Analysis
- **Usage**: Used to display the distribution of data based on a five-number summary.

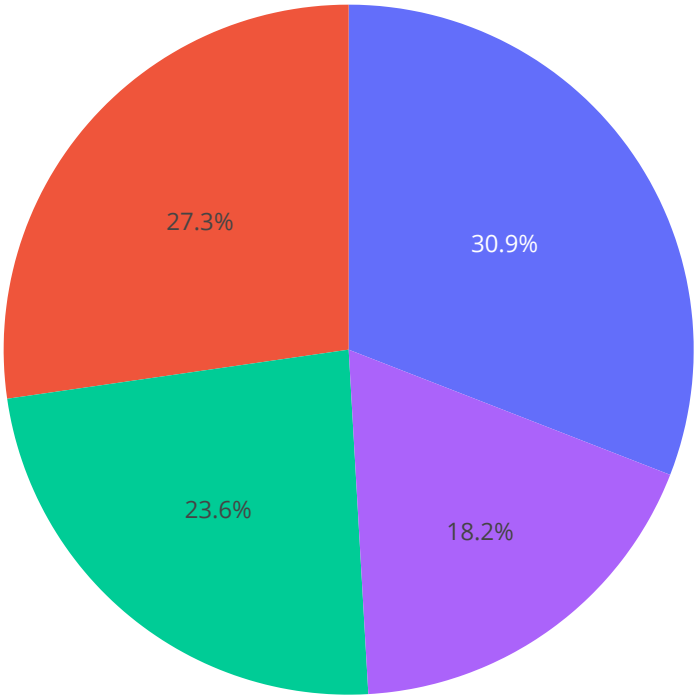
In [7]:
```
fig = go.Figure(data=go.Box(y=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))
fig.show()
```




### 6. **Pie Chart**

- **Category**: Proportional Data Analysis
- **Usage**: Used to show proportions of a whole.

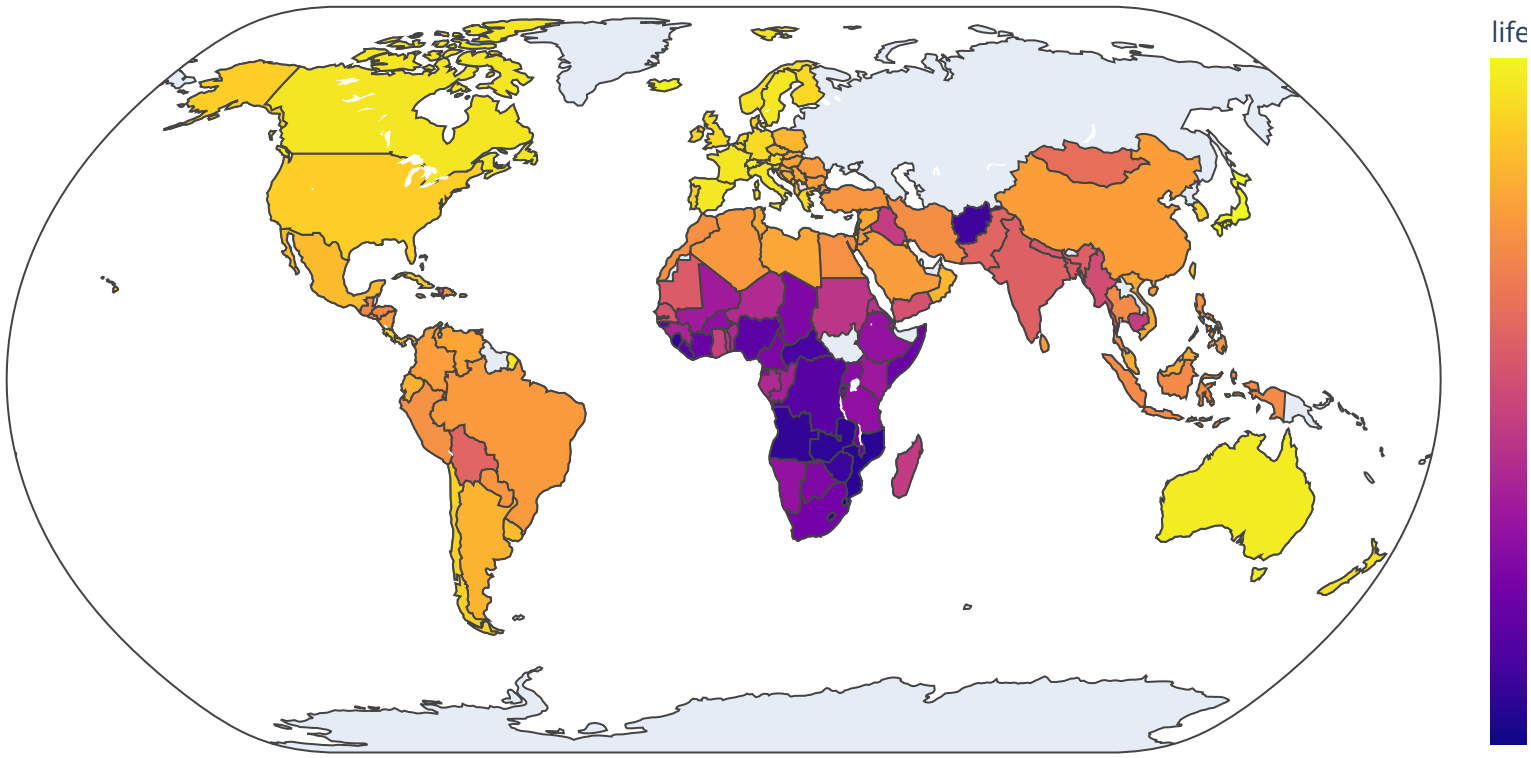
In [8]:
```
fig = go.Figure(data=go.Pie(labels=['A', 'B', 'C', 'D'], values=[10, 15, 13, 17]))
fig.show()
```




### 7. **Choropleth Map**

- **Category**: Geographic Data Visualization
- **Usage**: Used to represent the geographic distribution of data.

