Plotly Express in Python

Plotly Express is a terse, consistent, high-level API for creating figures.

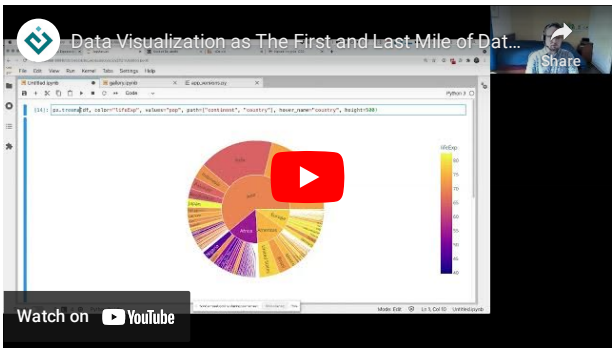
New to Plotly?

Overview

The `plotly.express` module (usually imported as `px`) contains functions that can create entire figures at once, and is referred to as Plotly Express or PX. Plotly Express is a built-in part of the `plotly` library, and is the recommended starting point for creating most common figures. Every Plotly Express function uses graph objects internally and returns a `plotly.graph_objects.Figure` instance. Throughout the `plotly` documentation, you will find the Plotly Express way of building figures at the top of any applicable page, followed by a section on how to use graph objects to build similar figures. Any figure created in a single function call with Plotly Express could be created using graph objects alone, but with between 5 and 100 times more code.

Plotly Express provides more than 30 functions for creating different types of figures. The API for these functions was carefully designed to be as consistent and easy to learn as possible, making it easy to switch from a scatter plot to a bar chart to a histogram to a sunburst chart throughout a data exploration session. *Scroll down for a gallery of Plotly Express plots, each made in a single function call.*

Here is a talk from the SciPy 2021 conference that gives a good introduction to Plotly Express and Dash:



Plotly Express currently includes the following functions:

- **Basics**: scatter, line, area, bar, funnel, timeline
- **Part-of-Whole**: pie, sunburst, treemap, icicle, funnel_area
- **1D Distributions**: histogram, box, violin, strip, ecdf
- **2D Distributions**: density_heatmap, density_contour
- **Matrix or Image Input**: imshow
- **3-Dimensional**: scatter_3d, line_3d
- **Multidimensional**: scatter_matrix, parallel_coordinates, parallel_categories
- **Tile Maps**: scatter_mapbox, line_mapbox, choropleth_mapbox, density_mapbox
- **Outline Maps**: scatter_geo, line_geo, choropleth
- **Polar Charts**: scatter_polar, line_polar, bar_polar
- **Ternary Charts**: scatter_ternary, line_ternary

High-Level Features

The Plotly Express API in general offers the following features:

- **A single entry point into** `plotly`: just `import plotly.express as px` and get access to all the plotting functions, plus built-in demo datasets under `px.data` and built-in color scales and sequences under `px.color`. Every PX function returns a `plotly.graph_objects.Figure` object, so you can edit it using all the same methods like `update_layout` and `add_trace`.
- **Sensible, Overridable Defaults**: PX functions will infer sensible defaults wherever possible, and will always let you override them.
- **Flexible Input Formats**: PX functions accept input in a variety of formats, from `lists` and `dicts` to long-form or wide-form Pandas DataFrames to numpy arrays and xarrays to GeoPandas GeoDataFrames.
- **Automatic Trace and Layout configuration**: PX functions will create one trace per animation frame for each unique combination of data values mapped to discrete color, symbol, line-dash, facet-row and/or facet-column. Traces' legendgroup and showlegend attributes are set such that only one legend item appears per unique combination of discrete color, symbol and/or line-dash. Traces are automatically linked to a correctly-configured subplot of the appropriate type.

- **Automatic Figure Labelling**: PX functions label axes, legends and colorbars based in the input `DataFrame` or `xarray`, and provide extra control with the `labels` argument.
- **Automatic Hover Labels**: PX functions populate the hover-label using the labels mentioned above, and provide extra control with the `hover_name` and `hover_data` arguments.
- **Styling Control**: PX functions read styling information from the default figure template, and support commonly-needed cosmetic controls like `category_orders` and `color_discrete_map` to precisely control categorical variables.
- **Uniform Color Handling**: PX functions automatically switch between continuous and categorical color based on the input type.
- **Faceting**: the 2D-cartesian plotting functions support row, column and wrapped facetting with `facet_row`, `facet_col` and `facet_col_wrap` arguments.
- **Marginal Plots**: the 2D-cartesian plotting functions support marginal distribution plots with the `marginal`, `marginal_x` and `marginal_y` arguments.
- **A Pandas backend**: the 2D-cartesian plotting functions are available as a Pandas plotting backend so you can call them via `df.plot()`.
- **Trendlines**: `px.scatter` supports built-in trendlines with accessible model output.
- **Animations**: many PX functions support simple animation support via the `animation_frame` and `animation_group` arguments.
- **Automatic WebGL switching**: for sufficiently large scatter plots, PX will automatically use WebGL for hardware-accelerated rendering.

Plotly Express in Dash

Dash is the best way to build analytical apps in Python using Plotly figures. To run the app below, run `pip install dash`, click "Download" to get the code and run `python app.py`.

Get started with the official Dash docs and **learn how to effortlessly** style **&** deploy **apps like this with** Dash Enterprise**.**

```
from dash import Dash, dcc, html, Input, Output
import plotly.express as px

app = Dash(__name__)

app.layout = html.Div([
    html.H4('Analysis of Iris data using scatter matrix'),
    dcc.Dropdown(
        id="dropdown",
        options=['sepal_length', 'sepal_width', 'petal_length', 'petal_width'],
        value=['sepal_length', 'sepal_width'],
        multi=True
    ),
    dcc.Graph(id="graph"),
])


@app.callback(
    Output("graph", "figure"),
    Input("dropdown", "value"))
def update_bar_chart(dims):
    df = px.data.iris() # replace with your own data source
    fig = px.scatter_matrix(
        df, dimensions=dims, color="species")
    return fig
```

DOWNLOAD

Analysis of Iris data using scatter matrix

× sepal_length × sepal_width

WebGL is not supported by your browser - visit https://get.webgl.org for more info

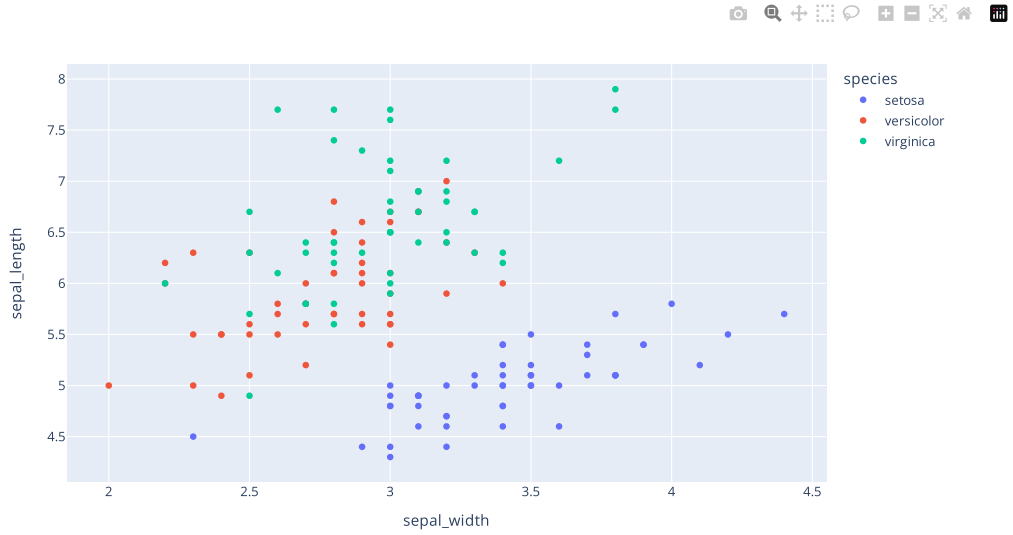
Gallery

The following set of figures is just a sampling of what can be done with Plotly Express.

