Plotly: An Overview

What is Plotly?

- Plotly is an open-source data visualization library for Python, R, JavaScript, and other languages.
- It helps create interactive, web-based graphs and charts.
- Commonly used for creating interactive plots, dashboards, and data visualizations.

Main Features of Plotly:

- **Interactive Plots**: Allows zooming, panning, and hovering over data points.
- **Wide Variety of Charts**: Supports many types of charts such as line plots, scatter plots, bar charts, histograms, maps, and 3D plots.
- **Customization**: Provides various styling options like colors, markers, and labels for customizing the look of charts.
- **Exporting**: Charts can be exported as images or embedded into websites.
- **Integrations**: Works well with popular frameworks like Dash for building dashboards, and integrates with Jupyter notebooks.
- **Cross-language Support**: Works with Python, R, MATLAB, and JavaScript, among other languages.

Why is Plotly Majorly Used?

- **Interactivity**: The ability to interact with the plots makes it useful for data exploration and presentations.
- **Web-ready Visuals**: Since it's web-based, you can easily share visualizations on websites or through web apps.
- **Ease of Use**: Simple and quick to create complex, beautiful visualizations without needing to write a lot of code.
- **Dash Integration**: Plotly is used alongside Dash to create analytical web applications without extensive front-end programming.

Disadvantages of Plotly:

- **Learning Curve**: It may take some time to learn all the features, especially for beginners.
- **Performance**: May become slow with very large datasets or complex visualizations.
- **Limited Offline Use**: Interactive plots need to be hosted online for some advanced features.
- **Size of Charts**: Plotly’s charts can be heavier in terms of file size compared to some other libraries.

Other Products from Plotly:

- **Dash**: A Python framework for building analytical web applications with no need for front-end code.
  - Used for creating dashboards that update in real-time and work with Plotly visualizations.
  - Helps in building interactive applications with minimal code.
- **Chart Studio**: An online tool for creating, storing, and sharing interactive charts and dashboards.
  - Allows collaboration and storing charts in the cloud.

Pricing:

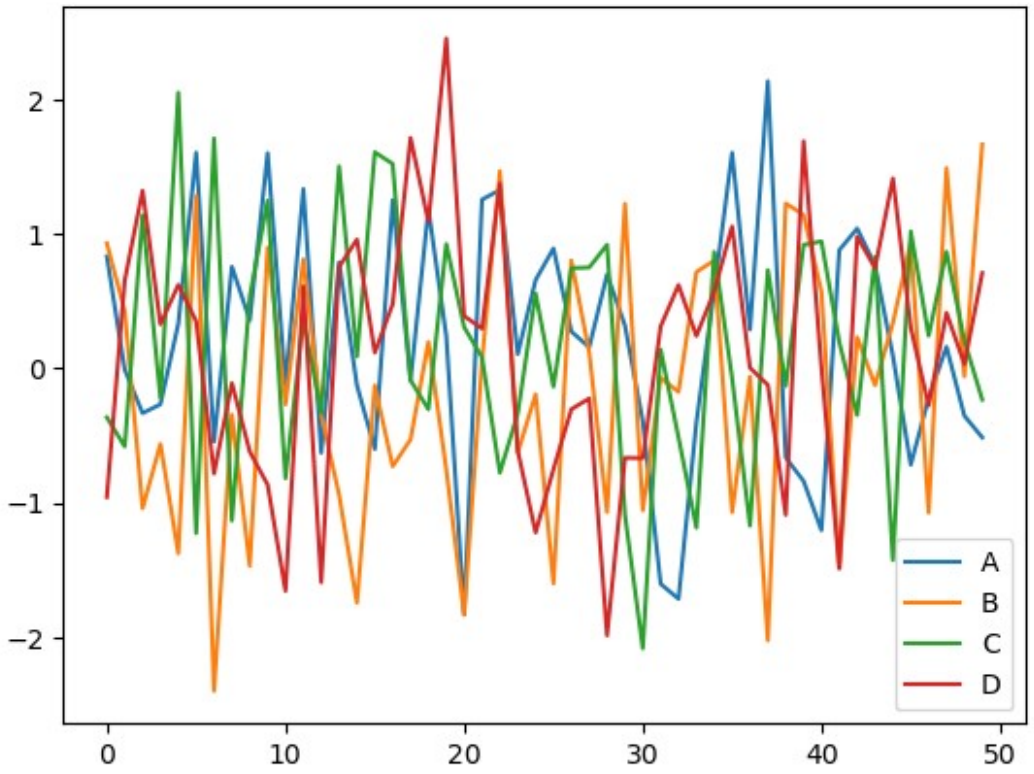
- Plotly is free to use for creating visualizations.
- It is free until you want to **host** your visualizations online (for example, using Dash apps on a server). Hosting might require a paid plan depending on your needs.

```
# installing plotly and cufflinks

# pip install cufflinks

# Importing major libraries

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

import plotly.express as px
import cufflinks as cf

import warnings
warnings.filterwarnings('ignore')

from plotly.offline import download_plotlyjs, init_notebook_mode,plot,
iplot
init_notebook_mode(connected=True)
cf.go_offline()

arr1 = np.random.randn(50,4)
df1 = pd.DataFrame(arr1,columns=['A','B','C','D'])
df1.head()
```

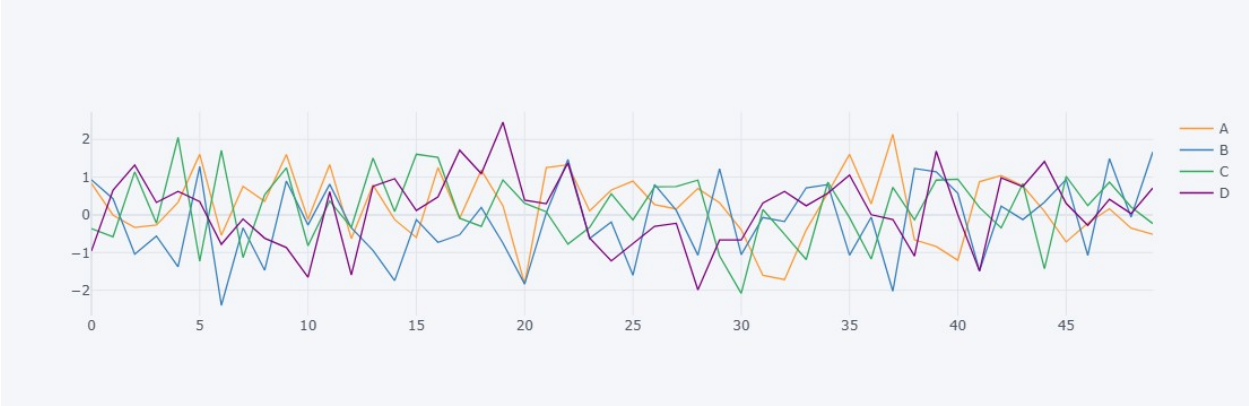
	A	B	C	D
0	0.834283	0.932043	-0.363921	-0.957393
1	-0.008373	0.427847	-0.580131	0.655117
2	-0.332619	-1.040922	1.141170	1.323756
3	-0.266466	-0.561670	-0.220349	0.329207
4	0.332846	-1.376237	2.053395	0.622690

```
df1.plot()
```

```
<Axes: >
```




```
df1.iplot()
```




```
# Importing Datasets

stocks = px.data.stocks()
gap = px.data.gapminder()
iris = px.data.iris()
tips = px.data.tips()
```

```
flights = sns.load_dataset('flights')
mpg = sns.load_dataset('mpg')
wind= px.data.wind()
attention = sns.load_dataset('attention')
```

Line Plot in Plotly Express

- **What it is:** A graph that connects data points with lines, used to visualize trends over time or relationships between variables.

Key Features:

- **Basic Syntax:**

```
    fig = px.line(df, x='x_column', y='y_column')
    fig.show()
```

Common Uses:

- **Trend Analysis:** Shows how values change over time (e.g., stock prices, population growth).
- **Time Series Data:** Plots dates or years on the x-axis to observe patterns over time.

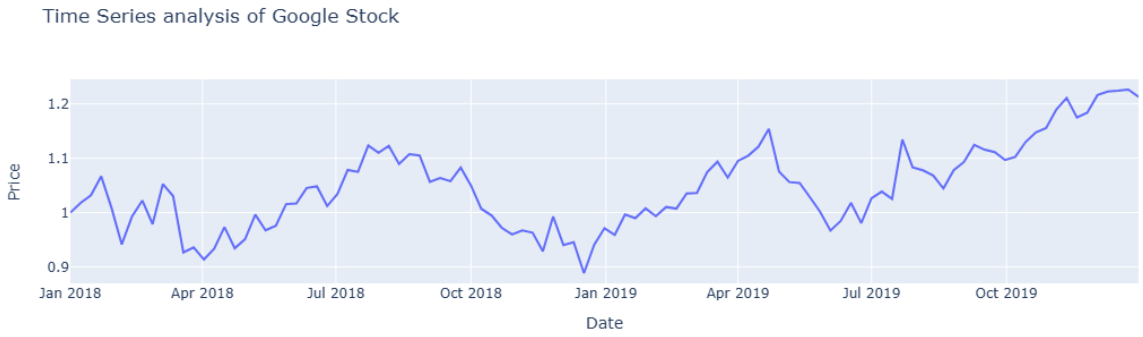
Customization:

- **Add Markers:** Use `markers=True` to display points on the lines.
- **Line Style:** Use `line_dash` for dashed or dotted lines.
- **Faceting:** Create subplots for different categories using `facet_col` or `facet_row`.

```
stocks.head()
```

	date	GOOG	AAPL	AMZN	FB	NFLX	MSFT
0	2018-01-01	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1	2018-01-08	1.018172	1.011943	1.061881	0.959968	1.053526	1.015988
2	2018-01-15	1.032008	1.019771	1.053240	0.970243	1.049860	1.020524
3	2018-01-22	1.066783	0.980057	1.140676	1.016858	1.307681	1.066561
4	2018-01-29	1.008773	0.917143	1.163374	1.018357	1.273537	1.040708

```
px.line(stocks,x='date',y='GOOG',labels={'date':'Date','GOOG':'Price'}
,title='Time Series analysis of Google Stock')
```




```
px.line(stocks,x='date',y=['GOOG','AAPL','MSFT'],labels={'date':'Date'
,'value':'Price'},title='Google vs Apple vs Microsoft over the years')
```

Bar Plot in Plotly Express

- **What it is:** A bar plot displays data using rectangular bars, with the length of the bar representing the value. It is useful for comparing categories or groups.