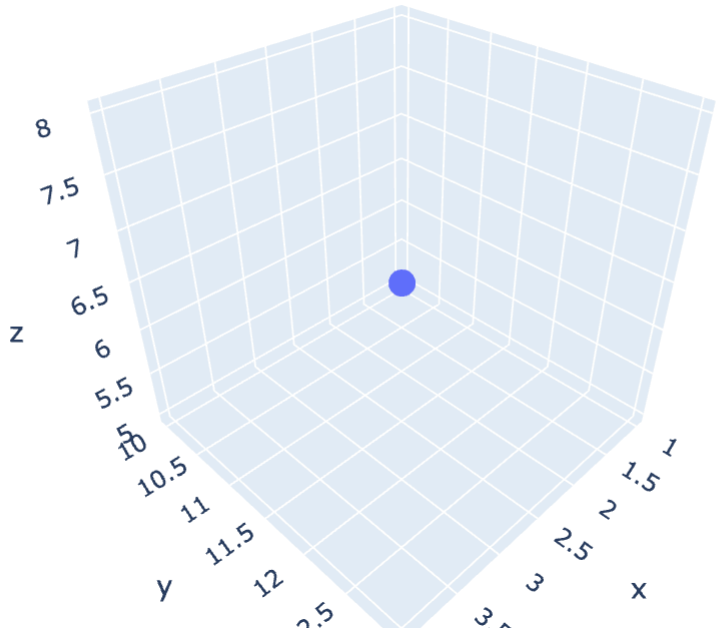
In [9]:
```
df = px.data.gapminder().query("year == 2007")
fig = px.choropleth(df, locations="iso_alpha", color="lifeExp", hover_name="country", projection="natural ear
fig.show()
```




### 8. **3D Scatter Plot**

- **Category**: Multi-dimensional Data Visualization
- **Usage**: Used to observe relationships between three variables.

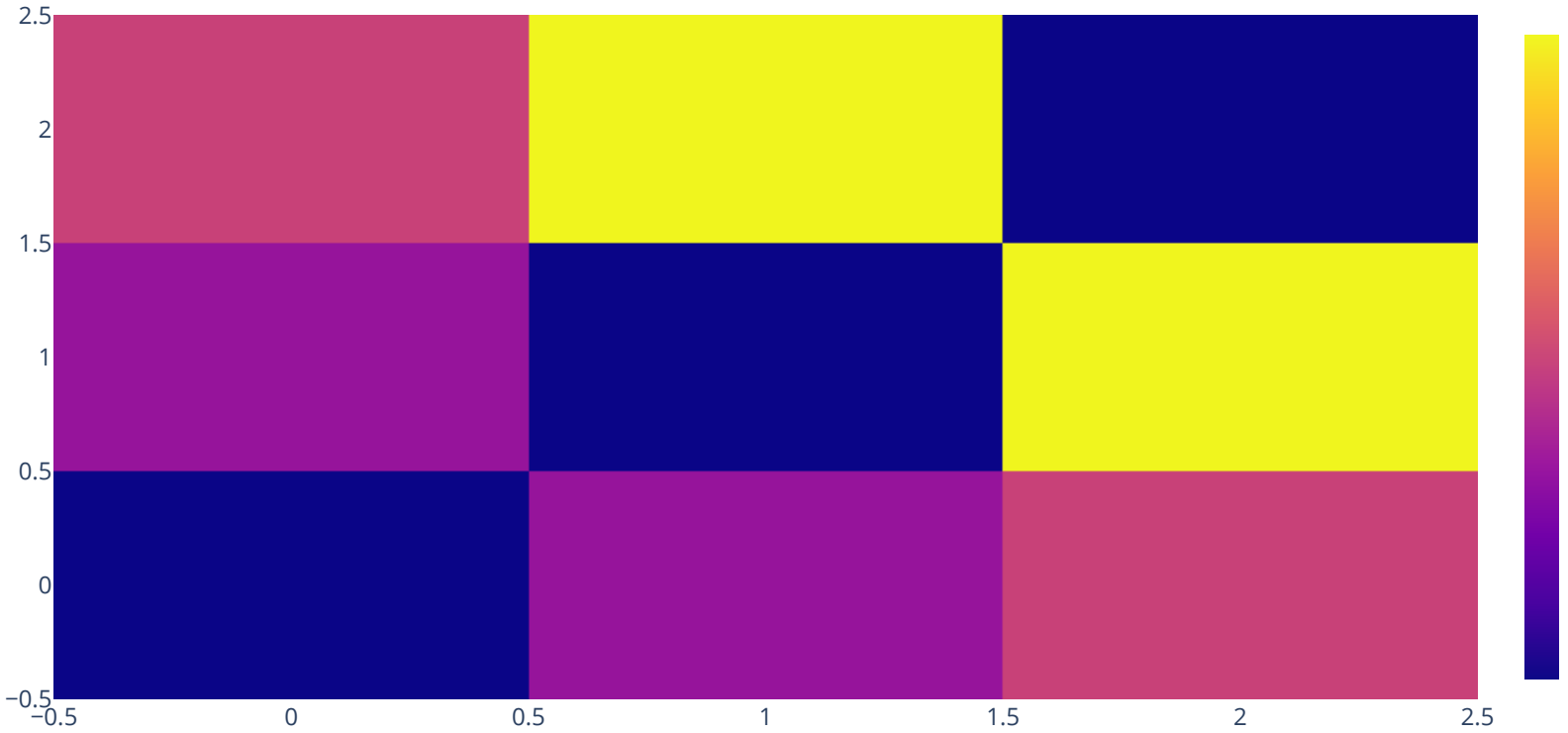
In [10]:
```
fig = go.Figure(data=[go.Scatter3d(x=[1, 2, 3, 4], y=[10, 11, 12, 13], z=[5, 6, 7, 8], mode='markers')])
fig.show()
```




### 9. **Heatmap**

- **Category**: Correlation Analysis, Data Intensity Visualization
- **Usage**: Used to represent data through variations in coloring.