Scatter, Line, Area and Bar Charts

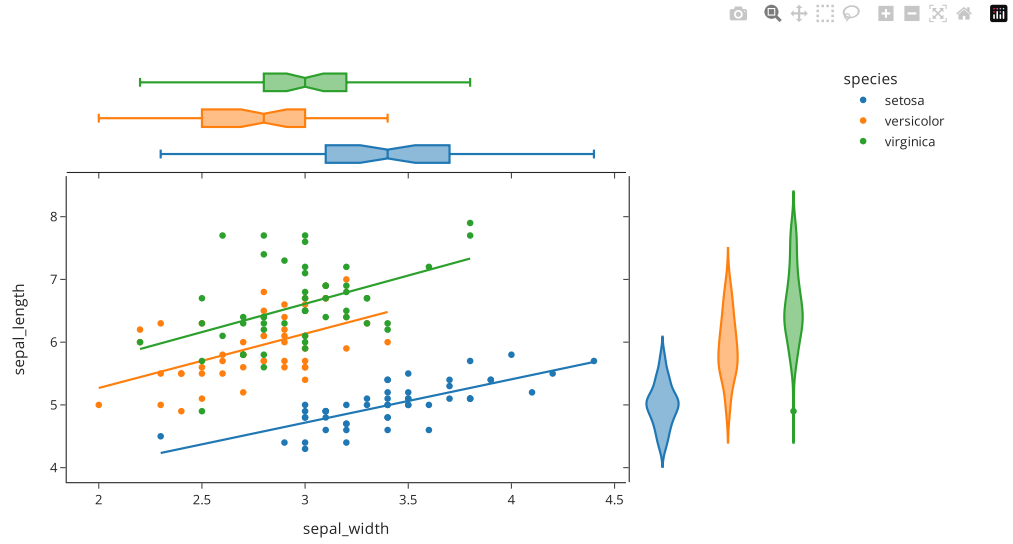
Read more about scatter plots **and** discrete color**.**

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



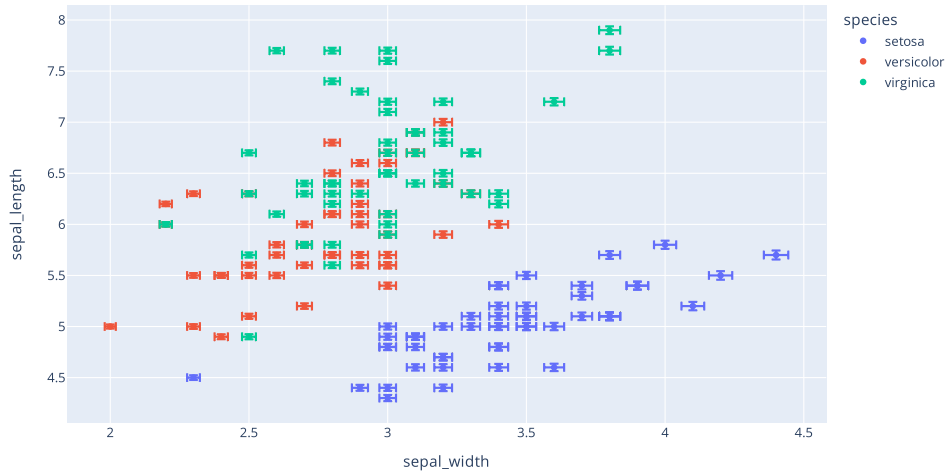
Read more about trendlines **and** templates **and** marginal distribution plots**.**

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", marginal_y="violin",
           marginal_x="box", trendline="ols", template="simple_white")
fig.show()
```



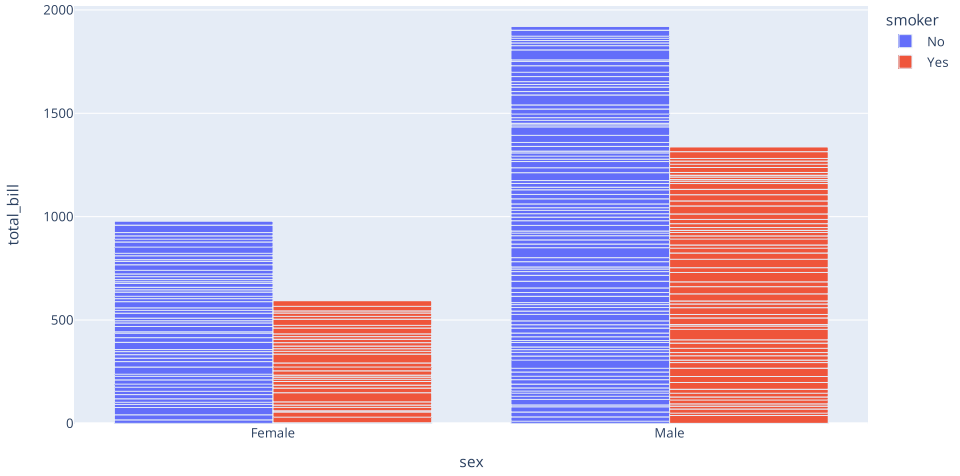
Read more about error bars**.**

```
import plotly.express as px
df = px.data.iris()
df["e"] = df["sepal_width"]/100
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", error_x="e", error_y="e")
fig.show()
```

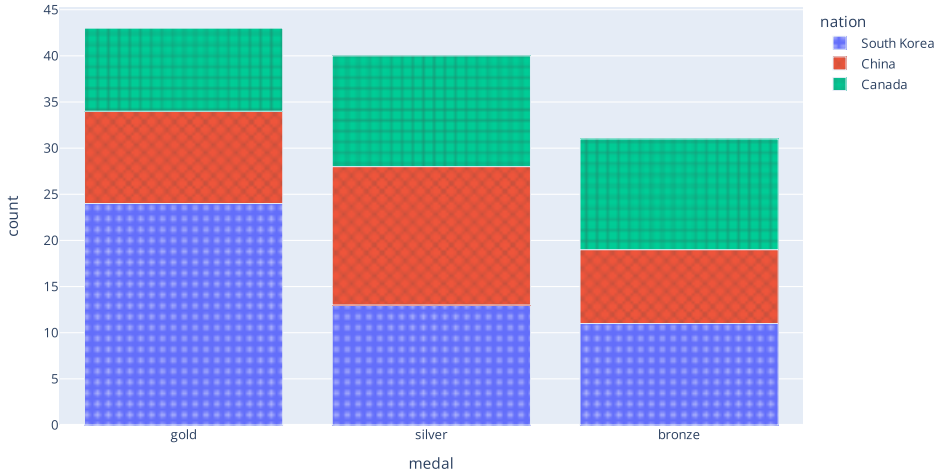


Read more about bar charts.