Key Features:

- **Basic Syntax:**

```
fig = px.bar(df, x='x_column', y='y_column')
fig.show()
```

Common Uses:

- **Category Comparison:** Shows how different categories or groups compare in terms of size or frequency.
- **Aggregated Data:** Useful for visualizing sums, averages, or counts of categories.

Customization:

- **Color Grouping:** Use the `color` argument to differentiate bars by categories.
- **Orientation:** Switch between vertical and horizontal bars using `orientation='h'`.

- **Bar Size:** Control the width of bars using `barmode` for stacked or grouped bars.

Example:

```
fig = px.bar(df, x='country', y='population', color='continent')
fig.show()
```

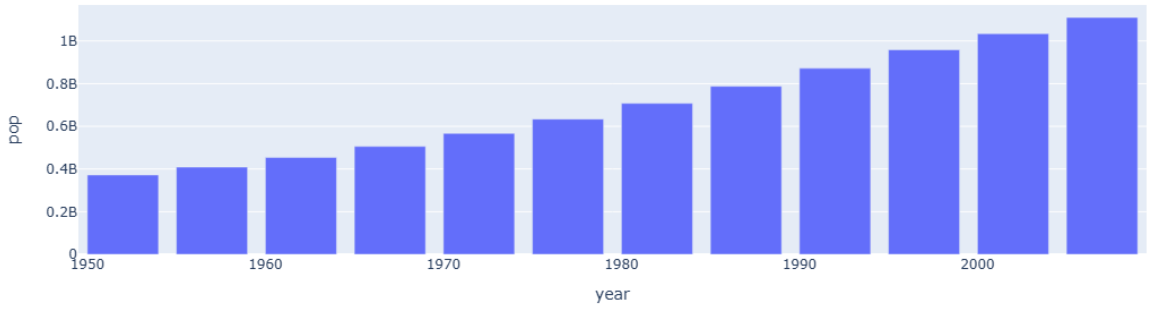
Summary:

- **Easy to create** for comparing values across categories.
- **Highly customizable** with options for colors, orientations, and bar grouping.
- Great for **categorical data and summaries**.

```
gap_ind = px.data.gapminder().query('country=="India"')

gap_ind.head()
```

```
    country continent  year  lifeExp        pop   gdpPercap iso_alpha
iso_num
696   India      Asia  1952   37.373  372000000  546.565749       IND
356
697   India      Asia  1957   40.249  409000000  590.061996       IND
356
698   India      Asia  1962   43.605  454000000  658.347151       IND
356
699   India      Asia  1967   47.193  506000000  700.770611       IND
356
700   India      Asia  1972   50.651  567000000  724.032527       IND
356
```

```
px.bar(gap_ind,x='year',y='pop')
```

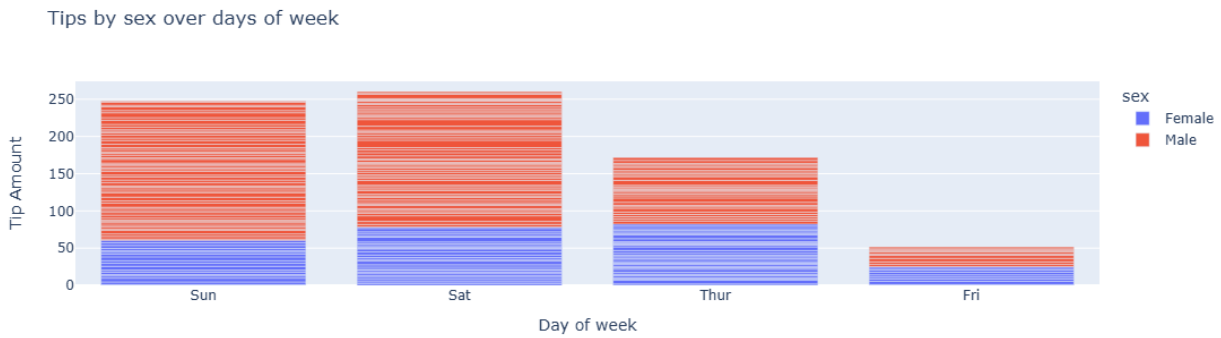



```
tips.head()
```

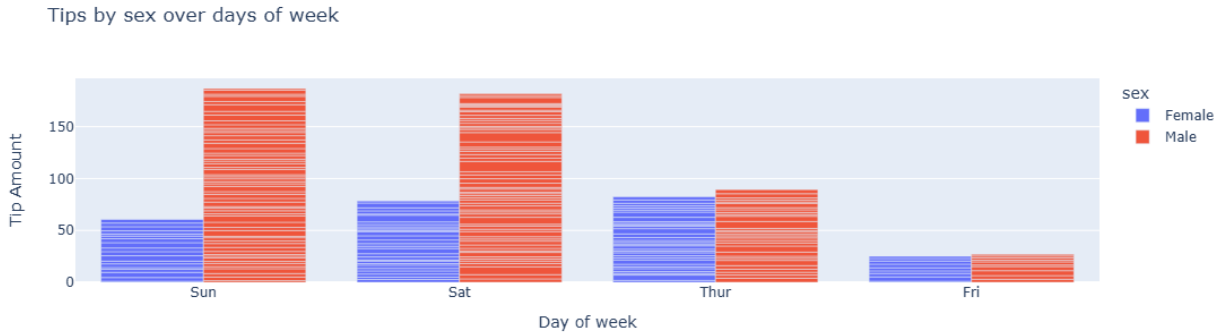
```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
```

```
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

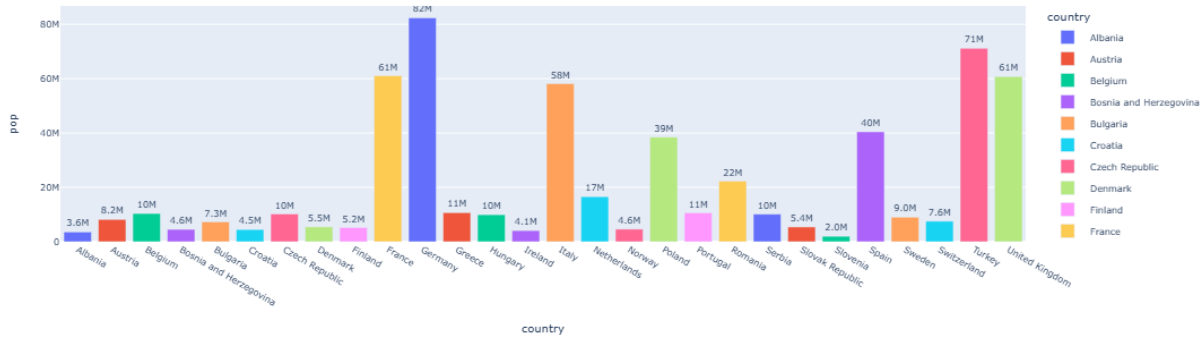
```
# stacked column chart
px.bar(tips,x='day',y='tip',color='sex',labels={'day':'Day of
week','tip':'Tip Amount'},title='Tips by sex over days of week')
```



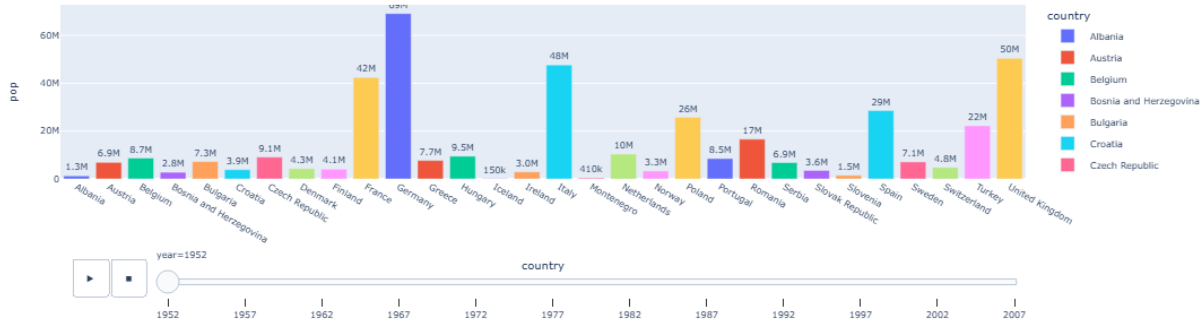

```
# clustered column chart
# Barmode
px.bar(tips,x='day',y='tip',color='sex',labels={'day':'Day of
week','tip':'Tip Amount'},title='Tips by sex over days of
week',barmode='group')
```




```
europe = px.data.gapminder().query('continent=="Europe" and year ==
2007 and pop>2.e6')

fig = px.bar(europe,x='country',y='pop',color='country',text='pop')
fig.update_traces(texttemplate='%{text:.2s}',textposition='outside')
fig.update_layout(font=dict(size=8))
fig.show()
```