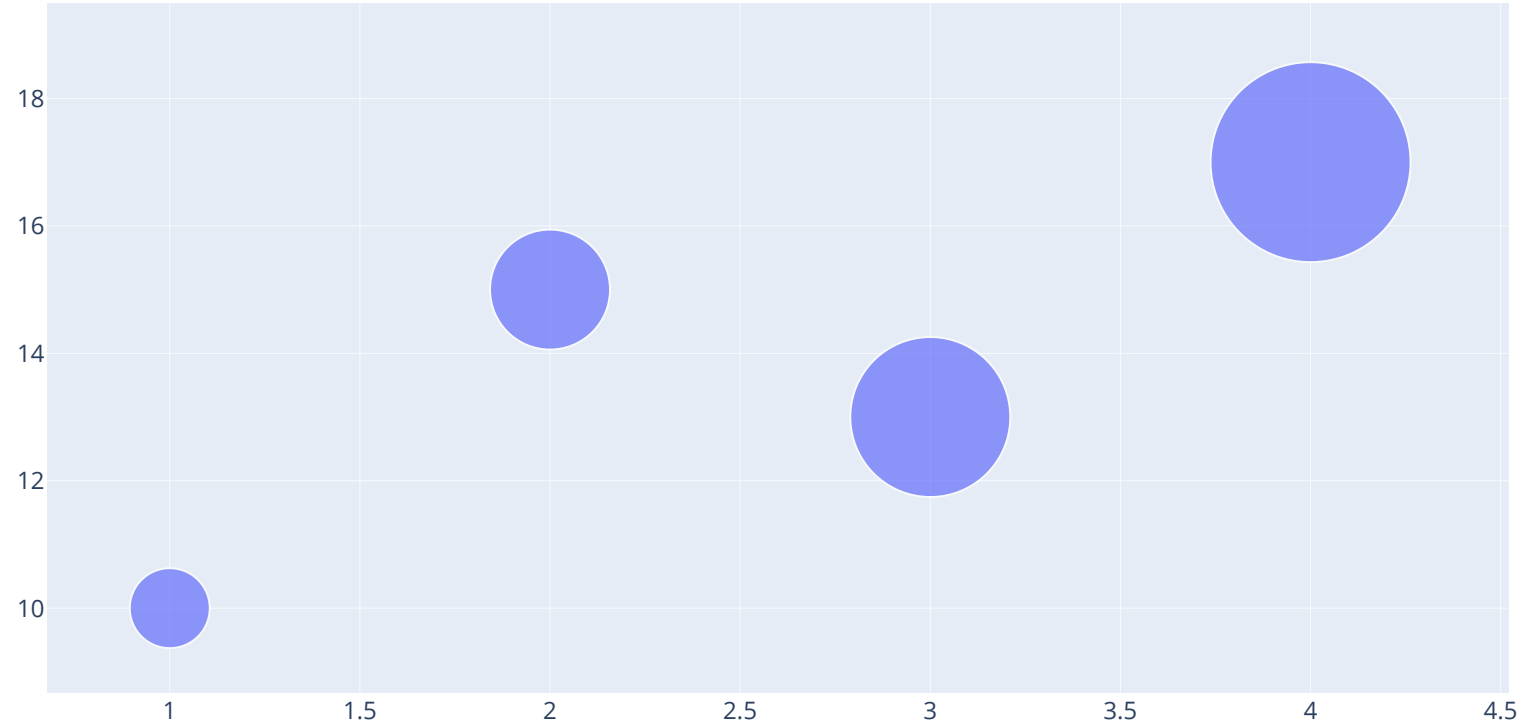
In [11]:
```
fig = go.Figure(data=go.Heatmap(z=[[1, 20, 30], [20, 1, 60], [30, 60, 1]]))
fig.show()
```




### 10. **Bubble Chart**

- **Category**: Comparative Analysis
- **Usage**: Used to compare entities based on three variables.

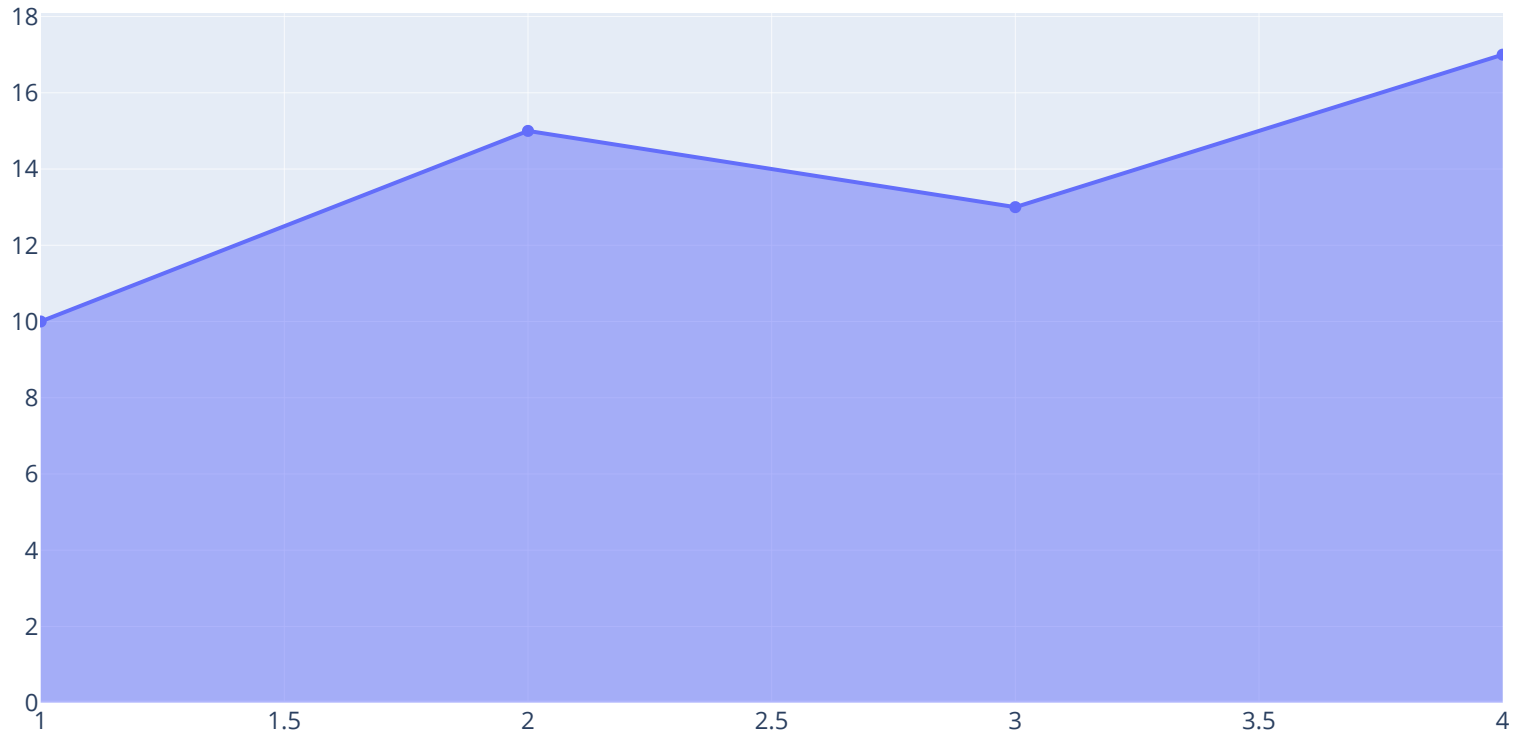
In [12]:
```
fig = go.Figure(data=go.Scatter(x=[1, 2, 3, 4], y=[10, 15, 13, 17],
                                mode='markers',
                                marker=dict(size=[40, 60, 80, 100])))
fig.show()
```




### 11. **Area Chart**

- **Category**: Trend Analysis
- **Usage**: Used to display cumulative totals over time.