```
import plotly.express as px
df = px.data.tips()
fig = px.bar(df, x="sex", y="total_bill", color="smoker", barmode="group")
fig.show()
```

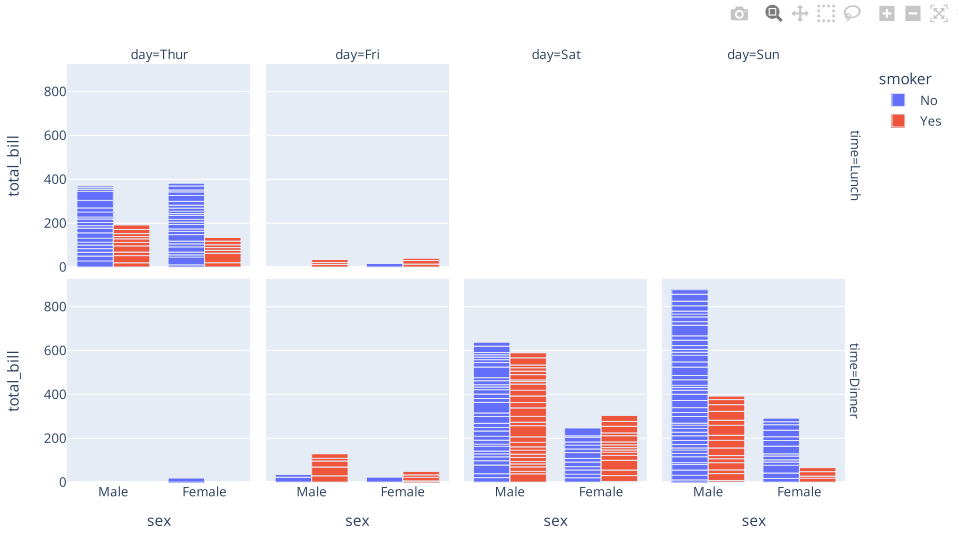



```
import plotly.express as px
df = px.data.medals_long()

fig = px.bar(df, x="medal", y="count", color="nation",
             pattern_shape="nation", pattern_shape_sequence=[".", "x", "+"])
fig.show()
```



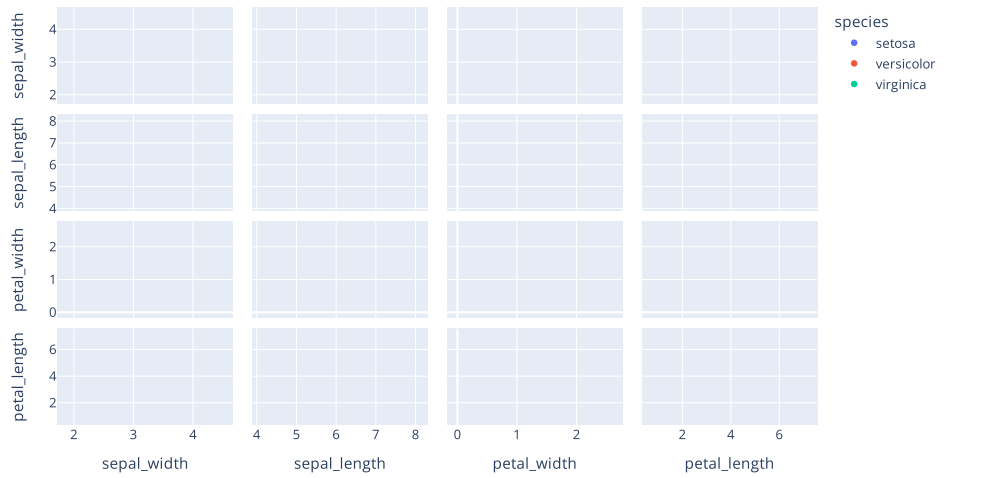
Read more about facet plots.

```
import plotly.express as px
df = px.data.tips()
fig = px.bar(df, x="sex", y="total_bill", color="smoker", barmode="group", facet_row="time", facet_col="day",
       category_orders={"day": ["Thur", "Fri", "Sat", "Sun"], "time": ["Lunch", "Dinner"]})
fig.show()
```



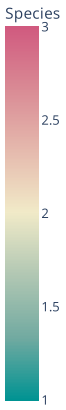
Read more about scatterplot matrices (SPLOMs).

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter_matrix(df, dimensions=["sepal_width", "sepal_length", "petal_width", "petal_length"], color="specie
s")
fig.show()
```

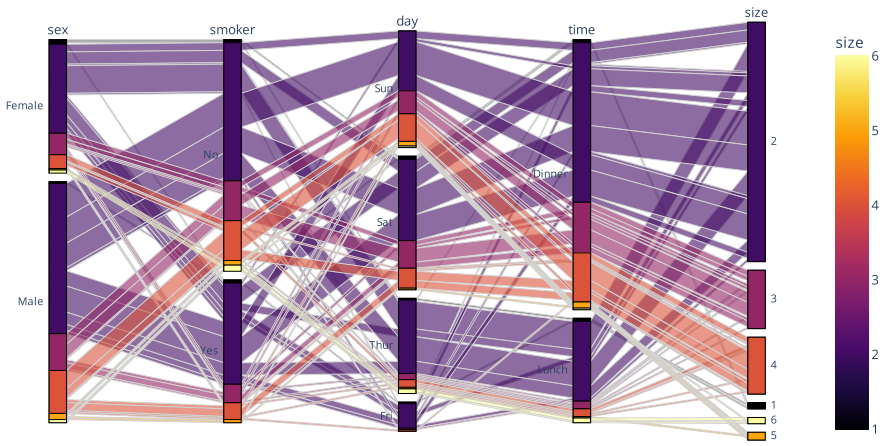


Read more about parallel coordinates **and** parallel categories**, as well as** continuous color**.**

```
import plotly.express as px
df = px.data.iris()
fig = px.parallel_coordinates(df, color="species_id", labels={"species_id": "Species",
                  "sepal_width": "Sepal Width", "sepal_length": "Sepal Length",
                  "petal_width": "Petal Width", "petal_length": "Petal Length", },
                    color_continuous_scale=px.colors.diverging.Tealrose, color_continuous_midpoint=2)
fig.show()
```

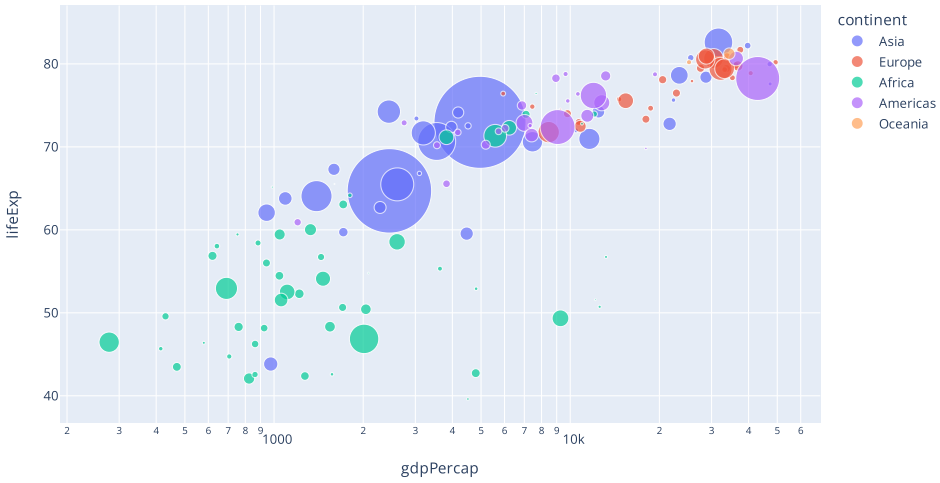



```
import plotly.express as px
df = px.data.tips()
fig = px.parallel_categories(df, color="size", color_continuous_scale=px.colors.sequential.Inferno)
fig.show()
```