```
# animated plots

europe_all = px.data.gapminder().query('continent=="Europe"')

fig = px.bar(europe_all,
x='country',y='pop',color='country',text='pop',animation_frame='year')
fig.update_traces(texttemplate='%{text:.2s}',textposition='outside')
fig.update_layout(font=dict(size=8))
fig.show()
```



Scatter Plot in Plotly Express

- **What it is:** A scatter plot uses dots to represent values for two different numeric variables. It is helpful for identifying relationships or correlations between variables.

Key Features:

- **Basic Syntax:**

```
fig = px.scatter(df, x='x_column', y='y_column')
fig.show()
```

Common Uses:

- **Relationship Analysis:** Used to observe how two variables are related (e.g., height vs. weight).

- **Identifying Patterns:** Helps identify trends, clusters, or outliers in the data.

Customization:

- **Color Grouping:** Use the `color` argument to differentiate points by categories.
- **Size Control:** Adjust the size of points using the `size` argument to reflect another variable.
- **Hover Data:** Add more information when hovering over points using `hover_data`.

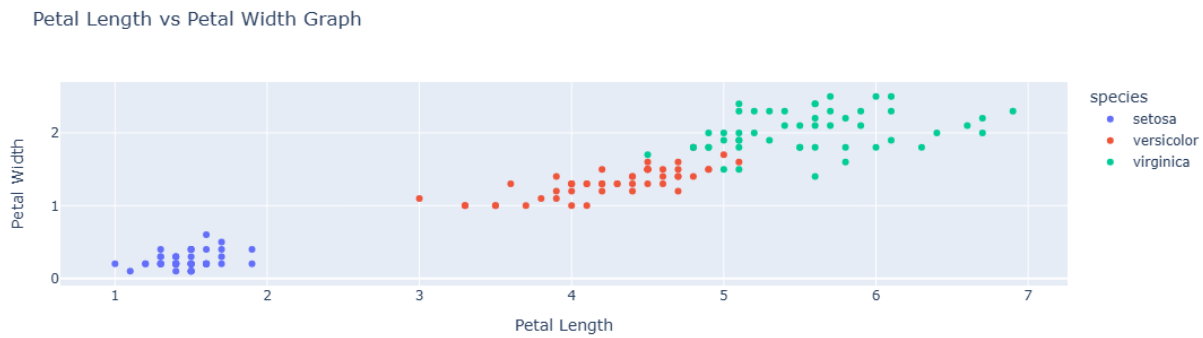
Example:

```
fig = px.scatter(df, x='GDP', y='LifeExpectancy', color='Region',
size='Population', hover_data=['Country'])
fig.show()
```

Summary:

- **Effective for exploring relationships** between two variables.
- **Highly customizable** with options for color, size, and hover information.
- Great for **visualizing correlations and identifying trends** in datasets.

```
px.scatter(iris,x='petal_length',y='petal_width',color='species',
           title='Petal Length vs Petal Width
Graph',labels={'petal_length':'Petal Length','petal_width':'Petal
Width'})
```

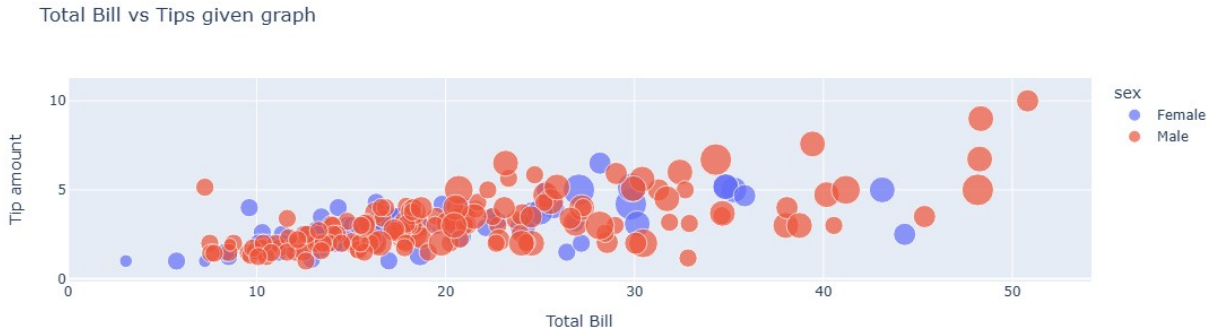



```
# Bubble Plot
tips
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
..	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2

```
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]
```

```
px.scatter(tips,x='total_bill',y='tip',size='size',color='sex',
          title='Total Bill vs Tips given
graph',labels={'total_bill':'Total Bill','tip':'Tip amount'})
```



Pie Chart in Plotly Express

- **What it is:** A pie chart is a circular statistical graphic divided into slices to illustrate numerical proportions. Each slice represents a category's contribution to the total.

Key Features:

- **Basic Syntax:**

```
fig = px.pie(df, names='category_column', values='value_column')
fig.show()
```

Common Uses:

- **Proportional Representation:** Ideal for displaying the percentage share of different categories in a dataset (e.g., market share).
- **Simple Comparisons:** Good for showing how parts relate to a whole.

Customization:

- **Color Customization:** Use the `color` argument to specify colors for different categories.
- **Hover Information:** Enhance user experience by adding `hover_data` for more details on each slice.
- **Exploding Slices:** Highlight specific slices by using the `pull` argument to create an "exploded" effect.

Example:

```
fig = px.pie(df, names='fruit', values='sales', title='Fruit Sales
Distribution', color='fruit')
fig.show()
```

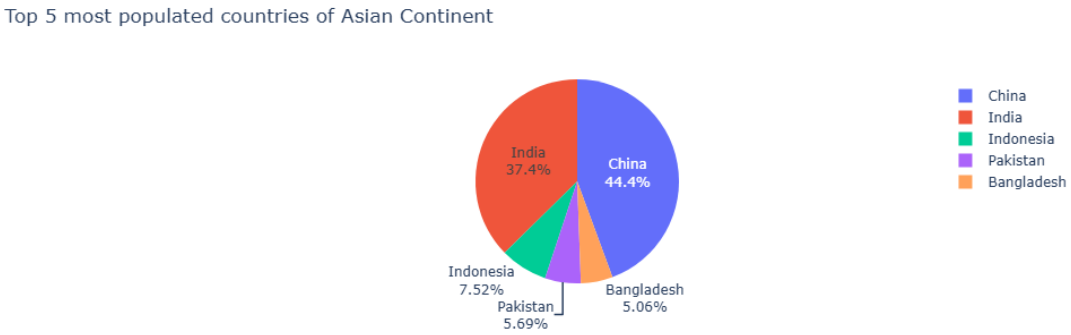
Summary:

- **Simple and intuitive** for showing parts of a whole.
- **Customizable** with colors, hover data, and slice effects.
- Best for **small datasets** with limited categories to avoid clutter.

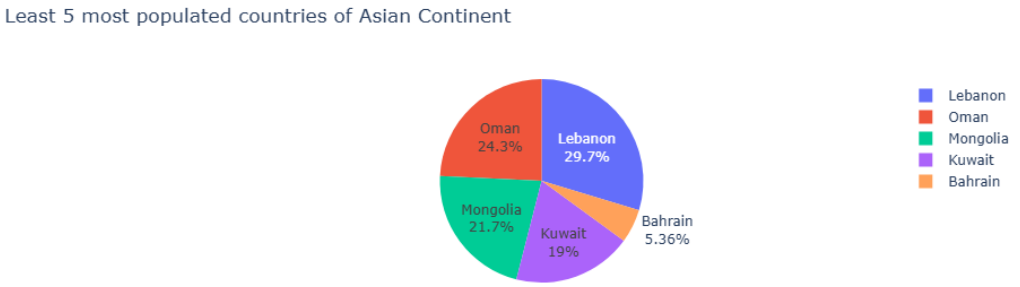
```
asia = px.data.gapminder().query('year==2007 and continent=="Asia"')

asia_sorted = asia.sort_values(by='pop',ascending=False)

asia_sorted.head()
```

```
         country continent  year  lifeExp         pop    gdpPercap
iso_alpha  \
299        China      Asia  2007   72.961  1318683096  4959.114854
CHN
707        India      Asia  2007   64.698  1110396331  2452.210407
IND
719    Indonesia      Asia  2007   70.650   223547000  3540.651564
IDN
1175    Pakistan      Asia  2007   65.483   169270617  2605.947580
PAK
107   Bangladesh      Asia  2007   64.062   150448339  1391.253792
BGD

      iso_num
299       156
707       356
719       360
1175      586
107        50
```

```
fig =
px.pie(asia_sorted.head(),values='pop',names='country',title='Top 5
most populated countries of Asian Continent')

fig.update_traces(
    hovertemplate='%{label}:%{percent}',
    textinfo='label+percent',
    textfont=dict(size=12)
)
```




```
fig =
px.pie(asia_sorted.tail(),values='pop',names='country',title='Least 5
most populated countries of Asian Continent')

fig.update_traces(
    hovertemplate='%{label}:%{percent}',
    textinfo='label+percent',
    textfont=dict(size=12)
)
```



Histogram in Plotly Express

- **What it is:** A histogram is a graphical representation that organizes a group of data points into user-specified ranges (bins). It is used to display the distribution of a continuous variable.

Key Features:

- **Basic Syntax:**

```
fig = px.histogram(df, x='value_column', nbins=number_of_bins)
fig.show()
```

Common Uses:

- **Data Distribution:** Helps to visualize the frequency distribution of a dataset (e.g., the distribution of heights in a population).
- **Identifying Patterns:** Useful for spotting the shape of data distribution, such as normal, skewed, or bimodal distributions.