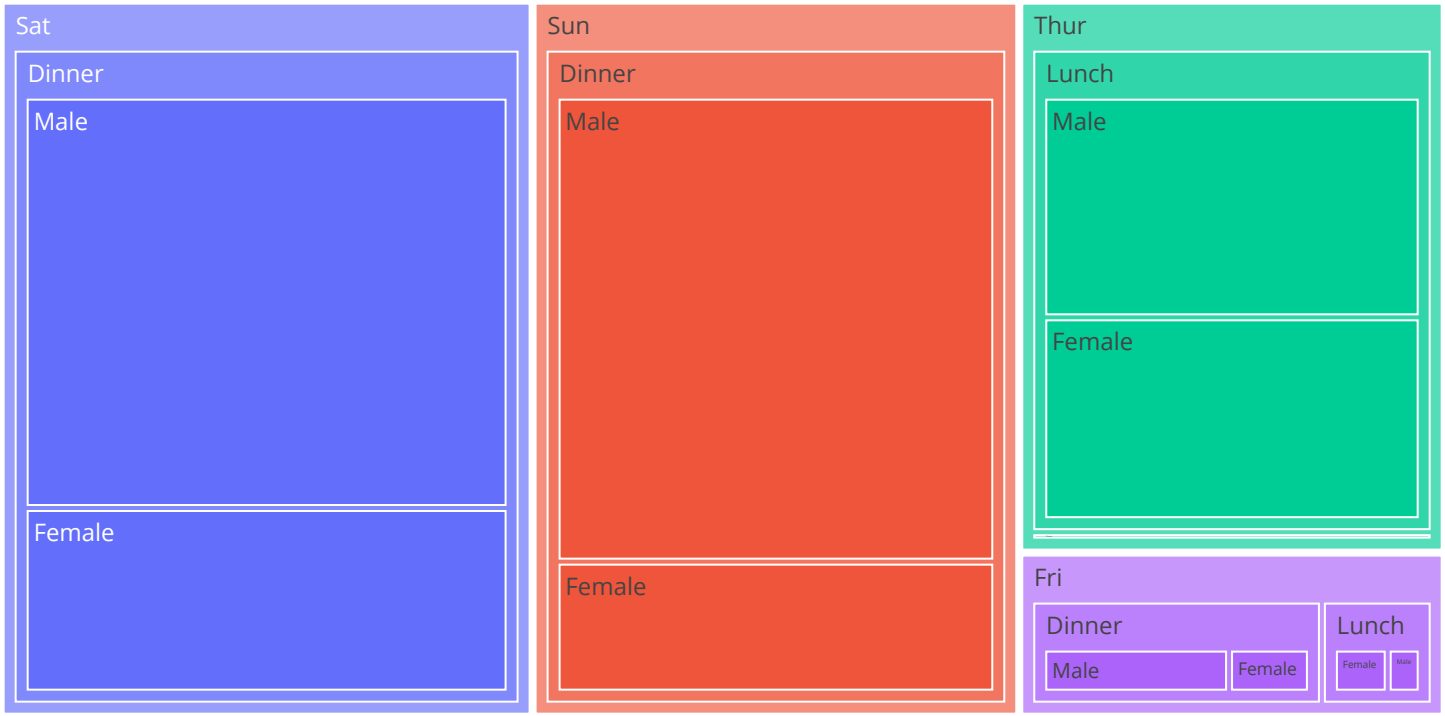
In [13]:
```
fig = go.Figure(data=go.Scatter(x=[1, 2, 3, 4], y=[10, 15, 13, 17], fill='tozeroy'))
fig.show()
```




## 12. Treemap

- **Category**: Hierarchical Data Visualization
- **Usage**: Used to visualize hierarchical data through nested rectangles.

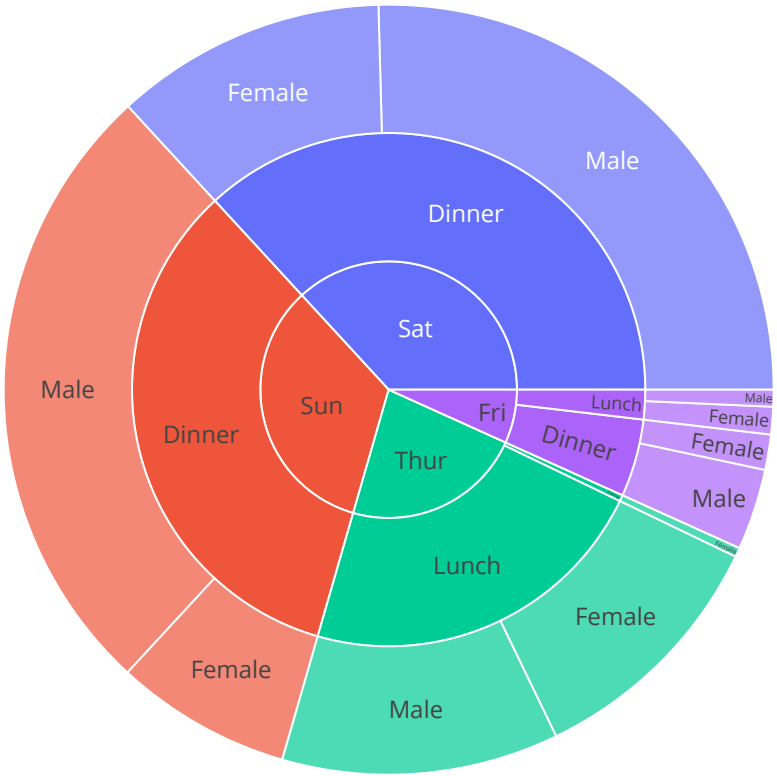
In [14]:
```
df = px.data.tips()
fig = px.treemap(df, path=['day', 'time', 'sex'], values='total_bill')
fig.show()
```




### 13. **Sunburst Chart**

- **Category**: Hierarchical Data Visualization
- **Usage**: Used to visualize hierarchical data through concentric circles.