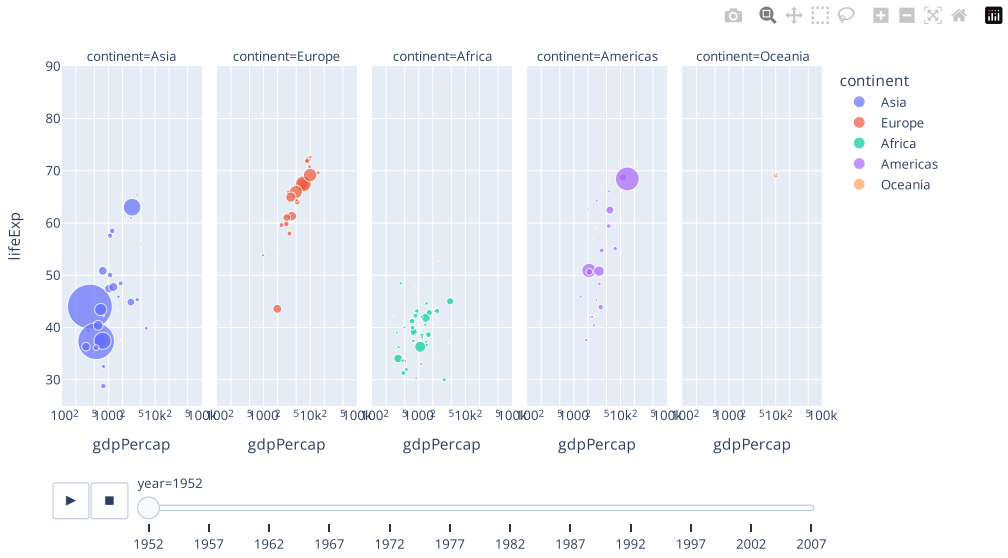
Read more about hover labels.

```
import plotly.express as px
df = px.data.gapminder()
fig = px.scatter(df.query("year==2007"), x="gdpPercap", y="lifeExp", size="pop", color="continent",
           hover_name="country", log_x=True, size_max=60)
fig.show()
```



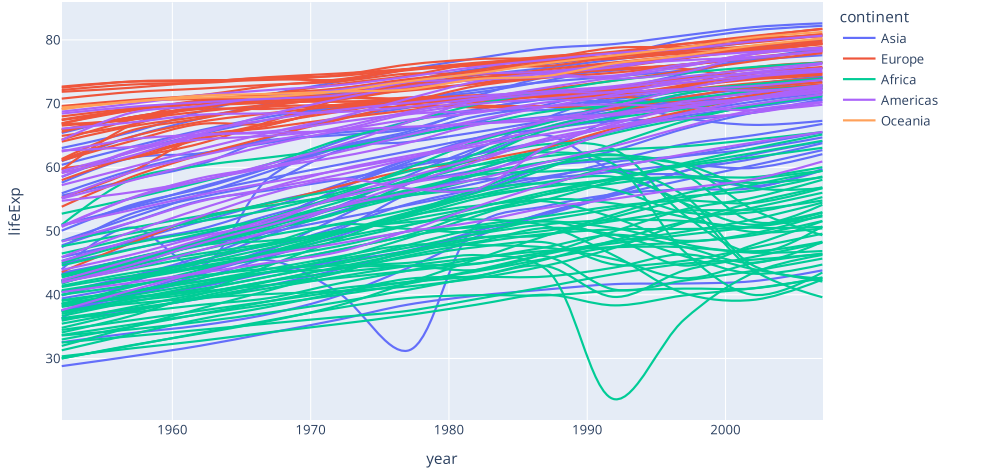
Read more about animations.

```
import plotly.express as px
df = px.data.gapminder()
fig = px.scatter(df, x="gdpPercap", y="lifeExp", animation_frame="year", animation_group="country",
           size="pop", color="continent", hover_name="country", facet_col="continent",
           log_x=True, size_max=45, range_x=[100,100000], range_y=[25,90])
fig.show()
```



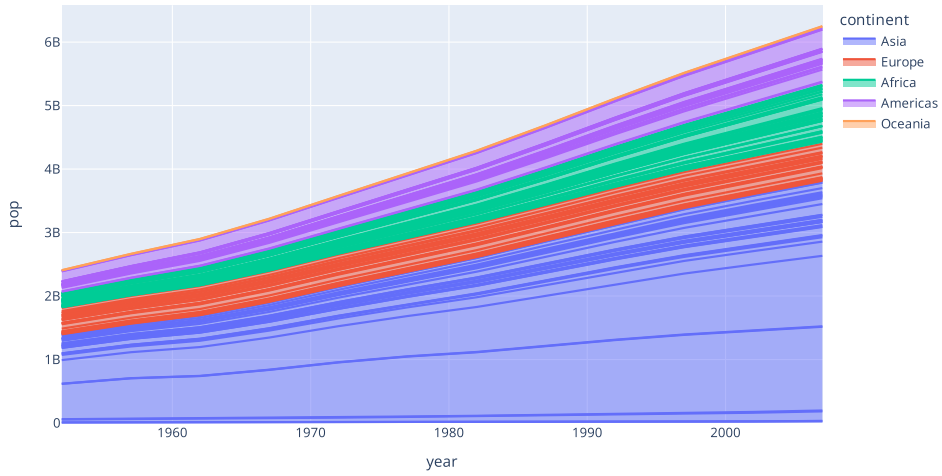
Read more about line charts.

```
import plotly.express as px
df = px.data.gapminder()
fig = px.line(df, x="year", y="lifeExp", color="continent", line_group="country", hover_name="country",
        line_shape="spline", render_mode="svg")
fig.show()
```



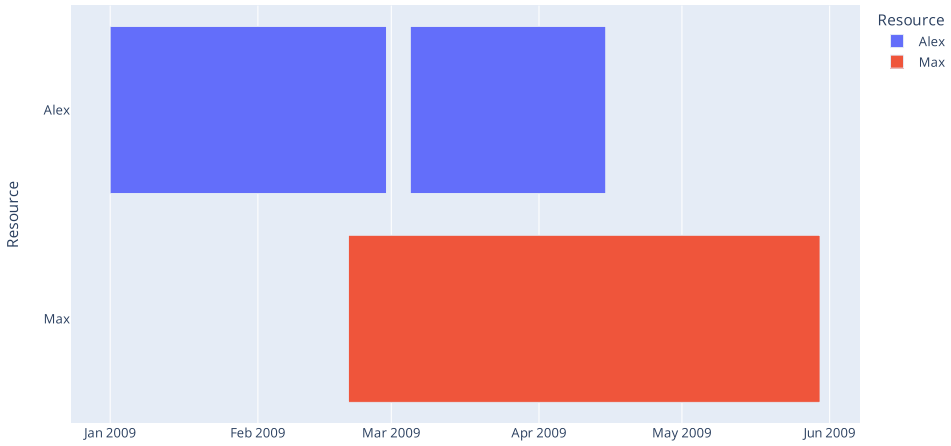
Read more about area charts.

```
import plotly.express as px
df = px.data.gapminder()
fig = px.area(df, x="year", y="pop", color="continent", line_group="country")
fig.show()
```



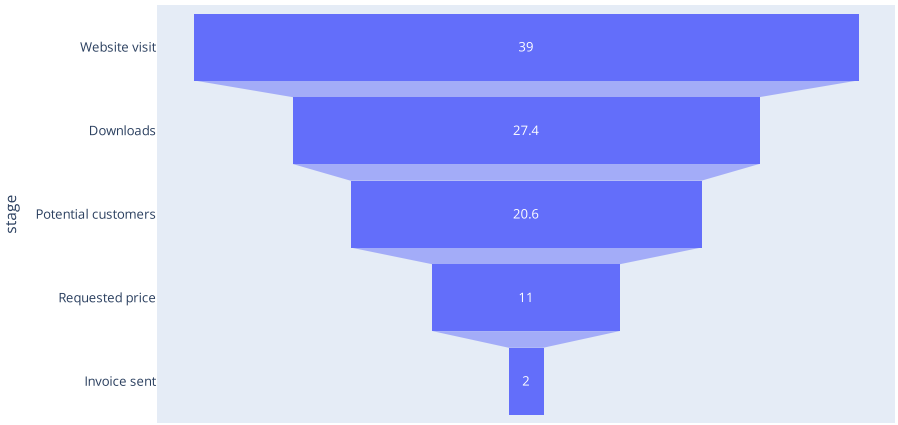
Read more about timeline/Gantt charts.