Customization:

- **Number of Bins:** Control the number of bins using the `nbins` argument to adjust granularity.
- **Color Grouping:** Use the `color` argument to differentiate data by categories within the same histogram.
- **Overlaying Density Plot:** Combine a histogram with a density plot for better insights by using the `marginal` argument (e.g., `marginal='violin'`).

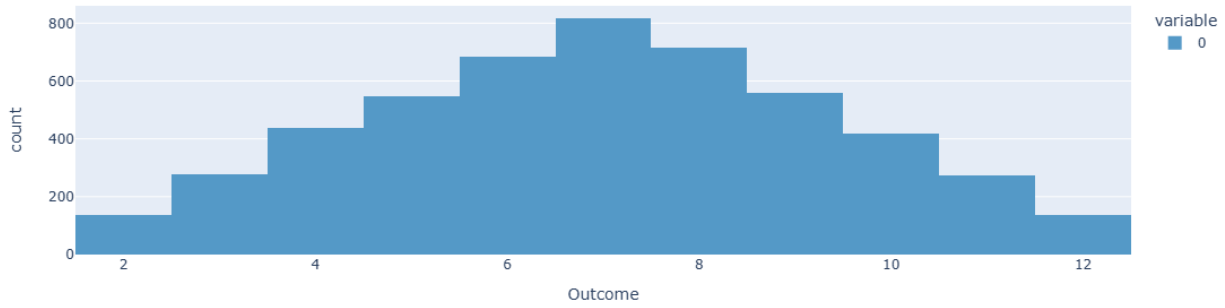
Example:

```
fig = px.histogram(df, x='age', nbins=20, color='gender', title='Age
Distribution by Gender')
fig.show()
```

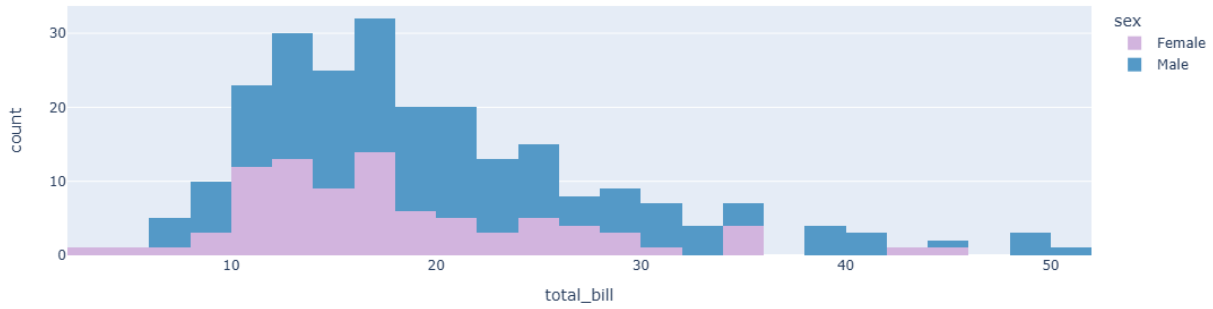
Summary:

- **Effective for visualizing distributions** of continuous data.
- **Highly customizable** with options for bin size, colors, and overlaying other plots.
- Great for **understanding data characteristics** and identifying patterns.

```
dice1 = np.random.randint(1,7,5000)
dice2 = np.random.randint(1,7,5000)
dice = dice1+dice2

px.histogram(dice,nbins=11,labels={'value':'Outcome'},color_discrete_s
equence=['#5499c7'])
```




```
px.histogram(tips,x='total_bill',color_discrete_sequence=['#d2b4de','#
5499c7'],color='sex')
```



Box Plot in Plotly Express

- **What it is:** A box plot (or whisker plot) summarizes the distribution of a dataset by displaying its central tendency, variability, and outliers. It visualizes the median, quartiles, and potential outliers in the data.

Key Features:

- **Basic Syntax:**

```
fig = px.box(df, x='category_column', y='value_column')
fig.show()
```

Common Uses:

- **Comparative Analysis:** Ideal for comparing distributions between different groups or categories (e.g., comparing test scores across different classes).
- **Outlier Detection:** Easily highlights outliers, which are data points significantly higher or lower than the rest.

Customization:

- **Color Grouping:** Use the `color` argument to differentiate box plots by categories.
- **Orientation:** Change the orientation of the box plot to horizontal by using `orientation='h'`.
- **Adding Points:** Display individual data points on top of the box plot using the `points` argument (e.g., `points='all'` to show all points).

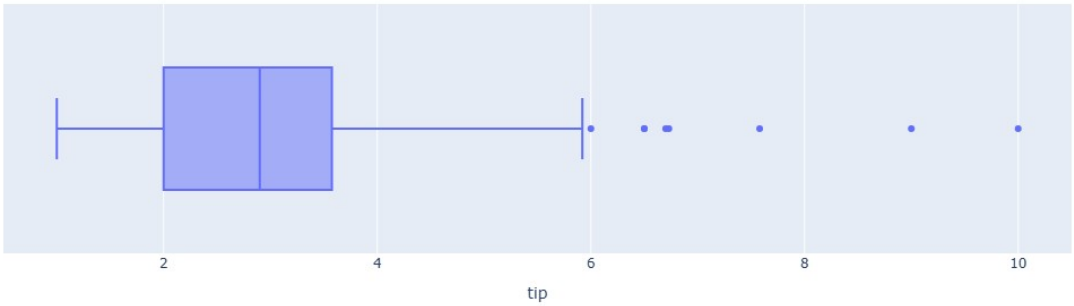
Example:

```
fig = px.box(df, x='species', y='sepal_length', color='species',
title='Sepal Length Distribution by Species')
fig.show()
```

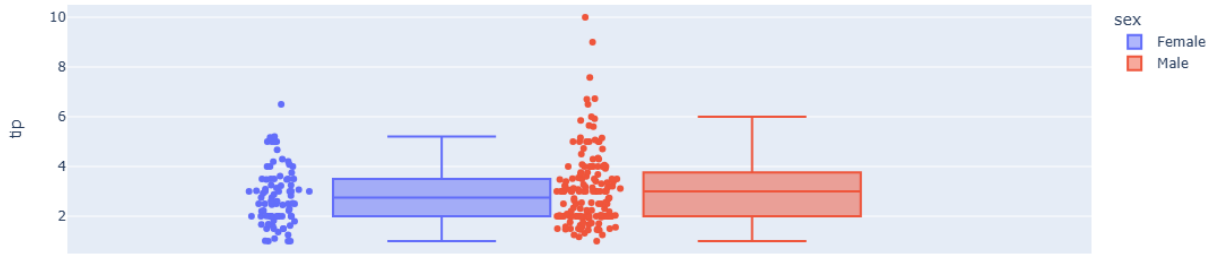
Summary:

- **Effective for summarizing** data distributions and spotting outliers.
- **Highly customizable** with options for colors, orientations, and additional point displays.
- Great for **visualizing the spread and skewness** of datasets across different categories.

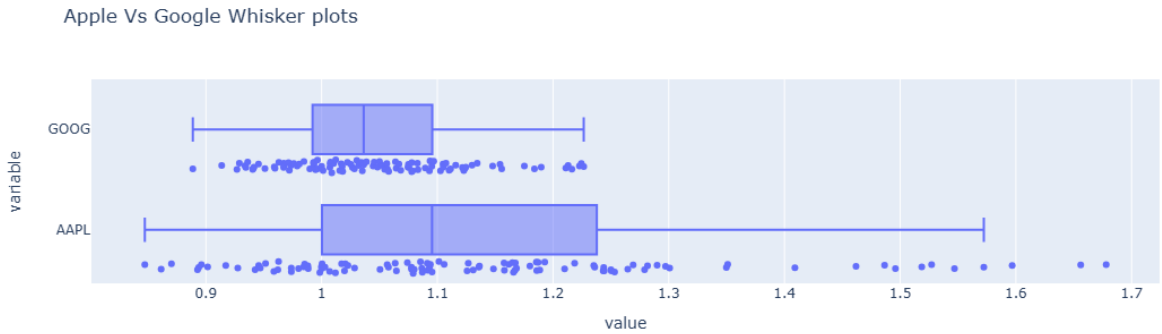
```
px.box(tips,x='tip')
```




```
px.box(tips,y='tip',color='sex',points='all')
```




```
px.box(stocks,x=['AAPL','GOOG'], points='all',title='Apple Vs Google Whisker plots')
```



Violin Plot in Plotly Express

- **What it is:** A violin plot combines a box plot with a density plot. It visualizes the distribution of a dataset, showing its probability density at different values, which helps understand the underlying distribution of the data.

Key Features:

- **Basic Syntax:**

```
fig = px.violin(df, x='category_column', y='value_column')
fig.show()
```

Common Uses:

- **Distribution Analysis:** Ideal for visualizing the distribution of data across different categories, providing insights into the density and spread of the data (e.g., visualizing test scores by class).
- **Comparative Analysis:** Useful for comparing distributions between multiple groups or categories.

Customization:

- **Color Grouping:** Use the `color` argument to differentiate violin plots by categories.
- **Box Plot Overlay:** Add a box plot on top of the violin plot using the `box=True` argument for better visual representation of the median and quartiles.
- **Violin Width:** Adjust the width of the violin plots to enhance clarity or emphasize differences using the `scale` argument (e.g., `scale='area'`, `scale='count'`).