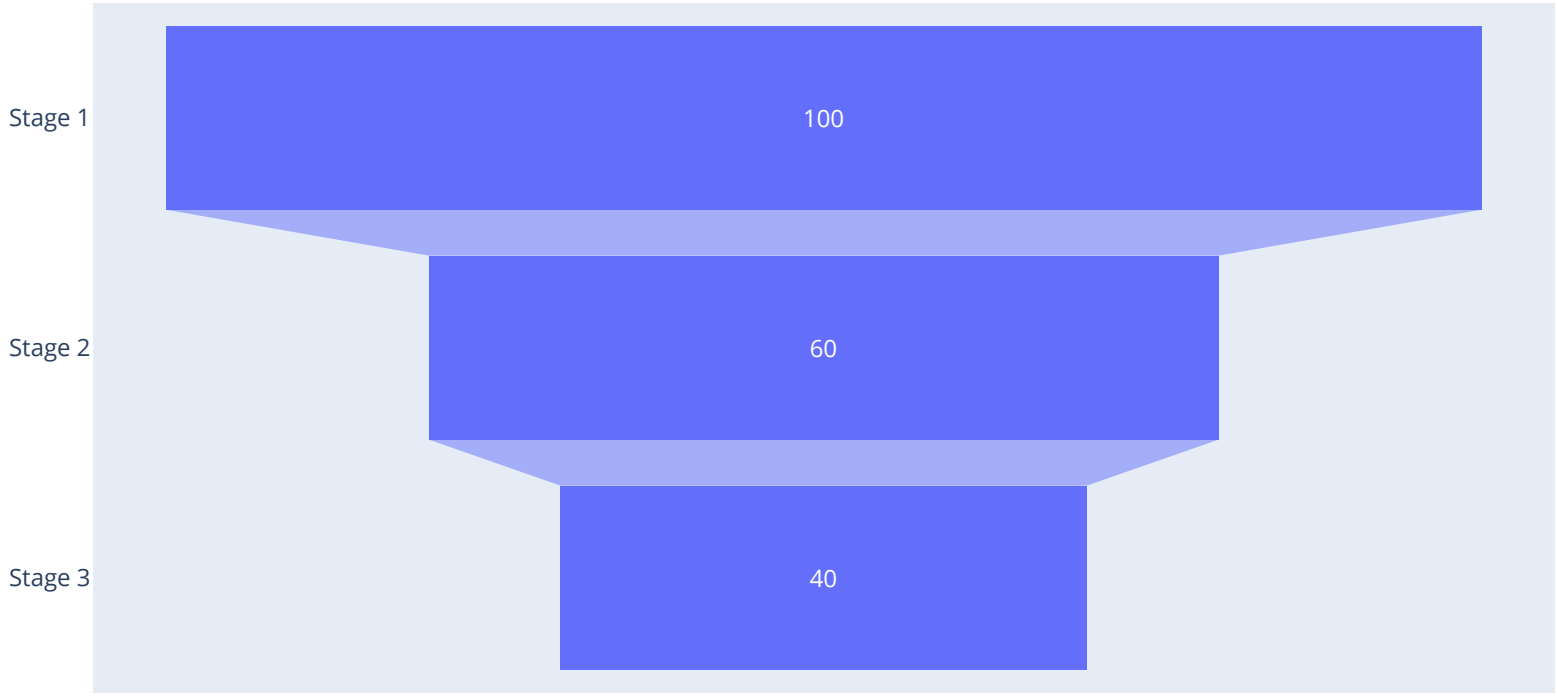
In [15]:
```
df = px.data.tips()
fig = px.sunburst(df, path=['day', 'time', 'sex'], values='total_bill')
fig.show()
```




## 14. **Funnel Chart**

- **Category**: Process Visualization
- **Usage**: Used to visualize stages in a process.

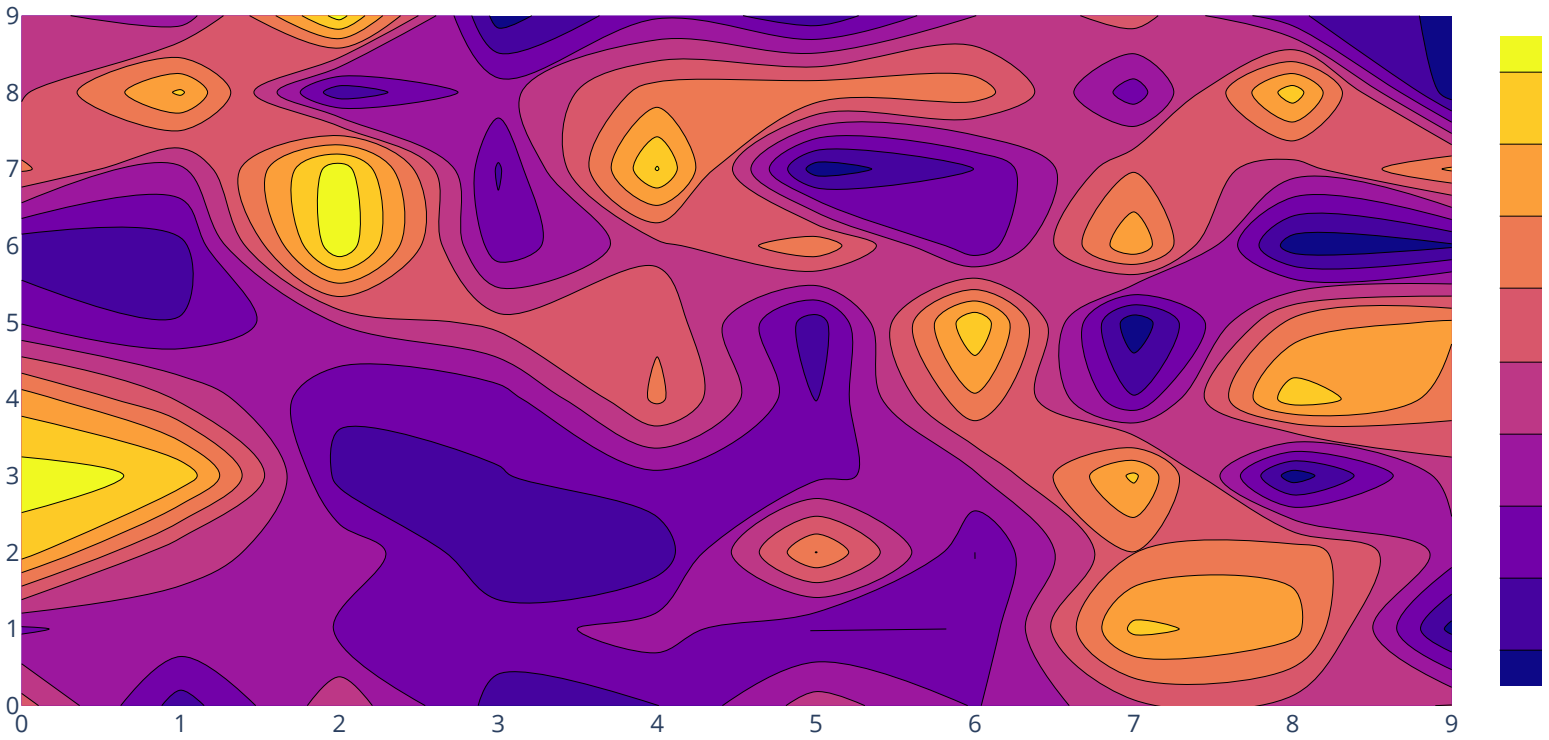
In [16]:
```
fig = go.Figure(go.Funnel(y=['Stage 1', 'Stage 2', 'Stage 3'], x=[100, 60, 40]))
fig.show()
```




### 15. **Contour Plot**

- **Category**: Statistical Analysis
- **Usage**: Used to represent three-dimensional data in two dimensions using contour lines.