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

df = pd.DataFrame([
    dict(Task="Job A", Start='2009-01-01', Finish='2009-02-28', Resource="Alex"),
    dict(Task="Job B", Start='2009-03-05', Finish='2009-04-15', Resource="Alex"),
    dict(Task="Job C", Start='2009-02-20', Finish='2009-05-30', Resource="Max")
])

fig = px.timeline(df, x_start="Start", x_end="Finish", y="Resource", color="Resource")
fig.show()
```



Read more about funnel charts.

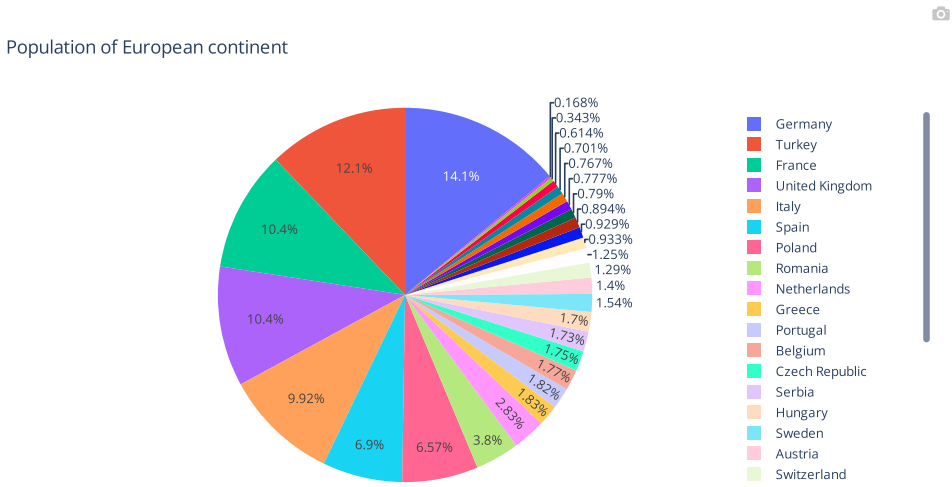
```
import plotly.express as px
data = dict(
    number=[39, 27.4, 20.6, 11, 2],
    stage=["Website visit", "Downloads", "Potential customers", "Requested price", "Invoice sent"])
fig = px.funnel(data, x='number', y='stage')
fig.show()
```



Part to Whole Charts

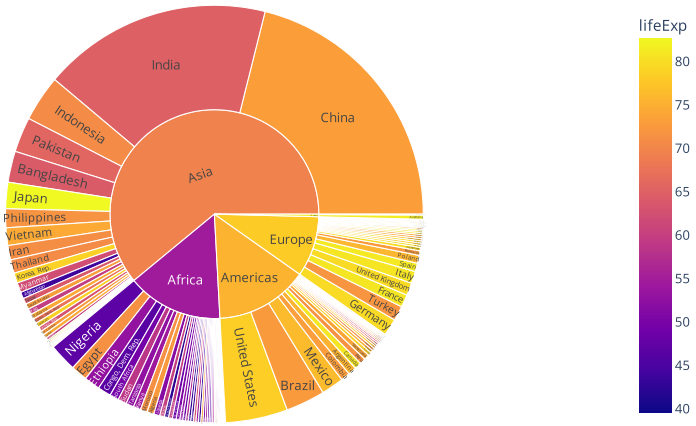
Read more about pie charts.

```
import plotly.express as px
df = px.data.gapminder().query("year == 2007").query("continent == 'Europe'")
df.loc[df['pop'] < 2.e6, 'country'] = 'Other countries' # Represent only large countries
fig = px.pie(df, values='pop', names='country', title='Population of European continent')
fig.show()
```



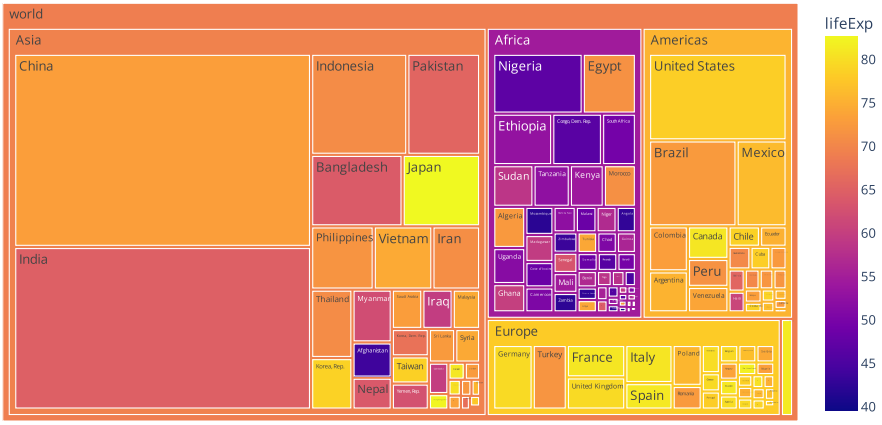
Read more about sunburst charts.

```
import plotly.express as px

df = px.data.gapminder().query("year == 2007")
fig = px.sunburst(df, path=['continent', 'country'], values='pop',
                  color='lifeExp', hover_data=['iso_alpha'])
fig.show()
```



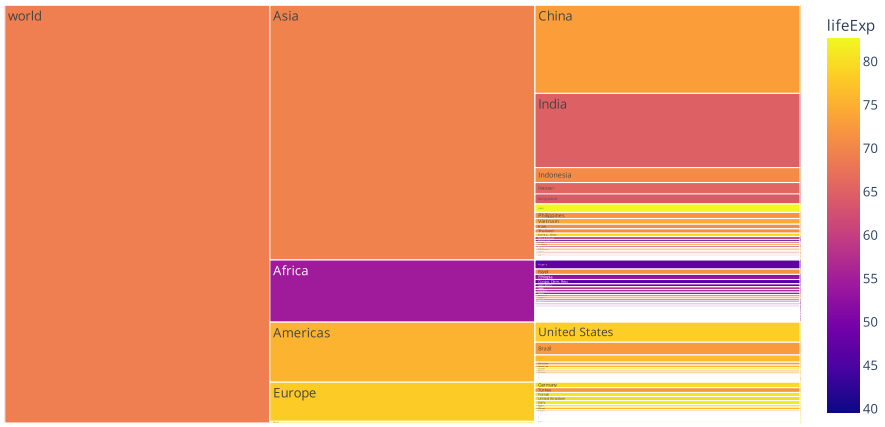
Read more about treemaps.