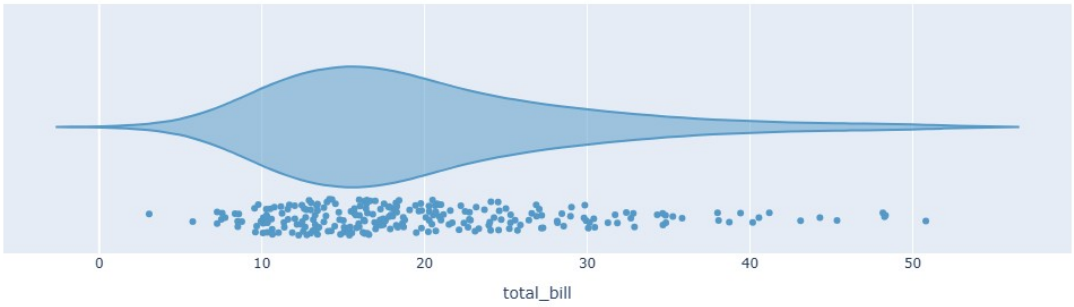
Example:

```
fig = px.violin(df, x='species', y='petal_length', color='species',
box=True, title='Petal Length Distribution by Species')
fig.show()
```

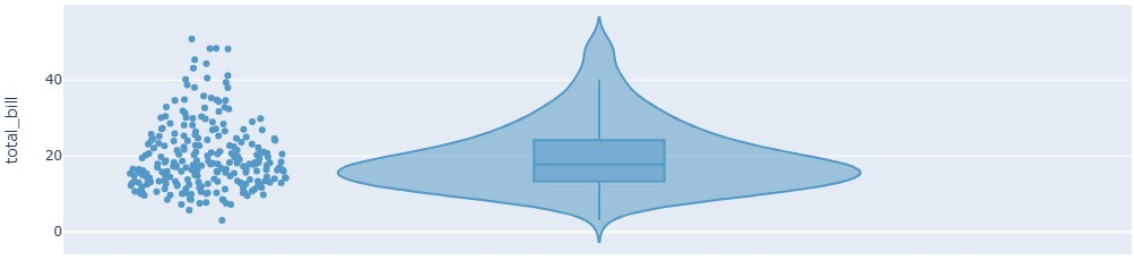
Summary:

- **Effective for visualizing distributions** and density in datasets.
- **Highly customizable** with options for colors, overlays, and scaling.
- Great for **understanding the shape and spread** of data across different categories.

```
px.violin(tips,x='total_bill',points='all',color_discrete_sequence=['#
5499c7'])
```




```
px.violin(tips,y='total_bill',points='all',color_discrete_sequence=['#5499c7'],box=True)
```



Density Heatmap in Plotly Express

- **What it is:** A density heatmap visualizes the density of data points in a two-dimensional space, representing the concentration of data in specific areas. It provides insights into how data points are distributed across different values of two variables.

Key Features:

- **Basic Syntax:**

```
fig = px.density_heatmap(df, x='x_column', y='y_column', z='z_column')
fig.show()
```

Common Uses:

- **Data Distribution Visualization:** Ideal for showing the density of data points, helping to identify areas with a high concentration of values (e.g., customer distribution across a geographic area).
- **Relationship Analysis:** Useful for observing relationships between two continuous variables and identifying patterns.

Customization:

- **Color Scale:** Customize the color scale to represent density levels more intuitively using the `color_continuous_scale` argument (e.g., `'Viridis'`, `'Cividis'`).
- **Opacity Control:** Adjust the opacity of the heatmap using the `opacity` argument to enhance clarity in overlapping areas.
- **Adding Marginals:** Overlay marginal histograms or density plots using the `marginal` argument (e.g., `marginal='histogram'`).

Example:

```
fig = px.density_heatmap(df, x='longitude', y='latitude',
z='population',
                         color_continuous_scale='Viridis',
title='Population Density Heatmap')
fig.show()
```

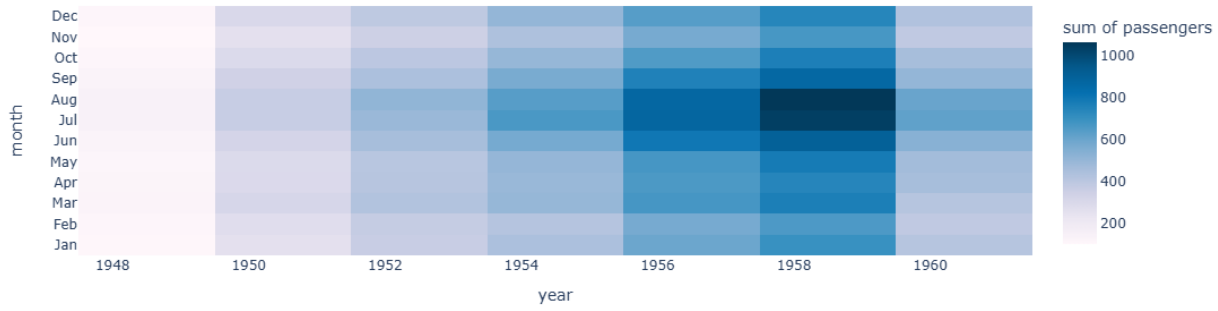
Summary:

- **Effective for visualizing data concentrations** across two dimensions.
- **Highly customizable** with options for color scales, opacity, and marginal overlays.
- Great for **identifying patterns and relationships** in large datasets.

```
flights.head()
```

```
   year month  passengers
0  1949   Jan         112
1  1949   Feb         118
2  1949   Mar         132
3  1949   Apr         129
4  1949   May         121
```

```
px.density_heatmap(flights,x='year',y='month',z='passengers',

hover_data=flights.columns,color_continuous_scale='PuBu')
```



3D Scatter Plot in Plotly Express

- **What it is:** A 3D scatter plot visualizes data points in three-dimensional space, representing three continuous variables. It helps to understand the relationships and distribution of data across three dimensions.

Key Features:

- **Basic Syntax:**

```
fig = px.scatter_3d(df, x='x_column', y='y_column', z='z_column',
color='color_column')
fig.show()
```

Common Uses:

- **Relationship Analysis:** Ideal for exploring relationships between three variables (e.g., analyzing the correlation between height, weight, and age).
- **Data Distribution Visualization:** Useful for visualizing how data points are distributed in three-dimensional space, helping to identify clusters and outliers.

Customization:

- **Marker Size:** Control the size of the markers using the `size` argument to represent another variable (e.g., `size='size_column'`).
- **Color Customization:** Use the `color` argument to differentiate data points by categories or groups.
- **Camera View:** Adjust the camera perspective using the `camera` argument for better visualization (e.g., `camera=dict(eye=dict(x=1.2, y=1.2, z=1.2))`).

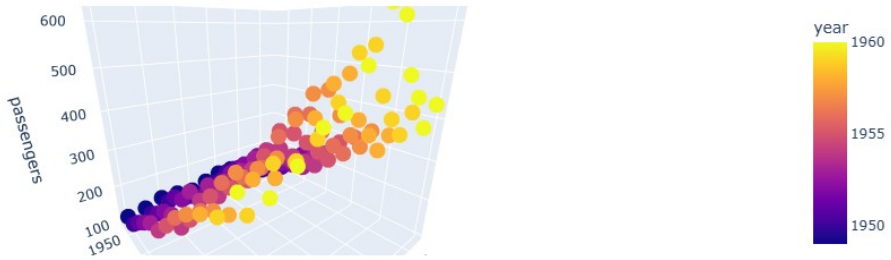
Example:

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

Summary:

- **Effective for visualizing relationships** among three continuous variables.
- **Highly customizable** with options for marker size, colors, and camera angles.
- Great for **identifying clusters and trends** in multidimensional datasets.

```
px.scatter_3d(flights,x='year',y='month',z='passengers',color='year')
```



3D Line Plot in Plotly Express

- **What it is:** A 3D line plot connects data points in three-dimensional space with lines, allowing for the visualization of trends or relationships across three continuous variables.

Key Features:

- **Basic Syntax:**

```
fig = px.line_3d(df, x='x_column', y='y_column', z='z_column',
color='color_column')
fig.show()
```

Common Uses:

- **Trend Analysis:** Ideal for visualizing trends over three dimensions (e.g., analyzing changes in temperature over time and altitude).
- **Relationship Exploration:** Useful for exploring relationships between three variables, helping to identify patterns.

Customization:

- **Line Width and Style:** Control the appearance of the lines using the `line_shape` argument to choose between straight or curved lines (e.g., `line_shape='spline'`).
- **Color Grouping:** Use the `color` argument to differentiate lines based on categories or groups.
- **Camera Angle:** Adjust the perspective of the plot using the `camera` argument for better visualization (e.g., `camera=dict(eye=dict(x=1, y=1, z=1))`).

Example:

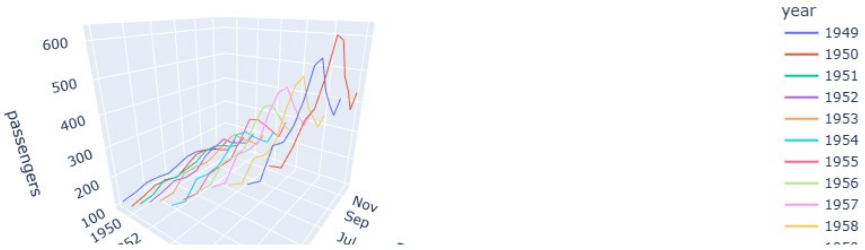
```
import pandas as pd
import numpy as np

# Create a sample dataset
t = np.linspace(0, 10, 100)
x = t
y = np.sin(t)
```

```
z = np.cos(t)
df = pd.DataFrame({'x': x, 'y': y, 'z': z})

fig = px.line_3d(df, x='x', y='y', z='z', title='3D Line Plot of Sine and Cosine Waves')
fig.show()
```

Summary:

- **Effective for visualizing trends** and relationships among three continuous variables over a continuous range.
- **Highly customizable** with options for line styles, colors, and camera perspectives.
- Great for **understanding the behavior of data in a three-dimensional context**.

```
px.line_3d(flights,x='year',y='month',z='passengers',color='year')
```



Scatter Matrix in Plotly Express

- **What it is:** A scatter matrix (also known as a pair plot) is a grid of scatter plots that visualizes the relationships between multiple variables in a dataset. Each scatter plot shows the relationship between a pair of variables, allowing for easy identification of correlations and patterns.