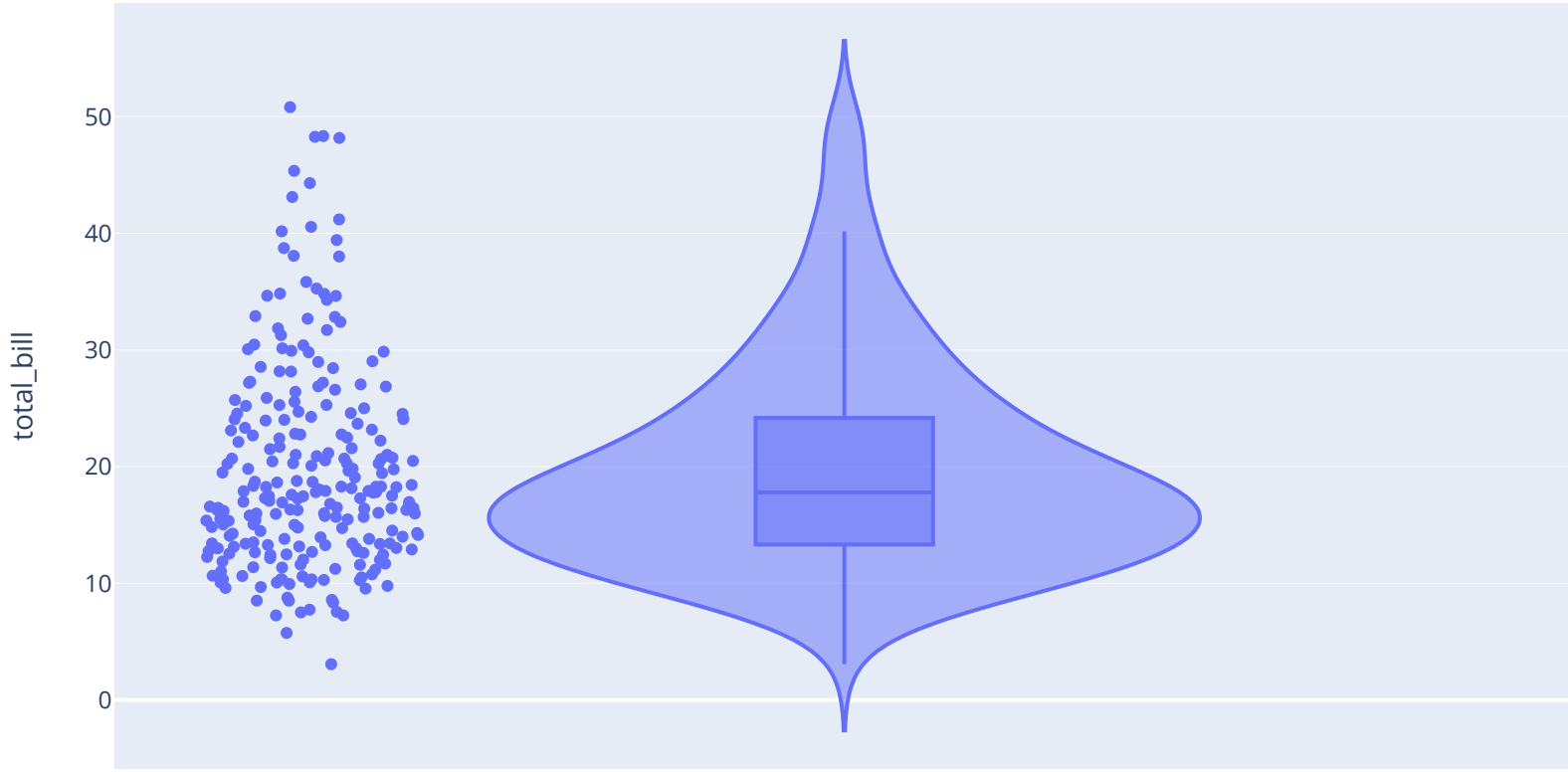
In [17]:
```
z = np.random.rand(10, 10)
fig = go.Figure(data=go.Contour(z=z))
fig.show()
```




### 16. **Violin Plot**

- **Category**: Statistical Analysis
- **Usage**: Used to visualize the distribution of data and its probability density.

In [18]:
```
df = px.data.tips()
fig = px.violin(df, y="total_bill", box=True, points="all")
fig.show()
```