```
import plotly.express as px
import numpy as np
df = px.data.gapminder().query("year == 2007")
fig = px.treemap(df, path=[px.Constant('world'), 'continent', 'country'], values='pop',
                 color='lifeExp', hover_data=['iso_alpha'])
fig.show()
```



Read more about icicle charts.

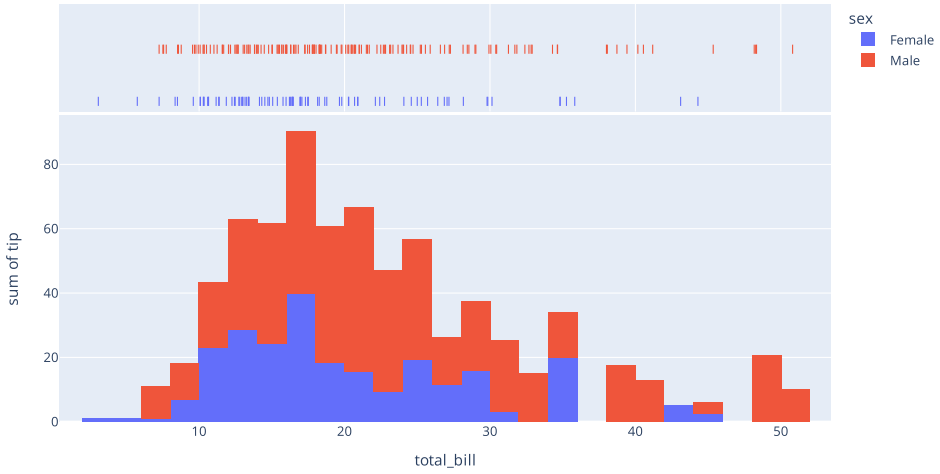
```
import plotly.express as px
import numpy as np
df = px.data.gapminder().query("year == 2007")
fig = px.icicle(df, path=[px.Constant('world'), 'continent', 'country'], values='pop',
                 color='lifeExp', hover_data=['iso_alpha'])
fig.show()
```



Distributions

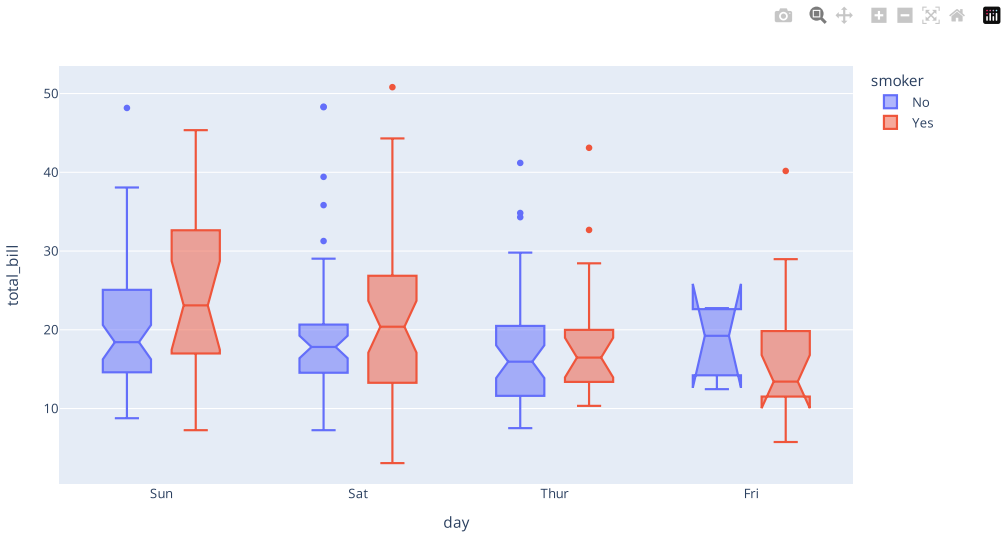
Read more about histograms.

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", y="tip", color="sex", marginal="rug", hover_data=df.columns)
fig.show()
```



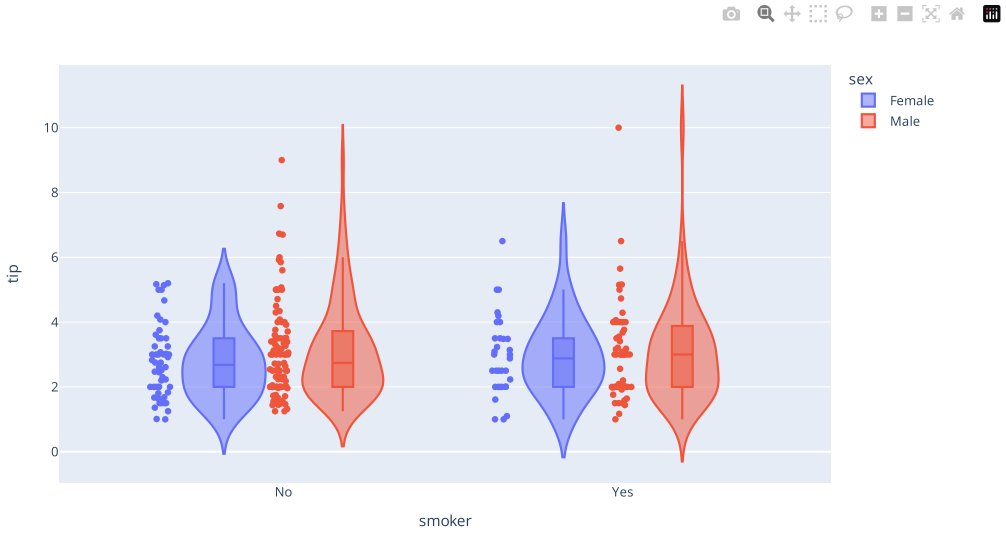
Read more about box plots.

```
import plotly.express as px
df = px.data.tips()
fig = px.box(df, x="day", y="total_bill", color="smoker", notched=True)
fig.show()
```



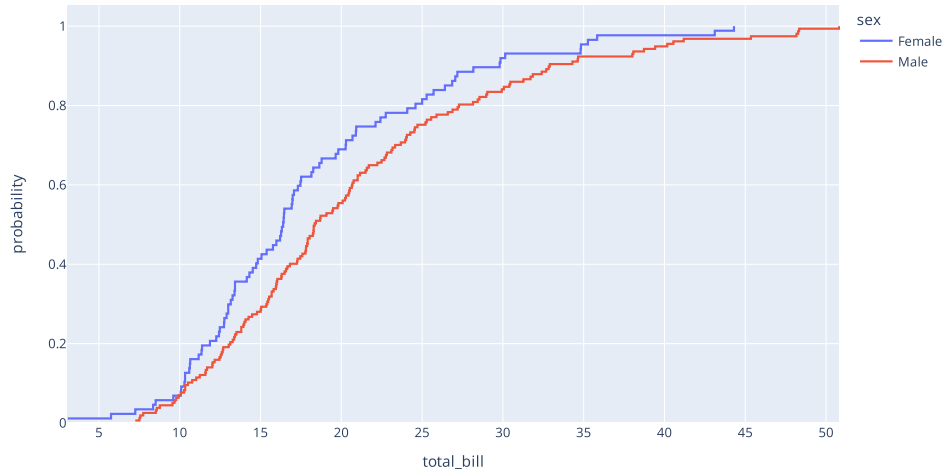
Read more about violin plots.

```
import plotly.express as px
df = px.data.tips()
fig = px.violin(df, y="tip", x="smoker", color="sex", box=True, points="all", hover_data=df.columns)
fig.show()
```



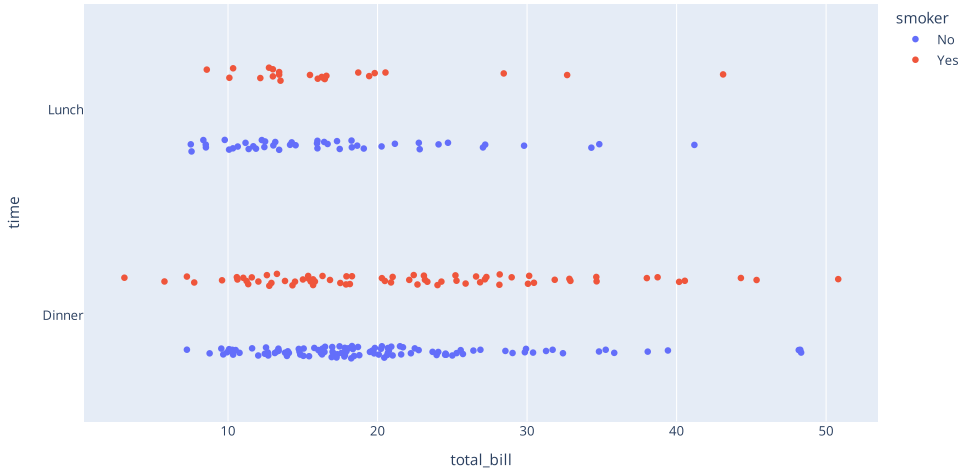
Read more about Empirical Cumulative Distribution Function (ECDF) charts.

```
import plotly.express as px
df = px.data.tips()
fig = px.ecdf(df, x="total_bill", color="sex")
fig.show()
```