Key Features:

- **Basic Syntax:**

```
fig = px.scatter_matrix(df, dimensions=['col1', 'col2', 'col3', 'col4'], color='color_column')
fig.show()
```

Common Uses:

- **Multivariate Analysis:** Ideal for exploring relationships among multiple variables simultaneously, helping to identify correlations and trends.
- **Outlier Detection:** Useful for spotting outliers and unusual patterns within the data.

Customization:

- **Color Grouping:** Use the `color` argument to differentiate data points based on categories or groups.
- **Diagonal Histograms:** Customize the diagonal plots to show histograms or density plots using the `diagonal_visible` and `diagonal_type` arguments.
- **Symbol Customization:** Modify the markers' symbols using the `symbol` argument for better differentiation among groups.

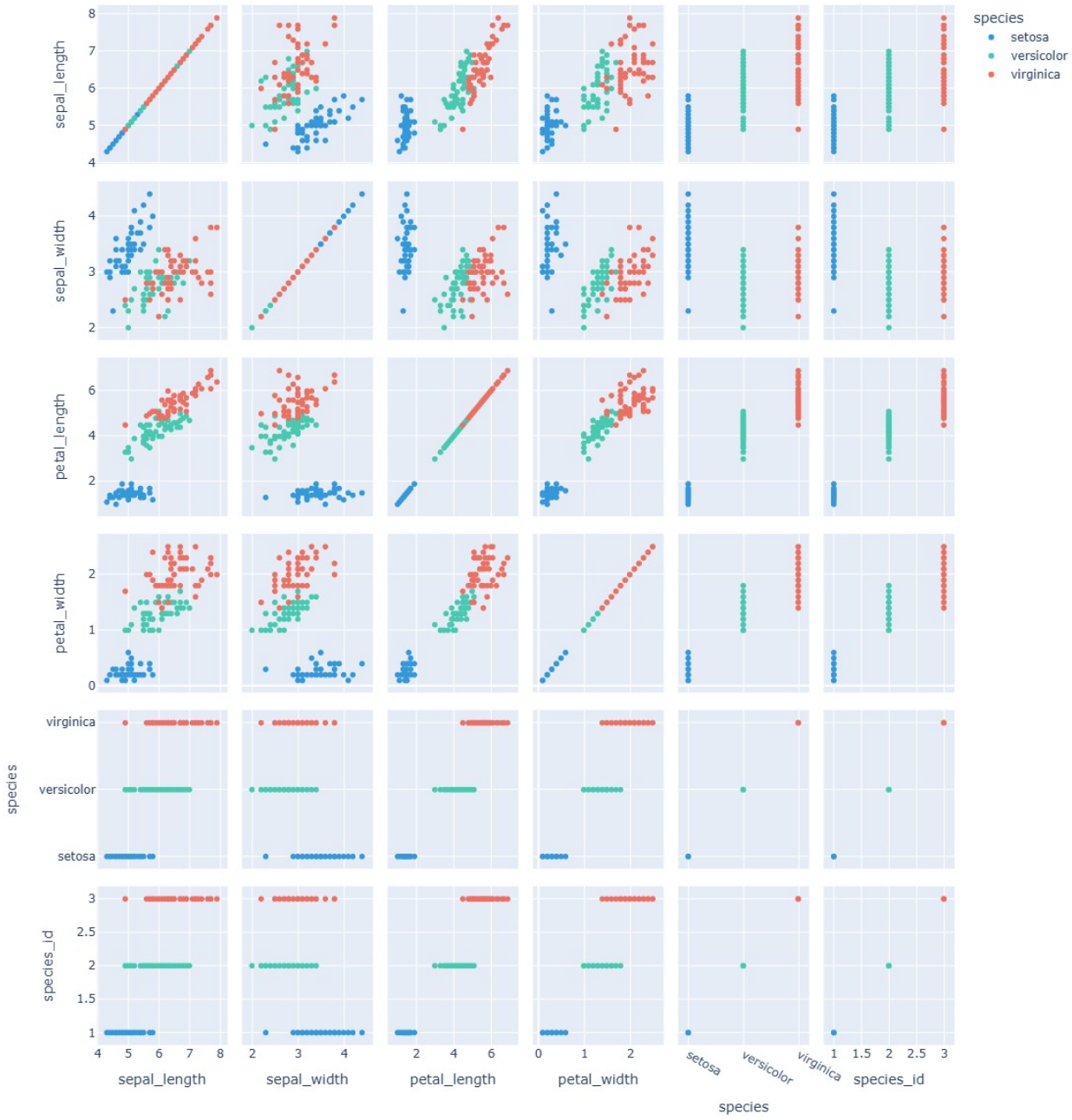
Example:

```
import plotly.express as px
from sklearn.datasets import load_iris
import pandas as pd

# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target

# Create a scatter matrix
fig = px.scatter_matrix(df, dimensions=iris.feature_names,
color='species',
                        title='Iris Dataset Scatter Matrix')
fig.show()
```

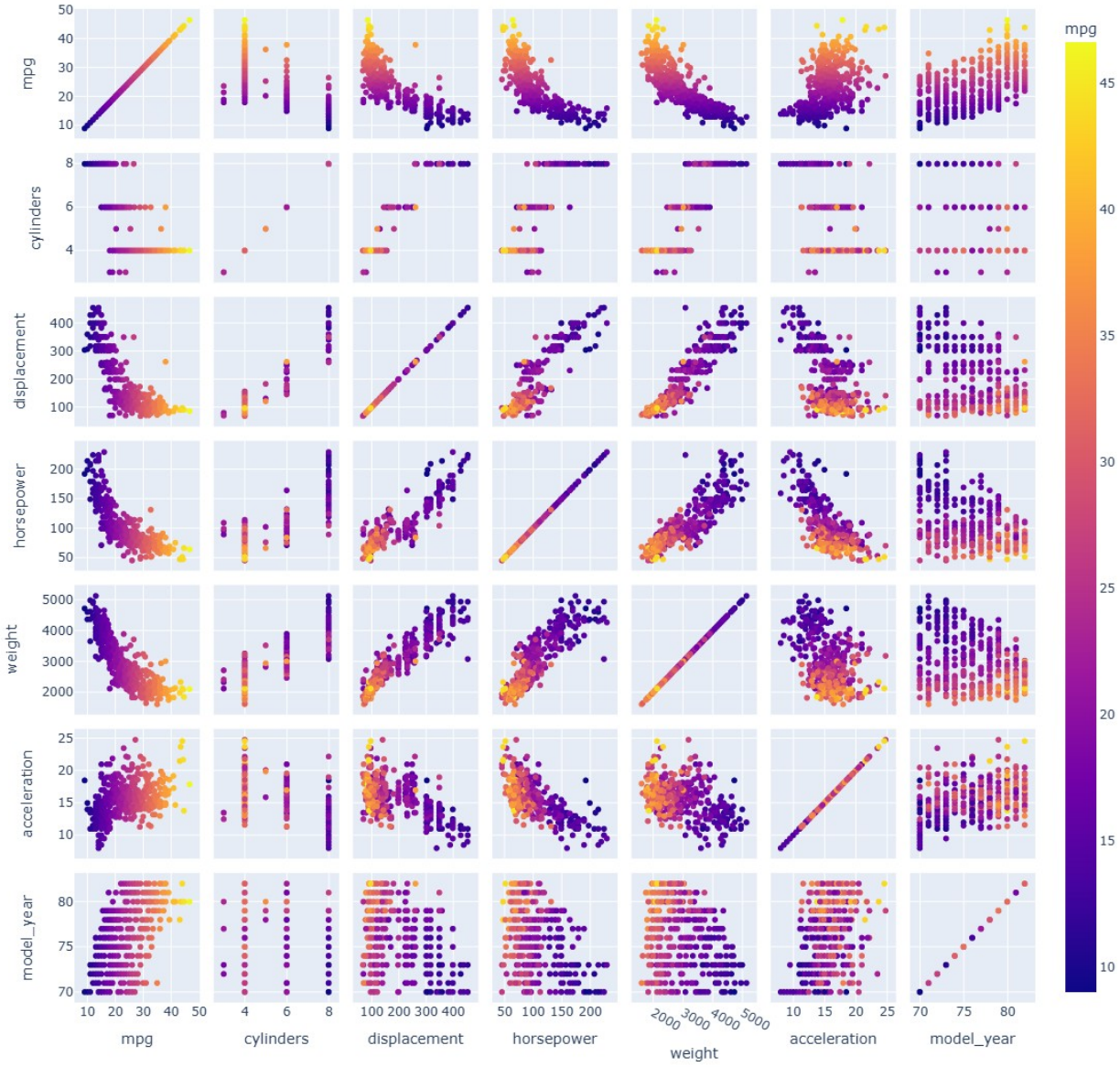
Summary:

- **Effective for visualizing relationships** among multiple variables in a dataset.
- **Highly customizable** with options for colors, symbols, and diagonal plot types.
- Great for **identifying correlations, trends, and outliers** in multivariate datasets.

```
fig =px.scatter_matrix(iris,
color='species',color_discrete_sequence=['#3498db','#48c9b0','#ec7063'
])

fig.update_layout(
    width=1200,
    height=1200,
    autosize=False
)
fig.show()
```




```
fig= px.scatter_matrix(mpg.drop(['name','origin'],axis=1),color='mpg')
fig.update_layout(
    width=1100,
    height=1100,
    autosize=False
)
fig.show()
```



Map Scatter Plot in Plotly Express

- **What it is:** A map scatter plot visualizes data points on a geographic map, using their coordinates (latitude and longitude) to represent their location. Each point can be customized based on other variables, such as size and color.