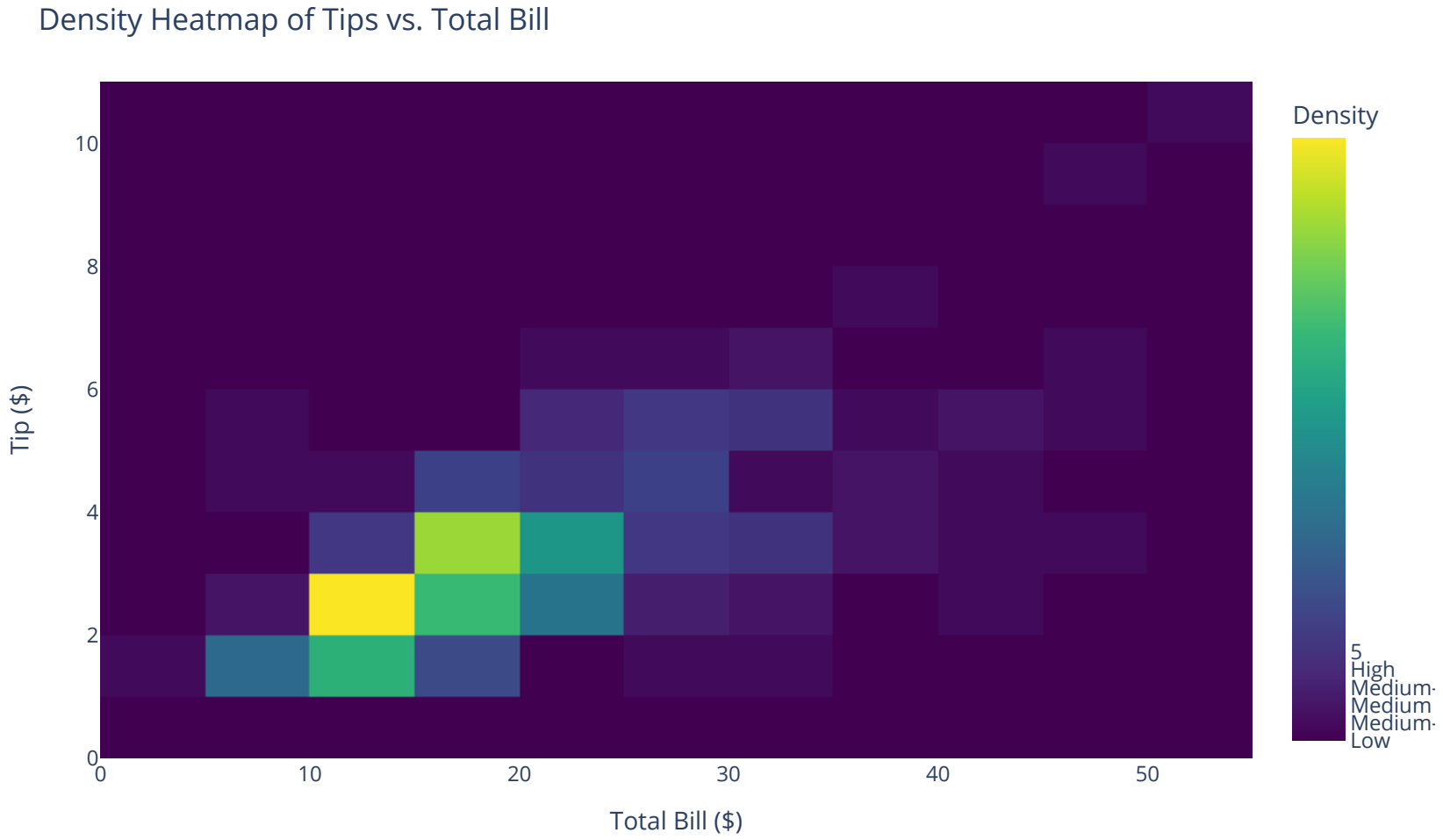



### 17. **Density Heatmap**

- **Category**: Data Density Visualization
- **Usage**: Used to show the distribution of data over a continuous interval or time period.

In [19]:
```
# Create the density heatmap
fig = px.density_heatmap(df,
                         x="total_bill",
                         y="tip",
                         color_continuous_scale="Viridis",  # Color scale for better visualization
```