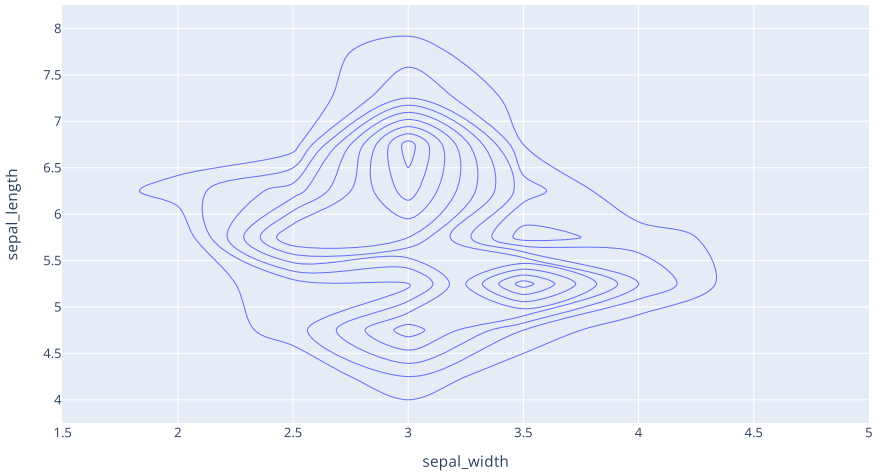
Read more about strip charts.

```
import plotly.express as px
df = px.data.tips()
fig = px.strip(df, x="total_bill", y="time", orientation="h", color="smoker")
fig.show()
```



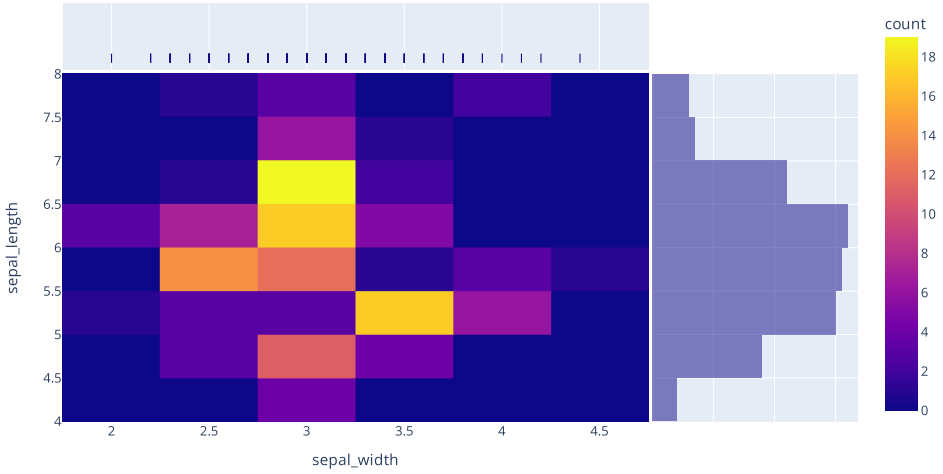
Read more about density contours, also known as 2D histogram contours.

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



Read more about density heatmaps, also known as 2D histograms.

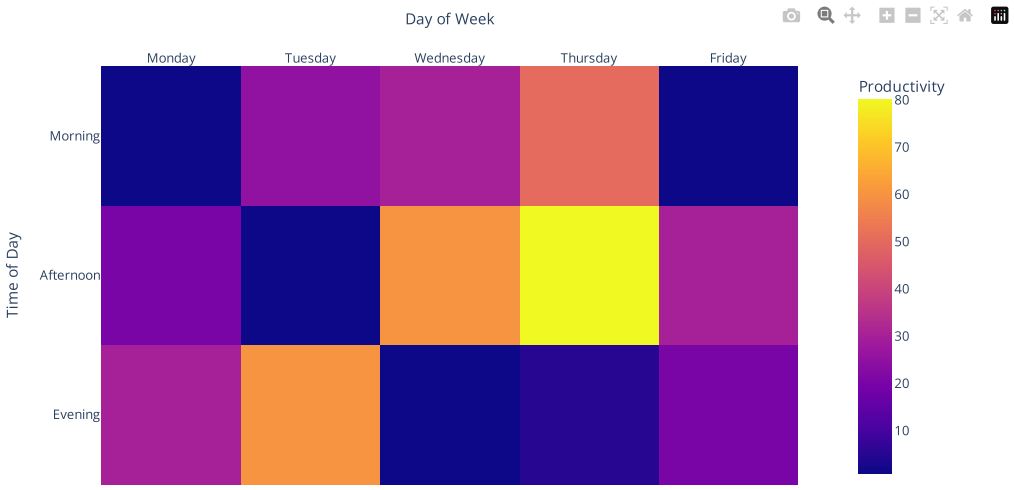
```
import plotly.express as px
df = px.data.iris()
fig = px.density_heatmap(df, x="sepal_width", y="sepal_length", marginal_x="rug", marginal_y="histogram")
fig.show()
```



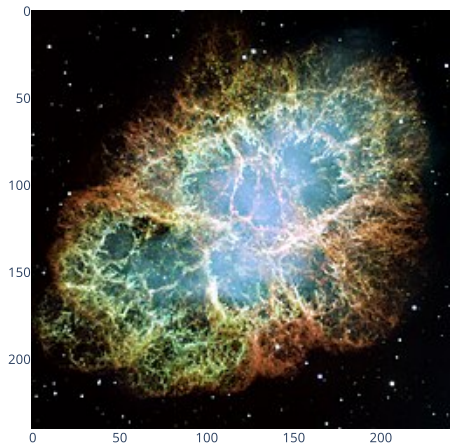
Images and Heatmaps

Read more about heatmaps and images.

```
import plotly.express as px
data=[[1, 25, 30, 50, 1], [20, 1, 60, 80, 30], [30, 60, 1, 5, 20]]
fig = px.imshow(data,
                labels=dict(x="Day of Week", y="Time of Day", color="Productivity"),
                x=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'],
                y=['Morning', 'Afternoon', 'Evening']
               )
fig.update_xaxes(side="top")
fig.show()
```



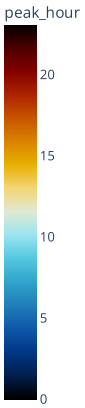

```
import plotly.express as px
from skimage import io
img = io.imread('https://upload.wikimedia.org/wikipedia/commons/thumb/0/00/Crab_Nebula.jpg/240px-Crab_Nebula.jpg')
fig = px.imshow(img)
fig.show()
```



Tile Maps

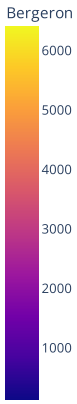
Read more about tile maps **and** point on tile maps**.**

```
import plotly.express as px
df = px.data.carshare()
fig = px.scatter_mapbox(df, lat="centroid_lat", lon="centroid_lon", color="peak_hour", size="car_hours",
                  color_continuous_scale=px.colors.cyclical.IceFire, size_max=15, zoom=10,
                  mapbox_style="carto-positron")
fig.show()
```



Read more about tile map GeoJSON choropleths.