Key Features:

- **Basic Syntax:**

```
fig = px.scatter_geo(df, lat='latitude_column',
lon='longitude_column',
                     size='size_column', color='color_column',
```

```
                          hover_name='hover_name_column',
                          title='Map Scatter Plot')
    fig.show()
```

Common Uses:

- **Geographic Distribution:** Ideal for visualizing how data points are distributed geographically, such as population density or sales figures across different regions.
- **Spatial Analysis:** Useful for analyzing relationships and patterns based on location, helping to identify geographic trends.

Customization:

- **Marker Size:** Use the `size` argument to represent another variable, making it easier to understand the magnitude of different points.
- **Color Scale:** Customize colors based on a variable to show different categories or intensities using the `color` argument.
- **Hover Information:** Use the `hover_name` argument to display additional information when hovering over the points.

Example:

```
import plotly.express as px

# Sample dataset with geographical coordinates
data = {
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston',
'Phoenix'],
    'Latitude': [40.7128, 34.0522, 41.8781, 29.7604, 33.4484],
    'Longitude': [-74.0060, -118.2437, -87.6298, -95.3698, -112.0740],
    'Population': [8419600, 3980400, 2716000, 2328000, 1664000]
}

df = pd.DataFrame(data)

# Create a map scatter plot
fig = px.scatter_geo(df, lat='Latitude', lon='Longitude',
                     size='Population', color='City',
                     hover_name='City',
                     title='Map Scatter Plot of US Cities')
fig.show()
```

Summary:

- **Effective for visualizing the geographic distribution** of data points based on their coordinates.
- **Highly customizable** with options for marker size, colors, and hover information.
- Great for **understanding geographic trends** and spatial relationships in datasets.

```
year_2007 = px.data.gapminder().query('year==2007')
```

```
px.scatter_geo(year_2007,locations='iso_alpha',color='continent',

hover_name='country',hover_data=year_2007.columns,size='pop',
              projection='orthographic')
```

Choropleth Map in Plotly Express

- **What it is:** A choropleth map is a type of geographical map where regions are shaded or patterned in proportion to the value of a particular variable being represented, such as population density, election results, or economic indicators. It provides a visual representation of data across geographical areas.

Key Features:

- **Basic Syntax:**

```
fig = px.choropleth(df, geojson=geojson_data,
locations='location_column',
                    color='value_column',
                    hover_name='hover_name_column',
                    title='Choropleth Map Title',
                    color_continuous_scale='Viridis')
fig.update_geos(fitbounds="locations", visible=False)
fig.show()
```

Common Uses:

- **Data Visualization:** Ideal for visualizing statistical data across different regions (e.g., states, countries, districts).
- **Comparison of Regions:** Useful for comparing values between different geographical areas.

Customization:

- **Color Scale:** Change the color scale using `color_continuous_scale` to enhance visual appeal and clarity (e.g., `color_continuous_scale='Blues'`).
- **Hover Information:** Display additional information with the `hover_name` argument to provide context for each region.

- **Geojson Data:** Use GeoJSON files for custom geographical boundaries, allowing for more detailed and specific mapping.

Example:

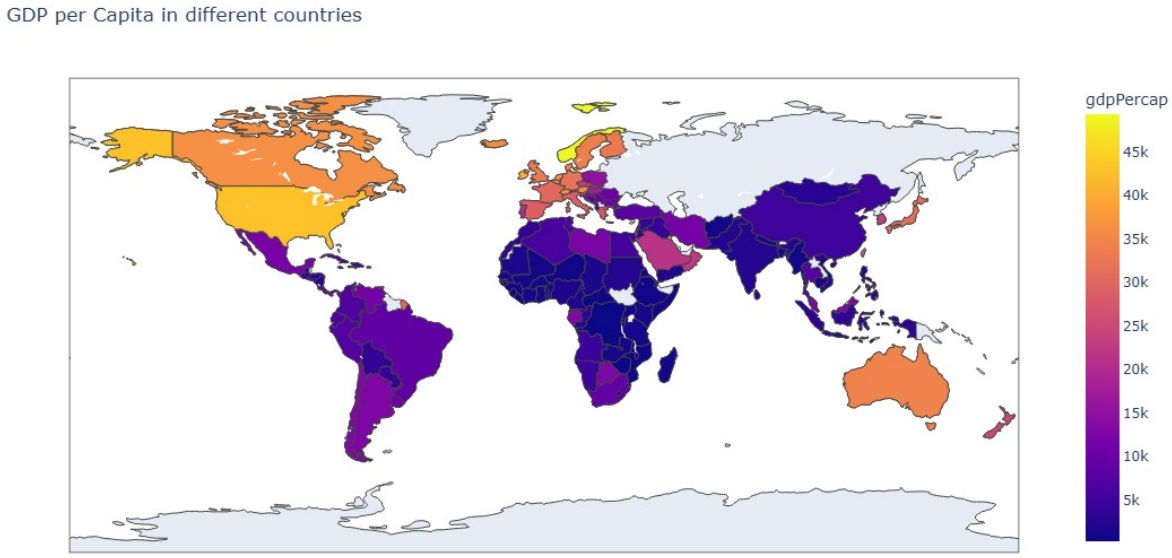
```
import plotly.express as px
import pandas as pd

# Sample dataset
data = {
    'Country': ['USA', 'Canada', 'Mexico', 'Brazil', 'Argentina'],
    'Value': [331, 38, 128, 213, 45]  # Example data: population in
millions
}

df = pd.DataFrame(data)

# World map GeoJSON
import plotly.express as px
import json
import requests

# Get the GeoJSON data
url =
"https://raw.githubusercontent.com/python-visualization/folium/master/
examples/data/us-states.json"
geojson_data = requests.get(url).json()

# Create a choropleth map
fig = px.choropleth(df, geojson=geojson_data,
                    locations='Country',
                    color='Value',
                    hover_name='Country',
                    title='Choropleth Map of Country Population',
                    color_continuous_scale='Viridis')
fig.update_geos(fitbounds="locations", visible=False)
fig.show()
```

Summary:

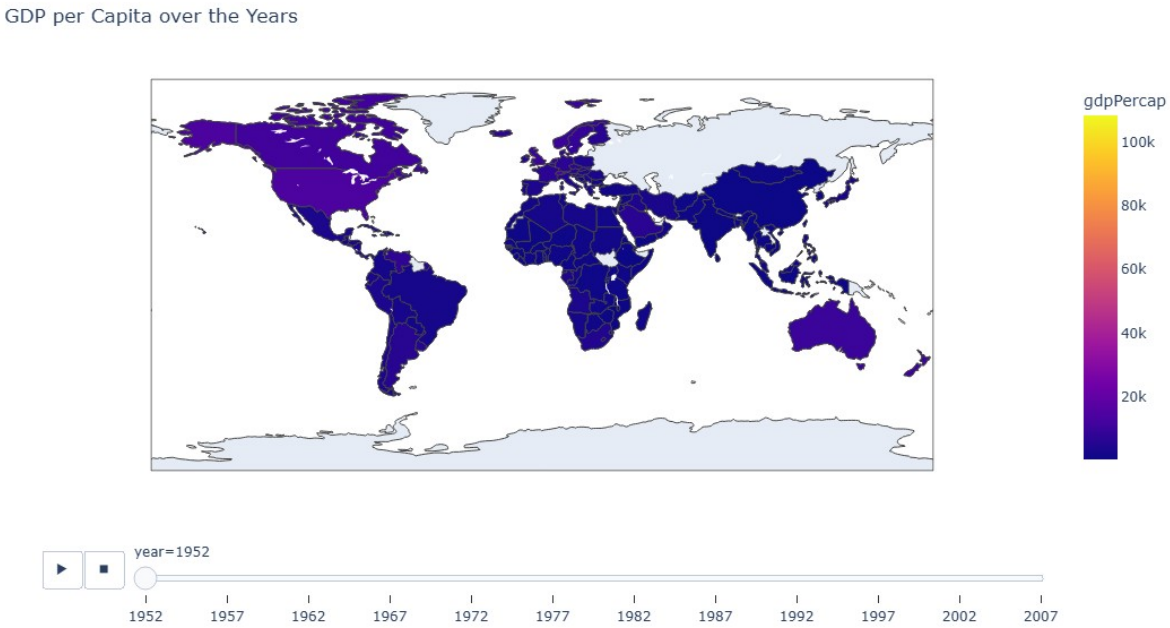
- **Effective for visualizing data across geographical regions**, highlighting patterns and trends.
- **Highly customizable** with options for color scales, hover information, and geographical boundaries.
- Great for **comparing and analyzing data** based on geographical distribution.

```
fig = px.choropleth(year_2007,
locations='iso_alpha',color='gdpPercap',hover_data=year_2007.columns,
                    title='GDP per Capita in different countries')
fig.update_layout(
    width=1000,
```

```
    height=600,
    autosize=True
)
fig.show()
```




```
fig = px.choropleth(gap, locations='iso_alpha',color='gdpPercap',

hover_data=gap.columns,animation_frame='year',title='GDP per Capita
over the Years')
fig.update_layout(
    width=1000,
    height=600,
    autosize=True
)
fig.show()
```



Polar Charts in Plotly Express

- **What it is:** A polar chart is a type of chart that represents data in a circular format, where each point is defined by an angle and a radius. It's commonly used to visualize data with a cyclic nature, such as wind direction, seasonal trends, or any data that can be represented in a circular layout.