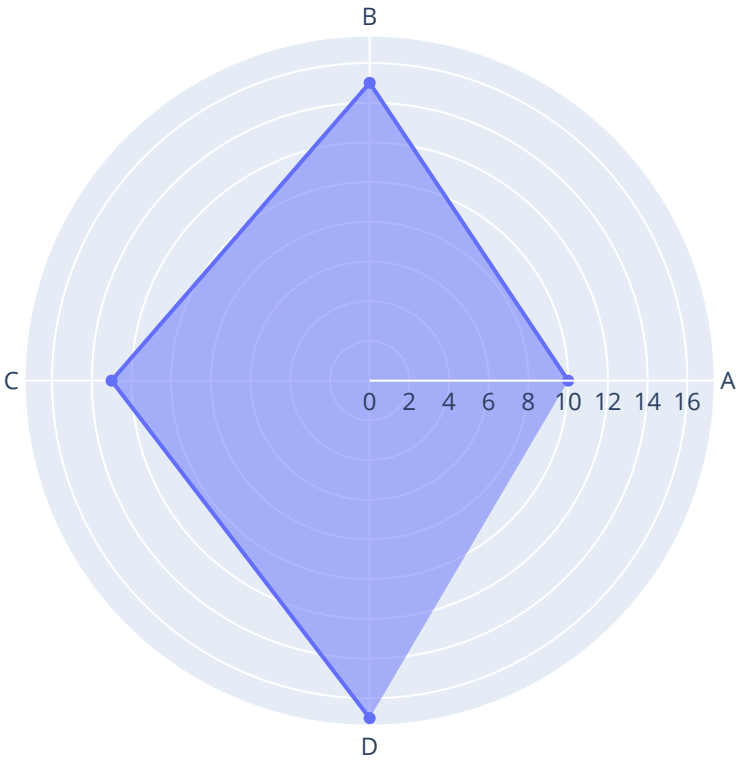
```
                          labels={"total_bill": "Total Bill ($)", "tip": "Tip ($)"}  # Adding meaningful label
                         )

# Update the layout with titles and labels
fig.update_layout(
    title="Density Heatmap of Tips vs. Total Bill",
    xaxis_title="Total Bill ($)",
    yaxis_title="Tip ($)",
    coloraxis_colorbar=dict(
        title="Density",
        tickvals=[0, 1, 2, 3, 4, 5],
        ticktext=["Low", "Medium-Low", "Medium", "Medium-High", "High"]
    )
)

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




### 18. **Radar Chart**

- **Category**: Comparative Analysis
- **Usage**: Used to compare multiple variables.

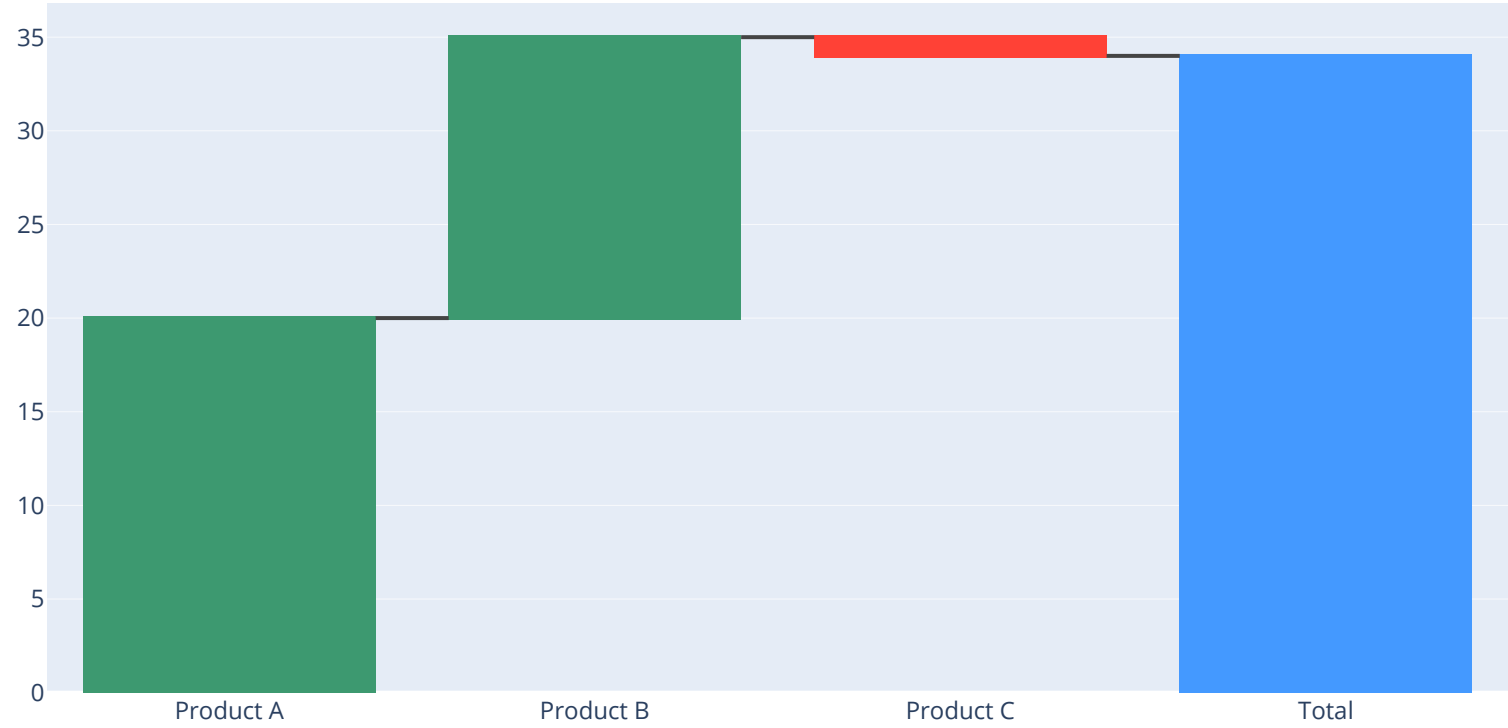
In [20]:
```
fig = go.Figure(data=go.Scatterpolar(r=[10, 15, 13, 17], theta=['A', 'B', 'C', 'D'], fill='toself'))
fig.show()
```




### 19. **Waterfall Chart**

- *Category*: Financial Data Analysis
- *Usage*: Used to show cumulative effects of sequentially introduced positive or negative values.

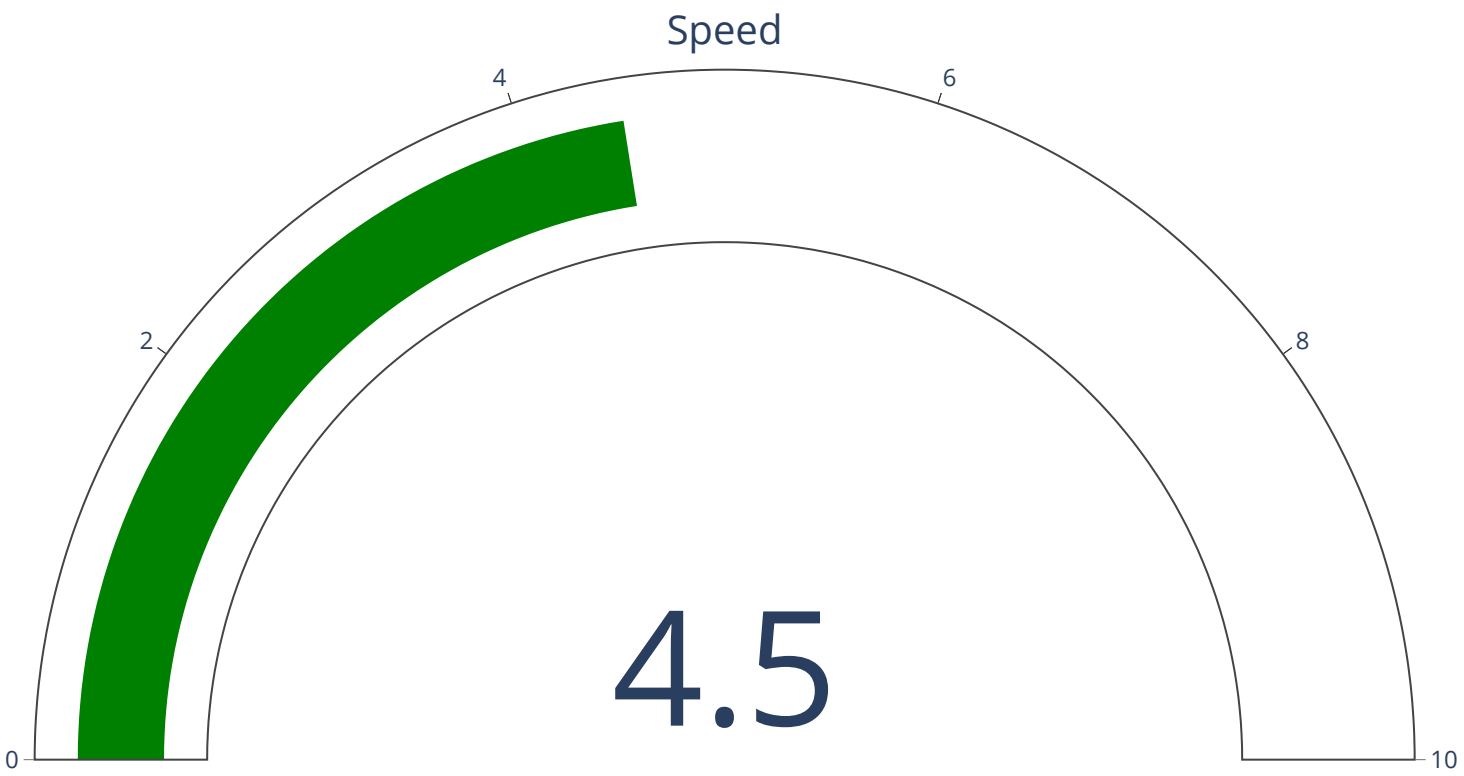
In [21]:
```
fig = go.Figure(go.Waterfall(name="20", orientation="v",
                             measure=["relative", "relative", "relative", "total"],
                             x=["Product A", "Product B","Product C", "Total"], y=[20, 15, -1, 34]))
fig.show()
```




## 20. **Gauge Chart**

- **Category**: Performance Measurement
- **Usage**: Used to display a single data point within a defined range.

In [22]:
```
fig = go.Figure(go.Indicator(mode="gauge+number",
                             value=4.5, title={'text': "Speed"},
                             gauge={'axis': {'range': [None, 10]}}))
fig.show()
```




## Conclusion

Plotly provides a comprehensive suite of visualization tools suitable for various data analysis and presentation needs. From simple scatter plots to complex 3D visualizations and interactive dashboards, Plotly makes it easy to create high-quality, interactive visualizations for a wide range of applications.