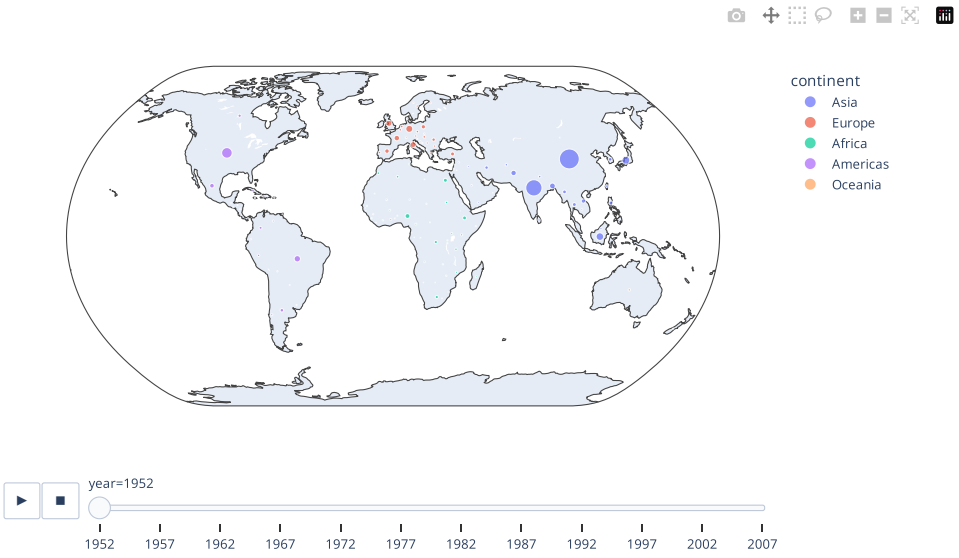
```
import plotly.express as px

df = px.data.election()
geojson = px.data.election_geojson()

fig = px.choropleth_mapbox(df, geojson=geojson, color="Bergeron",
                           locations="district", featureidkey="properties.district",
                           center={"lat": 45.5517, "lon": -73.7073},
                           mapbox_style="carto-positron", zoom=9)
fig.show()
```



Outline Maps

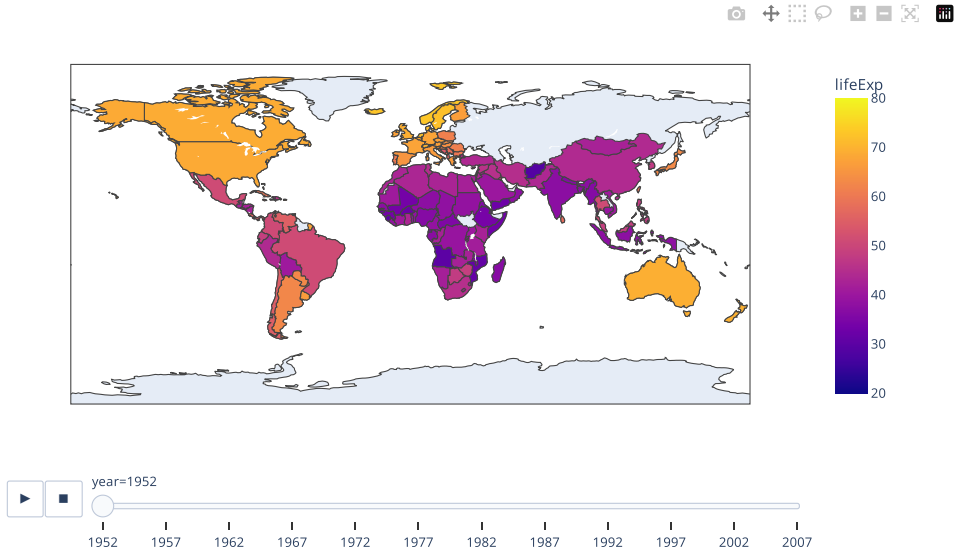
Read more about outline symbol maps.

```
import plotly.express as px
df = px.data.gapminder()
fig = px.scatter_geo(df, locations="iso_alpha", color="continent", hover_name="country", size="pop",
               animation_frame="year", projection="natural earth")
fig.show()
```



Read more about choropleth maps.

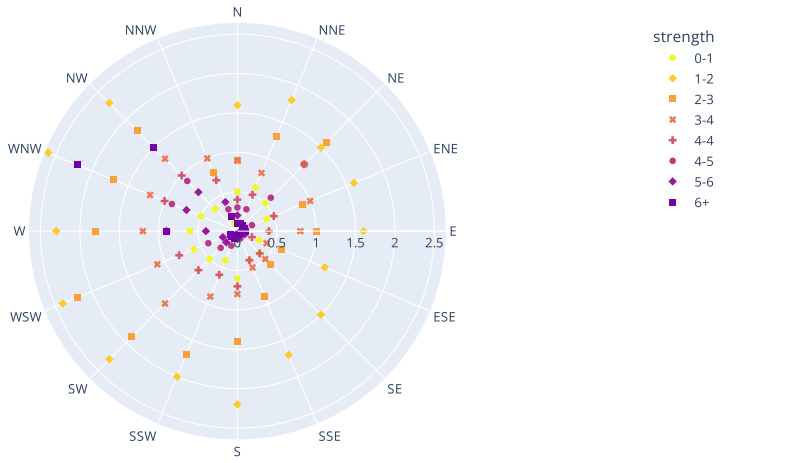
```
import plotly.express as px
df = px.data.gapminder()
fig = px.choropleth(df, locations="iso_alpha", color="lifeExp", hover_name="country", animation_frame="year", range_
color=[20,80])
fig.show()
```



Polar Coordinates

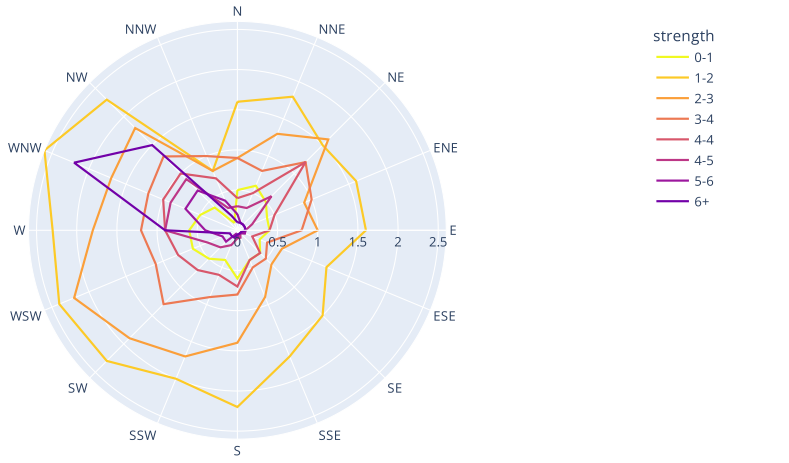
Read more about polar plots.

```
import plotly.express as px
df = px.data.wind()
fig = px.scatter_polar(df, r="frequency", theta="direction", color="strength", symbol="strength",
            color_discrete_sequence=px.colors.sequential.Plasma_r)
fig.show()
```



Read more about radar charts.

```
import plotly.express as px
df = px.data.wind()
fig = px.line_polar(df, r="frequency", theta="direction", color="strength", line_close=True,
            color_discrete_sequence=px.colors.sequential.Plasma_r)
fig.show()
```



Read more about polar bar charts.

```
import plotly.express as px
df = px.data.wind()
fig = px.bar_polar(df, r="frequency", theta="direction", color="strength", template="plotly_dark",
            color_discrete_sequence= px.colors.sequential.Plasma_r)
fig.show()
```