Key Features:

- **Types of Polar Charts:**
  - **Bar Polar Chart:** Uses bars to represent data, similar to a bar chart but in a circular layout.
  - **Scatter Polar Chart:** Displays individual data points in a polar coordinate system.

Basic Syntax for Bar Polar Chart:

```
fig = px.bar_polar(df, r='radius_column', theta='angle_column',
                   color='color_column',
                   title='Polar Bar Chart Title')
fig.show()
```

Basic Syntax for Scatter Polar Chart:

```
fig = px.scatter_polar(df, r='radius_column', theta='angle_column',
                       color='color_column',
                       title='Polar Scatter Chart Title')
fig.show()
```

Common Uses:

- **Wind Direction:** Ideal for visualizing wind speed and direction in meteorological data.

- **Seasonal Trends:** Useful for representing data that varies cyclically over time, like sales by month or time of day.

Customization:

- **Color Coding:** Use the `color` argument to differentiate categories within the data.
- **Markers and Lines:** Customize the appearance of points or lines in scatter plots using additional arguments.
- **Add Titles and Labels:** Include titles and axis labels to provide context and enhance understanding.

Example of Polar Bar Chart:

```
import plotly.express as px
import pandas as pd

# Sample data
data = {
    'direction': ['N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW'],
    'frequency': [15, 25, 30, 20, 10, 5, 10, 5]
}

df = pd.DataFrame(data)

# Create a polar bar chart
fig = px.bar_polar(df, r='frequency', theta='direction',
                   title='Wind Frequency by Direction')
fig.show()
```

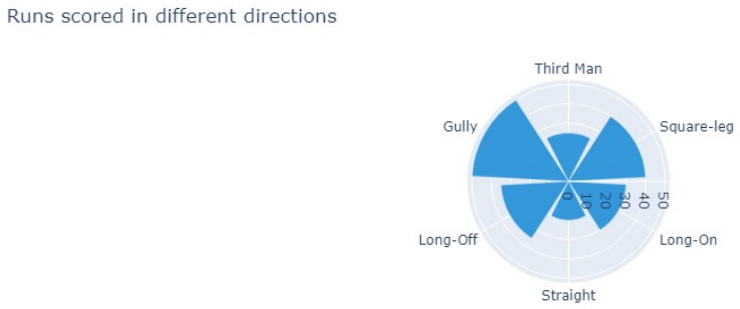
Example of Polar Scatter Chart:

```
import plotly.express as px
import pandas as pd

# Sample data
data = {
    'angle': [0, 45, 90, 135, 180, 225, 270, 315],
    'radius': [10, 15, 20, 25, 30, 25, 20, 15],
    'strength': [1, 2, 3, 4, 5, 4, 3, 2]
}

df = pd.DataFrame(data)

# Create a polar scatter chart
fig = px.scatter_polar(df, r='radius', theta='angle',
                       color='strength',
                       title='Strength by Angle')
fig.show()
```

Summary:

- **Ideal for visualizing data** with cyclic or angular relationships.
- **Highly customizable** with options for colors, markers, and titles.
- Great for representing **trends and distributions** in a visually appealing circular format.

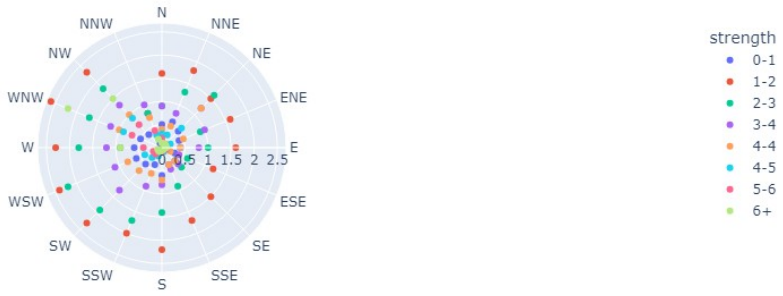
```
data = {'direction':['Third Man','Square-leg','Long-
On','Straight','Long-Off','Gully'],
       'runs':[25,40,30,20,35,50]}
cric = pd.DataFrame(data)

px.bar_polar(cric,r='runs',theta='direction',color_discrete_sequence=[
'#3498db'],
            title='Runs scored in different directions',
            # template='plotly_dark'
            )
#
color_discrete_sequence=[['#cd6155','#2980b9','#3498db','#16a085','#f3
9c12','#d35400']]
```




```
wind.head()
```

	direction	strength	frequency
0	N	0-1	0.5
1	NNE	0-1	0.6
2	NE	0-1	0.5
3	ENE	0-1	0.4
4	E	0-1	0.4

```
# scatterpolar

px.scatter_polar(wind, r='frequency',theta='direction',
color='strength')
# template='plotly_dark'
```



Facet Plot in Plotly Express

- **What it is:** A facet plot, also known as a small multiples plot, is a grid of plots that represent different subsets of data based on specific categorical variables. Each subplot (or facet) shows the same type of plot for a different category, allowing for easy comparison across multiple groups.

Key Features:

- **Basic Syntax:**

```
fig = px.scatter(df, x='x_column', y='y_column',
                 facet_col='facet_column',
                 facet_row='facet_row_column')
fig.show()
```

Common Uses:

- **Comparison Across Groups:** Ideal for comparing distributions, trends, or relationships between variables across different categories (e.g., species in a dataset, different time periods, etc.).
- **Exploratory Data Analysis (EDA):** Useful in EDA to visualize how relationships change across different subsets of data.

Customization:

- **Subplot Layout:** Use `facet_row` and `facet_col` to create a grid layout of subplots.
- **Titles and Labels:** Add titles and labels for each subplot to enhance clarity.
- **Coloring and Markers:** Customize colors and markers for better differentiation between categories.

Example:

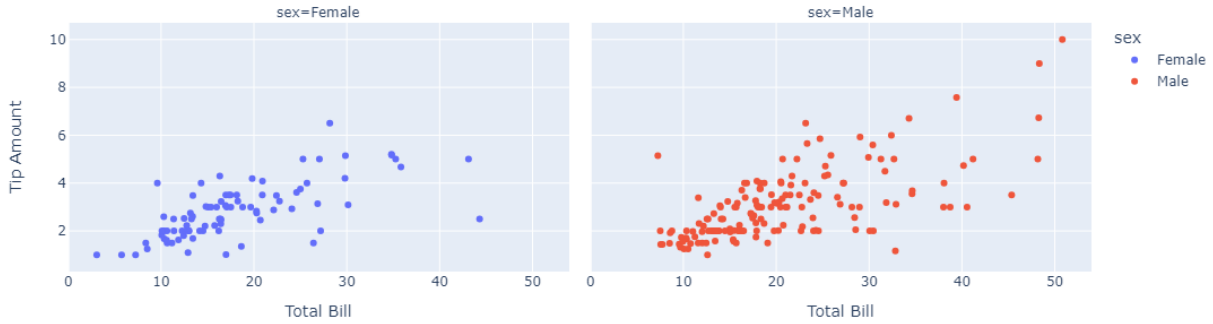
```
import plotly.express as px
import pandas as pd

# Sample dataset
df = px.data.iris()  # Load Iris dataset
```

```
# Create a facet plot (scatter plot in this case)
fig = px.scatter(df, x='sepal_width', y='sepal_length',
                 color='species',
                 facet_col='species',
                 title='Iris Dataset: Sepal Dimensions by Species')
fig.show()
```

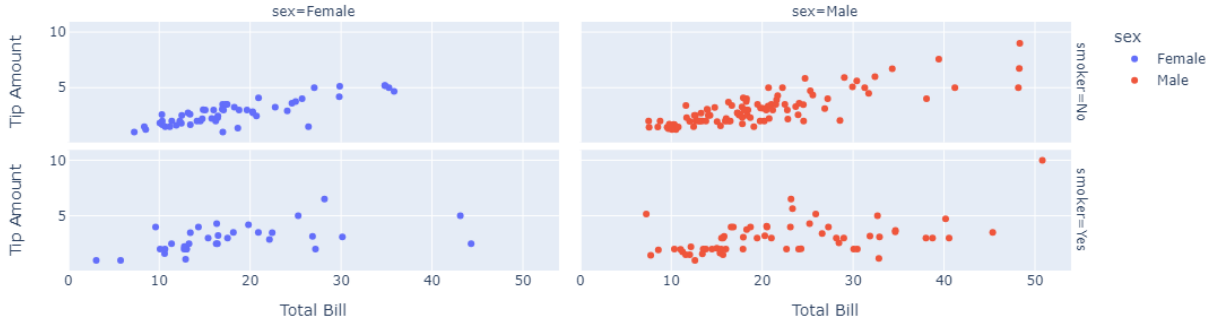
Summary:

- **Effective for comparing data** across multiple categories in a structured way.
- **Highly customizable** to suit specific visualization needs.
- Great for **detailed exploratory analysis**, allowing for quick visual insights across different subsets of data.

```
px.scatter(tips,x='total_bill',y='tip',color='sex',facet_col='sex',labels={'total_bill':'Total Bill','tip':"Tip Amount"})
```




```
px.scatter(tips,x='total_bill',y='tip',color='sex',facet_col='sex',facet_row='smoker',
           labels={'total_bill':'Total Bill','tip':"Tip Amount"})
# symbol='smoker'
```




```
attention.drop('Unnamed: 0',axis=1,inplace=True)
```

```
px.line(attention,x='solutions',y='score',
facet_col='subject',facet_col_wrap=5,title='Scores based on
Attention')
```